Fast.ai Computational Linear Algebra T extbook

From: fastai/numerical-linear-algebra

You can read an overview of this Numerical Linear Algebra course in this blog post. The course was originally taught in the University of San Francisco MS in Analytics graduate program. Course lecture videos are available on YouTube (note that the notebook numbers and video numbers do not line up, since some notebooks took longer than 1 video to cover).

You can ask questions about the course on our fast.ai forums.

0. Course Logistics ¶

Ask Questions ¶

Let me know how things are going. This is particularly important since I'm new to MSAN, I don't know everything you've seen/haven't seen.

Intro ¶

My background and linear algebra love :

- Swarthmore College : linear algebra convinced me to be a math major! (minors in CS & linguistics) I thought linear algebra was beautiful, but theoretical
- Duke University : Math PhD. Took numerical linear algebra. Enjoyed the course, but not my focus
- Research T riangle Institute : first time using linear algebra in practice (healthcare economics, markov chains)
- Quant : first time working with lots of data, decided to become a data scientist
- Uber : data scientist
- Hackbright : taught software engineering. Overhauled ML and collaborative filtering lectures
- fast.ai : co-founded to make deep learning more accessible. Deep Learning involves a TON of linear algebra

Teaching ¶

Teaching Approach

I'll be using a top-down teaching method, which is different from how most math courses operate. Typically, in a bottom-up approach, you first learn all the separate components you will be using, and then you gradually build them up into more complex structures. The problems with this are that students often lose motivation, don't have a sense of the "big picture", and don't know what they'll need.

If you took the fast.ai deep learning course, that is what we used. You can hear more about my teaching philosophy in this blog post or in this talk.

Harvard Professor David Perkins has a book, Making Learning Whole in which he uses baseball as an analogy. We don't require kids to memorize all the rules of baseball and understand all the technical details before we let them play the game. Rather, they start playing with a just general sense of it, and then gradually learn more rules/details as time goes on.

All that to say, don't worry if you don't understand everything at first! You're not supposed to. We will start using some "black boxes" or matrix decompositions that haven't yet been explained, and then we'll dig into the lower level details later.

To start, focus on what things DO, not what they ARE.

People learn by:

1. doing (coding and building)
2. explaining what they've learned (by writing or helping others)

Text Book

The book Numerical Linear Algebra by Trefethen and Bau is recommended. The MSAN program has a few copies on hand.

A secondary book is Numerical Methods by Greenbaum and Chartier.

Basics ¶

Office hours : 2:00-4:00 on Friday afternoons. Email me if you need to meet at other times.

My contact info: rachel@fast.ai

Class Slack: #numerical_lin_alg

Email me if you will need to miss class.

Jupyter Notebooks will be available on Github at: https://github.com/fastai/numerical-linear-algebra Please pull/download before class. Some parts are removed for you to fill in as you follow along in class . Be sure to let me know THIS WEEK if you are having any problems running the notebooks from your own computer. You may want to make a separate copy, because running Jupyter notebooks causes them to change, which can create github conflicts the next time you pull.

Check that you have MathJax running (which renders LaTeX, used for math equations) by running the following cell:

$$e^{\theta i} = \cos(\theta) + i\sin(\theta)$$

check that you can import:

In [2]:

```
import numpy as np
import sklearn
```

Grading Rubric :

Assignment	Percent
Attendance	10%
Homework	20%
Writing: proposal	10%
Writing: draft	15%
Writing: final	15%
Final Exam	30%

Honor Code

No cheating nor plagiarism is allowed, please see below for more details.

On Laptops

I ask you to be respectful of me and your classmates and to refrain from surfing the web or using social media (facebook, twitter, etc) or messaging programs during class. It is absolutely forbidden to use instant messaging programs, email, etc. during class lectures or quizzes.

Syllabus ¶

Topics Covered:

1. Why are we here?

- Matrix and Tensor Products
- Matrix Decompositions
- Accuracy
- Memory use
- Speed
- Parallelization & Vectorization

2. Topic Modeling with NMF and SVD

- Topic Frequency-Inverse Document Frequency (TF-IDF)
- Singular Value Decomposition (SVD)
- Non-negative Matrix Factorization (NMF)
- Stochastic Gradient Descent (SGD)
- Intro to PyTorch
- Truncated SVD, Randomized SVD

3. Background Removal with Robust PCA

- Robust PCA
- Randomized SVD
- LU factorization

4. Compressed Sensing for CT scans with Robust Regression

- L1 regularization

5. Predicting Health Outcomes with Linear Regression

- Linear regression
- Polynomial Features
- Speeding up with Numba
- Regularization and Noise
- Implementing linear regression 4 ways

6. PageRank with Eigen Decompositions

- Power Method
- QR Algorithm
- Arnoldi Iteration

7. QR Factorization

- Gram-Schmidt
- Householder
- Stability

Writing Assignment ¶

Writing Assignment: Writing about technical concepts is a hugely valuable skill. I want you to write a technical blog post related to numerical linear algebra. A blog is like a resume, only better. Technical writing is also important in creating documentation, sharing your work with co-workers, applying to speak at conferences, and practicing for interviews. (You don't actually have to publish it, although I hope you do, and please send me the link if you do.)

- List of ideas here
- Always cite sources, use quote marks around quotes. Do this even as you are first gathering sources and taking notes. If you plagiarize parts of someone else's work, you will fail.
- Can be done in a Jupyter Notebook (Jupyter Notebooks can be turned into blog posts) or a Kaggle Kernel

For the proposal, write a brief paragraph about the problem/topic/experiment you plan to research/test and write about. You need to include 4 sources that you plan to use: these can include Trefethen, other blog posts, papers, or books. Include a sentence about each source, stating what it's in it.

Feel free to ask me if you are wondering if your topic idea is suitable!

Excellent T echnical Blogs ¶

Examples of great technical blog posts:

- Peter Norvig (more here)
- Stephen Merity
- Julia Evans (more here)
- Julia Ferraioli
- Edwin Chen
- Slav Ivanov
- Brad Kenstler
- find more on twitter

Deadlines ¶

Assignment	Dates
Homeworks	TBA
Writing: proposal	5/30
Writing: draft	6/15
Writing: final	6/27
Final Exam	6/29

Linear Algebra ¶

We will review some linear algebra in class. However, if you find there are concepts you feel rusty on, you may want to review on your own. Here are some resources:

- 3Blue1Brown Essence of Linear Algebra videos about geometric intuition (fantastic! gorgeous!)
- Lectures 1-6 of Trefethen
- Immersive linear algebra free online textbook with interactive graphics
- Chapter 2 of Ian Goodfellow's Deep Learning Book

USF Policies ¶

Academic Integrity

USF upholds the standards of honesty and integrity from all members of the academic community. All students are expected to know and adhere to the University's Honor Code. You can find the full text of the code online. The policy covers:

- Plagiarism: intentionally or unintentionally representing the words or ideas of another person as your own; failure to properly cite references; manufacturing references.
- Working with another person when independent work is required.
- Submission of the same paper in more than one course without the specific permission of each instructor.
- Submitting a paper written (entirely or even a small part) by another person or obtained from the internet.
- Plagiarism is plagiarism: it does not matter if the source being copied is on the Internet, from a book or textbook, or from quizzes or problem sets written up by other students.
- The penalties for violation of the policy may include a failing grade on the assignment, a failing grade in the course, and/or a referral to the Academic Integrity Committee.

Students with Disabilities

If you are a student with a disability or disabling condition, or if you think you may have a disability, please contact USF Student Disability Services (SDS) at 415 422-2613 within the first week of class, or immediately upon onset of disability, to speak with a disability specialist. If you are determined eligible for reasonable accommodations, please meet with your disability specialist so they can arrange to have your accommodation letter sent to me, and we will discuss your needs for this course. For more information, please visit this website or call (415) 422-2613.

Behavioral Expectations

All students are expected to behave in accordance with the Student Conduct Code and other University policies. Open discussion and disagreement is encouraged when done respectfully and in the spirit of academic discourse. There are also a variety of behaviors that, while not against a specific University policy, may create disruption in this course. Students whose behavior is disruptive or who fail to comply with the instructor may be dismissed from the class for the remainder of the class period and may need to meet with the instructor or Dean prior to returning to the next class period. If necessary, referrals may also be made to the Student Conduct process for violations of the Student Conduct Code.

Counseling and Psychological Services

Our diverse staff offers brief individual, couple, and group counseling to student members of our community. CAPS services are confidential and free of charge. Call 415-422-6352 for an initial consultation appointment. Having a crisis at 3 AM? We are still here for you. Telephone consultation through CAPS After Hours is available between the hours of 5:00 PM to 8:30 AM; call the above number and press 2.

Confidentiality , Mandatory Reporting, and Sexual Assault

As an instructor, one of my responsibilities is to help create a safe learning environment on our campus. I also have a mandatory reporting responsibility related to my role as a faculty member. I am required to share information regarding sexual misconduct or information about a crime that may have occurred on USFs campus with the University. Here are other resources:

- To report any sexual misconduct, students may visit Anna Bartkowski (UC 5th floor) or see many other options by visiting this website
- Students may speak to someone confidentially, or report a sexual assault confidentially by contacting Counseling and Psychological Services at 415-422-6352
- To find out more about reporting a sexual assault at USF, visit USF's Callisto website
- For an off-campus resource, contact San Francisco Women Against Rape 415-647-7273

You can read an overview of this Numerical Linear Algebra course in this blog post. The course was originally taught in the University of San Francisco MS in Analytics graduate program. Course lecture videos are available on YouTube (note that the notebook numbers and video numbers do not line up, since some notebooks took longer than 1 video to cover).

You can ask questions about the course on our fast.ai forums.

1. Why are we here? ¶

Note: Future lessons have a lot more code than this one

Why study Numerical Linear Algebra? ¶

Key Question of this course : How can we do matrix computations with acceptable speed and acceptable accuracy?

A list of the Top 10 Algorithms of science and engineering during the 20th century includes: the matrix decompositions approach to linear algebra. It also includes the QR algorithm, which we'll cover, and Krylov iterative methods which we'll see an example of. (See here for another take)

In putting together this issue of *Computing in Science & Engineering*, we knew three things: it would be difficult to list just 10 algorithms; it would be fun to assemble the authors and read their papers; and, whatever we came up with in the end, it would be controversial. We tried to assemble the 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century. Following is our list (here, the list is in chronological order; however, the articles appear in no particular order):

- Metropolis Algorithm for Monte Carlo
- Simplex Method for Linear Programming
- Krylov Subspace Iteration Methods
- The Decompositional Approach to Matrix Computations
- The Fortran Optimizing Compiler
- QR Algorithm for Computing Eigenvalues
- Quicksort Algorithm for Sorting
- Fast Fourier Transform
- Integer Relation Detection
- Fast Multipole Method

(source: Top 10 Algorithms)

There are 4 things to keep in mind when choosing or designing an algorithm for matrix computations:

- Memory Use
- Speed
- Accuracy
- Scalability/Parallelization

Often there will be trade-offs between these categories.

Motivation ¶

Matrices are everywhere-- anything that can be put in an Excel spreadsheet is a matrix, and language and pictures can be represented as matrices as well. Knowing what options there are for matrix algorithms, and how to navigate compromises, can make enormous differences to your solutions. For instance, an approximate matrix computation can often be thousands of times faster than an exact one.

It's not just about knowing the contents of existing libraries, but knowing how they work too. That's because often you can make variations to an algorithm that aren't supported by your library, giving you the performance or accuracy that you need. In addition, this field is moving very quickly at the moment, particularly in areas related to deep learning , recommendation systems , approximate algorithms , and graph analytics , so you'll often find there's recent results that could make big differences in your project, but aren't in your library.

Knowing how the algorithms really work helps to both debug and accelerate your solution.

Matrix Computations ¶

There are two key types of matrix computation, which get combined in many different ways. These are:

- Matrix and tensor products
- Matrix decompositions

So basically we're going to be combining matrices, and pulling them apart again!

Matrix and T ensor Products ¶

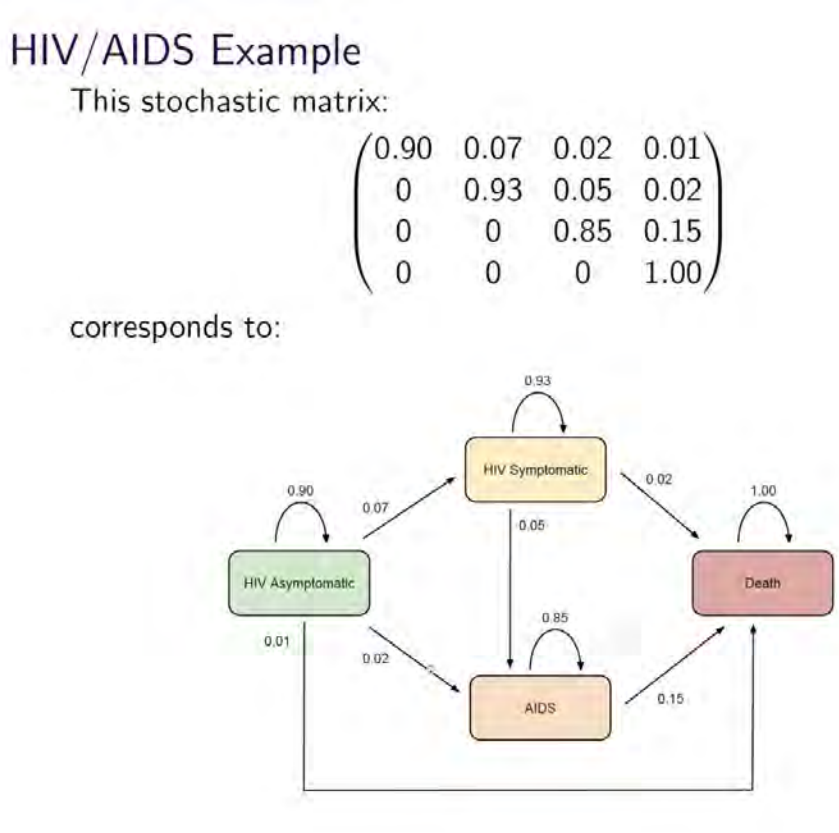
Matrix-V ector Products: ¶

The matrix below gives the probabilities of moving from 1 health state to another in 1 year. If the current health states for a group are:

- 85% asymptomatic
- 10% symptomatic
- 5% AIDS

- 0% death

what will be the % in each health state in 1 year?



(Source: Concepts of Markov Chains)

Answer ¶

In [6]:

```
import numpy as np
```

In [19]:

```
#Exercise: Use Numpy to compute the answer to the above
```

Out[19]:

```
array([[ 0.765 ],
       [ 0.1525],
       [ 0.0645],
       [ 0.018 ]])
```

Matrix-Matrix Products ¶

Three people denoted by P_1, P_2, P_3 intend to buy some rolls, buns, cakes and bread. Each of them needs these commodities in differing amounts and can buy them in two shops S_1, S_2. Which shop is the best for every person P_1, P_2, P_3 to pay as little as possible? The individual prices and desired quantities of the commodities are given in the following tables:

Demanded quantity of foodstuff:

	roll	bun	cake	bread
P_1	6	5	3	1
P_2	3	6	2	2
P_3	3	4	3	1

Prices in shops S_1 and S_2:

	S_1	S_2
roll	1.50	1.00
bun	2.00	2.50
cake	5.00	4.50
bread	16.00	17.00

For example, the amount spent by the person P_1 in the shop S_1 is:

$$6 \cdot 1.50 + 5 \cdot 2 + 3 \cdot 5 + 1 \cdot 16 = 50$$

and in the shop S_2 :

$$6 \cdot 1 + 5 \cdot 2.50 + 3 \cdot 4.50 + 1 \cdot 17 = 49,$$

(Source: Several Simple Real-world Applications of Linear Algebra Tools)

Answer ¶

In [23]:

```
#Exercise: Use Numpy to compute the answer to the above
```

Out[23]:

```
array([[ 50. ,  49. ],
       [ 58.5,  61. ],
       [ 43.5,  43.5]])
```

Image Data ¶

Images can be represented by matrices.

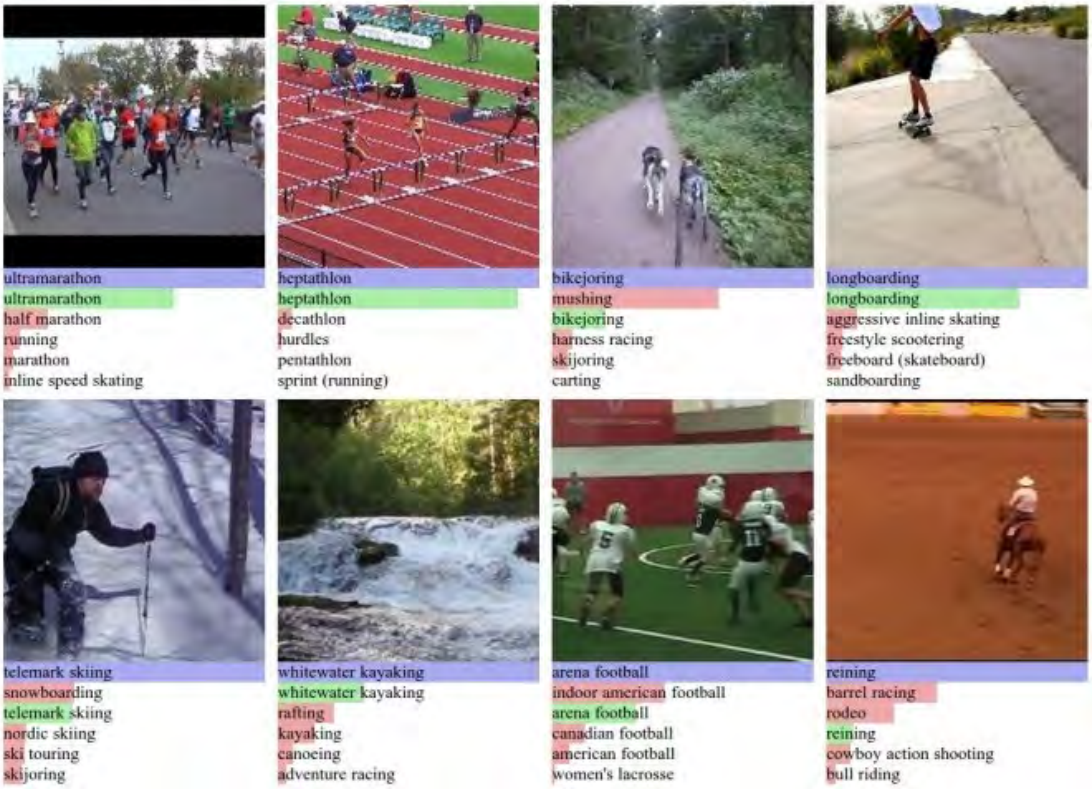
```
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   1  12   0  11  39 137  37   0 152 147  84   0   0   0
  0   0   1   0   0   0  41 160 250 255 235 162 255 238 206  11  13   0
  0   0   0  16   9   9 150 251  45  21 184 159 154 255 233  40   0   0
 10   0   0   0   0   0 145 146   3  10   0  11 124 253 255 107   0   0
  0   0   3   0   4  15 236 216   0   0  38 109 247 240 169   0  11   0
  1   0   2   0   0   0 253 253  23  62 224 241 255 164   0   5   0   0
  6   0   0   4   0   3 252 250 228 255 255 234 112  28   0   2  17   0
  0   2   1   4   0  21 255 253 251 255 172  31   8   0   1   0   0   0
  0   0   4   0 163 225 251 255 229 120   0   0   0   0   0  11   0   0
  0   0  21 162 255 255 254 255 126   6   0  10  14   6   0   0   9   0
  3  79 242 255 141  66 255 245 189   7   8   0   0   5   0   0   0   0
 26 221 237  98   0  67 251 255 144   0   8   0   0   7   0   0  11   0
125 255 141   0  87 244 255 208   3   0   0  13   0   1   0   1   0   0
145 248 228 116 235 255 141  34   0  11   0   1   0   0   0   1   3   0
 85 237 253 246 255 210  21   1   0   1   0   0   6   2   4   0   0   0
  6  23 112 157 114  32   0   0   0   0   2   0   8   0   7   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

(Source: Adam Geitgey)

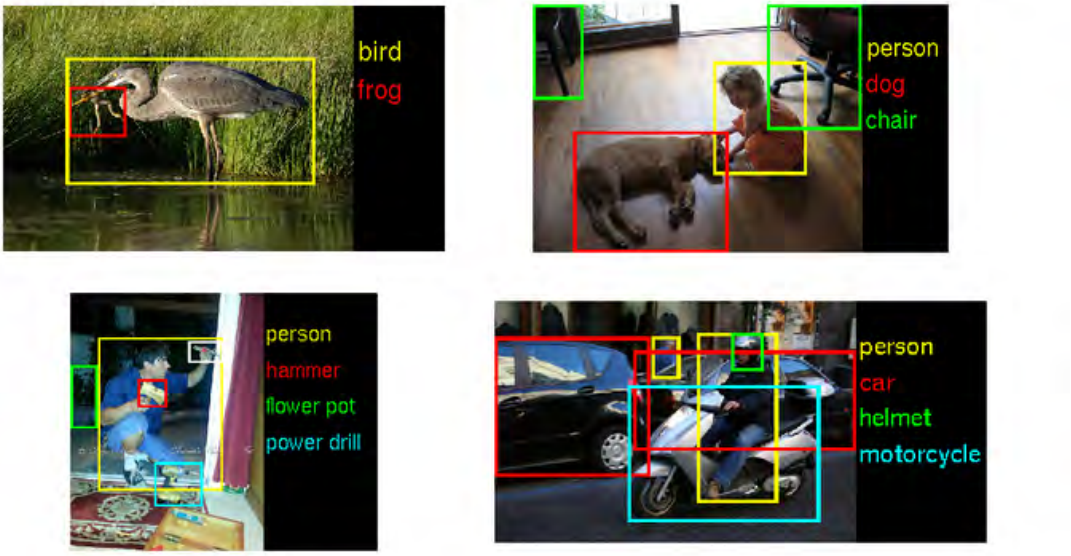
Convolution ¶

Convolutions are the heart of convolutional neural networks (CNNs), a type of deep learning, responsible for the huge advances in image recognitionin the last few years. They are now increasingly being used for speech as well, such as Facebook AI's results for speech translation which are 9x faster than RNNs (the current most popular approach for speech translation).

Computers are now more accurate than people at classifying images.



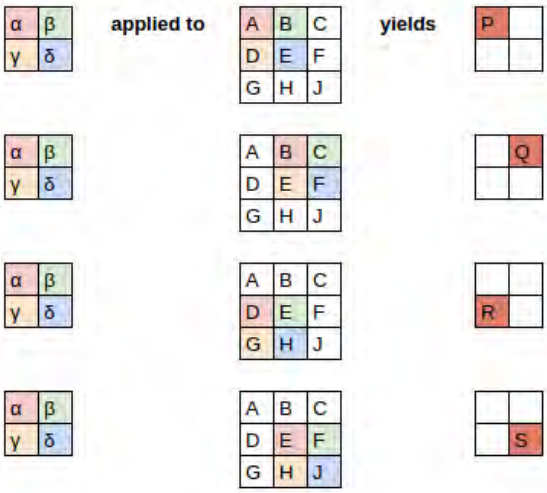
(Source: Andrej Karpathy)



(Source: Nvidia)

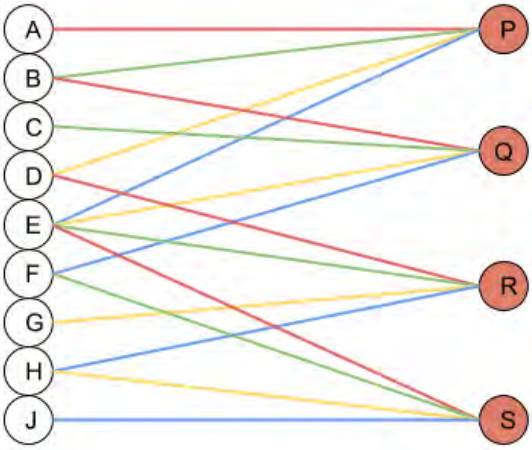
You can think of a convolution as a special kind of matrix product

The 3 images below are all from an excellent blog post written by a fast.ai student on CNNs from Different Viewpoints:



A convolution applies a filter to each section of an image:

Neural Network Viewpoint:



Matrix Multiplication Viewpoint:

Let's see how convolutions can be used for edge detection in this notebook(originally from the fast.ai Deep Learning Course)

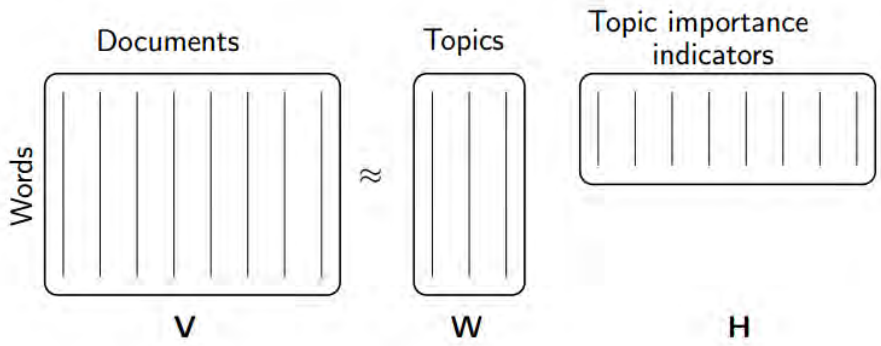
Matrix Decompositions ¶

We will be talking about Matrix Decompositions every day of this course, and will cover the below examples in future lessons:

- Topic Modeling (NMF and SVD. SVD uses QR) A group of documents can be represented by a term-document matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth ...
ANTHONY	157	73	0	0	0	1
BRUTUS	4	157	0	2	0	0
CAESAR	232	227	0	2	1	0
CALPURNIA	0	10	0	0	0	0
CLEOPATRA	57	0	0	0	0	0
MERCY	2	0	3	8	5	8
WORSER	2	0	1	1	1	5

(source: Introduction to Information

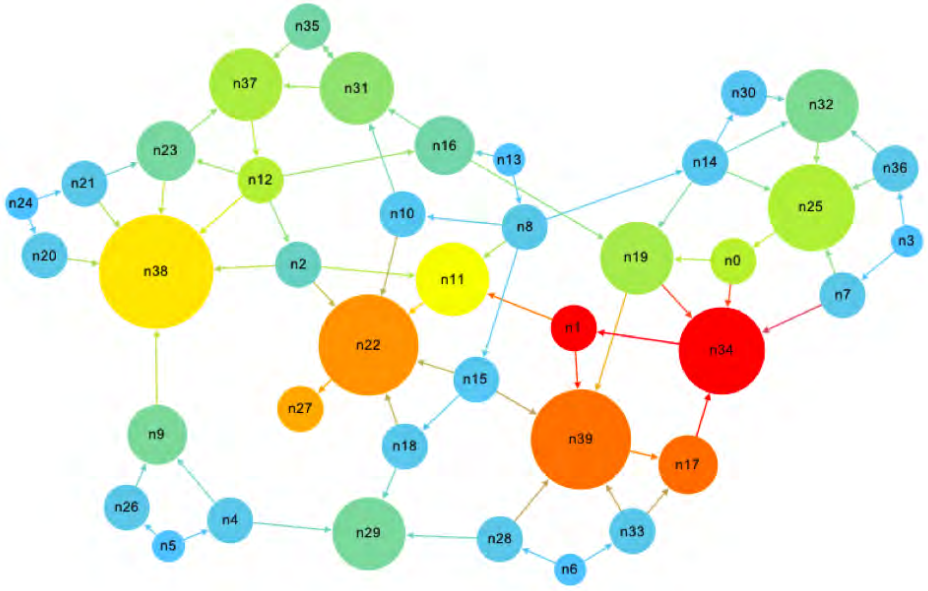


Retrieval)

(source: NMF Tutorial)

- Background removal (robust PCA, which uses truncated SVD)

- Google's PageRank Algorithm (eigen decomposition)



(source: What is in PageRank?)

- List of other decompositions and some applications matrix factorization jungle

Accuracy ¶

Floating Point Arithmetic ¶

To understand accuracy, we first need to look at how computers (which are finite and discrete) store numbers (which are infinite and continuous)

Exercise ¶

Take a moment to look at the function f below. Before you try running it, write on paper what the output would be of $x_1 = f(\frac{1}{10})$. Now, (still on paper) plug that back into f and calculate $x_2 = f(x_1)$. Keep going for 10 iterations.

This example is taken from page 107 of Numerical Methods, by Greenbaum and Chartier.

In [1]:

```
def f(x):
    if x <= 1/2:
```

```
        return 2 * x
    if x > 1/2:
        return 2*x - 1
```

Only after you've written down what you think the answer should be, run the code below:

In [ ]:

```
x = 1/10
for i in range(80):
    print(x)
    x = f(x)
```

What went wrong?

Problem: math is continuous & infinite, but computers are discrete & finite ¶

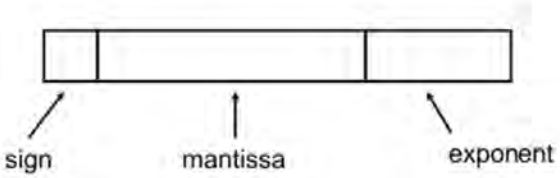
Two Limitations of computer representations of numbers:

1. they can't be arbitrarily large or small
2. there must be gaps between them

The reason we need to care about accuracy, is because computers can't store infinitely accurate numbers. It's possible to create calculations that give very wrong answers (particularly when repeating an operation many times, since each operation could multiply the error).

How computers store numbers:

- Floating point numbers have three parts:
  sign, mantissa, and exponent
- The base (radix) is assumed (usually base 2).
- The sign is a single bit (0 for positive number, 1 for negative).



The mantissa can also be referred to as the significand.

IEEE Double precision arithmetic:

- Numbers can be as large as 1.79×10^{308} and as small as 2.23×10^{-308}.
- The interval $[1, 2]$ is represented by discrete subset:

$$1,\ 1+2^{-52},\ 1+2\times 2^{-52},\ 1+3\times 2^{-52},\ \ldots, 2$$

- The interval $[2, 4]$ is represented:

$$2,\ 2+2^{-51},\ 2+2\times 2^{-51},\ 2+3\times 2^{-51},\ \ldots, 4$$

Floats and doubles are not equidistant:

Source: What you never wanted to know about floating point but will be forced to find out

Machine Epsilon

Half the distance between 1 and the next larger number. This can vary by computer. IEEE standards for double precision specify

$$\varepsilon_{machine} = 2^{-53} \approx 1.11 \times 10^{-16}$$

Two important properties of Floating Point Arithmetic :

- The difference between a real number x and its closest floating point approximation $fl(x)$ is always smaller than $\varepsilon_{machine}$ in relative terms. For some ε, where Undefined control sequence \lvert,

$$fl(x) = x \cdot (1 + \varepsilon)$$

- Where is any operation ($+, -, \times, \div$), and Undefined control sequence \circledast is its floating point analogue, $$ x \circledast y = (x y)(1 + \varepsilon)$$ for some ε, where $\lvert \varepsilon \rvert \leq \varepsilon{machine}$ $That is, every operation of floating point arithmetic is exact up to a relative error of size at most \varepsilon{machine}$

History ¶

Floating point arithmetic may seem like a clear choice in hindsight, but there have been many, many ways of storing numbers:

- fixed-point arithmetic
- logarithmic and semilogarithmic number systems
- continued-fractions
- rational numbers
- possibly infinite strings of rational numbers
- level-index number systems
- fixed-slash and floating-slash number systems
- 2-adic numbers

For references, see Chapter 1 (which is free) of the Handbook of Floating-Point Arithmetic. Yes, there is an entire 16 chapter book on floating point!

Timeline History of Floating Point Arithmetic:

- ~1600 BC: Babylonian radix-60 system was earliest floating-point system (Donald Knuth). Represented the significand of a radix-60 floating-point representation (if ratio of two numbers is a power of 60, represented the same)
- 1630 Slide rule. Manipulate only significands (radix-10)
- 1914 Leonardo Torres y Quevedo described an electromechanical implementation of Babbage's Analytical Engine with Floating Point Arithmetic.
- 1941 First real, modern implementation. Konrad Zuse's Z3 computer. Used radix-2, with 14 bit significand, 7 bit exponents, and 1 sign bit.
- 1985 IEEE 754-1985 Standard for Binary Floating-Point Arithmetic released. Has increased accuracy, reliability, and portability. William Kahan played leading role.

"Many different ways of approximating real numbers on computers have been introduced.. And yet, floating-point arithmetic is by far the most widely used way of representing real numbers in modern computers. Simulating an infinite, continuous set (the real numbers) with a finite set (the “machine numbers”) is not a straightforward task: clever compromises must be found between, speed, accuracy , dynamic range, ease of use and implementation, and memory . It appears that floating-point arithmetic, with adequately chosen parameters (radix, precision, extremal exponents, etc.), is a very good compromise for most numerical applications."

Although a radix value of 2 (binary) seems like the pretty clear winner now for computers, a variety of other radix values have been used at various point:

- radix-8 used by early machines PDP-10, Burroughs 570 and 6700
- radix-16 IBM 360
- radix-10 financial calculations, pocket calculators, Maple
- radix-3 Russian SETUN computer (1958). Benefits: minimizes beta x p (symbols x digits), for a fixed largest representable number beta^p - 1. Rounding = truncation
- radix-2 most common. Reasons: easy to implement. Studies have shown (with implicit leading bit) this gives better worst-case or average accuracy than all other radices.

Conditioning and Stability ¶

Since we can not represent numbers exactly on a computer (due to the finiteness of our storage, and the gaps between numbers in floating point architecture), it becomes important to know how small perturbations in the input to a problem impact the output.

"A stable algorithm gives nearly the right answer to nearly the right question." --Trefethen

Conditioning : perturbation behavior of a mathematical problem (e.g. least squares)

Stability : perturbation behavior of an algorithm used to solve that problem on a computer (e.g. least squares algorithms, householder, back substitution, gaussian elimination)

Example: Eigenvalues of a Matrix

In [37]:

```
import scipy.linalg as la

A = np.array([[1., 1000], [0, 1]])
B = np.array([[1, 1000], [0.001, 1]])

print(A)

print(B)
```

```
[[    1.  1000.]
 [    0.     1.]]
[[    1.     1000.   ]
 [    0.001     1.   ]]
```

In [30]:

```
np.set_printoptions(suppress=True, precision=4)
```

In [ ]:

```
wA, vrA = la.eig(A)
wB, vrB = la.eig(B)

wA, wB
```

Reminder: T wo properties of Floating Point Arithmetic

- The difference between a real number x and its closest floating point approximation $fl(x)$ is always smaller than $\varepsilon_{machine}$ in relative terms.
- Every operation $+, -, \times, \div$ of floating point arithmetic is exact up to a relative error of size at most $\varepsilon_{machine}$

Examples we'll see:

- Classical vs Modified Gram-Schmidt accuracy
- Gram-Schmidt vs. Householder (2 different ways of computing QR factorization), how orthogonal the answer is
- Condition of a system of equations

Approximation accuracy ¶

It's rare that we need to do highly accurate matrix computations at scale. In fact, often we're doing some kind of machine learning, and less accurate approaches can prevent overfitting.

If we accept some decrease in accuracy, then we can often increase speed by orders of magnitude (and/or decrease memory use) by using approximate algorithms. These algorithms typically give a correct answer with some probability. By rerunning the algorithm multiple times you can generally increase that probability multiplicatively!

Example : A bloom filter allows searching for set membership with 1% false positives, using <10 bits per element. This often represents reductions in memory use of thousands of times.

The false positives can be easily handled by having a second (exact) stage check all returned items - for rare items this can be very effective. For instance, many web browsers use a bloom filter to create a set of blocked pages (e.g. pages with viruses), since blocked web pages are only a small fraction of the whole web. A false positive can be handled here by taking anything returned by the bloom filter and checking against a web service with the full exact list. (See this bloom filter tutorial for more details).

Expensive Errors ¶

The below examples are from Greenbaum & Chartier.

European Space Agency spent 10 years and $7 billion on the Ariane 5 Rocket.

What can happen when you try to fit a 64 bit number into a 16 bit space (integer overflow):

In [3]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("PK_yguLapgA")
```

Out[3]:

Here is a floating point error that cost Intel $475 million:

COMPANY NEWS

COMPANY NEWS; Flaw Undermines Accuracy of Pentium Chips

By JOHN MARKOFF
Published: November 24, 1994

Correction Appended

SAN FRANCISCO, Nov. 23— An elusive circuitry error is causing a chip used in millions of computers to generate inaccurate results in certain rare cases, heightening anxiety among many scientists and engineers who rely on their machines for precise calculations.

FACEBOOK
TWITTER
GOOGLE+
EMAIL
SHARE
PRINT
REPRINTS

The flaw, an error in division, has been found in the Pentium, the current top microprocessor of the Intel Corporation, the world's largest chip maker. The chip, in several different configurations, is used in many computers sold for home and business use, including those made by I.B.M., Compaq, Dell, Gateway 2000 and others.

The flaw appears in all Pentium chips now on the market, in certain types of division problems involving more than five significant digits, a mathematical term that can include numbers before and after a decimal point.

Intel declined to say how many Pentium chips it made or sold, but Dataquest, a market research company in San Jose, Calif., estimated that in 1994 Intel would sell 5.5 million to 6 million Pentiums, roughly 10 percent of the number of personal computers sold worldwide.

1994 NYTimes article about Intel Pentium Error

Resources : See Lecture 13 of Trefethen & Bau and Chapter 5 of Greenbaum & Chartier for more on Floating Point Arithmetic

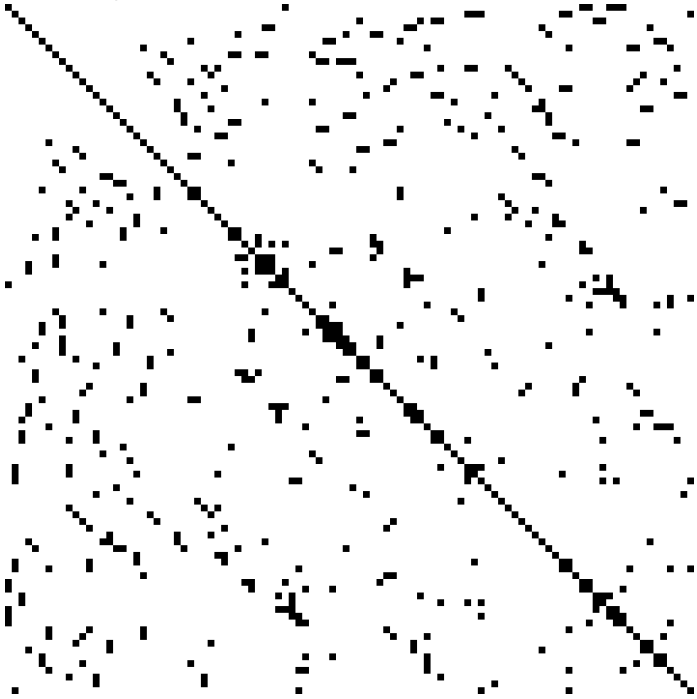
Memory Use ¶

Sparse vs Dense ¶

Above we covered how numbers are stored, now let's talk about how matrices are stored. A key way to save memory (and computation) is not to store all of your matrix. Instead, just store the non-zero elements. This is called sparse storage, and it is well suited to sparse matrices, that is, matrices where most elements are zero.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

Here is an example of the matrix from a finite element problem, which shows up in engineering (for instance, when modeling the air-flow around a plane). In this example, the non-zero elements are black and the zero elements are white:



Source

There are also special types of structured matrix, such as diagonal, tri-diagonal, hessenberg, and triangular, which each display particular patterns of sparsity, which can be leveraged to reduce memory and computation.

The opposite of a sparse matrix is a dense matrix, along with dense storage, which simply refers to a matrix containing mostly non-zeros, in which every element is stored explicitly. Since sparse matrices are helpful and common, numerical linear algebra focuses on maintaining sparsity through as many operations in a computation as possible.

Speed ¶

Speed differences come from a number of areas, particularly:

- Computational complexity
- Vectorization
- Scaling to multiple cores and nodes
- Locality

Computational complexity ¶

If you are unfamiliar with computational complexity and $\mathcal{O}$ notation, you can read about it on Interview Cake and practice on Codecademy. Algorithms are generally expressed in terms of computation complexity with respect to the number of rows and number of columns in the matrix. E.g. you may find an algorithm described as $\mathcal{O}(n^2m)$.

Vectorization ¶

Modern CPUs and GPUs can apply an operation to multiple elements at once on a single core. For instance, take the exponent of 4 floats in a vector in a single step. This is called SIMD. You will not be explicitly writing SIMD code (which tends to require assembly language or special C "intrinsics"), but instead will use vectorized operations in libraries like numpy, which in turn rely on specially tuned vectorized low level linear algebra APIs (in particular, BLAS, and LAPACK).

Matrix Computation Packages: BLAS and LAP ACK¶

BLAS (Basic Linear Algebra Subprograms): specification for low-level matrix and vector arithmetic operations. These are the standard building blocks for performing basic vector and matrix operations. BLAS originated as a Fortran library in 1979. Examples of BLAS libraries include: AMD Core Math Library (ACML), ATLAS, Intel Math Kernel Library (MKL), and OpenBLAS.

LAPACK is written in Fortran, provides routines for solving systems of linear equations, eigenvalue problems, and singular value problems. Matrix factorizations (LU, Cholesky, QR, SVD, Schur). Dense and banded matrices are handled, but not general sparse matrices. Real and complex, single and double precision.

1970s and 1980s: EISPACK (eigenvalue routines) and LINPACK (linear equations and linear least-squares routines) libraries

LAPACK original goal : make LINAPCK and EISPACK run efficiently on shared-memory vector and parallel processors and exploit cache on modern cache-based architectures (initially released in 1992). EISPACK and LINPACK ignore multi-layered memory hierarchies and spend too much time moving data around.

LAPACK uses highly optimized block operations implementations (which much be implemented on each machine) LAPACK written so as much of the computation as possible is performed by BLAS.

Locality ¶

Using slower ways to access data (e.g. over the internet) can be up to a billion times slower than faster ways (e.g. from a register). But there's much less fast storage than slow storage. So once we have data in fast storage, we want to do any computation required at that time, rather than having to load it multiple times each time we need it. In addition, for most types of storage its much faster to access data items that are stored next to each other, so we should try to always use any data stored nearby that we know we'll need soon. These two issues are known as locality.

Speed of different types of memory ¶

Here are some numbers everyone should know (from the legendary Jeff Dean):

- L1 cache reference 0.5 ns
- L2 cache reference 7 ns
- Main memory reference/RAM 100 ns
- Send 2K bytes over 1 Gbps network 20,000 ns
- Read 1 MB sequentially from memory 250,000 ns
- Round trip within same datacenter 500,000 ns
- Disk seek 10,000,000 ns
- Read 1 MB sequentially from network 10,000,000 ns
- Read 1 MB sequentially from disk 30,000,000 ns
- Send packet CA->Netherlands->CA 150,000,000 ns

And here is an updated, interactive version, which includes a timeline of how these numbers have changed.

Key take-away : Each successive memory type is (at least) an order of magnitude worse than the one before it. Disk seeks are very slow .

This video has a great example of showing several ways you could compute the blur of a photo, with various trade-offs. Don't worry about the C code that appears, just focus on the red and green moving pictures of matrix computation.

Although the video is about a new language called Halide, it is a good illustration the issues it raises are universal. Watch minutes 1-13:

In [1]:

```
from IPython.display import YouTubeVideo
YouTubeVideo("3uiEyEKji0M")
```

Out[1]:

Locality is hard. Potential trade-offs:

- redundant computation to save memory bandwidth
- sacrificing parallelism to get better reuse

Temporaries ¶

The issue of "temporaries" occurs when the result of a calculation is stored in a temporary variable in RAM, and then that variable is loaded to do another calculation on it. This is many orders of magnitude slower than simply keeping the data in cache or registers and doing all necessary computations before storing the final result in RAM. This is particularly an issue for us since numpy generally creates temporaries for every single operation or function it does. E.g. $a = b \cdot c^2 + ln(d)$ will create four temporaries (since there are four operations and functions).

Scaling to multiple cores and nodes ¶

We have a separate section for scalability, but it's worth noting that this is also important for speed - if we can't scale across all the computing resources we have, we'll be stuck with slower computation.

Scalability / parallelization ¶

Often we'll find that we have more data than we have memory to handle, or time to compute. In such a case we would like to be able to scale our algorithm across multiple cores (within one computer) or nodes (i.e. multiple computers on a network). We will not be tackling multi-node scaling in this course, although we will look at scaling across multiple cores (called parallelization). In general, scalable algorithms are those where the input can be broken up into smaller pieces, each of which are handled by a different core/computer, and then are put back together at the end.

You can read an overview of this Numerical Linear Algebra course in this blog post. The course was originally taught in the University of San Francisco MS in Analytics graduate program. Course lecture videos are available on YouTube (note that the notebook numbers and video numbers do not line up, since some notebooks took longer than 1 video to cover).

You can ask questions about the course on our fast.ai forums.

2. Topic Modeling with NMF and SVD ¶

Topic modeling is a great way to get started with matrix factorizations. We start with a term-document matrix :

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth ...
ANTHONY	157	73	0	0	0	1
BRUTUS	4	157	0	2	0	0
CAESAR	232	227	0	2	1	0
CALPURNIA	0	10	0	0	0	0
CLEOPATRA	57	0	0	0	0	0
MERCY	2	0	3	8	5	8
WORSER	2	0	1	1	1	5

(source: Introduction to Information Retrieval)

We can decompose this into one tall thin matrix times one wide short matrix (possibly with a diagonal matrix in between).

Notice that this representation does not take into account word order or sentence structure. It's an example of a bag of words approach.

Motivation ¶

Consider the most extreme case - reconstructing the matrix using an outer product of two vectors. Clearly, in most cases we won't be able to reconstruct the matrix exactly. But if we had one vector with the relative frequency of each vocabulary word out of the total word count, and one with the average number of words per document, then that outer product would be as close as we can get.

Now consider increasing that matrices to two columns and two rows. The optimal decomposition would now be to cluster the documents into two groups, each of which has as different a distribution of words as possible to each other, but as similar as possible amongst the documents in the cluster. We will call those two groups "topics". And we would cluster the words into two groups, based on those which most frequently appear in each of the topics.

In today's class ¶

We'll take a dataset of documents in several different categories, and find topics (consisting of groups of words) for them. Knowing the actual categories helps us evaluate if the topics we find make sense.

We will try this with two different matrix factorizations: Singular V alue Decomposition (SVD) and Non-negative Matrix Factorization (NMF)

In [4]:

```
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn import decomposition
from scipy import linalg
import matplotlib.pyplot as plt
```

In [5]:

```
%matplotlib inline
np.set_printoptions(suppress=True)
```

Additional Resources ¶

- Data source: Newsgroups are discussion groups on Usenet, which was popular in the 80s and 90s before the web really took off. This dataset includes 18,000 newsgroups posts with 20 topics.
- Chris Manning's book chapter on matrix factorization and LSI
- Scikit learn truncated SVD LSI details

Other Tutorials ¶

- Scikit-Learn: Out-of-core classification of text documents: uses Reuters-21578 dataset (Reuters articles labeled with ~100 categories), HashingVectorizer
- Text Analysis with Topic Models for the Humanities and Social Sciences: uses British and French Literature dataset of Jane Austen, Charlotte Bronte, Victor Hugo, and more

Set up data ¶

Scikit Learn comes with a number of built-in datasets, as well as loading utilities to load several standard external datasets. This is a great resource, and the datasets include Boston housing prices, face images, patches of forest, diabetes, breast cancer, and more. We will be using the newsgroups dataset.

Newsgroups are discussion groups on Usenet, which was popular in the 80s and 90s before the web really took off. This dataset includes 18,000 newsgroups posts with 20 topics.

In [6]:

```
categories = ['alt.atheism', 'talk.religion.misc', 'comp.graphics', 'sci.space']
remove = ('headers', 'footers', 'quotes')
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories, remove=remove)
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories, remove=remove)
```

In [8]:

```
newsgroups_train.filenames.shape, newsgroups_train.target.shape
```

Out[8]:

```
((2034,), (2034,))
```

Let's look at some of the data. Can you guess which category these messages are in?

In [55]:

```
print("\n".join(newsgroups_train.data[:3]))
```

```
Hi,

I've noticed that if you only save a model (with all your mapping planes
positioned carefully) to a .3DS file that when you reload it after restarting
3DS, they are given a default position and orientation.  But if you save
to a .PRJ file their positions/orientation are preserved.  Does anyone
know why this information is not stored in the .3DS file?  Nothing is
explicitly said in the manual about saving texture rules in the .PRJ file.
I'd like to be able to read the texture rule information, does anyone have
the format for the .PRJ file?

Is the .CEL file format available from somewhere?

Rych
```

```
Seems to be, barring evidence to the contrary, that Koresh was simply
another deranged fanatic who thought it neccessary to take a whole bunch of
folks with him, children and all, to satisfy his delusional mania. Jim
Jones, circa 1993.


Nope - fruitcakes like Koresh have been demonstrating such evil corruption
for centuries.

 >In article <1993Apr19.020359.26996@sq.sq.com>, msb@sq.sq.com (Mark Brader)

MB>                                                             So the
MB> 1970 figure seems unlikely to actually be anything but a perijove.

JG>Sorry, _perijoves_...I'm not used to talking this language.

Couldn't we just say periapsis or apoapsis?
```

hint: definition of perijove is the point in the orbit of a satellite of Jupiter nearest the planet's center

In [249]:

```
np.array(newsgroups_train.target_names)[newsgroups_train.target[:3]]
```

Out[249]:

```
array(['comp.graphics', 'talk.religion.misc', 'sci.space'],
      dtype='<U18')
```

The target attribute is the integer index of the category.

In [58]:

```
newsgroups_train.target[:10]
```

Out[58]:

```
array([1, 3, 2, 0, 2, 0, 2, 1, 2, 1])
```

In [11]:

```
num_topics, num_top_words = 6, 8
```

Next, scikit learn has a method that will extract all the word counts for us.

In [7]:

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

In [342]:

```
vectorizer = CountVectorizer(stop_words='english')
vectors = vectorizer.fit_transform(newsgroups_train.data).todense() # (documents, vocab)
vectors.shape #, vectors.nnz / vectors.shape[0], row_means.shape
```

Out[342]:

```
(2034, 26576)
```

In [343]:

```
print(len(newsgroups_train.data), vectors.shape)
```

```
2034 (2034, 26576)
```

In [303]:

```
vocab = np.array(vectorizer.get_feature_names())
```

In [304]:

```
vocab.shape
```

Out[304]:

```
(26576,)
```

In [18]:

```
vocab[7000:7020]
```

Out[18]:

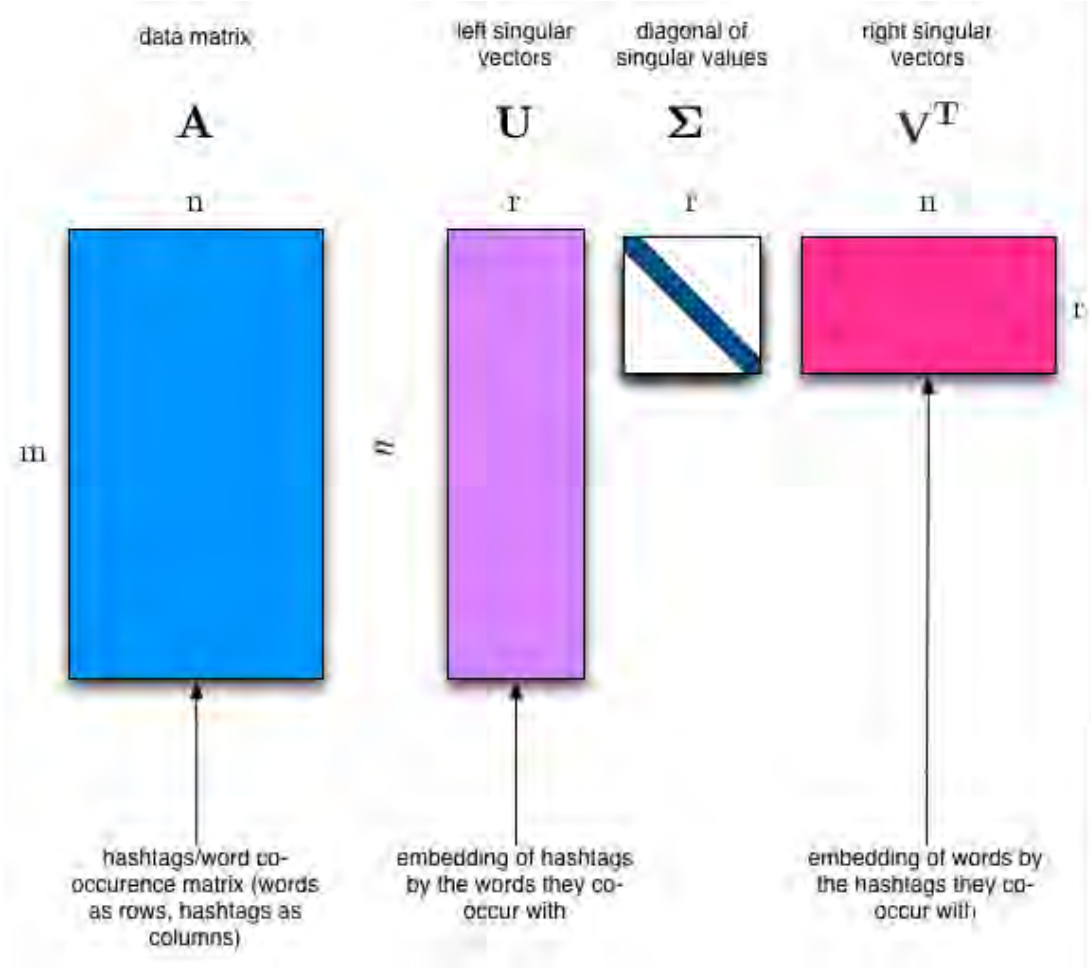
```
array(['cosmonauts', 'cosmos', 'cosponsored', 'cost', 'costa', 'costar',
       'costing', 'costly', 'costruction', 'costs', 'cosy', 'cote',
       'couched', 'couldn', 'council', 'councils', 'counsel', 'counselees',
       'counselor', 'count'],
      dtype='<U80')
```

Singular V alue Decomposition (SVD) ¶

"SVD is not nearly as famous as it should be." - Gilbert Strang

We would clearly expect that the words that appear most frequently in one topic would appear less frequently in the other - otherwise that word wouldn't make a good choice to separate out the two topics. Therefore, we expect the topics to be orthogonal .

The SVD algorithm factorizes a matrix into one matrix with orthogonal columns and one with orthogonal rows (along with a diagonal matrix, which contains the relative importance of each factor).



(source: Facebook Research: Fast Randomized SVD)

SVD is an exact decomposition , since the matrices it creates are big enough to fully cover the original matrix. SVD is extremely widely used in linear algebra, and specifically in data science, including:

- semantic analysis
- collaborative filtering/recommendations (winning entry for Netflix Prize)
- calculate Moore-Penrose pseudoinverse
- data compression
- principal component analysis (will be covered later in course)

In [344]:

```
%time U, s, Vh = linalg.svd(vectors, full_matrices=False)
```

```
CPU times: user 27.2 s, sys: 812 ms, total: 28 s
Wall time: 27.9 s
```

In [345]:

```
print(U.shape, s.shape, Vh.shape)
```

```
(2034, 2034) (2034,) (2034, 26576)
```

Confirm this is a decomposition of the input.

Answer ¶

In [346]:

```
#Exercise: confrim that U, s, Vh is a decomposition of the var Vectors
```

Out[346]:

```
True
```

Confirm that U, V are orthonormal

Answer ¶

In [246]:

```
#Exercise: Confirm that U, Vh are orthonormal
```

Out[246]:

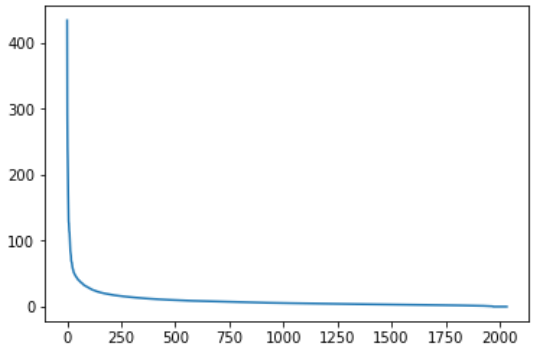
```
True
```

Topics ¶

What can we say about the singular values s?

In [96]:

```
plt.plot(s);
```

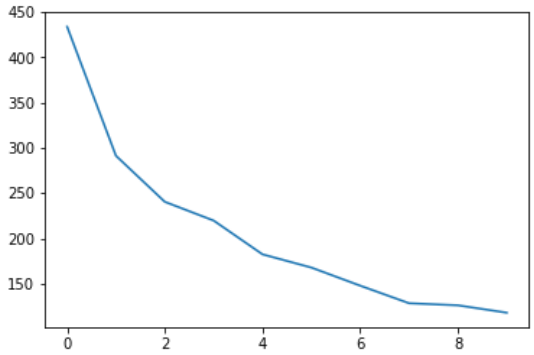


In [97]:

```
plt.plot(s[:10])
```

Out[97]:

```
[<matplotlib.lines.Line2D at 0x7fcada6c6828>]
```



In [52]:

```
num_top_words=8

def show_topics(a):
    top_words = lambda t: [vocab[i] for i in np.argsort(t)[:-num_top_words-1:-1]]
    topic_words = ([top_words(t) for t in a])
    return [' '.join(t) for t in topic_words]
```

In [347]:

```
show_topics(Vh[:10])
```

Out[347]:

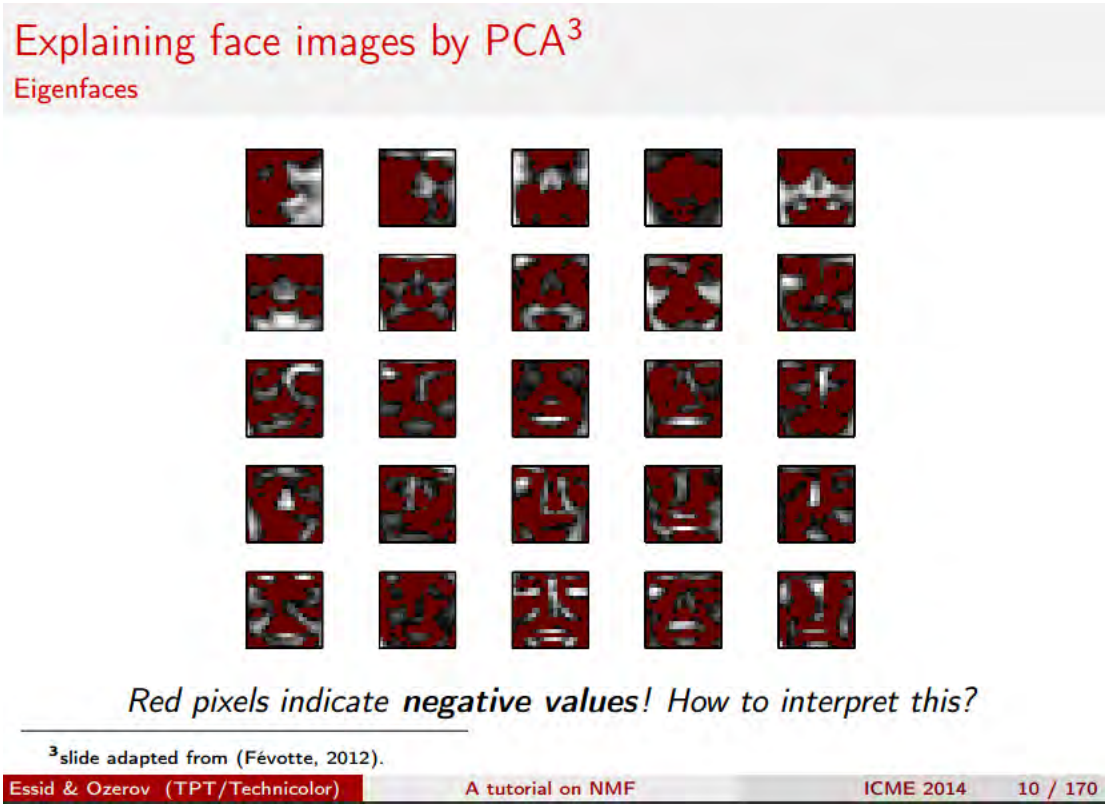
```
['critus ditto propagandist surname galacticentric kindergarten surreal imaginative',
 'jpeg gif file color quality image jfif format',
 'graphics edu pub mail 128 3d ray ftp',
 'jesus god matthew people atheists atheism does graphics',
 'image data processing analysis software available tools display',
 'god atheists atheism religious believe religion argument true',
 'space nasa lunar mars probe moon missions probes',
 'image probe surface lunar mars probes moon orbit',
 'argument fallacy conclusion example true ad argumentum premises',
 'space larson image theory universe physical nasa material']
```

We get topics that match the kinds of clusters we would expect! This is despite the fact that this is an unsupervised algorithm - which is to say, we never actually told the algorithm how our documents are grouped.

We will return to SVD in much more detail later. For now, the important takeaway is that we have a tool that allows us to exactly factor a matrix into orthogonal columns and orthogonal rows.

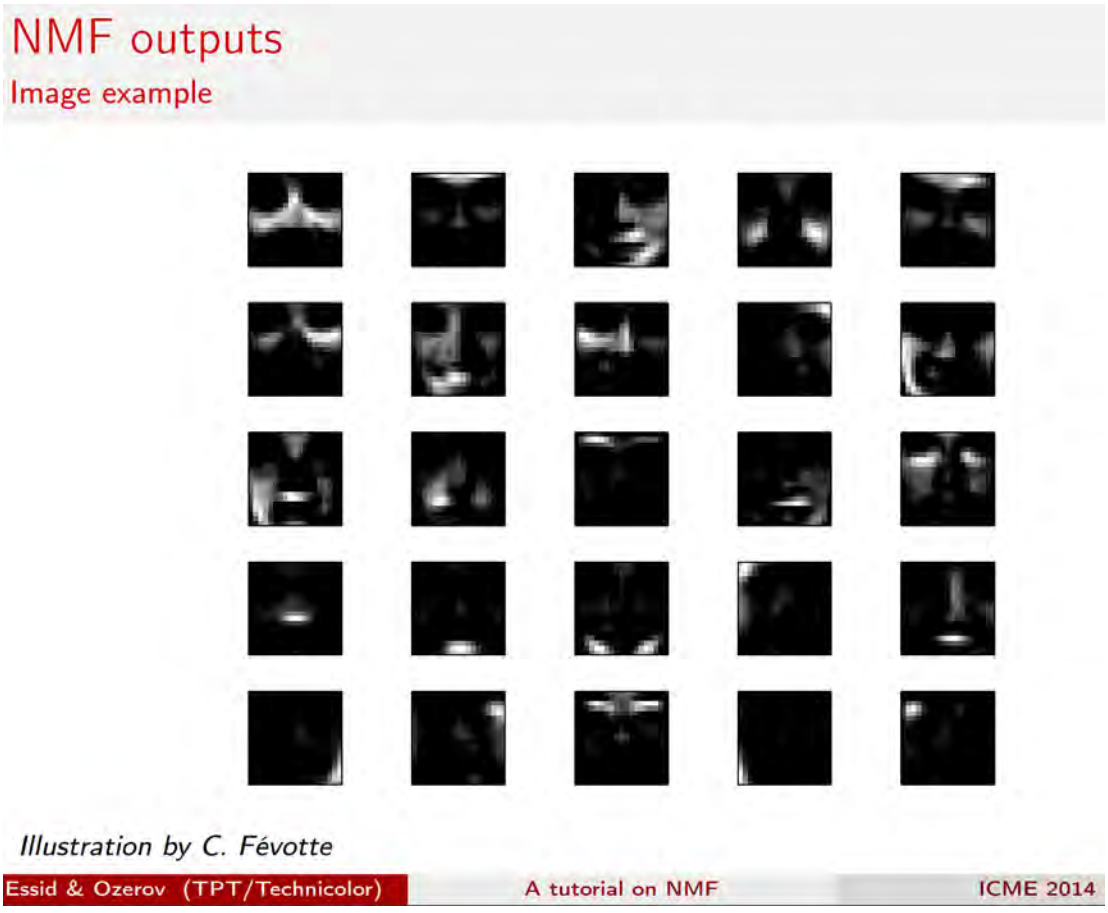
Non-negative Matrix Factorization (NMF) ¶

Motivation ¶



(source: NMF Tutorial)

A more interpretable approach:



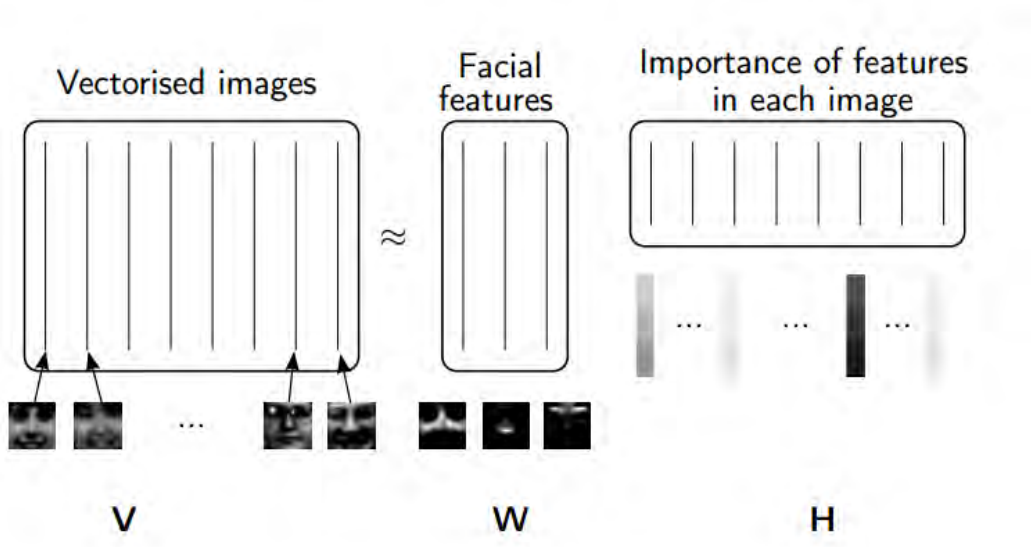
(source: NMF Tutorial)

Idea¶

Rather than constraining our factors to be orthogonal, another idea would to constrain them to be non-negative. NMF is a factorization of a non-negative data set V:

$$V = WH$$

into non-negative matrices W, H. Often positive factors will be more easily interpretable (and this is the reason behind NMF's popularity).

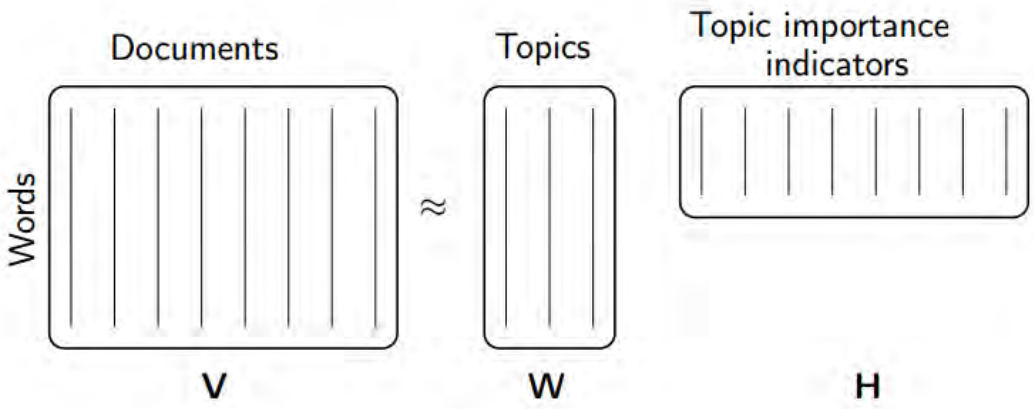


(source: NMF Tutorial)

Nonnegative matrix factorization (NMF) is a non-exact factorization that factors into one skinny positive matrix and one short positive matrix. NMF is NP-hard and non-unique. There are a number of variations on it, created by adding different constraints.

Applications of NMF ¶

- Face Decompositions
- Collaborative Filtering, eg movie recommendations
- Audio source separation
- Chemistry
- Bioinformatics and Gene Expression
- Topic Modeling (our problem!)



(source: NMF Tutorial)

More Reading :

- The Why and How of Nonnegative Matrix Factorization

NMF from sklearn ¶

First, we will use scikit-learn's implementation of NMF:

In [13]:

```
m,n=vectors.shape
d=5  # num topics
```

In [363]:

```
clf = decomposition.NMF(n_components=d, random_state=1)

W1 = clf.fit_transform(vectors)
H1 = clf.components_
```

In [296]:

```
show_topics(H1)
```

Out[296]:

```
['jpeg image gif file color images format quality',
 'edu graphics pub mail 128 ray ftp send',
 'space launch satellite nasa commercial satellites year market',
 'jesus matthew prophecy people said messiah david isaiah',
 'image data available software processing ftp edu analysis',
 'god atheists atheism religious believe people religion does']
```

TF-IDF¶

Topic Frequency-Inverse Document Frequency (TF-IDF) is a way to normalize term counts by taking into account how often they appear in a document, how long the document is, and how commmon/rare the term is.

TF = (# occurrences of term t in document) / (# of words in documents)

IDF = log(# of documents / # documents with term t in it)

In [ ]:

```
vectorizer_tfidf = TfidfVectorizer(stop_words='english')
vectors_tfidf = vectorizer_tfidf.fit_transform(newsgroups_train.data) # (documents, vocab)
```

In [263]:

```
W1 = clf.fit_transform(vectors_tfidf)
H1 = clf.components_
```

In [255]:

```
show_topics(H1)
```

Out[255]:

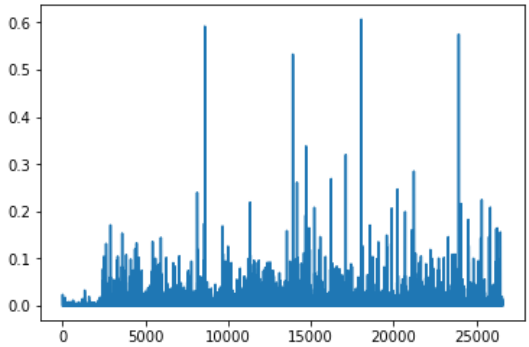
```
['don people just think like know say religion',
 'thanks graphics files image file program windows format',
 'space nasa launch shuttle orbit lunar moon earth',
 'ico bobbe tek beauchaine bronx manhattan sank queens',
 'god jesus bible believe atheism christian does belief',
 'objective morality values moral subjective science absolute claim']
```

In [26]:

```
plt.plot(clf.components_[0])
```

Out[26]:

```
[<matplotlib.lines.Line2D at 0x7f0e1039a7b8>]
```



In [27]:

```
clf.reconstruction_err_
```

Out[27]:

```
43.71292605795277
```

NMF in summary ¶

Benefits: Fast and easy to use!

Downsides: took years of research and expertise to create

Notes:

- For NMF, matrix needs to be at least as tall as it is wide, or we get an error with fit_transform
- Can use df_min in CountVectorizer to only look at words that were in at least k of the split texts

NMF from scratch in numpy , using SGD ¶

Gradient Descent ¶

The key idea of standard gradient descent :

1. Randomly choose some weights to start
2. Loop:
    - Use weights to calculate a prediction
    - Calculate the derivative of the loss
    - Update the weights
3. Repeat step 2 lots of times. Eventually we end up with some decent weights.

Key: We want to decrease our loss and the derivative tells us the direction of steepest descent .

Note that loss, error, and cost are all terms used to describe the same thing.

Let's take a look at the Gradient Descent Intro notebook (originally from the fast.ai deep learning course).

Stochastic Gradient Descent (SGD) ¶

Stochastic gradient descent is an incredibly useful optimization method (it is also the heart of deep learning, where it is used for backpropagation).

For standard gradient descent, we evaluate the loss using all of our data which can be really slow. In stochastic gradient descent, we evaluate our loss function on just a sample of our data (sometimes called a mini-batch). We would get different loss values on different samples of the data, so this is why it is stochastic. It turns out that this is still an effective way to optimize, and it's much more efficient!

We can see how this works in this excel spreadsheet (originally from the fast.ai deep learning course).

Resources :

- SGD Lecture from Andrew Ng's Coursera ML course
- fast.ai wiki page on SGD
- Gradient Descent For Machine Learning (Jason Brownlee- Machine Learning Mastery)
- An overview of gradient descent optimization algorithms

Applying SGD to NMF ¶

Goal: Decompose V $(m \times n)$ into

$$V \approx WH$$

where W $(m \times d)$ and H $(d \times n)$, $W,\ H\ >=\ 0$, and we've minimized the Frobenius norm of $V - WH$.

Approach : We will pick random positive W & H, and then use SGD to optimize.

To use SGD, we need to know the gradient of the loss function.

Sources :

- Optimality and gradients of NMF: http://users.wfu.edu/plemmons/papers/chu_ple.pdf
- Projected gradients: https://www.csie.ntu.edu.tw/~cjlin/papers/pgradnmf.pdf

In [272]:

```
lam=1e3
lr=1e-2
m, n = vectors_tfidf.shape
```

In [252]:

```
W1 = clf.fit_transform(vectors)
H1 = clf.components_
```

In [253]:

```
show_topics(H1)
```

Out[253]:

```
['jpeg image gif file color images format quality',
 'edu graphics pub mail 128 ray ftp send',
 'space launch satellite nasa commercial satellites year market',
 'jesus matthew prophecy people said messiah david isaiah',
 'image data available software processing ftp edu analysis',
 'god atheists atheism religious believe people religion does']
```

In [265]:

```
mu = 1e-6
def grads(M, W, H):
    R = W@H-M
    return R@H.T + penalty(W, mu)*lam, W.T@R + penalty(H, mu)*lam # dW, dH
```

In [266]:

```
def penalty(M, mu):
    return np.where(M>=mu,0, np.min(M - mu, 0))
```

In [267]:

```
def upd(M, W, H, lr):
    dW,dH = grads(M,W,H)
    W -= lr*dW; H -= lr*dH
```

In [268]:

```
def report(M,W,H):
    print(np.linalg.norm(M-W@H), W.min(), H.min(), (W<0).sum(), (H<0).sum())
```

In [348]:

```
W = np.abs(np.random.normal(scale=0.01, size=(m,d)))
H = np.abs(np.random.normal(scale=0.01, size=(d,n)))
```

In [349]:

```
report(vectors_tfidf, W, H)
```

```
44.4395133509 5.67503308167e-07 2.49717354504e-07 0 0
```

In [350]:

```
upd(vectors_tfidf,W,H,lr)
```

In [351]:

```
report(vectors_tfidf, W, H)
```

```
44.4194155587 -0.00186845669883 -0.000182969569359 509 788
```

In [352]:

```
for i in range(50):
    upd(vectors_tfidf,W,H,lr)
    if i % 10 == 0: report(vectors_tfidf,W,H)
```

```
44.4071645597 -0.00145791197281 -0.00012862260312 343 1174
44.352156176 -0.000549676823494 -9.16363641124e-05 218 4689
44.3020593384 -0.000284017335617 -0.000130903875061 165 9685
44.2468609535 -0.000279317810433 -0.000182173029912 169 16735
44.199218 -0.000290092649623 -0.000198140867356 222 25109
```

In [281]:

```
show_topics(H)
```

Out[281]:

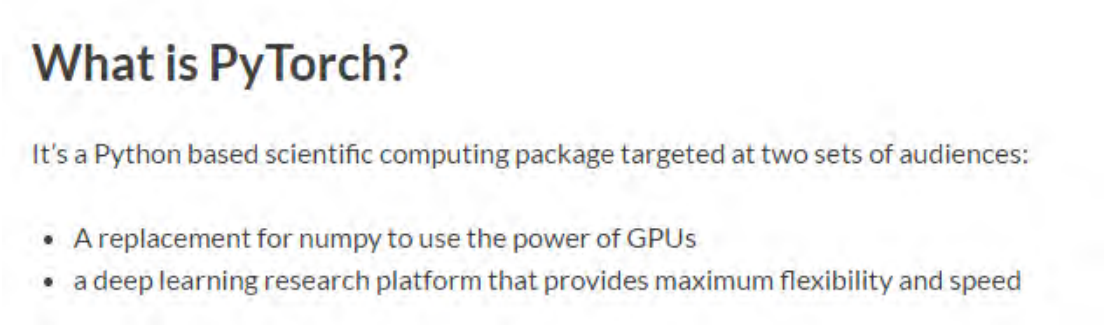
```
['cview file image edu files use directory temp',
 'moral like time does don software new years',
 'god jesus bible believe objective exist atheism belief',
 'thanks graphics program know help looking windows advance',
 'space nasa launch shuttle orbit station moon lunar',
 'people don said think ico tek bobbe bronx']
```

This is painfully slow to train! Lots of parameter fiddling and still slow to train (or explodes).

PyTorch ¶

PyTorch is a Python framework for tensors and dynamic neural networks with GPU acceleration. Many of the core contributors work on Facebook's AI team. In many ways, it is similar to Numpy, only with the increased parallelization of using a GPU.

From the PyTorch documentation:



Further learning ：If you are curious to learn what dynamic neural networks are, you may want to watch this talk by Soumith Chintala, Facebook AI researcher and core PyTorch contributor.

If you want to learn more PyTorch, you can try this tutorial or this learning by examples.

Note about GPUs ：If you are not using a GPU, you will need to remove the `.cuda()` from the methods below. GPU usage is not required for this course, but I thought it would be of interest to some of you. To learn how to create an AWS instance with a GPU, you can watch the fast.ai setup lesson.

In [282]:

```
import torch
import torch.cuda as tc
from torch.autograd import Variable
```

In [283]:

```
def V(M): return Variable(M, requires_grad=True)
```

In [284]:

```
v=vectors_tfidf.todense()
```

In [285]:

```
t_vectors = torch.Tensor(v.astype(np.float32)).cuda()
```

In [286]:

```
mu = 1e-5
```

In [287]:

```
def grads_t(M, W, H):
    R = W.mm(H)-M
    return (R.mm(H.t()) + penalty_t(W, mu)*lam,
        W.t().mm(R) + penalty_t(H, mu)*lam) # dW, dH

def penalty_t(M, mu):
    return (M<mu).type(tc.FloatTensor)*torch.clamp(M - mu, max=0.)

def upd_t(M, W, H, lr):
    dW,dH = grads_t(M,W,H)
    W.sub_(lr*dW); H.sub_(lr*dH)

def report_t(M,W,H):
    print((M-W.mm(H)).norm(2), W.min(), H.min(), (W<0).sum(), (H<0).sum())
```

In [354]:

```
t_W = tc.FloatTensor(m,d)
t_H = tc.FloatTensor(d,n)
t_W.normal_(std=0.01).abs_();
t_H.normal_(std=0.01).abs_();
```

In [355]:

```
d=6; lam=100; lr=0.05
```

In [356]:

```
for i in range(1000):
    upd_t(t_vectors,t_W,t_H,lr)
    if i % 100 == 0:
        report_t(t_vectors,t_W,t_H)
        lr *= 0.9
```

```
44.392791748046875 -0.0060190423391759396 -0.0004986411076970398 1505 2282
43.746803283691406 -0.009054708294570446 -0.011047963984310627 2085 23854
43.702056884765625 -0.008214150555431843 -0.014783496037125587 2295 24432
43.654273986816406 -0.006195350084453821 -0.006913348101079464 2625 22663
43.646759033203125 -0.004638500511646271 -0.003197424579411745 2684 23270
43.645320892333984 -0.005679543130099773 -0.00419645756483078 3242 24199
43.6449089050293 -0.0041352929547429085 -0.00843129213899374 3282 25030
43.64469528198242 -0.003943094052374363 -0.00745873199775815 3129 26220
43.64459991455078 -0.004347225651144981 -0.007400824688374996 3031 26323
43.64434051513672 -0.0039044099394232035 -0.0067480215802788734 3930 33718
```

In [291]:

```
show_topics(t_H.cpu().numpy())
```

Out[291]:

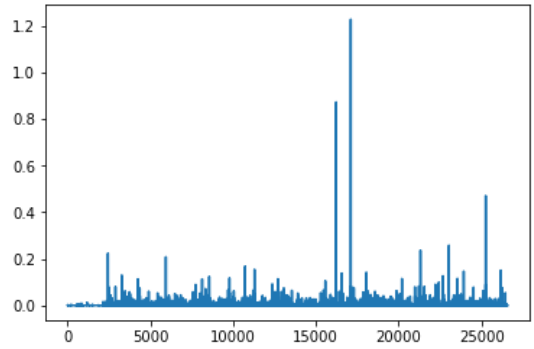
```
['objective morality values moral subjective science absolute claim',
 'god jesus bible believe atheism christian does belief',
 'ico bobbe tek bronx beauchaine manhattan sank queens',
 'thanks graphics files image file program windows know',
 'space nasa launch shuttle orbit lunar moon earth',
 'don people just think like know say religion']
```

In [292]:

```
plt.plot(t_H.cpu().numpy()[0])
```

Out[292]:

```
[<matplotlib.lines.Line2D at 0x7fe4173f1d68>]
```



In [1328]:

```
t_W.mm(t_H).max()
```

Out[1328]:

```
0.43389660120010376
```

In [1329]:

```
t_vectors.max()
```

Out[1329]:

```
0.9188119769096375
```

PyTorch: autograd ¶

Above, we used our knowledge of what the gradient of the loss function was to do SGD from scratch in PyTorch. However, PyTorch has an automatic differentiation package, autograd which we could use instead. This is really useful, in that we can use autograd on problems where we don't know what the derivative is.

The approach we use below is very general, and would work for almost any optimization problem.

In PyTorch, Variables have the same API as tensors, but Variables remember the operations used on to create them. This lets us take derivatives.

PyTorch Autograd Introduction ¶

Example taken from this tutorial in the official documentation.

In [375]:

```
x = Variable(torch.ones(2, 2), requires_grad=True)
print(x)
```

```
Variable containing:
 1  1
 1  1
[torch.FloatTensor of size 2x2]
```

In [376]:

```
print(x.data)
```

```
 1  1
 1  1
[torch.FloatTensor of size 2x2]
```

In [377]:

```
print(x.grad)
```

```
Variable containing:
 0  0
 0  0
```

```
[torch.FloatTensor of size 2x2]
```

In [378]:

```
y = x + 2
print(y)
```

```
Variable containing:
 3  3
 3  3
[torch.FloatTensor of size 2x2]
```

In [383]:

```
z = y * y * 3
out = z.sum()
print(z, out)
```

```
Variable containing:
 27  27
 27  27
[torch.FloatTensor of size 2x2]
 Variable containing:
 108
[torch.FloatTensor of size 1]
```

In [382]:

```
out.backward()
print(x.grad)
```

```
Variable containing:
 18  18
 18  18
[torch.FloatTensor of size 2x2]
```

Using Autograd for NMF ¶

In [167]:

```
lam=1e6
```

In [168]:

```
pW = Variable(tc.FloatTensor(m,d), requires_grad=True)
pH = Variable(tc.FloatTensor(d,n), requires_grad=True)
pW.data.normal_(std=0.01).abs_()
pH.data.normal_(std=0.01).abs_();
```

In [357]:

```
def report():
    W,H = pW.data, pH.data
    print((M-pW.mm(pH)).norm(2).data[0], W.min(), H.min(), (W<0).sum(), (H<0).sum())

def penalty(A):
    return torch.pow((A<0).type(tc.FloatTensor)*torch.clamp(A, max=0.), 2)

def penalize(): return penalty(pW).mean() + penalty(pH).mean()

def loss(): return (M-pW.mm(pH)).norm(2) + penalize()*lam
```

In [358]:

```
M = Variable(t_vectors).cuda()
```

In [359]:

```
opt = torch.optim.Adam([pW,pH], lr=1e-3, betas=(0.9,0.9))
lr = 0.05
report()
```

```
43.66044616699219 -0.0002547535696066916 -0.00046720390673726797 319 8633
```

How to apply SGD, using autograd:

In [361]:

```
for i in range(1000):
    opt.zero_grad()
    l = loss()
    l.backward()
    opt.step()
    if i % 100 == 99:
        report()
        lr *= 0.9     # learning rate annealling
```

```
43.628597259521484 -0.022899555042386055 -0.26526615023612976 692 82579
43.62860107421875 -0.021287493407726288 -0.2440912425518036 617 77552
43.628597259521484 -0.020111067220568657 -0.22828206419944763 576 77726
43.628604888916016 -0.01912039890885353 -0.21654289960861206 553 84411
43.62861251831055 -0.018248897045850754 -0.20736189186573029 544 75546
43.62862014770508 -0.01753264293074608 -0.19999365508556366 491 78949
43.62862777709961 -0.016773322597146034 -0.194113627076149 513 83822
43.628639221191406 -0.01622121036052704 -0.18905577063560486 485 74101
43.62863540649414 -0.01574397087097168 -0.18498440086841583 478 85987
43.628639221191406 -0.015293922275304794 -0.18137598037719727 487 74023
```

In [362]:

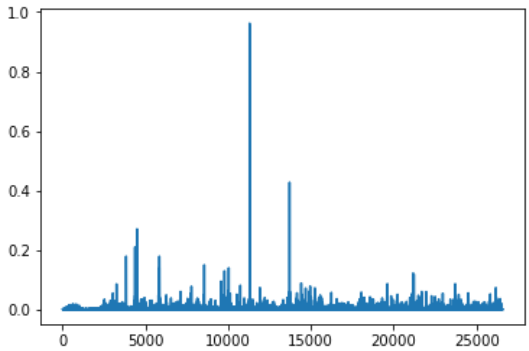
```
h = pH.data.cpu().numpy()
show_topics(h)
```

Out[362]:

```
['god jesus bible believe atheism christian belief does',
 'thanks graphics files image file windows program format',
 'just don think people like ve graphics religion',
 'objective morality values moral subjective science absolute claim',
 'ico bobbe tek beauchaine bronx manhattan sank queens',
 'space nasa shuttle launch orbit lunar moon data']
```

In [174]:

```
plt.plot(h[0]);
```



Comparing Approaches ¶

Scikit-Learn's NMF ¶

- Fast
- No parameter tuning
- Relies on decades of academic research, took experts a long time to implement

Factorization Algorithms

- **BD** - Bayesian nonnegative matrix factorization Gibbs sampler [Schmidt2009]
- **BMF** - Binary matrix factorization [Zhang2007]
- **ICM** - Iterated conditional modes nonnegative matrix factorization [Schmidt2009]
- **LFNMF** - Fisher nonnegative matrix factorization for learning local features [Wang2004], [Li2001]
- **LSNMF** - Alternating nonnegative least squares matrix factorization using projected gradient method for subproblems [Lin2007]
- **NMF** - Standard nonnegative matrix factorization with Euclidean / Kullback-Leibler update equations and Frobenius / divergence / connectivity cost functions [Lee2001], [Brunet2004]
- **NSNMF** - Nonsmooth nonnegative matrix factorization [Montano2006]
- **PMF** - Probabilistic nonnegative matrix factorization [Laurberg2008], [Hansen2008]
- **PSMF** - Probabilistic sparse matrix factorization [Dueck2005], [Dueck2004], [Srebro2001], [Li2007]
- **SNMF** - Sparse nonnegative matrix factorization based on alternating nonnegativity constrained least squares [Park2007]
- **SNMNMF** - Sparse network-regularized multiple nonnegative matrix factorization [Zhang2011]
- **PMFCC** - Penalized matrix factorization for constrained clustering [FWang2008]
- **SepNMF** - Separable nonnegative matrix factorization [Gillis2014], [Kumar2013], [Tepper2015], [Kapralov2016]

source: Python Nimfa

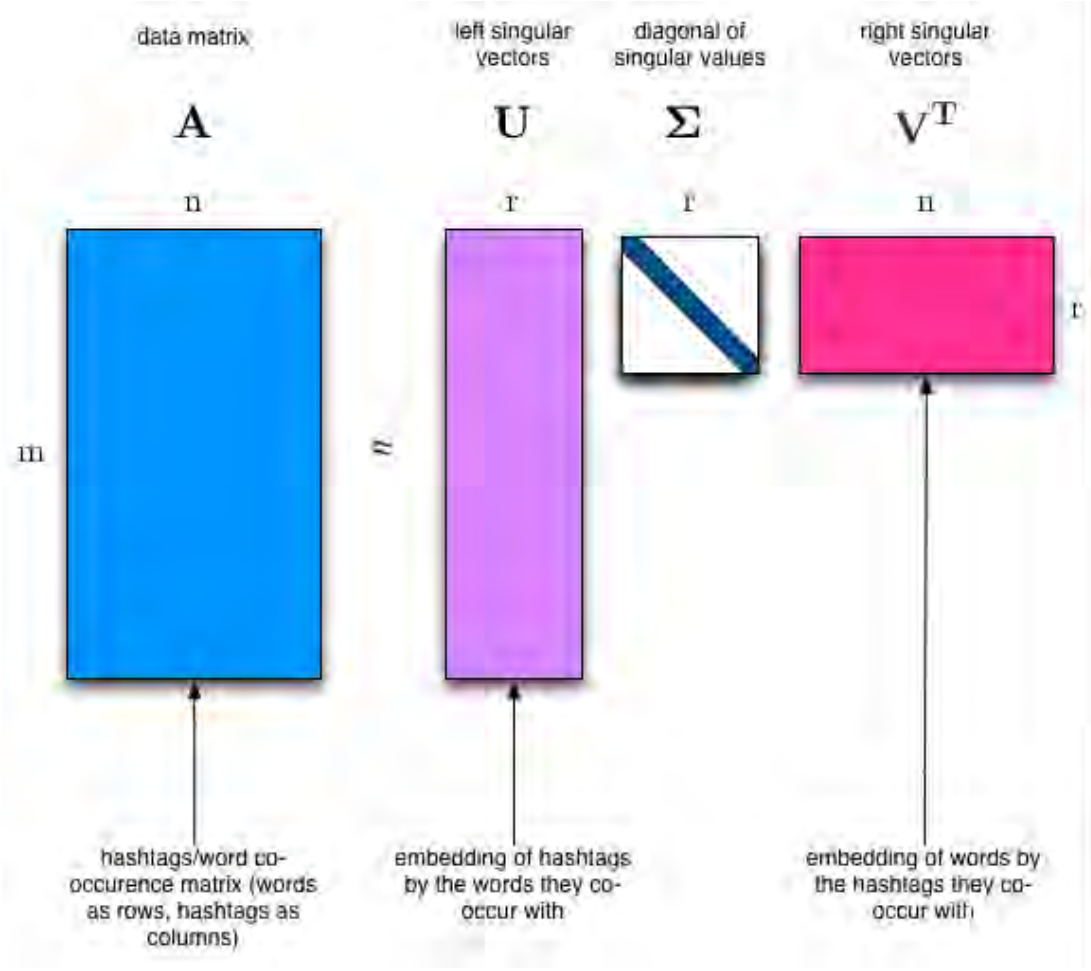
Documentation

Using PyT orch and SGD ¶

- Took us an hour to implement, didn't have to be NMF experts
- Parameters were fiddly
- Not as fast (tried in numpy and was so slow we had to switch to PyTorch)

Truncated SVD ¶

We saved a lot of time when we calculated NMF by only calculating the subset of columns we were interested in. Is there a way to get this benefit with SVD? Yes there is! It's called truncated SVD. We are just interested in the vectors corresponding to the largest singular values.



(source: Facebook Research: Fast Randomized SVD)

Shortcomings of classical algorithms for decomposition: ¶

- Matrices are "stupendously big"
- Data are often missing or inaccurate . Why spend extra computational resources when imprecision of input limits precision of the output?
- Data transfer now plays a major role in time of algorithms. Techniques the require fewer passes over the data may be substantially faster, even if they require more flops (flops = floating point operations).
- Important to take advantage of GPUs.

(source: Halko)

Advantages of randomized algorithms: ¶

- inherently stable
- performance guarantees do not depend on subtle spectral properties
- needed matrix-vector products can be done in parallel

(source: Halko)

Randomized SVD ¶

Reminder: full SVD is slow . This is the calculation we did above using Scipy's Linalg SVD:

In [384]:

```
vectors.shape
```

Out[384]:

```
(2034, 26576)
```

In [344]:

```
%time U, s, Vh = linalg.svd(vectors, full_matrices=False)
```

```
CPU times: user 27.2 s, sys: 812 ms, total: 28 s
Wall time: 27.9 s
```

In [345]:

```
print(U.shape, s.shape, Vh.shape)
```

```
(2034, 2034) (2034,) (2034, 26576)
```

Fortunately, there is a faster way:

In [175]:

```
%time u, s, v = decomposition.randomized_svd(vectors, 5)
```

```
CPU times: user 144 ms, sys: 8 ms, total: 152 ms
Wall time: 154 ms
```

The runtime complexity for SVD is $\mathcal{O}(\min(m^2n,\ mn^2))$

Question : How can we speed things up? (without new breakthroughs in SVD research)

Idea: Let's use a smaller matrix (with smaller n)!

Instead of calculating the SVD on our full matrix A which is $m \times n$, let's use $B = AQ$, which is just $m \times r$ and $r << n$

We haven't found a better general SVD method, we are just using the method we have on a smaller matrix.

In [175]:

```
%time u, s, v = decomposition.randomized_svd(vectors, 5)
```

```
CPU times: user 144 ms, sys: 8 ms, total: 152 ms
Wall time: 154 ms
```

In [177]:

```
u.shape, s.shape, v.shape
```

Out[177]:

```
((2034, 5), (5,), (5, 26576))
```

In [178]:

```
show_topics(v)
```

Out[178]:

```
['jpeg image edu file graphics images gif data',
 'jpeg gif file color quality image jfif format',
 'space jesus launch god people satellite matthew atheists',
 'jesus god matthew people atheists atheism does graphics',
 'image data processing analysis software available tools display']
```

Here are some results from Facebook Research:

Benchmarks

We consider a few cases. We consider

- Large dense matrices, up to 100,000 × 200,000 in size,
- Large sparse matrices, up to $10^7 \times 10^7$ in size with $O(10^9)$ non-zero elements,
- A comparison to Apache Spark's distributed SVD implementation.

Throughout, we compute a rank 10 approximation, and we run two iterations of the power method. For the dense case, we compare against `numpy.linalg.svd`, and the sparse case, we compare against `scipy.sparse.linalg.svds`.

Timings

$10^6 \times 10^5$ matrix with 10^7 non-zeros, 1 second vs 100 seconds with ARPACK,
$10^6 \times 10^5$ matrix with 10^8 non-zeros, 5 seconds vs 63 minutes with ARPACK,
$10^5 \times 10^5$ dense matrix in 120 seconds,
$9.4 * 10^7 \times 4 * 10^3$ matrix with $1.6 * 10^9$ non-zeros, 60 seconds on single machine server vs 50 seconds on a 68 machine cluster with Spark.

Johnson-Lindenstrauss Lemma : (from wikipedia) a small set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved.

It is desirable to be able to reduce dimensionality of data in a way that preserves relevant structure. The Johnson–Lindenstrauss lemma is a classic result of this type.

Implementing our own Randomized SVD ¶

In [112]:

```
from scipy import linalg
```

The method `randomized_range_finder` finds an orthonormal matrix whose range approximates the range of A (step 1 in our algorithm above). To do so, we use the LU and QR factorizations, both of which we will be covering in depth later.

I am using the scikit-learn.extmath.randomized_svd source code as a guide.

In [182]:

```
# computes an orthonormal matrix whose range approximates the range of A
# power_iteration_normalizer can be safe_sparse_dot (fast but unstable), LU (imbetween), or QR (slow but most accurate)
def randomized_range_finder(A, size, n_iter=5):
    Q = np.random.normal(size=(A.shape[1], size))

    for i in range(n_iter):
        Q, _ = linalg.lu(A @ Q, permute_l=True)
        Q, _ = linalg.lu(A.T @ Q, permute_l=True)

    Q, _ = linalg.qr(A @ Q, mode='economic')
    return Q
```

And here's our randomized SVD method:

In [236]:

```
def randomized_svd(M, n_components, n_oversamples=10, n_iter=4):

    n_random = n_components + n_oversamples

    Q = randomized_range_finder(M, n_random, n_iter)

    # project M to the (k + p) dimensional space using the basis vectors
    B = Q.T @ M
```

```
    # compute the SVD on the thin matrix: (k + p) wide
    Uhat, s, V = linalg.svd(B, full_matrices=False)
    del B
    U = Q @ Uhat

    return U[:, :n_components], s[:n_components], V[:n_components, :]
```

In [237]:

```
u, s, v = randomized_svd(vectors, 5)
```

In [238]:

```
%time u, s, v = randomized_svd(vectors, 5)
```

```
CPU times: user 136 ms, sys: 0 ns, total: 136 ms
Wall time: 137 ms
```

In [239]:

```
u.shape, s.shape, v.shape
```

Out[239]:

```
((2034, 5), (5,), (5, 26576))
```

In [247]:

```
show_topics(v)
```

Out[247]:

```
['jpeg image edu file graphics images gif data',
 'edu graphics data space pub mail 128 3d',
 'space jesus launch god people satellite matthew atheists',
 'space launch satellite commercial nasa satellites market year',
 'image data processing analysis software available tools display']
```

Write a loop to calculate the error of your decomposition as you vary the # of topics. Plot the result

Answer ¶

In [248]:

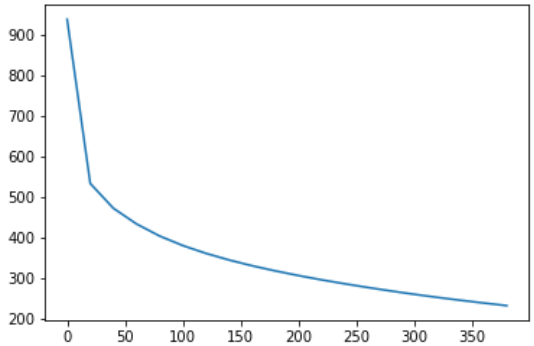
```
#Exercise: Write a loop to calculate the error of your decomposition as you vary the # of topics
```

In [242]:

```
plt.plot(range(0,n*step,step), error)
```

Out[242]:

```
[<matplotlib.lines.Line2D at 0x7fe3f8a1b438>]
```



Further Resources :

- a whole course on randomized algorithms

More Details ¶

Here is a process to calculate a truncated SVD, described in Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions and summarized in this blog post:

1. Compute an approximation to the range of A. That is, we want Q with r orthonormal columns such that

$$A \approx QQ^T A$$

2. Construct $B = Q^T A$, which is small ($r \times n$)

3. Compute the SVD of B by standard methods (fast since B is smaller than A), $B = S\,\Sigma V^T$

4. Since

$$A \approx QQ^T A = Q(S\,\Sigma V^T)$$

if we set $U = QS$, then we have a low rank approximation $A \approx U\Sigma V^T$.

So how do we find Q (in step 1)? ¶

To estimate the range of A, we can just take a bunch of random vectors w_i, evaluate the subspace formed by Aw_i. We can form a matrix W with the w_i as it's columns. Now, we take the QR decomposition of $AW = QR$, then the columns of Q form an orthonormal basis for AW, which is the range of A.

Since the matrix AW of the product has far more rows than columns and therefore, approximately, orthonormal columns. This is simple probability - with lots of rows, and few columns, it's unlikely that the columns are linearly dependent.

The QR Decomposition ¶

We will be learning about the QR decomposition in depth later on. For now, you just need to know that $A = QR$, where Q consists of orthonormal columns, and R is upper triangular. Trefethen says that the QR decomposition is the most important idea in numerical linear algebra! We will definitely be returning to it.

How should we choose r?¶

Suppose our matrix has 100 columns, and we want 5 columns in U and V. To be safe, we should project our matrix onto an orthogonal basis with a few more rows and columns than 5 (let's use 15). At the end, we will just grab the first 5 columns of U and V

So even although our projection was only approximate, by making it a bit bigger than we need, we can make up for the loss of accuracy (since we're only taking a subset later).

In [175]:

```
%time u, s, v = decomposition.randomized_svd(vectors, 5)
```

```
CPU times: user 144 ms, sys: 8 ms, total: 152 ms
Wall time: 154 ms
```

In [176]:

```
%time u, s, v = decomposition.randomized_svd(vectors.todense(), 5)
```

```
CPU times: user 2.38 s, sys: 592 ms, total: 2.97 s
Wall time: 2.96 s
```

End¶

You can read an overview of this Numerical Linear Algebra course in this blog post. The course was originally taught in the University of San Francisco MS in Analytics graduate program. Course lecture videos are available on YouTube (note that the notebook numbers and video numbers do not line up, since some notebooks took longer than 1 video to cover).

You can ask questions about the course on our fast.ai forums.

3. Background Removal with Robust PCA ¶

Our goal today :



Getting Started ¶

Let's use the real video 003 dataset from BMC 2012 Background Models Challenge Dataset

Other sources of datasets:

- Human Activity Video Datasets
- Background Subtraction Website (a few links on this site are broken/outdated, but many work)

Import needed libraries:

In [31]:

```
import moviepy.editor as mpe
# from IPython.display import display
from glob import glob
```

In [32]:

```
import sys, os
import numpy as np
import scipy
```

In [33]:

```
%matplotlib inline
import matplotlib.pyplot as plt
```

In [34]:

```
# MAX_ITERS = 10
TOL = 1.0e-8
```

In [35]:

```
video = mpe.VideoFileClip("data/Video_003.avi")
```

In [179]:

```
video.subclip(0,50).ipython_display(width=300)
```

```
100%|█████████| 350/351 [00:00<00:00, 1258.81it/s]
```

Out[179]:

In [36]:

```
video.duration
```

Out[36]:

```
113.57
```

Helper Methods ¶

In [37]:

```
def create_data_matrix_from_video(clip, k=5, scale=50):
    return np.vstack([scipy.misc.imresize(rgb2gray(clip.get_frame(i/float(k))).astype(int),
                      scale).flatten() for i in range(k * int(clip.duration))]).T
```

In [38]:

```
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
```

In [83]:

```
def plt_images(M, A, E, index_array, dims, filename=None):
    f = plt.figure(figsize=(15, 10))
    r = len(index_array)
    pics = r * 3
    for k, i in enumerate(index_array):
        for j, mat in enumerate([M, A, E]):
            sp = f.add_subplot(r, 3, 3*k + j + 1)
            sp.axis('Off')
            pixels = mat[:,i]
            if isinstance(pixels, scipy.sparse.csr_matrix):
                pixels = pixels.todense()
            plt.imshow(np.reshape(pixels, dims), cmap='gray')
    return f
```

In [95]:

```
def plots(ims, dims, figsize=(15,20), rows=1, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims)
    f = plt.figure(figsize=figsize)
    for i in range(len(ims)):
        sp = f.add_subplot(rows, len(ims)//rows, i+1)
        sp.axis('Off')
        plt.imshow(np.reshape(ims[i], dims), cmap="gray")
```

Load and view the data ¶

An image from 1 moment in time is 60 pixels by 80 pixels (when scaled). We can unroll that picture into a single tall column. So instead of having a 2D picture that is 60×80, we have a $1 \times 4,800$ column

This isn't very human-readable, but it's handy because it lets us stack the images from different times on top of one another, to put a video all into 1 matrix. If we took the video image every tenth of a second for 113 seconds (so 11,300 different images, each from a different point in time), we'd have a 11300×4800 matrix, representing the video!

In [40]:

```
scale = 25   # Adjust scale to change resolution of image
dims = (int(240 * (scale/100)), int(320 * (scale/100)))
```

In [41]:

```
M = create_data_matrix_from_video(video, 100, scale)
# M = np.load("high_res_surveillance_matrix.npy")
```

In [13]:

```
print(dims, M.shape)
```

```
(60, 80) (4800, 11300)
```

In [79]:

```
plt.imshow(np.reshape(M[:,140], dims), cmap='gray');
```

Since `create_data_from_matrix` is somewhat slow, we will save our matrix. In general, whenever you have slow pre-processing steps, it's a good idea to save the results for future use.

In [42]:

```
np.save("low_res_surveillance_matrix.npy", M)
```

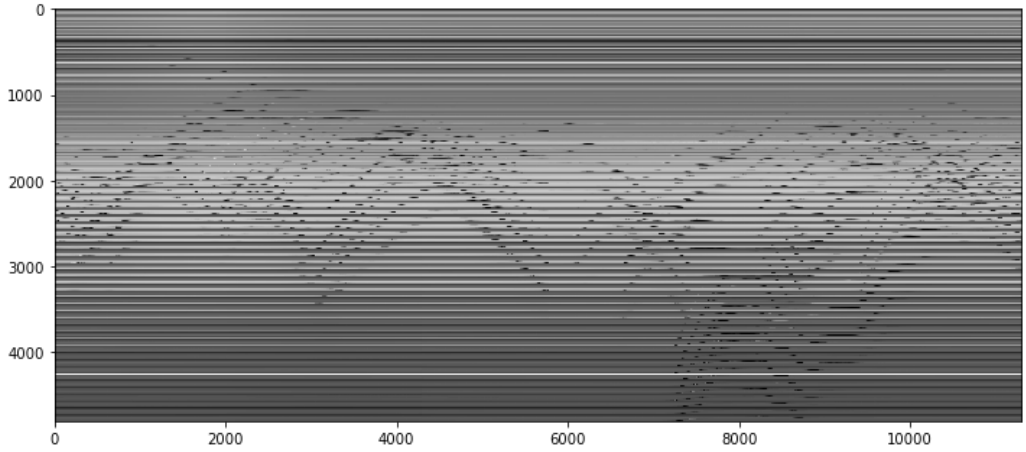
Note: High-res M is too big to plot, so only run the below with the low-res version

In [33]:

```
plt.figure(figsize=(12, 12))
plt.imshow(M, cmap='gray')
```

Out[33]:

```
<matplotlib.image.AxesImage at 0x7f601f315fd0>
```



Questions：What are those wavy black lines? What are the horizontal lines?

In [ ]:

```
plt.imsave(fname="image1.jpg", arr=np.reshape(M[:,140], dims), cmap='gray')
```

SVD¶

Review from Lesson 2 ：

- What kind of matrices are returned by SVD?
- What is a way to speed up truncated SVD?

A first attempt with SVD ¶

In [43]:

```
from sklearn import decomposition
```

In [44]:

```
u, s, v = decomposition.randomized_svd(M, 2)
```

In [45]:

```
u.shape, s.shape, v.shape
```

Out[45]:

```
((4800, 2), (2,), (2, 11300))
```

In [46]:

```
low_rank = u @ np.diag(s) @ v
```

In [47]:

```
low_rank.shape
```

Out[47]:

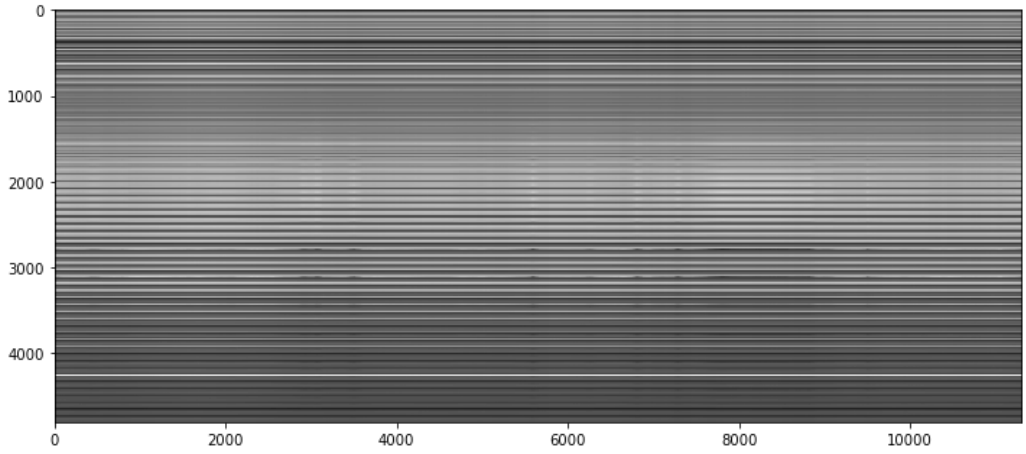
```
(4800, 11300)
```

In [19]:

```
plt.figure(figsize=(12, 12))
plt.imshow(low_rank, cmap='gray')
```

Out[19]:

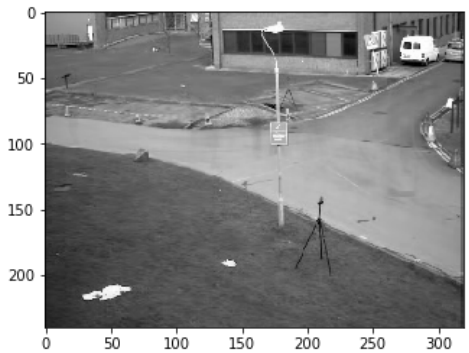
```
<matplotlib.image.AxesImage at 0x7f0d483260b8>
```



The below images were created with high-res data. Very slow to process:

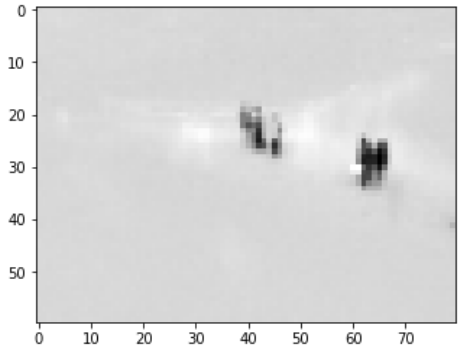
In [18]:

```
plt.imshow(np.reshape(low_rank[:,140], dims), cmap='gray');
```



In [21]:

```
plt.imshow(np.reshape(M[:,550] - low_rank[:,550], dims), cmap='gray');
```



Rank 1 Approximation ¶

In [22]:

```
u, s, v = decomposition.randomized_svd(M, 1)
```

In [23]:

```
u.shape, s.shape, v.shape
```

Out[23]:

```
((4800, 1), (1,), (1, 11300))
```

In [24]:

```
low_rank = u @ np.diag(s) @ v
```

In [25]:

```
low_rank.shape
```

Out[25]:

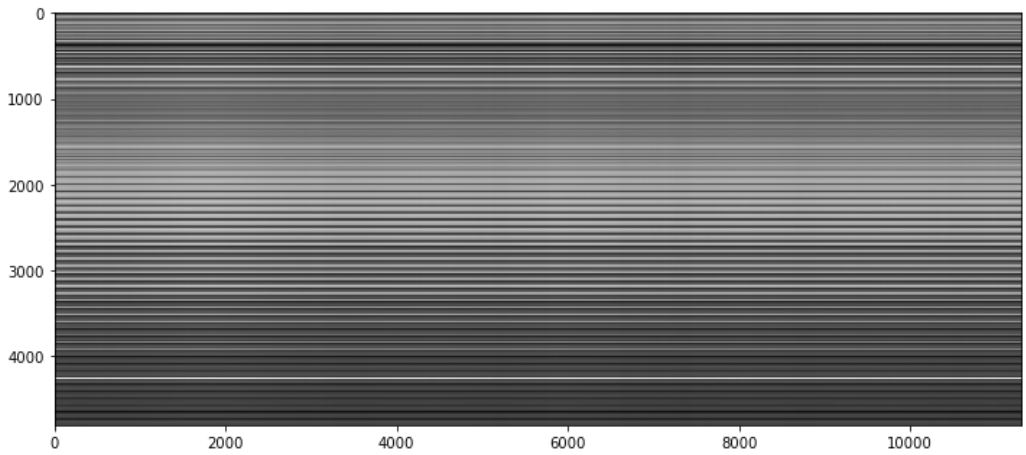
```
(4800, 11300)
```

In [26]:

```
plt.figure(figsize=(12, 12))
plt.imshow(low_rank, cmap='gray')
```

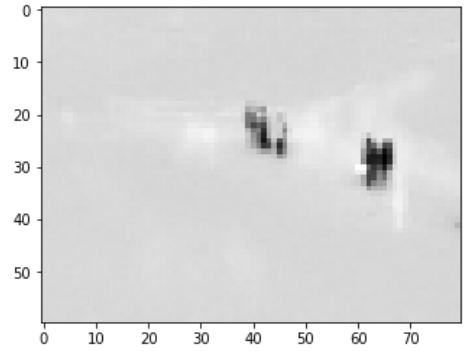
Out[26]:

```
<matplotlib.image.AxesImage at 0x7f0d627902b0>
```



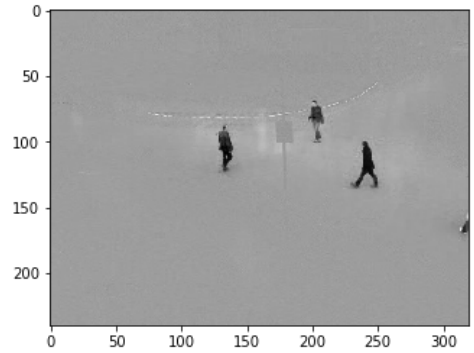
In [27]:

```
plt.imshow(np.reshape(M[:,550] - low_rank[:,550], dims), cmap='gray');
```



In [35]:

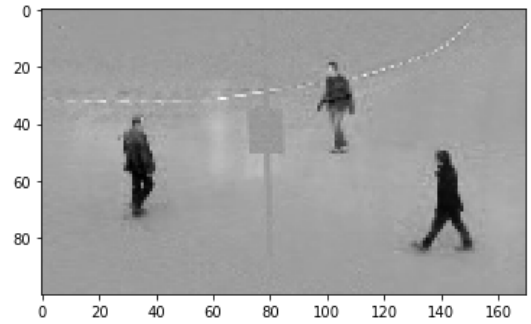
```
plt.imshow(np.reshape(M[:,140] - low_rank[:,140], dims), cmap='gray');
```



Let's zoom in on the people:

In [36]:

```
plt.imshow(np.reshape(M[:,140] - low_rank[:,140], dims)[50:150,100:270], cmap='gray');
```



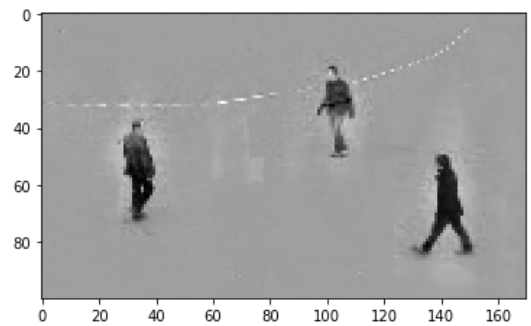
This is amazing for such a simple approach! We get somewhat better results through a more complicated algorithm below.

In [33]:

```
plt.imshow(np.reshape(S[:,140], dims)[50:150,100:270], cmap='gray')
```

Out[33]:

```
<matplotlib.image.AxesImage at 0x7f25ce9476a0>
```



Principal Component Analysis (PCA) ¶

When dealing with high-dimensional data sets, we often leverage on the fact that the data has low intrinsic dimensionality in order to alleviate the curse of dimensionality and scale (perhaps it lies in a low-dimensional subspace or lies on a low-dimensional manifold). Principal component analysis is handy for eliminating dimensions. Classical PCA seeks the best rank-k estimate L of M (minimizing $||M - L||$ where L has rank-k). Truncated SVD makes this calculation!

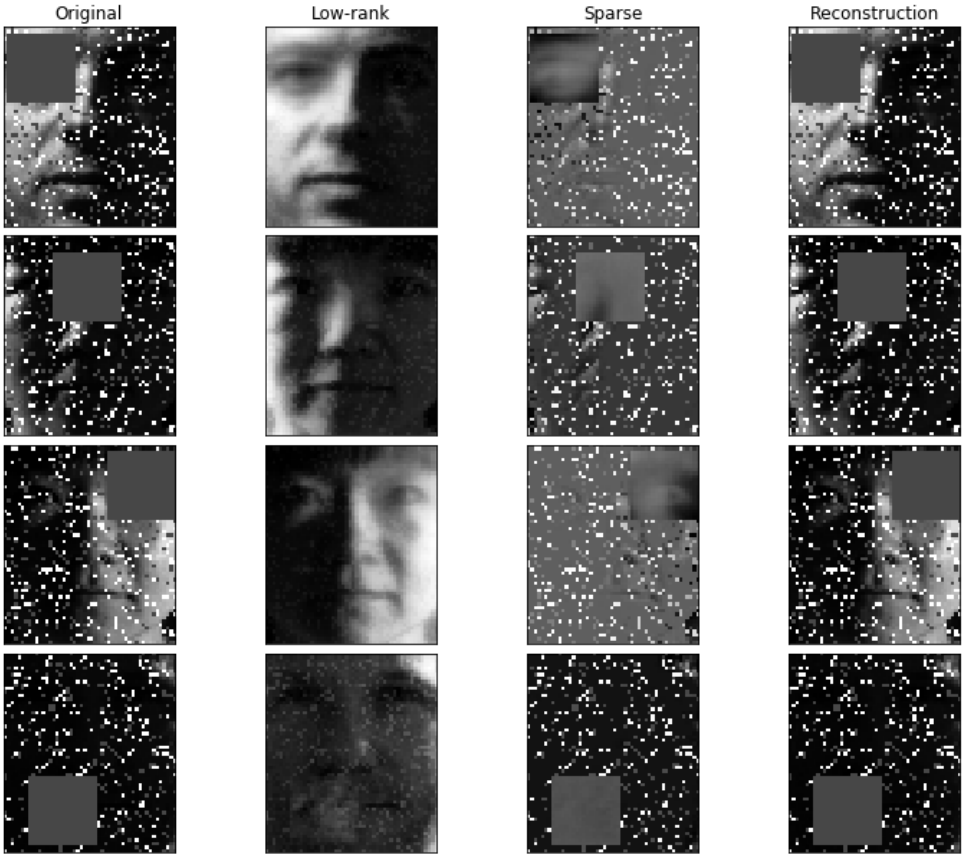
Traditional PCA can handle small noise, but is brittle with respect to grossly corrupted observations-- even one grossly corrupt observation can significantly mess up answer.

Robust PCA factors a matrix into the sum of two matrices, $M = L + S$, where M is the original matrix, L is low-rank , and S is sparse . This is what we'll be using for the background removal problem! Low-rank means that the matrix has a lot of redundant information-- in this case, it's the background, which is the same in every scene (talk about redundant info!). Sparse means that the matrix has mostly zero entries-- in this case, see how the picture of the foreground (the people) is mostly empty. (In the case of corrupted data, S is capturing the corrupted entries).

Applications of Robust PCA ¶

- Video Surveillance
- Face Recognition photos from this excellent tutorial. The dataset here consists of images of the faces of several people taken from the same angle but with different illumniations.

(Source: Jean Kossaifi)

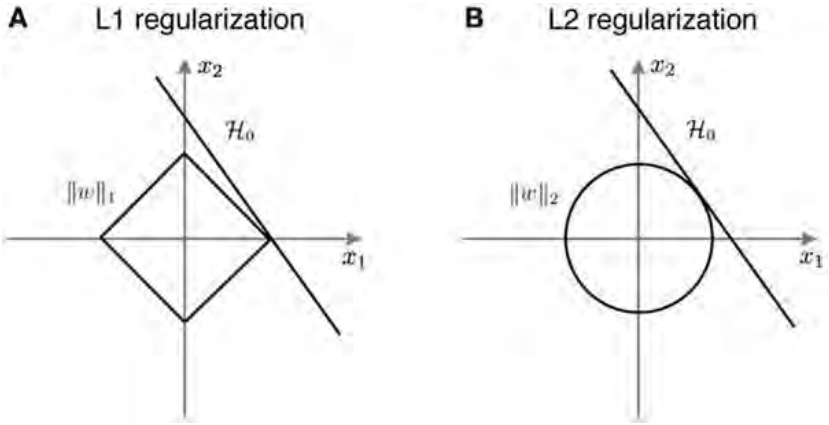


(Source: Jean Kossaifi)

- Latent Semantic Indexing: L captures common words used in all documents while S captures the few key words that best distinguish each document from others
- Ranking and Collaborative Filtering: a small portion of the available rankings could be noisy and even tampered with (see Netflix RAD - Outlier Detection on Big Data on the official netflix blog)

L1 norm induces sparsity ¶

The unit ball Undefined control sequence \lVert is a diamond in the L1 norm. It's extrema are the corners:



(Source)

A similar perspective is to look at the contours of the loss function:

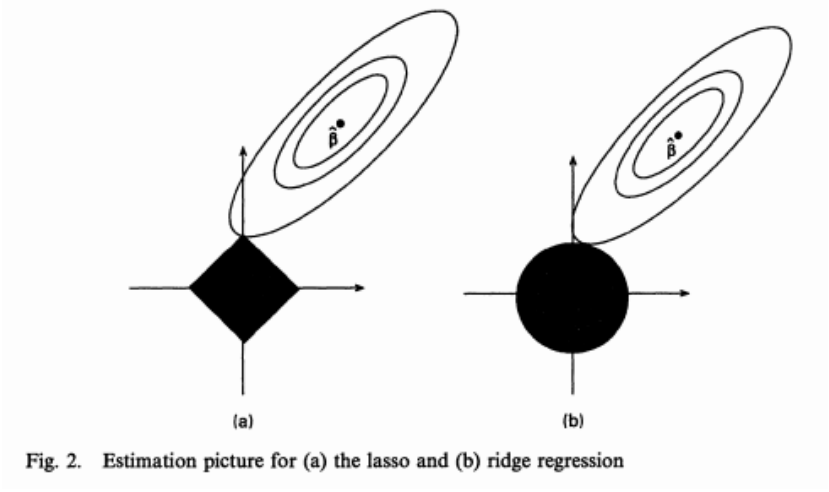


Fig. 2. Estimation picture for (a) the lasso and (b) ridge regression

(Source)

Optimization Problem ¶

Robust PCA can be written:

Undefined control sequence \lVert

where:

- Undefined control sequence \lVert is the L1 norm . Minimizing the L1 norm results in sparse values. For a matrix, the L1 norm is equal to the maximum absolute column norm.
- Undefined control sequence \lVert is the nuclear norm , which is the L1 norm of the singular values. Trying to minimize this results in sparse singular values --> low rank.

Implementing an algorithm from a paper ¶

Source ¶

We will use the general primary component pursuit algorithm from this Robust PCA paper (Candes, Li, Ma, Wright), in the specific form of Matrix Completion via the Inexact ALM Method found in section 3.1 of this paper (Lin, Chen, Ma). I also referenced the implemenations found here and here.

The Good Parts ¶

Section 1 of Candes, Li, Ma, Wright is nicely written, and section 5 Algorithms is our key interest. You don't need to know the math or understand the proofs to implement an algorithm from a paper . You will need to try different things and comb through resources for useful tidbits. This field has more theoretical researchers and less pragmatic advice. It took months to find what I needed and get this working.

The algorithm shows up on page 29:

Algorithm 1 (Principal Component Pursuit by Alternating Directions [32, 51])

initialize: $S_0 = Y_0 = 0, \mu > 0.$
while not converged **do**
  compute $L_{k+1} = \mathcal{D}_\mu(M - S_k - \mu^{-1}Y_k)$;
  compute $S_{k+1} = \mathcal{S}_{\lambda\mu}(M - L_{k+1} + \mu^{-1}Y_k)$;
  compute $Y_{k+1} = Y_k + \mu(M - L_{k+1} - S_{k+1})$;
end while
output: $L, S.$

needed definitions of $\mathcal{S}$, the Shrinkage operator, and $\mathcal{D}$, the singular value thresholding operator:

The ALM method operates on the *augmented Lagrangian*

$$l(L, S, Y) = \|L\|_* + \lambda\|S\|_1 + \langle Y, M - L - S\rangle + \frac{\mu}{2}\|M - L - S\|_F^2. \tag{5.1}$$

A generic Lagrange multiplier algorithm [5] would solve PCP by repeatedly setting $(L_k, S_k) = \arg\min_{L,S} l(L, S, Y_k)$, and then updating the Lagrange multiplier matrix via $Y_{k+1} = Y_k + \mu(M - L_k - S_k)$.

For our low-rank and sparse decomposition problem, we can avoid having to solve a sequence of convex programs by recognizing that $\min_L l(L, S, Y)$ and $\min_S l(L, S, Y)$ both have very simple and efficient solutions. Let $\mathcal{S}_\tau : \mathbb{R} \to \mathbb{R}$ denote the shrinkage operator $\mathcal{S}_\tau[x] = \text{sgn}(x)\max(|x| - \tau, 0)$, and extend it to matrices by applying it to each element. It is easy to show that

$$\arg\min_S l(L, S, Y) = \mathcal{S}_{\lambda\mu}(M - L + \mu^{-1}Y). \tag{5.2}$$

Similarly, for matrices X, let $\mathcal{D}_\tau(X)$ denote the singular value thresholding operator given by $\mathcal{D}_\tau(X) = U\mathcal{S}_\tau(\Sigma)V^*$, where $X = U\Sigma V^*$ is any singular value decomposition. It is not difficult to show that

$$\arg\min_L l(L, S, Y) = \mathcal{D}_\mu(M - S - \mu^{-1}Y). \tag{5.3}$$

Thus, a more practical strategy is to first minimize l with respect to L (fixing S), then minimize l with respect to S (fixing L), and then finally update the Lagrange multiplier matrix Y based on the residual $M - L - S$, a strategy that is summarized as Algorithm 1 below.

Section 3.1 of Chen, Lin, Ma contains a faster vartiation of this:

Algorithm 5 (RPCA via the Inexact ALM Method)

Input: Observation matrix $D \in \mathbb{R}^{m\times n}$, λ.
1: $Y_0 = D/J(D)$; $E_0 = 0$; $\mu_0 > 0$; $\rho > 1$; $k = 0$.
2: **while** not converged **do**
3: // Lines 4-5 solve $A_{k+1} = \arg\min_A L(A, E_k, Y_k, \mu_k)$.
4: $(U, S, V) = \text{svd}(D - E_k + \mu_k^{-1}Y_k)$;
5: $A_{k+1} = U\mathcal{S}_{\mu_k^{-1}}[S]V^T$.
6: // Line 7 solves $E_{k+1} = \arg\min_E L(A_{k+1}, E, Y_k, \mu_k)$.
7: $E_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}[D - A_{k+1} + \mu_k^{-1}Y_k]$.
8: $Y_{k+1} = Y_k + \mu_k(D - A_{k+1} - E_{k+1})$.
9: Update μ_k to μ_{k+1}.
10: $k \leftarrow k + 1$.
11: **end while**
Output: (A_k, E_k).

And Section 4 has some very helpful implementation details on how many singular values to calculate (as well as how to choose the parameter values):

$$\text{sv}_{k+1} = \begin{cases} \text{svp}_k + 1, & \text{if } \text{svp}_k < \text{sv}_k, \\ \min(\text{svp}_k + \text{round}(0.05d), d), & \text{if } \text{svp}_k = \text{sv}_k, \end{cases} \tag{17}$$

If you want to learn more of the theory: ¶

- Convex Optimization by Stephen Boyd (Stanford Prof):
  - OpenEdX Videos
  - Jupyter Notebooks
- Alternating Direction Method of Multipliers (more Stephen Boyd)

Robust PCA (via Primary Component Pursuit) ¶

Methods ¶

We will use Facebook's Fast Randomized PCA library.

In [48]:

```
from scipy import sparse
from sklearn.utils.extmath import randomized_svd
import fbpca
```

In [49]:

```
TOL=1e-9
MAX_ITERS=3
```

In [50]:

```
def converged(Z, d_norm):
    err = np.linalg.norm(Z, 'fro') / d_norm
    print('error: ', err)
    return err < TOL
```

In [51]:

```
def shrink(M, tau):
    S = np.abs(M) - tau
    return np.sign(M) * np.where(S>0, S, 0)
```

In [52]:

```
def _svd(M, rank): return fbpca.pca(M, k=min(rank, np.min(M.shape)), raw=True)
```

In [53]:

```
def norm_op(M): return _svd(M, 1)[1][0]
```

In [54]:

```
def svd_reconstruct(M, rank, min_sv):
    u, s, v = _svd(M, rank)
    s -= min_sv
    nnz = (s > 0).sum()
    return u[:,:nnz] @ np.diag(s[:nnz]) @ v[:nnz], nnz
```

In [97]:

```
def pcp(X, maxiter=10, k=10): # refactored
    m, n = X.shape
    trans = m<n
    if trans: X = X.T; m, n = X.shape

    lamda = 1/np.sqrt(m)
    op_norm = norm_op(X)
    Y = np.copy(X) / max(op_norm, np.linalg.norm( X, np.inf) / lamda)
    mu = k*1.25/op_norm; mu_bar = mu * 1e7; rho = k * 1.5

    d_norm = np.linalg.norm(X, 'fro')
    L = np.zeros_like(X); sv = 1

    examples = []

    for i in range(maxiter):
        print("rank sv:", sv)
        X2 = X + Y/mu

        # update estimate of Sparse Matrix by "shrinking/truncating": original - low-rank
        S = shrink(X2 - L, lamda/mu)

        # update estimate of Low-rank Matrix by doing truncated SVD of rank sv & reconstructing.
        # count of singular values > 1/mu is returned as svp
        L, svp = svd_reconstruct(X2 - S, sv, 1/mu)

        # If svp < sv, you are already calculating enough singular values.
        # If not, add 20% (in this case 240) to sv
        sv = svp + (1 if svp < sv else round(0.05*n))
```

```
        # residual
        Z = X - L - S
        Y += mu*Z; mu *= rho

        examples.extend([S[140,:], L[140,:]])

        if m > mu_bar: m = mu_bar
        if converged(Z, d_norm): break

    if trans: L=L.T; S=S.T
    return L, S, examples
```

the algorithm again (page 29 of Candes, Li, Ma, and Wright)

Algorithm 1 (Principal Component Pursuit by Alternating Directions [32, 51])

initialize: $S_0 = Y_0 = 0, \mu > 0$.
while not converged **do**
  compute $L_{k+1} = \mathcal{D}_\mu(M - S_k - \mu^{-1}Y_k)$;
  compute $S_{k+1} = \mathcal{S}_{\lambda\mu}(M - L_{k+1} + \mu^{-1}Y_k)$;
  compute $Y_{k+1} = Y_k + \mu(M - L_{k+1} - S_{k+1})$;
end while
output: L, S.

Results ¶

In [71]:

```
m, n = M.shape
round(m * .05)
```

Out[71]:

```
240
```

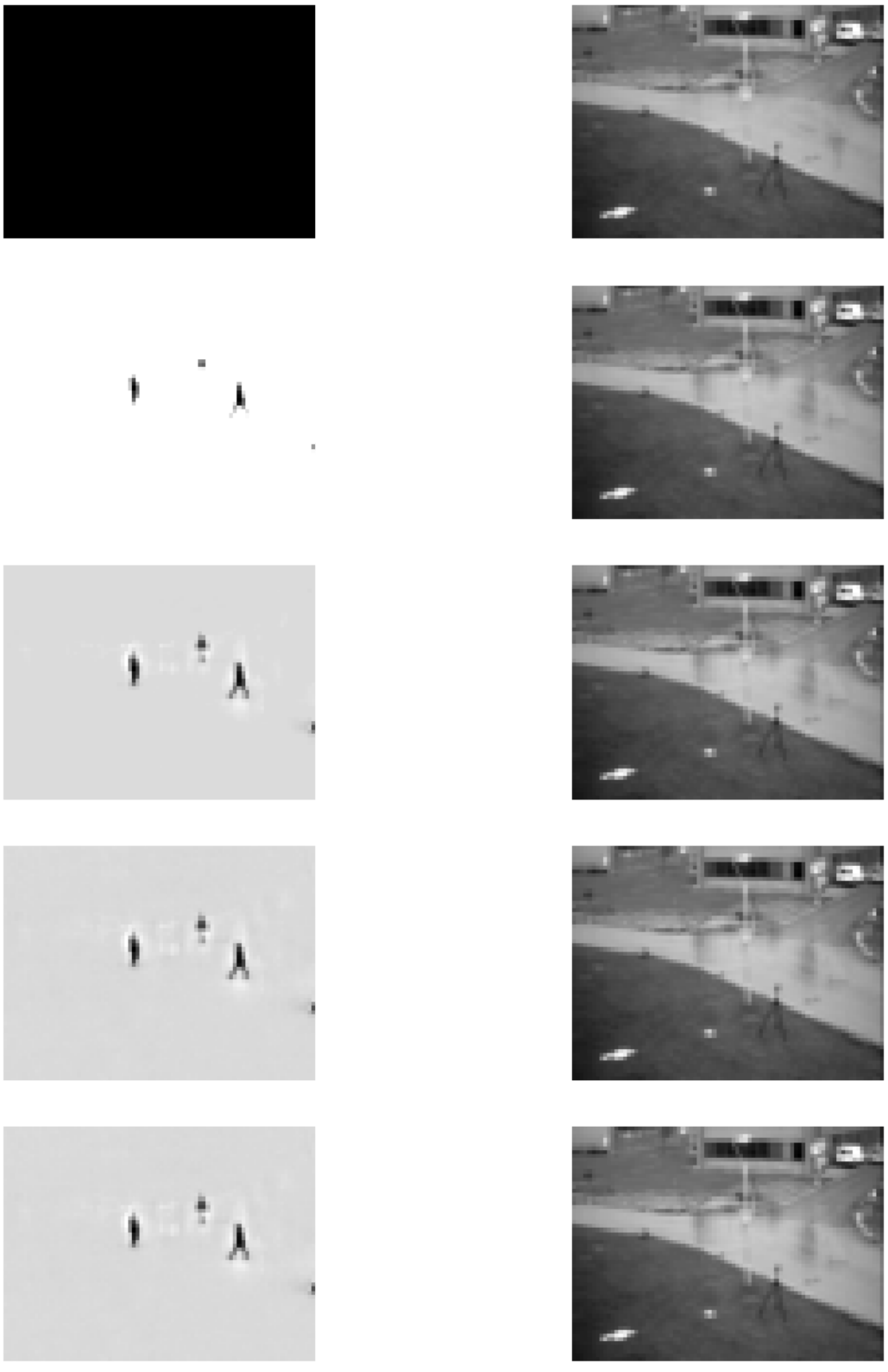
In [78]:

```
L, S, examples =  pcp(M, maxiter=5, k=10)
```

```
rank sv: 1
error:  0.131637937114
rank sv: 241
error:  0.0458515689278
rank sv: 49
error:  0.00591314217762
rank sv: 289
error:  0.000567221885441
rank sv: 529
error:  2.4633786172e-05
```

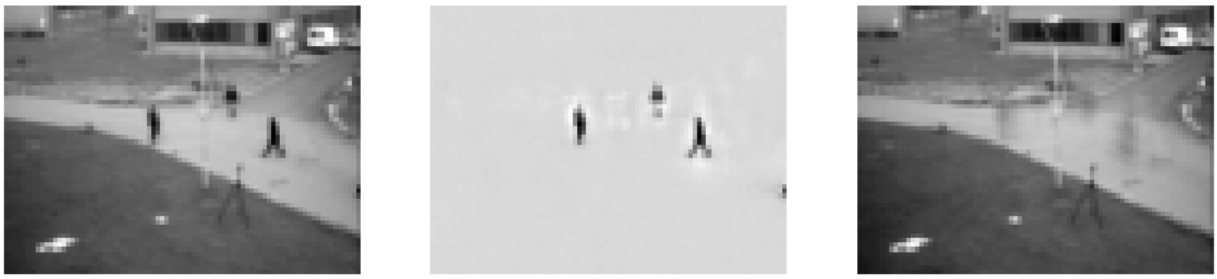
In [96]:

```
plots(examples, dims, rows=5)
```



In [92]:

```
f = plt_images(M, S, L, [140], dims)
```



In [37]:

```
np.save("high_res_L.npy", L)
np.save("high_res_S.npy", S)
```

In [38]:

```
f = plt_images(M, S, L, [0, 100, 1000], dims)
```

Extracting a bit of the foreground is easier than identifying the background. To accurately get the background, you need to remove all the foreground, not just parts of it

LU Factorization ¶

Both `fbpca` and our own `randomized_range_finder` methods used LU factorization, which factors a matrix into the product of a lower triangular matrix and an upper triangular matrix.

Gaussian Elimination ¶

This section is based on lectures 20-22 in Trefethen.

If you are unfamiliar with Gaussian elimination or need a refresher, watch this Khan Academy video.

Let's use Gaussian Elimination by hand to review:

A =

$$\begin{pmatrix} 1 & -2 & -2 & -3 \\ 3 & -9 & 0 & -9 \\ -1 & 2 & 4 & 7 \\ -3 & -6 & 26 & 2 \end{pmatrix}$$

Answer: ¶

$$LU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -3 & 4 & -2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -2 & -2 & -3 \\ 0 & -3 & 6 & 0 \\ 0 & 0 & 2 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Above example is from Lectures 20, 21 of Trefethen.

Gaussian Elimination transform a linear system into an upper triangular one by applying linear transformations on the left. It is triangular triangularization.

$L_{m-1} \dots L_2 L_1 A = U$

L is unit lower-triangular: all diagonal entries are 1

In [206]:

```
def LU(A):
    U = np.copy(A)
    m, n = A.shape
    L = np.eye(n)
    for k in range(n-1):
        for j in range(k+1,n):
            L[j,k] = U[j,k]/U[k,k]
            U[j,k:n] -= L[j,k] * U[k,k:n]
    return L, U
```

In [207]:

```
A = np.array([[2,1,1,0],[4,3,3,1],[8,7,9,5],[6,7,9,8]]).astype(np.float)
```

In [208]:

```
L, U = LU(A)
```

In [44]:

```
np.allclose(A, L @ U)
```

Out[44]:

```
True
```

The LU factorization is useful!

Solving Ax = b becomes LUx = b:

1. find A = LU
2. solve Ly = b
3. solve Ux = y

Work

Work for Gaussian Elimination: $2 \cdot \frac{1}{3} n^3$

Memory

Above, we created two new matrices, L and U. However, we can store the values of L and U in our matrix A (overwriting the original matrix). Since the diagonal of L is all 1s, it doesn't need to be stored. Doing factorizations or computations in-place is a common technique in numerical linear algebra to save memory. Note: you wouldn't want to do this if you needed to use your original matrix A again in the future. One of the homework questions is to rewrite the LU method to operate in place.

Consider the matrix

$$A = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix}$$

In [128]:

```
A = np.array([[1e-20, 1], [1,1]])
```

By hand, use Gaussian Elimination to calculate what L and U are:

Answer ¶

In [ ]:

```
#Exercise:
```

In [123]:

```
np.set_printoptions(suppress=True)
```

In [127]:

```
#Exercise:
```

In [129]:

```
L2, U2 = LU(A)
```

```
[[  1.00000000e-20   1.00000000e+00]
 [  0.00000000e+00  -1.00000000e+20]]
```

In [130]:

```
L2, U2
```

Out[130]:

```
(array([[  1.00000000e+00,   0.00000000e+00],
        [  1.00000000e+20,   1.00000000e+00]]),
 array([[  1.00000000e-20,   1.00000000e+00],
        [  0.00000000e+00,  -1.00000000e+20]]))
```

In [84]:

```
np.allclose(L1, L2)
```

Out[84]:

```
True
```

In [85]:

```
np.allclose(U1, U2)
```

Out[85]:

```
True
```

In [86]:

```
np.allclose(A, L2 @ U2)
```

Out[86]:

```
False
```

This is the motivation for LU factorization with pivoting .

This also illustrates that LU factorization is stable, but not backward stable. (spoiler alert: even with partial pivoting, LU is "explosively unstable" for certain matrices, yet stable in practice)

Stability ¶

An algorithm $\hat{f}$ for a problem f is stable if for each x,

Undefined control sequence \lVert

for some y with

Undefined control sequence \lVert

A stable algorithm gives nearly the right answer to nearly the right question (Trefethen, pg 104)

To translate that:

- right question: x
- nearly the right question: y
- right answer: f
- right answer to nearly the right question: $f(y)$

Backwards Stability ¶

Backwards stability is both stronger and simpler than stability.

An algorithm $\hat{f}$ for a problem f is backwards stable if for each x,

$$\hat{f}(x) = f(y)$$

for some y with

Undefined control sequence \lVert

A backwards stable algorithm gives exactly the right answer to nearly the right question (Trefethen, pg 104)

Translation:

- right question: x
- nearly the right question: y
- right answer: f
- right answer to nearly the right question: $f(y)$

LU factorization with Partial Pivoting ¶

Let's now look at the matrix

$$\hat{A} = \begin{bmatrix} 1 & 1 \\ 10^{-20} & 1 \end{bmatrix}$$

In [89]:

```
A = np.array([[1,1], [1e-20, 1]])
```

By hand, use Gaussian Elimination to calculate what L and U are:

Answer ¶

In [ ]:

```
#Exercise:
```

In [90]:

```
L, U = LU(A)
```

In [93]:

```
np.allclose(A, L @ U)
```

Out[93]:

```
True
```

Idea: We can switch the order of the rows around to get more stable answers! This is equivalent to multiplying by a permutation matrix P. For instance,

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 10^{-20} & 1 \end{bmatrix}$$

$$PA = \hat{A}$$

Apply Gaussian elimination to PA.

At each step, choose the largest value in column k, and move that row to be row k.

Homework ¶

In [100]:

```
def swap(a,b):
    temp = np.copy(a)
    a[:] = b
    b[:] = temp

a=np.array([1,2,3])
```

```
b=np.array([3,2,1])
swap(a,b)
a,b
```

In [102]:

```
#Exercise: re-write the LU factorization above to use pivoting
```

Example ¶

In [104]:

```
A = np.array([[2,1,1,0],[4,3,3,1],[8,7,9,5],[6,7,9,8]]).astype(np.float)
```

In [105]:

```
L, U, P = LU_pivot(A)
```

Can compare below to answers in Trefethen, page 159:

In [106]:

```
A
```

Out[106]:

```
array([[ 2.,  1.,  1.,  0.],
       [ 4.,  3.,  3.,  1.],
       [ 8.,  7.,  9.,  5.],
       [ 6.,  7.,  9.,  8.]])
```

In [107]:

```
U
```

Out[107]:

```
array([[ 8.        ,  7.        ,  9.        ,  5.        ],
       [ 0.        ,  1.75      ,  2.25      ,  4.25      ],
       [ 0.        ,  0.        , -0.28571429,  0.57142857],
       [ 0.        ,  0.        ,  0.        , -2.        ]])
```

In [114]:

```
P
```

Out[114]:

```
array([[ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.],
       [ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.]])
```

Partial pivoting permutes the rows. It is such a universal practice, that this is usually what is meant by LU factorization.

Complete pivoting permutes the rows and columns. Complete pivoting is significantly time-consuming and rarely used in practice.

Example ¶

Consider the system of equations:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 2 \\ 1 \end{bmatrix}$$

In [99]:

```
def make_matrix(n):
    A = np.eye(n)
    for i in range(n):
        A[i,-1] = 1
        for j in range(i):
            A[i,j] = -1
    return A
```

In [117]:

```
def make_vector(n):
    b = np.ones(n)
    b[-2] = 2
    return b
```

In [101]:

```
make_vector(7)
```

Out[101]:

```
array([ 1.,  1.,  1.,  1.,  1.,  2.,  1.])
```

Exercise ¶

Exercise: Let's use Gaussian Elimination on the 5×5 system.

Scipy has this funtionality as well. Let's look at the solution for the last 5 equations with $n = 10,\ 20,\ 30,\ 40,\ 50,\ 60$

In [131]:

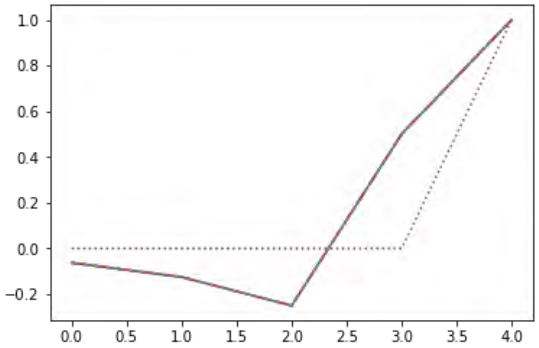
```
?scipy.linalg.solve
```

In [112]:

```
for n, ls in zip(range(10, 70, 10), ['--', ':', '-', '-.', '--', ':']):
    soln = scipy.linalg.lu_solve(scipy.linalg.lu_factor(make_matrix(n)), make_vector(n))
    plt.plot(soln[-5:], ls)
    print(soln)
```

```
[-0.00195312 -0.00390625 -0.0078125  -0.015625   -0.03125    -0.0625     -0.125
 -0.25        0.5         1.00195312]
[ -1.90734863e-06  -3.81469727e-06  -7.62939453e-06  -1.52587891e-05
  -3.05175781e-05  -6.10351562e-05  -1.22070312e-04  -2.44140625e-04
  -4.88281250e-04  -9.76562500e-04  -1.95312500e-03  -3.90625000e-03
  -7.81250000e-03  -1.56250000e-02  -3.12500000e-02  -6.25000000e-02
  -1.25000000e-01  -2.50000000e-01   5.00000000e-01   1.00000191e+00]
[ -1.86264515e-09  -3.72529030e-09  -7.45058060e-09  -1.49011612e-08
  -2.98023224e-08  -5.96046448e-08  -1.19209290e-07  -2.38418579e-07
  -4.76837158e-07  -9.53674316e-07  -1.90734863e-06  -3.81469727e-06
  -7.62939453e-06  -1.52587891e-05  -3.05175781e-05  -6.10351562e-05
  -1.22070312e-04  -2.44140625e-04  -4.88281250e-04  -9.76562500e-04
  -1.95312500e-03  -3.90625000e-03  -7.81250000e-03  -1.56250000e-02
  -3.12500000e-02  -6.25000000e-02  -1.25000000e-01  -2.50000000e-01
   5.00000000e-01   1.00000000e+00]
[ -1.81898940e-12  -3.63797881e-12  -7.27595761e-12  -1.45519152e-11
  -2.91038305e-11  -5.82076609e-11  -1.16415322e-10  -2.32830644e-10
  -4.65661287e-10  -9.31322575e-10  -1.86264515e-09  -3.72529030e-09
  -7.45058060e-09  -1.49011612e-08  -2.98023224e-08  -5.96046448e-08
  -1.19209290e-07  -2.38418579e-07  -4.76837158e-07  -9.53674316e-07
  -1.90734863e-06  -3.81469727e-06  -7.62939453e-06  -1.52587891e-05
  -3.05175781e-05  -6.10351562e-05  -1.22070312e-04  -2.44140625e-04
  -4.88281250e-04  -9.76562500e-04  -1.95312500e-03  -3.90625000e-03
  -7.81250000e-03  -1.56250000e-02  -3.12500000e-02  -6.25000000e-02
  -1.25000000e-01  -2.50000000e-01   5.00000000e-01   1.00000000e+00]
[ -1.77635684e-15  -3.55271368e-15  -7.10542736e-15  -1.42108547e-14
  -2.84217094e-14  -5.68434189e-14  -1.13686838e-13  -2.27373675e-13
  -4.54747351e-13  -9.09494702e-13  -1.81898940e-12  -3.63797881e-12
  -7.27595761e-12  -1.45519152e-11  -2.91038305e-11  -5.82076609e-11
  -1.16415322e-10  -2.32830644e-10  -4.65661287e-10  -9.31322575e-10
  -1.86264515e-09  -3.72529030e-09  -7.45058060e-09  -1.49011612e-08
  -2.98023224e-08  -5.96046448e-08  -1.19209290e-07  -2.38418579e-07
```

```
  -4.76837158e-07  -9.53674316e-07  -1.90734863e-06  -3.81469727e-06
  -7.62939453e-06  -1.52587891e-05  -3.05175781e-05  -6.10351562e-05
  -1.22070312e-04  -2.44140625e-04  -4.88281250e-04  -9.76562500e-04
  -1.95312500e-03  -3.90625000e-03  -7.81250000e-03  -1.56250000e-02
  -3.12500000e-02  -6.25000000e-02  -1.25000000e-01  -2.50000000e-01
   5.00000000e-01   1.00000000e+00]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  1.]
```



What is going on when $n = 60$?

Theorem : Let the factorization $PA = LU$ of a matrix A be computed by Gaussian Elimination with partial pivoting. The computed (by a computer with Floating Point Arithmetic) matrices $\hat{P}$, $\hat{L}$, and $\hat{U}$ satisfy

$$\hat{L}\hat{U} = \hat{P}A + \delta A, \quad \frac{\delta A}{A} = \mathcal{O}(\rho\varepsilon_{machine})$$

where ρ is the growth factor,

Undefined control sequence \lvert

For our matrix above, $\rho = 2^{m-1}$

Unstable in theory , stable in practice ¶

Stability of most algorithms (such as QR) is straightforward. Not the case for Gaussian Elimination with partial pivoting. Instability in Gaussian elimination (with or without pivoting) arises only if L and/or U is large relative to the size of A.

Trefethen: "Despite examples like (22.4), Gaussian elimination with partial pivoting is utterly stable in practice... In fifty years of computing, no matrix problems that excite an explosive instability are known to have arisen under natural circumstances." [although can easily be constructed as contrived examples]

Although some matrices cause instability, but extraordinarily small proportion of all matrices so "never" arise in practice for statistical reasons. "If you pick a billion matrices at random, you will almost certainly not find one for which Gaussian elimination is unstable."

Further Reading ¶

- Gaussian Elimination/LU factorization-- Trefethn Lecture 20
- Pivoting -- Trefethn Lecture 21
- Stability of Gaussian Elimination -- Trefethn Lecture 22

Follow up from last class ¶

What is going on with Randomized Projections? ¶

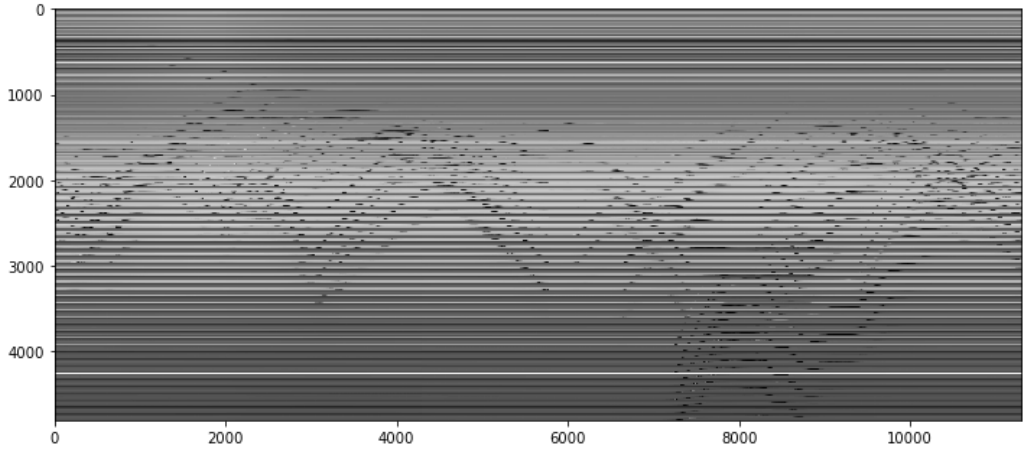
We are taking a linear combination (with random weights) of the columns in the matrix below:

In [33]:

```
plt.figure(figsize=(12, 12))
plt.imshow(M, cmap='gray')
```

Out[33]:

```
<matplotlib.image.AxesImage at 0x7f601f315fd0>
```



It's like a random weighted average. If you take several of these, you end up with columns that are orthonormal to each other.

Johnson-Lindenstrauss Lemma : (from wikipedia) a small set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved.

It is desirable to be able to reduce dimensionality of data in a way that preserves relevant structure. The Johnson–Lindenstrauss lemma is a classic result of this type.

History of Gaussian Elimination ¶

Fascinating history of Gaussian Elimination. Some highlights:

- First written record of Gaussian elimination from ~200 BC in the Chinese book Nine Chapters on Arithmetic
- The ancient Chinese used colored bamboo rods placed in columns on a "counting board"
- Japanese mathematician Seki Kowa (16432-1708) carried forward the Chinese elimintion methods and invented the determinant before 1683. Around the same time, Leibniz made similar discoveries independently, but neither Kowa nor Leibniz go credit for their discoveries.
- Gauss referred to the elimination method as being "commonly known" and never claimed to have invented it, although he may have invented the Cholesky decomposition

More history here

Speeding Up Gaussian Elimination ¶

Parallelized LU Decomposition LU decomposition can be fully parallelized

Randomized LU Decomposition (2016 article): The randomized LU is fully implemented to run on a standard GPU card without any GPU–CPU data transfer.

Scipy .linalg solve vs lu_solve ¶

In [142]:

```
n = 100
A = make_matrix(n)
b = make_vector(n)
```

This problem has a large growth factor $= 2^{59}$. We get the wrong answer using scipy.linalg.lu_solve, but the right answer with scipy.linalg.solve. What is scipy.linalg.solve doing?

In [143]:

```
print(scipy.linalg.lu_solve(scipy.linalg.lu_factor(A), b)[-5:])
print(scipy.linalg.solve(A, b)[-5:])
```

```
[ 0.  0.  0.  0.  1.]
[-0.0625 -0.125  -0.25    0.5     1.    ]
```

In [136]:

```
%%timeit
soln = scipy.linalg.lu_solve(scipy.linalg.lu_factor(A), b)
soln[-5:]
```

```
91.2 µs ± 192 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [137]:

```
%%timeit
soln = scipy.linalg.solve(A, b)
soln[-5:]
```

```
153 µs ± 5 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Looking at the source code for scipy, we see that it is calling the LAPACK routine `gesvx`. Here is the Fortran source code for sgesvx (`s` refers to single, there is also `dgesvx` for doubles and `cgesvx` for complex numbers). In the comments, we see that it is computing a reciprocal pivot growth factor, so it is taking into account this growth factor and doing something more complex than plain partial pivot LU factorization.

Block Matrices ¶

This is a follow-up to a question about block matrices asked in a previous class. But first,

Ordinary Matrix Multiplication ¶

Question : What is the computational complexity (big $\mathcal{O}$) of matrix multiplication for multiplying two $n \times n$ matrices $A \times B = C$?

You can learn (or refresh) about big $\mathcal{O}$ on Codecademy

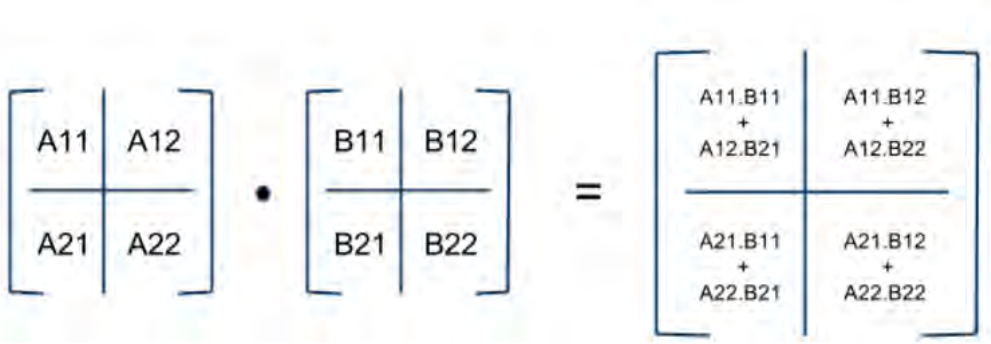
What this looks like:

```
for i=1 to n
    {read row i of A into fast memory}
    for j=1 to n
        {read col j of B into fast memory}
        for k=1 to n
            C[i,j] = C[i,j] + A[i,k] x B[k,j]
        {write C[i,j] back to slow memory}
```

Question : How many reads and writes are made?

Block Matrix Multiplication ¶

Divide A, B, C into $N \times N$ blocks of size $\frac{n}{N} \times \frac{n}{N}$



(Source)

What this looks like:

```
for i=1 to N
    for j=1 to N
        for k=1 to N
            {read block (i,k) of A}
            {read block (k,j) of B}
            block (i,j) of C += block of A times block of B
        {write block (i,j) of C back to slow memory}
```

Question 1 : What is the big-$\mathcal{O}$ of this?

Question 2 : How many reads and writes are made?

End ¶

You can read an overview of this Numerical Linear Algebra course in this blog post. The course was originally taught in the University of San Francisco MS in Analytics graduate program. Course lecture videos are available on YouTube (note that the notebook numbers and video numbers do not line up, since some notebooks took longer than 1 video to cover).

You can ask questions about the course on our fast.ai forums.

4. Compressed Sensing of CT Scans with Robust Regression ¶

Broadcasting ¶

The term broadcasting describes how arrays with different shapes are treated during arithmetic operations. The term broadcasting was first used by Numpy, although is now used in other libraries such as Tensorflow and Matlab; the rules can vary by library.

From the Numpy Documentation:

```
Broadcasting provides a means of vectorizing array operations so that looping
occurs in C instead of Python. It does this without making needless copies of data
and usually leads to efficient algorithm implementations.
```

The simplest example of broadcasting occurs when multiplying an array by a scalar.

In [718]:

```
a = np.array([1.0, 2.0, 3.0])
b = 2.0
a * b
```

Out[718]:

```
array([ 2.,  4.,  6.])
```

In [128]:

```
v=np.array([1,2,3])
print(v, v.shape)
```

```
[1 2 3] (3,)
```

In [129]:

```
m=np.array([v,v*2,v*3]); m, m.shape
```

Out[129]:

```
(array([[1, 2, 3],
        [2, 4, 6],
        [3, 6, 9]]), (3, 3))
```

In [133]:

```
n = np.array([m*1, m*5])
```

In [134]:

```
n
```

Out[134]:

```
array([[[ 1,  2,  3],
        [ 2,  4,  6],
        [ 3,  6,  9]],

       [[ 5, 10, 15],
        [10, 20, 30],
        [15, 30, 45]]])
```

In [136]:

```
n.shape, m.shape
```

Out[136]:

```
((2, 3, 3), (3, 3))
```

We can use broadcasting to add a matrix and an array:

In [48]:

```
m+v
```

Out[48]:

```
array([[ 2,  4,  6],
       [ 3,  6,  9],
       [ 4,  8, 12]])
```

Notice what happens if we transpose the array:

In [49]:

```
v1=np.expand_dims(v,-1); v1, v1.shape
```

Out[49]:

```
(array([[1],
        [2],
        [3]]), (3, 1))
```

In [50]:

```
m+v1
```

Out[50]:

```
array([[ 2,  3,  4],
       [ 4,  6,  8],
       [ 6,  9, 12]])
```

General Numpy Broadcasting Rules ¶

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions , and works its way forward. Two dimensions are compatible when

- they are equal, or
- one of them is 1

Arrays do not need to have the same number of dimensions. For example, if you have a $256 \times 256 \times 3$ array of RGB values, and you want to scale each color in the image by a different value, you can multiply the image by a one-dimensional array with 3 values. Lining up the sizes of the trailing axes of these arrays according to the broadcast rules, shows that they are compatible:

```
Image  (3d array): 256 x 256 x 3
Scale  (1d array):             3
Result (3d array): 256 x 256 x 3
```

Review ¶

In [165]:

```
v = np.array([1,2,3,4])
m = np.array([v,v*2,v*3])
A = np.array([5*m, -1*m])
```

In [166]:

```
v.shape, m.shape, A.shape
```

Out[166]:

```
((4,), (3, 4), (2, 3, 4))
```

Will the following operations work?

In [159]:

```
A
```

Out[159]:

```
array([[[ 5, 10, 15],
        [10, 20, 30],
        [15, 30, 45]],

       [[-1, -2, -3],
        [-2, -4, -6],
        [-3, -6, -9]]])
```

In [158]:

```
A + v
```

Out[158]:

```
array([[[ 6, 12, 18],
        [11, 22, 33],
        [16, 32, 48]],

       [[ 0,  0,  0],
        [-1, -2, -3],
        [-2, -4, -6]]])
```

In [167]:

```
A
```

Out[167]:

```
array([[[  5,  10,  15,  20],
        [ 10,  20,  30,  40],
        [ 15,  30,  45,  60]],

       [[ -1,  -2,  -3,  -4],
        [ -2,  -4,  -6,  -8],
        [ -3,  -6,  -9, -12]]])
```

In [168]:

```
A.T.shape
```

Out[168]:

```
(4, 3, 2)
```

In [169]:

```
A.T
```

Out[169]:

```
array([[[  5,  -1],
        [ 10,  -2],
        [ 15,  -3]],

       [[ 10,  -2],
        [ 20,  -4],
        [ 30,  -6]],

       [[ 15,  -3],
        [ 30,  -6],
        [ 45,  -9]],

       [[ 20,  -4],
```

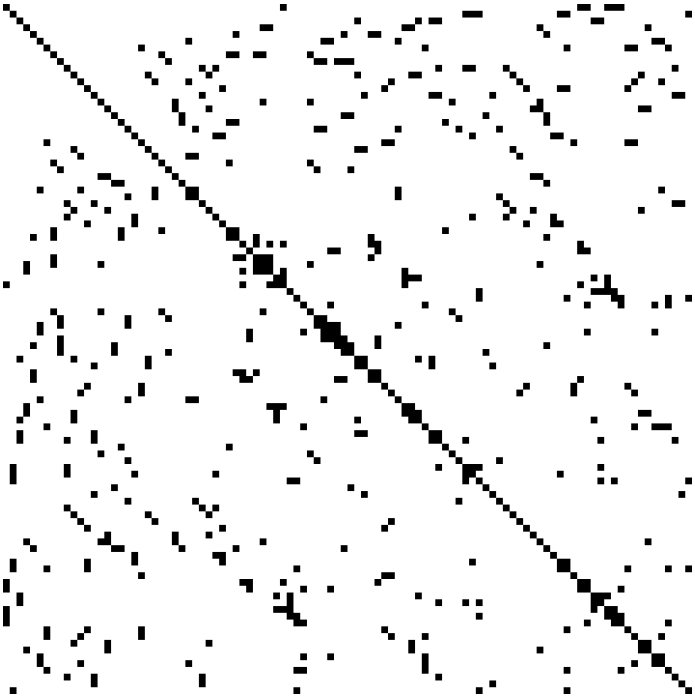
```
       [ 40,  -8],
       [ 60, -12]]])
```

Sparse Matrices (in Scipy) ¶

A matrix with lots of zeros is called sparse (the opposite of sparse is dense). For sparse matrices, you can save a lot of memory by only storing the non-zero values.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

Another example of a large, sparse matrix:



Source

There are the most common sparse storage formats:

- coordinate-wise (scipy calls COO)
- compressed sparse row (CSR)
- compressed sparse column (CSC)

Let's walk through these examples

There are actually many more formats as well.

A class of matrices (e.g, diagonal) is generally called sparse if the number of non-zero elements is proportional to the number of rows (or columns) instead of being proportional to the product rows x columns.

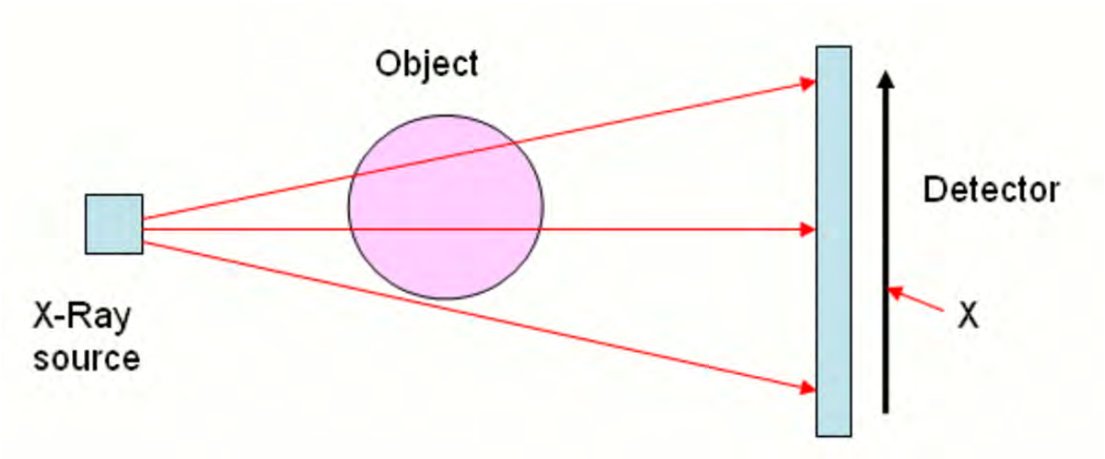
Scipy Implementation

From the Scipy Sparse Matrix Documentation

- To construct a matrix efficiently, use either dok_matrix or lil_matrix. The lil_matrix class supports basic slicing and fancy indexing with a similar syntax to NumPy arrays. As illustrated below, the COO format may also be used to efficiently construct matrices
- To perform manipulations such as multiplication or inversion, first convert the matrix to either CSC or CSR format.
- All conversions among the CSR, CSC, and COO formats are efficient, linear-time operations.

Today: CT scans ¶

"Can Maths really save your life? Of course it can!!" (lovely article)

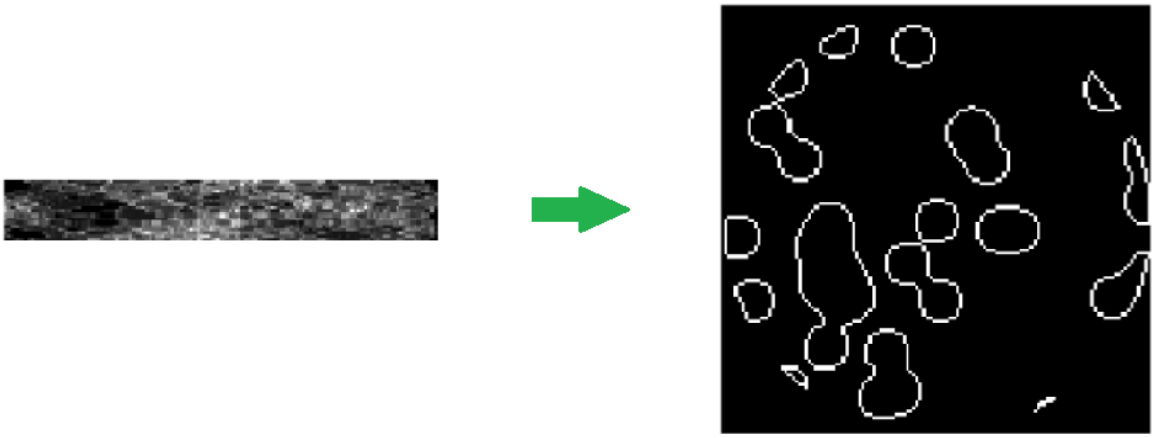


(CAT and CT scan refer to the same procedure. CT scan is the more modern term)

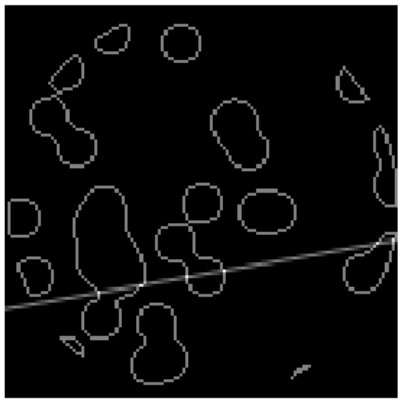
This lesson is based off the Scikit-Learn example Compressive sensing: tomography reconstruction with L1 prior (Lasso)

Our goal today ¶

Take the readings from a CT scan and construct what the original looks like.



For each x-ray (at a particular position and particular angle), we get a single measurement. We need to construct the original picture just from these measurements. Also, we don't want the patient to experience a ton of radiation, so we are gathering less data than the area of the picture.



Review ¶

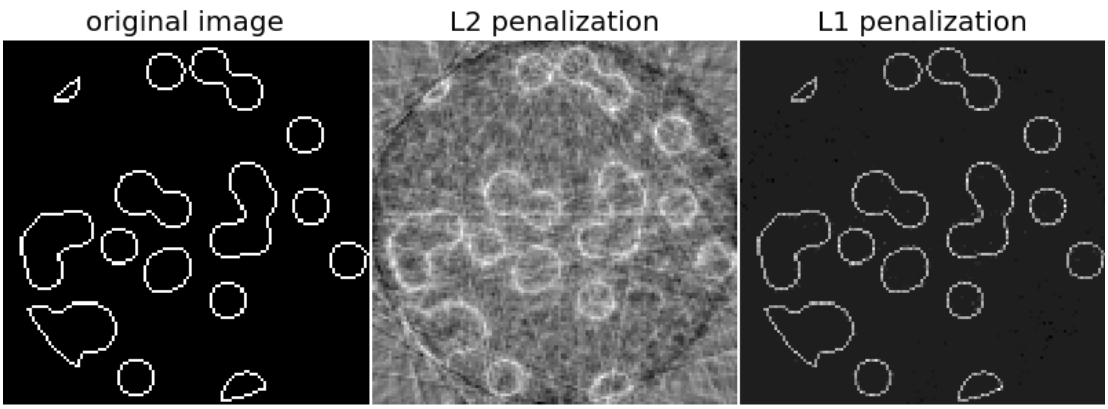
In the previous lesson, we used Robust PCA for background removal of a surveillance video. We saw that this could be written as the optimization problem:

Undefined control sequence \lVert

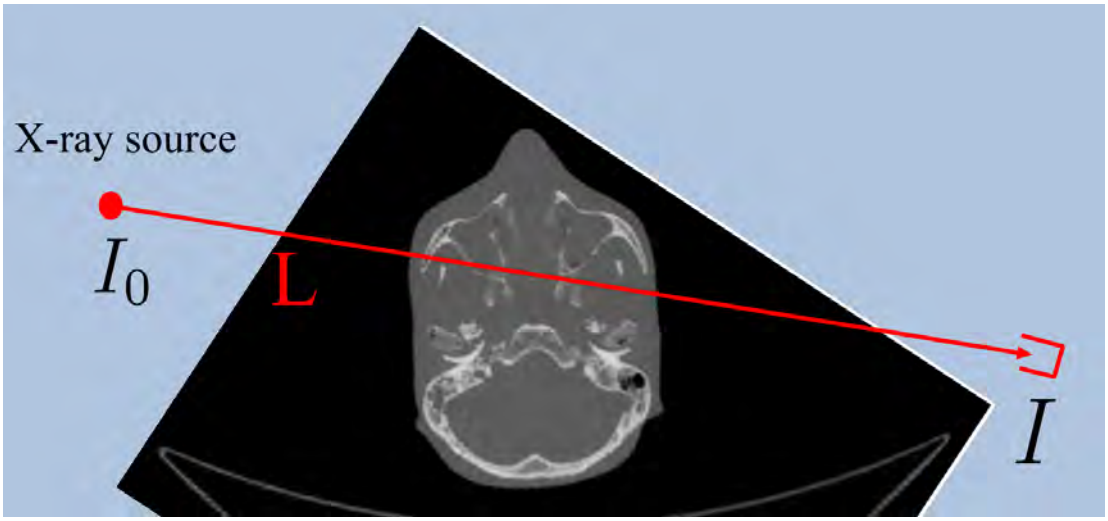
Question : Do you remember what is special about the L1 norm?

Today ¶

We will see that:



Resources: Compressed Sensing



Source

Imports ¶

In [1]:

```
%matplotlib inline
import numpy as np, matplotlib.pyplot as plt, math
from scipy import ndimage, sparse
```

In [2]:

```
np.set_printoptions(suppress=True)
```

Generate Data ¶

Intro ¶

We will use generated data today (not real CT scans). There is some interesting numpy and linear algebra involved in generating the data, and we will return to that later.

Code is from this Scikit-Learn example Compressive sensing: tomography reconstruction with L1 prior (Lasso)

Generate pictures ¶

In [3]:

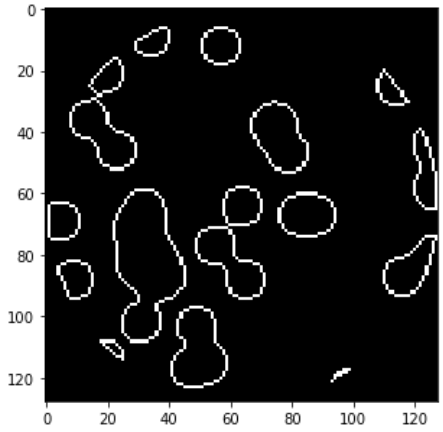
```
def generate_synthetic_data():
    rs = np.random.RandomState(0)
    n_pts = 36
    x, y = np.ogrid[0:l, 0:l]
    mask_outer = (x - l / 2) ** 2 + (y - l / 2) ** 2 < (l / 2) ** 2
    mx,my = rs.randint(0, l, (2,n_pts))
    mask = np.zeros((l, l))
    mask[mx,my] = 1
    mask = ndimage.gaussian_filter(mask, sigma=l / n_pts)
    res = (mask > mask.mean()) & mask_outer
    return res ^ ndimage.binary_erosion(res)
```

In [208]:

```
l = 128
data = generate_synthetic_data()
```

In [209]:

```
plt.figure(figsize=(5,5))
plt.imshow(data, cmap=plt.cm.gray);
```



What generate_synthetic_data() is doing ¶

In [155]:

```
l=8; n_pts=5
rs = np.random.RandomState(0)
```

In [156]:

```
x, y = np.ogrid[0:l, 0:l]; x,y
```

Out[156]:

```
(array([[0],
        [1],
        [2],
        [3],
        [4],
        [5],
        [6],
        [7]]), array([[0, 1, 2, 3, 4, 5, 6, 7]]))
```

In [170]:

```
x + y
```

Out[170]:

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 1,  2,  3,  4,  5,  6,  7,  8],
       [ 2,  3,  4,  5,  6,  7,  8,  9],
```

```
       [ 3,  4,  5,  6,  7,  8,  9, 10],
       [ 4,  5,  6,  7,  8,  9, 10, 11],
       [ 5,  6,  7,  8,  9, 10, 11, 12],
       [ 6,  7,  8,  9, 10, 11, 12, 13],
       [ 7,  8,  9, 10, 11, 12, 13, 14]])
```

In [157]:

```
(x - l/2) ** 2
```

Out[157]:

```
array([[ 16.],
       [  9.],
       [  4.],
       [  1.],
       [  0.],
       [  1.],
       [  4.],
       [  9.]])
```

In [59]:

```
(x - l/2) ** 2 + (y - l/2) ** 2
```

Out[59]:

```
array([[ 32.,  25.,  20.,  17.,  16.,  17.,  20.,  25.],
       [ 25.,  18.,  13.,  10.,   9.,  10.,  13.,  18.],
       [ 20.,  13.,   8.,   5.,   4.,   5.,   8.,  13.],
       [ 17.,  10.,   5.,   2.,   1.,   2.,   5.,  10.],
       [ 16.,   9.,   4.,   1.,   0.,   1.,   4.,   9.],
       [ 17.,  10.,   5.,   2.,   1.,   2.,   5.,  10.],
       [ 20.,  13.,   8.,   5.,   4.,   5.,   8.,  13.],
       [ 25.,  18.,  13.,  10.,   9.,  10.,  13.,  18.]])
```

In [60]:

```
mask_outer = (x - l/2) ** 2 + (y - l/2) ** 2 < (l/2) ** 2; mask_outer
```

Out[60]:

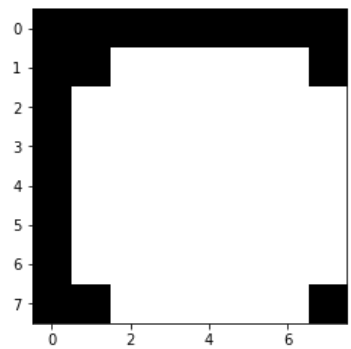
```
array([[False, False, False, False, False, False, False, False],
       [False, False,  True,  True,  True,  True,  True, False],
       [False,  True,  True,  True,  True,  True,  True,  True],
       [False,  True,  True,  True,  True,  True,  True,  True],
       [False,  True,  True,  True,  True,  True,  True,  True],
       [False,  True,  True,  True,  True,  True,  True,  True],
       [False,  True,  True,  True,  True,  True,  True,  True],
       [False, False,  True,  True,  True,  True,  True, False]], dtype=bool)
```

In [61]:

```
plt.imshow(mask_outer, cmap='gray')
```

Out[61]:

```
<matplotlib.image.AxesImage at 0x7efcd9303278>
```



In [62]:

```
mask = np.zeros((1, 1))
mx,my = rs.randint(0, 1, (2,n_pts))
mask[mx,my] = 1; mask
```

Out[62]:

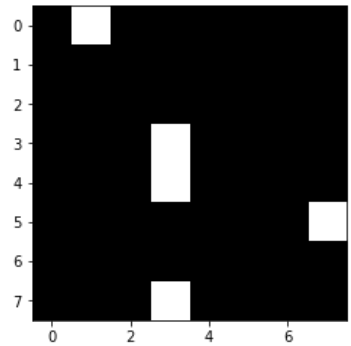
```
array([[ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.]])
```

In [63]:

```
plt.imshow(mask, cmap='gray')
```

Out[63]:

```
<matplotlib.image.AxesImage at 0x7efcd9293940>
```



In [64]:

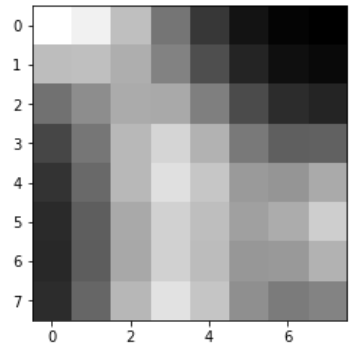
```
mask = ndimage.gaussian_filter(mask, sigma=1 / n_pts)
```

In [65]:

```
plt.imshow(mask, cmap='gray')
```

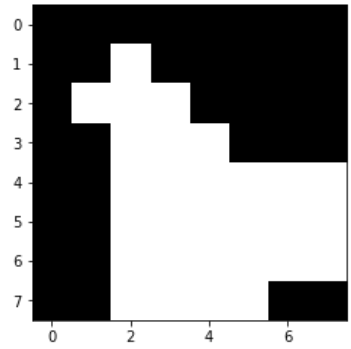
Out[65]:

```
<matplotlib.image.AxesImage at 0x7efcd922c0b8>
```



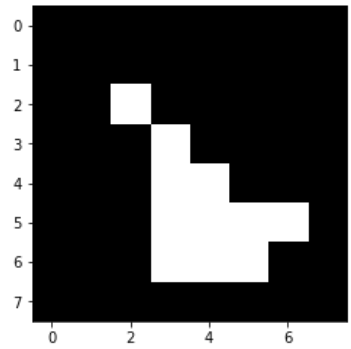
In [66]:

```
res = np.logical_and(mask > mask.mean(), mask_outer)
plt.imshow(res, cmap='gray');
```



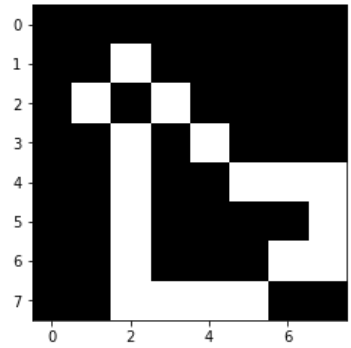
In [67]:

```
plt.imshow(ndimage.binary_erosion(res), cmap='gray');
```



In [68]:

```
plt.imshow(res ^ ndimage.binary_erosion(res), cmap='gray');
```



Generate Projections ¶

Code¶

In [72]:

```
def _weights(x, dx=1, orig=0):
    x = np.ravel(x)
    floor_x = np.floor((x - orig) / dx)
    alpha = (x - orig - floor_x * dx) / dx
    return np.hstack((floor_x, floor_x + 1)), np.hstack((1 - alpha, alpha))


def _generate_center_coordinates(l_x):
    X, Y = np.mgrid[:l_x, :l_x].astype(np.float64)
    center = l_x / 2.
    X += 0.5 - center
    Y += 0.5 - center
    return X, Y
```

In [73]:

```
def build_projection_operator(l_x, n_dir):
    X, Y = _generate_center_coordinates(l_x)
    angles = np.linspace(0, np.pi, n_dir, endpoint=False)
```

```
    data_inds, weights, camera_inds = [], [], []
    data_unravel_indices = np.arange(l_x ** 2)
    data_unravel_indices = np.hstack((data_unravel_indices,
                                      data_unravel_indices))
    for i, angle in enumerate(angles):
        Xrot = np.cos(angle) * X - np.sin(angle) * Y
        inds, w = _weights(Xrot, dx=1, orig=X.min())
        mask = (inds >= 0) & (inds < l_x)
        weights += list(w[mask])
        camera_inds += list(inds[mask] + i * l_x)
        data_inds += list(data_unravel_indices[mask])
    proj_operator = sparse.coo_matrix((weights, (camera_inds, data_inds)))
    return proj_operator
```

Projection operator ¶

In [210]:

```
l = 128
```

In [211]:

```
proj_operator = build_projection_operator(l, l // 7)
```

In [212]:

```
proj_operator
```

Out[212]:

```
<2304x16384 sparse matrix of type '<class 'numpy.float64'>'
        with 555378 stored elements in COOrdinate format>
```

dimensions: angles (l//7), positions (l), image for each (l x l)

In [213]:

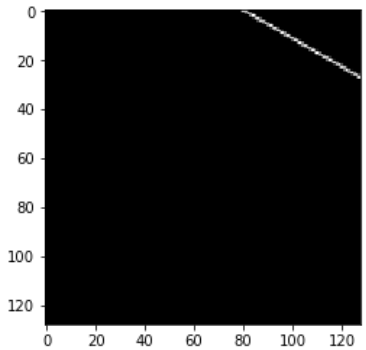
```
proj_t = np.reshape(proj_operator.todense().A, (l//7,l,l,l))
```

The first coordinate refers to the angle of the line, and the second coordinate refers to the location of the line.

The lines for the angle indexed with 3:

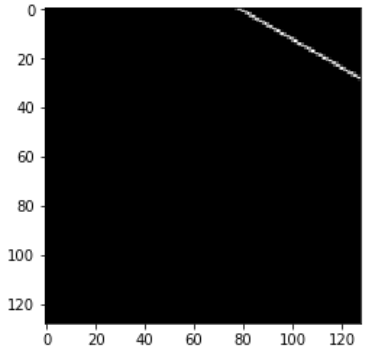
In [214]:

```
plt.imshow(proj_t[3,0], cmap='gray');
```



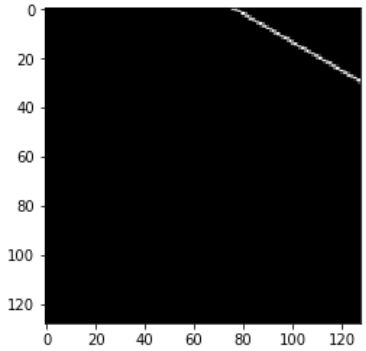
In [215]:

```
plt.imshow(proj_t[3,1], cmap='gray');
```



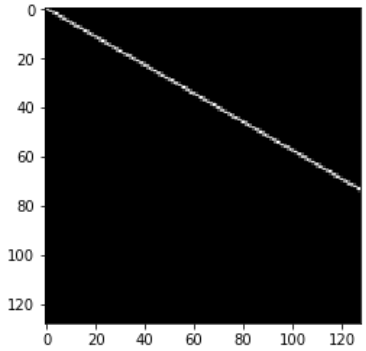
In [216]:

```
plt.imshow(proj_t[3,2], cmap='gray');
```



In [217]:

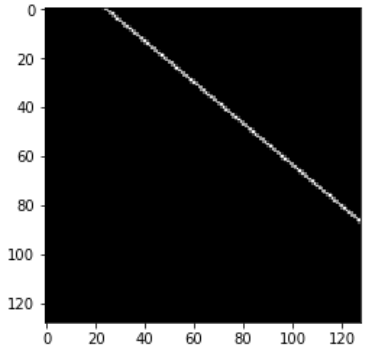
```
plt.imshow(proj_t[3,40], cmap='gray');
```



Other lines at vertical location 40:

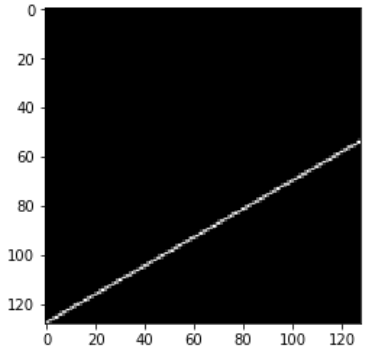
In [218]:

```
plt.imshow(proj_t[4,40], cmap='gray');
```



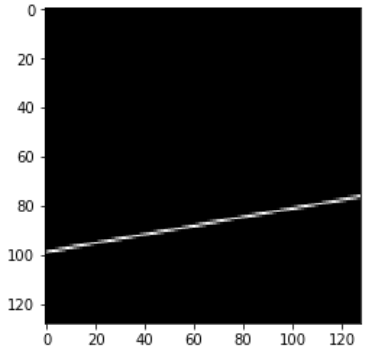
In [219]:

```
plt.imshow(proj_t[15,40], cmap='gray');
```



In [220]:

```
plt.imshow(proj_t[17,40], cmap='gray');
```

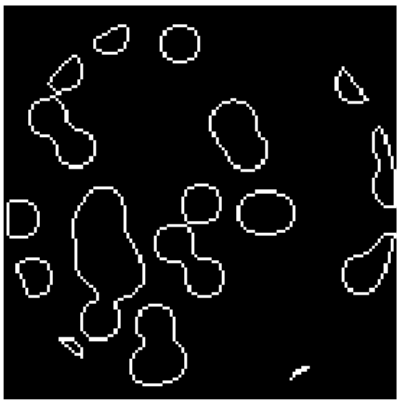


Intersection between x-rays and data ¶

Next, we want to see how the line intersects with our data. Remember, this is what the data looks like:

In [221]:

```
plt.figure(figsize=(5,5))
plt.imshow(data, cmap=plt.cm.gray)
plt.axis('off')
plt.savefig("https://raw.githubusercontent.com/fastai/numerical-linear-algebra/master/nbs/images/data.png")
```



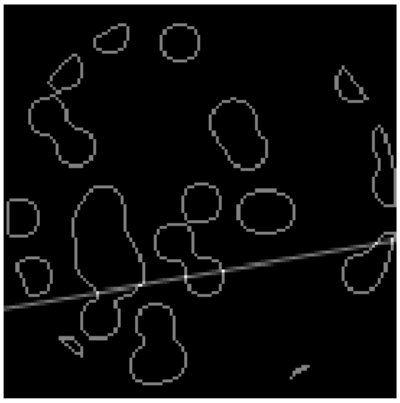
In [222]:

```
proj = proj_operator @ data.ravel()[:, np.newaxis]
```

An x-ray at angle 17, location 40 passing through the data:

In [223]:

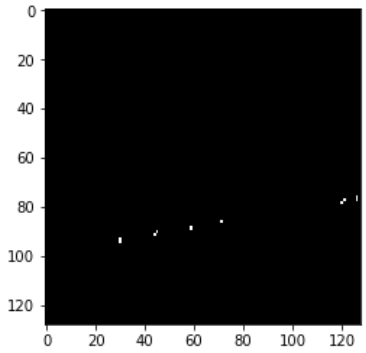
```
plt.figure(figsize=(5,5))
plt.imshow(data + proj_t[17,40], cmap=plt.cm.gray)
plt.axis('off')
plt.savefig("https://raw.githubusercontent.com/fastai/numerical-linear-algebra/master/nbs/images/data_xray.png")
```



Where they intersect:

In [224]:

```
both = data + proj_t[17,40]
plt.imshow((both > 1.1).astype(int), cmap=plt.cm.gray);
```



The intensity of an x-ray at angle 17, location 40 passing through the data:

In [225]:

```
np.resize(proj, (l//7,l))[17,40]
```

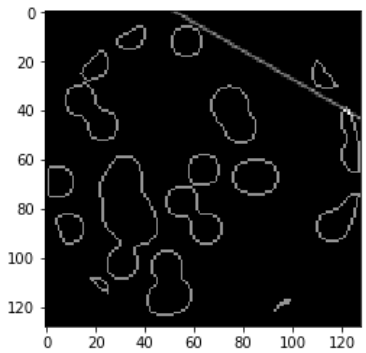
Out[225]:

```
6.4384498372605989
```

The intensity of an x-ray at angle 3, location 14 passing through the data:

In [226]:

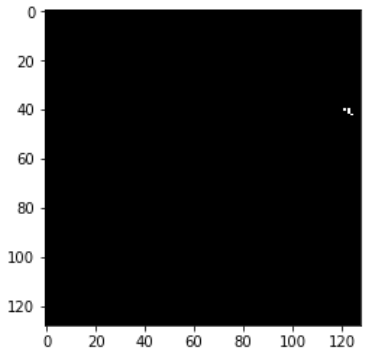
```
plt.imshow(data + proj_t[3,14], cmap=plt.cm.gray);
```



Where they intersect:

In [227]:

```
both = data + proj_t[3,14]
plt.imshow((both > 1.1).astype(int), cmap=plt.cm.gray);
```



The measurement from the CT scan would be a small number here:

In [228]:

```
np.resize(proj, (l//7,l))[3,14]
```

Out[228]:

```
2.1374953737965541
```

In [229]:

```
proj += 0.15 * np.random.randn(*proj.shape)
```

About *args ¶

In [230]:

```
a = [1,2,3]
b= [4,5,6]
```

In [231]:

```
c = list(zip(a, b))
```

In [232]:

```
c
```

Out[232]:

```
[(1, 4), (2, 5), (3, 6)]
```

In [233]:

```
list(zip(*c))
```

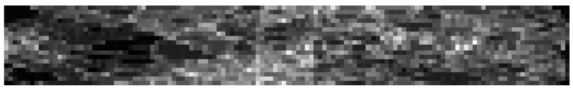
Out[233]:

```
[(1, 2, 3), (4, 5, 6)]
```

The Projection (CT readings) ¶

In [234]:

```
plt.figure(figsize=(7,7))
plt.imshow(np.resize(proj, (l//7,l)), cmap='gray')
plt.axis('off')
plt.savefig("https://raw.githubusercontent.com/fastai/numerical-linear-algebra/master/nbs/images/proj.png")
```



Regresssion ¶

Now we will try to recover the data just from the projections (the measurements of the CT scan)

Linear Regression: $Ax = b$¶

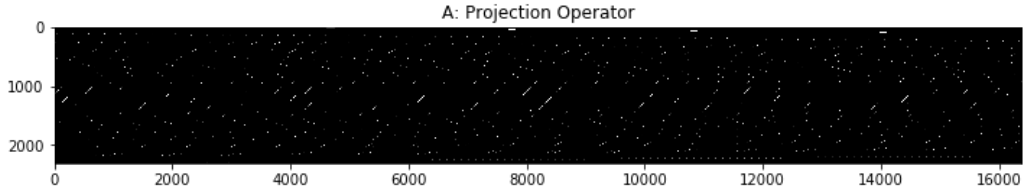
Our matrix A is the projection operator. This was our 4d matrix above (angle, location, x, y) of the different x-rays:

In [203]:

```
plt.figure(figsize=(12,12))
plt.title("A: Projection Operator")
plt.imshow(proj_operator.todense().A, cmap='gray')
```

Out[203]:

```
<matplotlib.image.AxesImage at 0x7efcd4199a20>
```



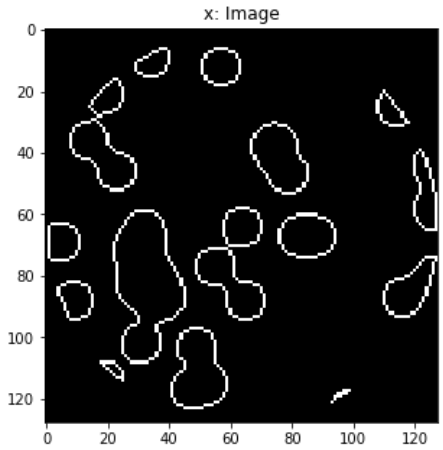
We are solving for x, the original data. We (un)ravel the 2D data into a single column.

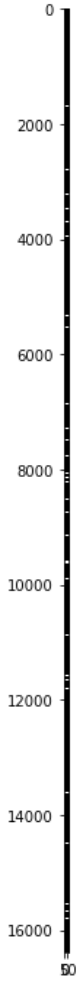
In [202]:

```
plt.figure(figsize=(5,5))
plt.title("x: Image")
plt.imshow(data, cmap='gray')

plt.figure(figsize=(4,12))
# I am tiling the column so that it's easier to see
plt.imshow(np.tile(data.ravel(), (80,1)).T, cmap='gray')
```

Out[202]:

```
<matplotlib.image.AxesImage at 0x7efcd3903be0>
```





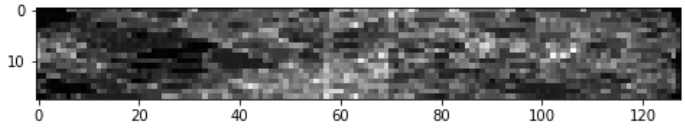
Our vector b is the (un)raveled matrix of measurements:

In [238]:

```
plt.figure(figsize=(8,8))
plt.imshow(np.resize(proj, (l//7,l)), cmap='gray')

plt.figure(figsize=(10,10))
plt.imshow(np.tile(proj.ravel(), (20,1)).T, cmap='gray')
```

Out[238]:

```
<matplotlib.image.AxesImage at 0x7efcd3bebf28>
```



Scikit Learn Linear Regression ¶

In [84]:

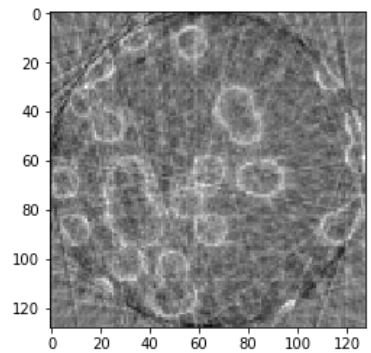
```
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
```

In [126]:

```
# Reconstruction with L2 (Ridge) penalization
rgr_ridge = Ridge(alpha=0.2)
rgr_ridge.fit(proj_operator, proj.ravel())
rec_l2 = rgr_ridge.coef_.reshape(l, l)
plt.imshow(rec_l2, cmap='gray')
```

Out[126]:

```
<matplotlib.image.AxesImage at 0x7efcd453d5c0>
```



In [179]:

```
18*128
```

Out[179]:

```
2304
```

In [ ]:

```
18 x 128 x 128 x 128
```

In [178]:

```
proj_operator.shape
```

Out[178]:

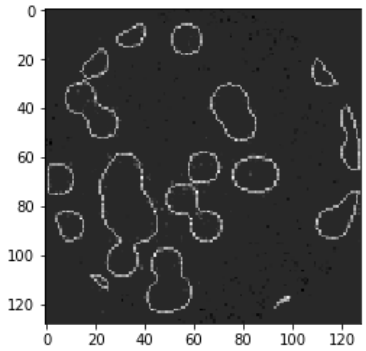
```
(2304, 16384)
```

In [87]:

```
# Reconstruction with L1 (Lasso) penalization
# the best value of alpha was determined using cross validation
# with LassoCV
rgr_lasso = Lasso(alpha=0.001)
rgr_lasso.fit(proj_operator, proj.ravel())
rec_l1 = rgr_lasso.coef_.reshape(l, l)
plt.imshow(rec_l1, cmap='gray')
```

Out[87]:

```
<matplotlib.image.AxesImage at 0x7efcd4919cf8>
```



The L1 penalty works significantly better than the L2 penalty here!

You can read an overview of this Numerical Linear Algebra course in this blog post. The course was originally taught in the University of San Francisco MS in Analytics graduate program. Course lecture videos are available on YouTube (note that the notebook numbers and video numbers do not line up, since some notebooks took longer than 1 video to cover).

You can ask questions about the course on our fast.ai forums.

5. Health Outcomes with Linear Regression ¶

In [26]:

```
from sklearn import datasets, linear_model, metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
import math, scipy, numpy as np
from scipy import linalg
```

Diabetes Dataset ¶

We will use a dataset from patients with diabates. The data consists of 442 samples and 10 variables (all are physiological characteristics), so it is tall and skinny. The dependent variable is a quantitative measure of disease progression one year after baseline.

This is a classic dataset, famously used by Efron, Hastie, Johnstone, and Tibshirani in their Least Angle Regression paper, and one of the many datasets included with scikit-learn.

In [27]:

```
data = datasets.load_diabetes()
```

In [28]:

```
feature_names=['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

In [29]:

```
trn,test,y_trn,y_test = train_test_split(data.data, data.target, test_size=0.2)
```

In [30]:

```
trn.shape, test.shape
```

Out[30]:

```
((353, 10), (89, 10))
```

Linear regression in Scikit Learn ¶

Consider a system $X\beta = y$, where X has more rows than columns. This occurs when you have more data samples than variables. We want to find $\hat{\beta}$ that minimizes:

$$||X\beta - y||_2$$

Let's start by using the sklearn implementation:

In [173]:

```
regr = linear_model.LinearRegression()
%timeit regr.fit(trn, y_trn)
```

```
458 µs ± 62.4 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [7]:

```
pred = regr.predict(test)
```

It will be helpful to have some metrics on how good our prediciton is. We will look at the mean squared norm (L2) and mean absolute error (L1).

In [128]:

```
def regr_metrics(act, pred):
    return (math.sqrt(metrics.mean_squared_error(act, pred)),
     metrics.mean_absolute_error(act, pred))
```

In [129]:

```
regr_metrics(y_test, regr.predict(test))
```

Out[129]:

```
(75.36166834955054, 60.629082113104403)
```

Polynomial Features ¶

Linear regression finds the best coefficients β_i for:

$$x_0\beta_0 + x_1\beta_1 + x_2\beta_2 = y$$

Adding polynomial features is still a linear regression problem, just with more terms:

$$x_0\beta_0 + x_1\beta_1 + x_2\beta_2 + x_0^2\beta_3 + x_0x_1\beta_4 + x_0x_2\beta_5 + x_1^2\beta_6 + x_1x_2\beta_7 + x_2^2\beta_8 = y$$

We need to use our original data X to calculate the additional polynomial features.

In [172]:

```
trn.shape
```

Out[172]:

```
(353, 10)
```

Now, we want to try improving our model's performance by adding some more features. Currently, our model is linear in each variable, but we can add polynomial features to change this.

In [130]:

```
poly = PolynomialFeatures(include_bias=False)
```

In [131]:

```
trn_feat = poly.fit_transform(trn)
```

In [132]:

```
', '.join(poly.get_feature_names(feature_names))
```

Out[132]:

```
'age, sex, bmi, bp, s1, s2, s3, s4, s5, s6, age^2, age sex, age bmi, age bp, age s1, age s2, age s3, age s4, age s5, age s6, sex^2,
```

In [133]:

```
trn_feat.shape
```

Out[133]:

```
(353, 65)
```

In [134]:

```
regr.fit(trn_feat, y_trn)
```

Out[134]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [135]:

```
regr_metrics(y_test, regr.predict(poly.fit_transform(test)))
```

Out[135]:

```
(55.747345922929185, 42.836164292252235)
```

Time is squared in #features and linear in #points, so this will get very slow!

In [136]:

```
%timeit poly.fit_transform(trn)
```

```
635 µs ± 9.25 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Speeding up feature generation ¶

We would like to speed this up. We will use Numba, a Python library that compiles code directly to C.

Numba is a compiler .

Resources ¶

This tutorial from Jake VanderPlas is a nice introduction. Here Jake implements a non-trivial algorithm (non-uniform fast Fourier transform) with Numba.

Cython is another alternative. I've found Cython to require more knowledge to use than Numba (it's closer to C), but to provide similar speed-ups to Numba.

The Takeaway

So numba is 1000 times faster than a pure python implementation, and only marginally slower than nearly identical cython code. There are some caveats here: first of all, I have years of experience with cython, and only an hour's experience with numba. I've used every optimization I know for the cython version, and just the basic vanilla syntax for numba. There are likely ways to tweak the numba version to make it even faster, as indicated in the comments of this post.

All in all, I should say I'm very impressed. Using numba, I added just a *single line* to the original python code, and was able to attain speeds competetive with a highly-optimized (and significantly less "pythonic") cython implementation. Based on this, I'm extremely excited to see what numba brings in the future.

Here is a thorough answer on the differences between an Ahead Of Time (AOT) compiler, a Just In Time (JIT) compiler, and an interpreter.

Experiments with vectorization and native code ¶

Let's first get aquainted with Numba, and then we will return to our problem of polynomial features for regression on the diabates data set.

In [140]:

```
%matplotlib inline
```

In [141]:

```
import math, numpy as np, matplotlib.pyplot as plt
from pandas_summary import DataFrameSummary
from scipy import ndimage
```

In [142]:

```
from numba import jit, vectorize, guvectorize, cuda, float32, void, float64
```

We will show the impact of:

- Avoiding memory allocations and copies (slower than CPU calculations)
- Better locality
- Vectorization

If we use numpy on whole arrays at a time, it creates lots of temporaries, and can't use cache. If we use numba looping through an array item at a time, then we don't have to allocate large temporary arrays, and can reuse cached data since we're doing multiple calculations on each array item.

In [175]:

```
# Untype and Unvectorized
def proc_python(xx,yy):
    zz = np.zeros(nobs, dtype='float32')
    for j in range(nobs):
        x, y = xx[j], yy[j]
        x = x*2 - ( y * 55 )
        y = x + y*2
        z = x + y + 99
        z = z * ( z - .88 )
        zz[j] = z
    return zz
```

In [176]:

```
nobs = 10000
x = np.random.randn(nobs).astype('float32')
y = np.random.randn(nobs).astype('float32')
```

In [177]:

```
%timeit proc_python(x,y)   # Untyped and unvectorized
```

```
49.8 ms ± 1.19 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Numpy ¶

Numpy lets us vectorize this:

In [146]:

```
# Typed and Vectorized
def proc_numpy(x,y):
    z = np.zeros(nobs, dtype='float32')
    x = x*2 - ( y * 55 )
    y = x + y*2
    z = x + y + 99
    z = z * ( z - .88 )
    return z
```

In [147]:

```
np.allclose( proc_numpy(x,y), proc_python(x,y), atol=1e-4 )
```

Out[147]:

```
True
```

In [148]:

```
%timeit proc_numpy(x,y)    # Typed and vectorized
```

```
35.9 µs ± 166 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Numba ¶

Numba offers several different decorators. We will try two different ones:

- `@jit`: very general
- `@vectorize`: don't need to write a for loop. useful when operating on vectors of the same size

First, we will use Numba's jit (just-in-time) compiler decorator, without explicitly vectorizing. This avoids large memory allocations, so we have better locality:

In [149]:

```
@jit()
def proc_numba(xx,yy,zz):
    for j in range(nobs):
        x, y = xx[j], yy[j]
        x = x*2 - ( y * 55 )
        y = x + y*2
        z = x + y + 99
        z = z * ( z - .88 )
        zz[j] = z
    return zz
```

In [150]:

```
z = np.zeros(nobs).astype('float32')
np.allclose( proc_numpy(x,y), proc_numba(x,y,z), atol=1e-4 )
```

Out[150]:

```
True
```

In [151]:

```
%timeit proc_numba(x,y,z)
```

```
6.4 µs ± 17.6 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Now we will use Numba's `vectorize` decorator. Numba's compiler optimizes this in a smarter way than what is possible with plain Python and Numpy.

In [152]:

```
@vectorize
def vec_numba(x,y):
    x = x*2 - ( y * 55 )
    y = x + y*2
    z = x + y + 99
    return z * ( z - .88 )
```

In [153]:

```
np.allclose(vec_numba(x,y), proc_numba(x,y,z), atol=1e-4 )
```

Out[153]:

```
True
```

In [154]:

```
%timeit vec_numba(x,y)
```

```
5.82 µs ± 14.4 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Numba is amazing . Look how fast this is!

Numba polynomial features ¶

In [155]:

```
@jit(nopython=True)
def vec_poly(x, res):
    m,n=x.shape
    feat_idx=0
    for i in range(n):
        v1=x[:,i]
        for k in range(m): res[k,feat_idx] = v1[k]
        feat_idx+=1
        for j in range(i,n):
            for k in range(m): res[k,feat_idx] = v1[k]*x[k,j]
            feat_idx+=1
```

Row-Major vs Column-Major Storage ¶

From this blog post by Eli Bendersky:

"The row-major layout of a matrix puts the first row in contiguous memory, then the second row right after it, then the third, and so on. Column-major layout puts the first column in contiguous memory, then the second, etc.... While knowing which layout a particular data set is using is critical for good performance, there's no single answer to the question which layout 'is better' in general.

"It turns out that matching the way your algorithm works with the data layout can make or break the performance of an application.

"The short takeaway is: always traverse the data in the order it was laid out ."

Column-major layout : Fortran, Matlab, R, and Julia

Row-major layout : C, C++, Python, Pascal, Mathematica

In [156]:

```
trn = np.asfortranarray(trn)
test = np.asfortranarray(test)
```

In [157]:

```
m,n=trn.shape
n_feat = n*(n+1)//2 + n
trn_feat = np.zeros((m,n_feat), order='F')
test_feat = np.zeros((len(y_test), n_feat), order='F')
```

In [158]:

```
vec_poly(trn, trn_feat)
vec_poly(test, test_feat)
```

In [159]:

```
regr.fit(trn_feat, y_trn)
```

Out[159]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [160]:

```
regr_metrics(y_test, regr.predict(test_feat))
```

Out[160]:

```
(55.74734592292935, 42.836164292252306)
```

In [161]:

```
%timeit vec_poly(trn, trn_feat)
```

```
7.33 µs ± 19.8 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Recall, this was the time from the scikit learn implementation PolynomialFeatures, which was created by experts:

In [136]:

```
%timeit poly.fit_transform(trn)
```

```
635 µs ± 9.25 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [162]:

```
605/7.7
```

Out[162]:

```
78.57142857142857
```

This is a big deal! Numba is amazing ! With a single line of code, we are getting a 78x speed-up over scikit learn (which was optimized by experts).

Regularization and noise ¶

Regularization is a way to reduce over-fitting and create models that better generalize to new data.

Regularization ¶

Lasso regression uses an L1 penalty, which pushes towards sparse coefficients. The parameter α is used to weight the penalty term. Scikit Learn's LassoCV performs cross validation with a number of different values for α.

Watch this Coursera video on Lasso regression for more info.

In [163]:

```
reg_regr = linear_model.LassoCV(n_alphas=10)
```

In [164]:

```
reg_regr.fit(trn_feat, y_trn)
```

```
/home/jhoward/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:484: ConvergenceWarning: Objective di
  ConvergenceWarning)
```

Out[164]:

```
LassoCV(alphas=None, copy_X=True, cv=None, eps=0.001, fit_intercept=True,
    max_iter=1000, n_alphas=10, n_jobs=1, normalize=False, positive=False,
```

```
    precompute='auto', random_state=None, selection='cyclic', tol=0.0001,
    verbose=False)
```

In [165]:

```
reg_regr.alpha_
```

Out[165]:

```
0.0098199431661591518
```

In [166]:

```
regr_metrics(y_test, reg_regr.predict(test_feat))
```

Out[166]:

```
(50.0982471642817, 40.065199085003101)
```

Noise ¶

Now we will add some noise to the data

In [167]:

```
idxs = np.random.randint(0, len(trn), 10)
```

In [168]:

```
y_trn2 = np.copy(y_trn)
y_trn2[idxs] *= 10 # label noise
```

In [169]:

```
regr = linear_model.LinearRegression()
regr.fit(trn, y_trn)
regr_metrics(y_test, regr.predict(test))
```

Out[169]:

```
(51.1766253181518, 41.415992803872754)
```

In [170]:

```
regr.fit(trn, y_trn2)
regr_metrics(y_test, regr.predict(test))
```

Out[170]:

```
(62.66110319520415, 53.21914420254862)
```

Huber loss is a loss function that is less sensitive to outliers than squared error loss. It is quadratic for small error values, and linear for large values.

Undefined control sequence \lvert

In [171]:

```
hregr = linear_model.HuberRegressor()
hregr.fit(trn, y_trn2)
regr_metrics(y_test, hregr.predict(test))
```

Out[171]:

```
(51.24055602541746, 41.670840571376822)
```

End ¶

You can read an overview of this Numerical Linear Algebra course in this blog post. The course was originally taught in the University of San Francisco MS in Analytics graduate program. Course lecture videos are available on YouTube (note that the notebook numbers and video numbers do not line up,

since some notebooks took longer than 1 video to cover).

You can ask questions about the course on our fast.ai forums.

6. How to Implement Linear Regression ¶

In the previous lesson, we calculated the least squares linear regression for a diabetes dataset, using scikit learn's implementation. Today, we will look at how we could write our own implementation.

In [2]:

```
from sklearn import datasets, linear_model, metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
import math, scipy, numpy as np
from scipy import linalg
```

In [3]:

```
np.set_printoptions(precision=6)
```

In [4]:

```
data = datasets.load_diabetes()
```

In [5]:

```
feature_names=['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

In [6]:

```
trn,test,y_trn,y_test = train_test_split(data.data, data.target, test_size=0.2)
```

In [7]:

```
trn.shape, test.shape
```

Out[7]:

```
((353, 10), (89, 10))
```

In [8]:

```
def regr_metrics(act, pred):
    return (math.sqrt(metrics.mean_squared_error(act, pred)),
     metrics.mean_absolute_error(act, pred))
```

How did sklearn do it? ¶

How is sklearn doing this? By checking the source code, you can see that in the dense case, it calls scipy.linalg.lstqr, which is calling a LAPACK method:

```
    Options are ``'gelsd'``, ``'gelsy'``, ``'gelss'``. Default
    (``'gelsd'``) is a good choice.  However, ``'gelsy'`` can be slightly
    faster on many problems.  ``'gelss'`` was used historically.  It is
    generally slow but uses less memory.
```

- gelsd: uses SVD and a divide-and-conquer method
- gelsy: uses QR factorization
- gelss: uses SVD

Scipy Sparse Least Squares ¶

We will not get into too much detail about the sparse version of least squares. Here is a bit of info if you are interested:

Scipy sparse lsqr uses an iterative method called Golub and Kahan bidiagonalization.

from scipy sparse lsqr source code: Preconditioning is another way to reduce the number of iterations. If it is possible to solve a related system `M*x = b` efficiently, where M approximates A in some helpful way (e.g. M - A has low rank or its elements are small relative to those of A), LSQR may converge

more rapidly on the system `A*M(inverse)*z = b`, after which x can be recovered by solving M*x = z.

If A is symmetric, LSQR should not be used! Alternatives are the symmetric conjugate-gradient method (cg) and/or SYMMLQ. SYMMLQ is an implementation of symmetric cg that applies to any symmetric A and will converge more rapidly than LSQR. If A is positive definite, there are other implementations of symmetric cg that require slightly less work per iteration than SYMMLQ (but will take the same number of iterations).

linalg.lstqr ¶

The sklearn implementation handled adding a constant term (since the y-intercept is presumably not 0 for the line we are learning) for us. We will need to do that by hand now:

In [9]:

```
trn_int = np.c_[trn, np.ones(trn.shape[0])]
test_int = np.c_[test, np.ones(test.shape[0])]
```

Since `linalg.lstsq` lets us specify which LAPACK routine we want to use, lets try them all and do some timing comparisons:

In [10]:

```
%timeit coef, _,_,_ = linalg.lstsq(trn_int, y_trn, lapack_driver="gelsd")
```

```
290 µs ± 9.24 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [11]:

```
%timeit coef, _,_,_ = linalg.lstsq(trn_int, y_trn, lapack_driver="gelsy")
```

```
140 µs ± 91.7 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [12]:

```
%timeit coef, _,_,_ = linalg.lstsq(trn_int, y_trn, lapack_driver="gelss")
```

```
199 µs ± 228 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Naive Solution ¶

Recall that we want to find $\hat{x}$ that minimizes:

$$||Ax - b||_2$$

Another way to think about this is that we are interested in where vector b is closest to the subspace spanned by A (called the range of A). This is the projection of b onto A. Since $b - A\hat{x}$ must be perpendicular to the subspace spanned by A, we see that

$$A^T(b - A\hat{x}) = 0$$

(we are using A^T because we want to multiply each column of A by $b - A\hat{x}$

This leads us to the normal equations:

$$x = (A^TA)^{-1}A^Tb$$

In [13]:

```
def ls_naive(A, b):
    return np.linalg.inv(A.T @ A) @ A.T @ b
```

In [14]:

```
%timeit coeffs_naive = ls_naive(trn_int, y_trn)
```

```
45.8 µs ± 4.65 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [15]:

```
coeffs_naive = ls_naive(trn_int, y_trn)
regr_metrics(y_test, test_int @ coeffs_naive)
```

Out[15]:

```
(57.94102134545707, 48.053565198516438)
```

Normal Equations (Cholesky) ¶

Normal equations:

$$A^T Ax = A^T b$$

If A has full rank, the pseudo-inverse $(A^T A)^{-1} A^T$ is a square, hermitian positive definite matrix. The standard way of solving such a system is Cholesky Factorization, which finds upper-triangular R s.t. $A^T A = R^T R$.

The following steps are based on Algorithm 11.1 from Trefethen:

In [16]:

```
A = trn_int
```

In [17]:

```
b = y_trn
```

In [18]:

```
AtA = A.T @ A
Atb = A.T @ b
```

Warning: Numpy and Scipy default to different upper/lower for Cholesky

In [19]:

```
R = scipy.linalg.cholesky(AtA)
```

In [196]:

```
np.set_printoptions(suppress=True, precision=4)
R
```

Out[196]:

```
array([[  0.9124,   0.1438,   0.1511,   0.3002,   0.2228,   0.188 ,
         -0.051 ,   0.1746,   0.22  ,   0.2768,  -0.2583],
       [  0.    ,   0.8832,   0.0507,   0.1826,  -0.0251,   0.0928,
         -0.3842,   0.2999,   0.0911,   0.15  ,   0.4393],
       [  0.    ,   0.    ,   0.8672,   0.2845,   0.2096,   0.2153,
         -0.2695,   0.3181,   0.3387,   0.2894,  -0.005 ],
       [  0.    ,   0.    ,   0.    ,   0.7678,   0.0762,  -0.0077,
          0.0383,   0.0014,   0.165 ,   0.166 ,   0.0234],
       [  0.    ,   0.    ,   0.    ,   0.    ,   0.8288,   0.7381,
          0.1145,   0.4067,   0.3494,   0.158 ,  -0.2826],
       [  0.    ,   0.    ,   0.    ,   0.    ,   0.    ,   0.3735,
         -0.3891,   0.2492,  -0.3245,  -0.0323,  -0.1137],
       [  0.    ,   0.    ,   0.    ,   0.    ,   0.    ,   0.    ,
          0.6406,  -0.511 ,  -0.5234,  -0.172 ,  -0.9392],
       [  0.    ,   0.    ,   0.    ,   0.    ,   0.    ,   0.    ,
          0.    ,   0.2887,  -0.0267,  -0.0062,   0.0643],
       [  0.    ,   0.    ,   0.    ,   0.    ,   0.    ,   0.    ,
          0.    ,   0.    ,   0.2823,   0.0636,   0.9355],
       [  0.    ,   0.    ,   0.    ,   0.    ,   0.    ,   0.    ,
          0.    ,   0.    ,   0.    ,   0.7238,   0.0202],
       [  0.    ,   0.    ,   0.    ,   0.    ,   0.    ,   0.    ,
          0.    ,   0.    ,   0.    ,   0.    ,  18.7319]])
```

check our factorization:

In [20]:

```
np.linalg.norm(AtA - R.T @ R)
```

Out[20]:

```
4.5140158187158533e-16
```

$$A^TAx = A^Tb$$
$$R^TRx = A^Tb$$
$$R^Tw = A^Tb$$
$$Rx = w$$

In [21]:

```
w = scipy.linalg.solve_triangular(R, Atb, lower=False, trans='T')
```

It's always good to check that our result is what we expect it to be: (in case we entered the wrong params, the function didn't return what we thought, or sometimes the docs are even outdated)

In [22]:

```
np.linalg.norm(R.T @ w - Atb)
```

Out[22]:

```
1.1368683772161603e-13
```

In [23]:

```
coeffs_chol = scipy.linalg.solve_triangular(R, w, lower=False)
```

In [24]:

```
np.linalg.norm(R @ coeffs_chol - w)
```

Out[24]:

```
6.861429794408013e-14
```

In [25]:

```
def ls_chol(A, b):
    R = scipy.linalg.cholesky(A.T @ A)
    w = scipy.linalg.solve_triangular(R, A.T @ b, trans='T')
    return scipy.linalg.solve_triangular(R, w)
```

In [26]:

```
%timeit coeffs_chol = ls_chol(trn_int, y_trn)
```

```
111 µs ± 272 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [27]:

```
coeffs_chol = ls_chol(trn_int, y_trn)
regr_metrics(y_test, test_int @ coeffs_chol)
```

Out[27]:

```
(57.9410213454571, 48.053565198516438)
```

QR Factorization ¶

$$Ax = b$$
$$A = QR$$
$$QRx = b$$
$$Rx = Q^Tb$$

In [28]:

```
def ls_qr(A,b):
    Q, R = scipy.linalg.qr(A, mode='economic')
    return scipy.linalg.solve_triangular(R, Q.T @ b)
```

In [29]:

```
%timeit coeffs_qr = ls_qr(trn_int, y_trn)
```

```
205 µs ± 264 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [30]:

```
coeffs_qr = ls_qr(trn_int, y_trn)
regr_metrics(y_test, test_int @ coeffs_qr)
```

Out[30]:

```
(57.94102134545711, 48.053565198516452)
```

SVD¶

$$Ax = b$$
$$A = U\Sigma V$$
$$\Sigma V x = U^T b$$
$$\Sigma w = U^T b$$
$$x = V^T w$$

SVD gives the pseudo-inverse

In [253]:

```
def ls_svd(A,b):
    m, n = A.shape
    U, sigma, Vh = scipy.linalg.svd(A, full_matrices=False, lapack_driver='gesdd')
    w = (U.T @ b)/ sigma
    return Vh.T @ w
```

In [32]:

```
%timeit coeffs_svd = ls_svd(trn_int, y_trn)
```

```
1.11 ms ± 320 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [254]:

```
%timeit coeffs_svd = ls_svd(trn_int, y_trn)
```

```
266 µs ± 8.49 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [255]:

```
coeffs_svd = ls_svd(trn_int, y_trn)
regr_metrics(y_test, test_int @ coeffs_svd)
```

Out[255]:

```
(57.941021345457244, 48.053565198516687)
```

Random Sketching T echnique for Least Squares Regression ¶

Linear Sketching (Woodruff)

1. Sample a r x n random matrix S, r << n
2. Compute S A and S b
3. Find exact solution x to regression SA x = Sb

Timing Comparison ¶

In [244]:

```
import timeit
import pandas as pd
```

In [245]:

```
def scipylstq(A, b):
    return scipy.linalg.lstsq(A,b)[0]
```

In [246]:

```
row_names = ['Normal Eqns- Naive',
             'Normal Eqns- Cholesky',
             'QR Factorization',
             'SVD',
             'Scipy lstsq']

name2func = {'Normal Eqns- Naive': 'ls_naive',
             'Normal Eqns- Cholesky': 'ls_chol',
             'QR Factorization': 'ls_qr',
             'SVD': 'ls_svd',
             'Scipy lstsq': 'scipylstq'}
```

In [247]:

```
m_array = np.array([100, 1000, 10000])
n_array = np.array([20, 100, 1000])
```

In [248]:

```
index = pd.MultiIndex.from_product([m_array, n_array], names=['# rows', '# cols'])
```

In [249]:

```
pd.options.display.float_format = '{:,.6f}'.format
df = pd.DataFrame(index=row_names, columns=index)
df_error = pd.DataFrame(index=row_names, columns=index)
```

In [256]:

```
# %%prun
for m in m_array:
    for n in n_array:
        if m >= n:
            x = np.random.uniform(-10,10,n)
            A = np.random.uniform(-40,40,[m,n])   # removed np.asfortranarray
            b = np.matmul(A, x) + np.random.normal(0,2,m)
            for name in row_names:
                fcn = name2func[name]
                t = timeit.timeit(fcn + '(A,b)', number=5, globals=globals())
                df.set_value(name, (m,n), t)
                coeffs = locals()[fcn](A, b)
                reg_met = regr_metrics(b, A @ coeffs)
                df_error.set_value(name, (m,n), reg_met[0])
```

In [257]:

```
df
```

Out[257]:

# rows	100			1000			10000		
# cols	20	100	1000	20	100	1000	20	100	1000
Normal Eqns- Naive	0.001276	0.003634	NaN	0.000960	0.005172	0.293126	0.002226	0.021248	1.164655
Normal Eqns- Cholesky	0.001660	0.003958	NaN	0.001665	0.004007	0.093696	0.001928	0.010456	0.399464

# rows	100			1000			10000		
# cols	20	100	1000	20	100	1000	20	100	1000
QR Factorization	0.002174	0.006486	NaN	0.004235	0.017773	0.213232	0.019229	0.116122	2.208129
SVD	0.003880	0.021737	NaN	0.004672	0.026950	1.280490	0.018138	0.130652	3.433003
Scipy lstsq	0.004338	0.020198	NaN	0.004320	0.021199	1.083804	0.012200	0.088467	2.134780

In [252]:

```
df_error
```

Out[252]:

# rows	100			1000			10000		
# cols	20	100	1000	20	100	1000	20	100	1000
Normal Eqns- Naive	1.702742	0.000000	NaN	1.970767	1.904873	0.000000	1.978383	1.980449	1.884440
Normal Eqns- Cholesky	1.702742	0.000000	NaN	1.970767	1.904873	0.000000	1.978383	1.980449	1.884440
QR Factorization	1.702742	0.000000	NaN	1.970767	1.904873	0.000000	1.978383	1.980449	1.884440
SVD	1.702742	0.000000	NaN	1.970767	1.904873	0.000000	1.978383	1.980449	1.884440
Scipy lstsq	1.702742	0.000000	NaN	1.970767	1.904873	0.000000	1.978383	1.980449	1.884440

In [618]:

```
store = pd.HDFStore('least_squares_results.h5')
```

In [619]:

```
store['df'] = df
```

```
C:\Users\rache\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:2881: PerformanceWarning:
your performance may suffer as PyTables will pickle object types that it cannot
map directly to c-types [inferred_type->floating,key->block0_values] [items->[(100, 20), (100, 100), (100, 1000), (1000, 20), (1000,

  exec(code_obj, self.user_global_ns, self.user_ns)
```

Notes ¶

I used the magick %prun to profile my code.

Alternative: least absolute deviation (L1 regression)

- Less sensitive to outliers than least squares.
- No closed form solution, but can solve with linear programming

Conditioning & stability ¶

Condition Number ¶

Condition number is a measure of how small changes to the input cause the output to change.

Question : Why do we care about behavior with small changes to the input in numerical linear algebra?

The relative condition number is defined by

$$\kappa = \sup_{\delta x} \frac{\|\delta f\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|}$$

where δx is infinitesimal

According to Trefethen (pg. 91), a problem is well-conditioned if κ is small (e.g. $1, 10, 10^2$) and ill-conditioned if κ is large (e.g. $10^6, 10^{16}$)

Conditioning : perturbation behavior of a mathematical problem (e.g. least squares)

Stability : perturbation behavior of an algorithm used to solve that problem on a computer (e.g. least squares algorithms, householder, back substitution, gaussian elimination)

Conditioning example ¶

The problem of computing eigenvalues of a non-symmetric matrix is often ill-conditioned

In [178]:

```
A = [[1, 1000], [0, 1]]
B = [[1, 1000], [0.001, 1]]
```

In [179]:

```
wA, vrA = scipy.linalg.eig(A)
wB, vrB = scipy.linalg.eig(B)
```

In [180]:

```
wA, wB
```

Out[180]:

```
(array([ 1.+0.j,  1.+0.j]),
 array([  2.00000000e+00+0.j,  -2.22044605e-16+0.j]))
```

Condition Number of a Matrix ¶

The product $||A||||A^{-1}||$ comes up so often it has its own name: the condition number of A. Note that normally we talk about the conditioning of problems, not matrices.

The condition number of A relates to:

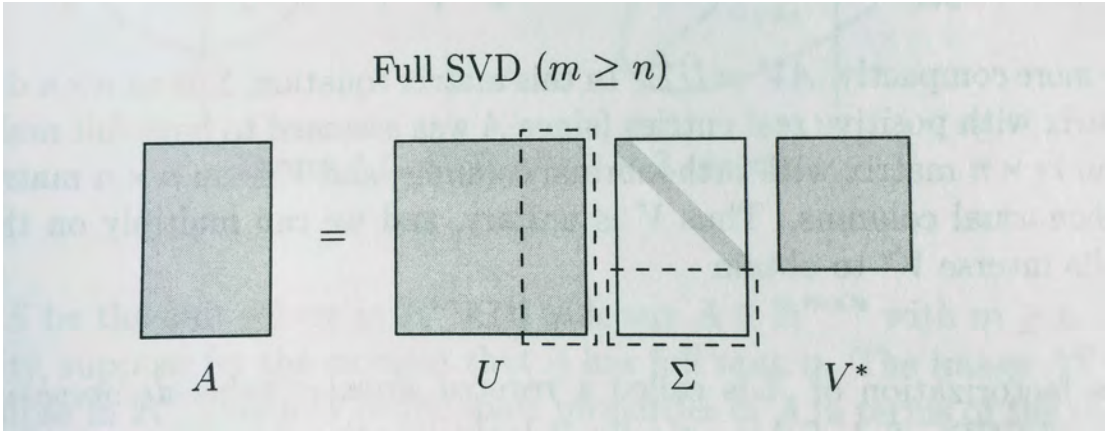
- computing b given A and x in $Ax = b$
- computing x given A and b in $Ax = b$

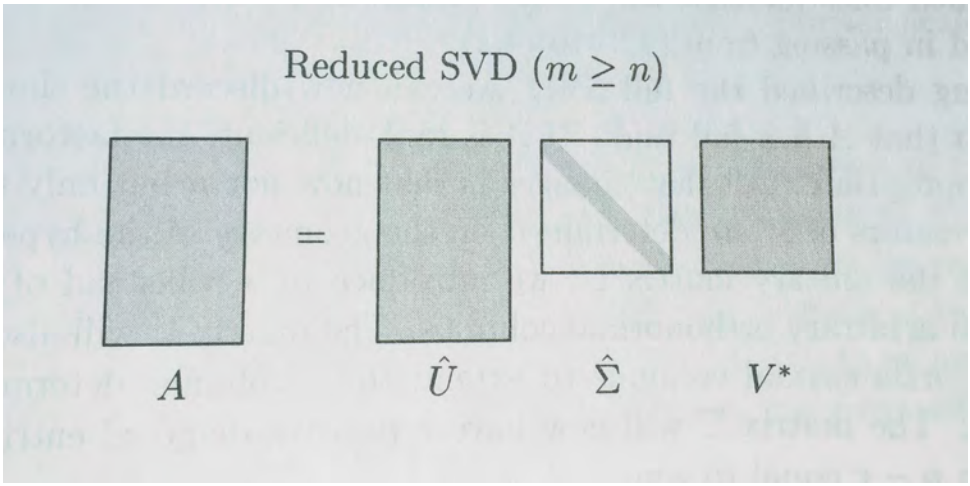
Loose ends from last time ¶

Full vs Reduced Factorizations ¶

SVD

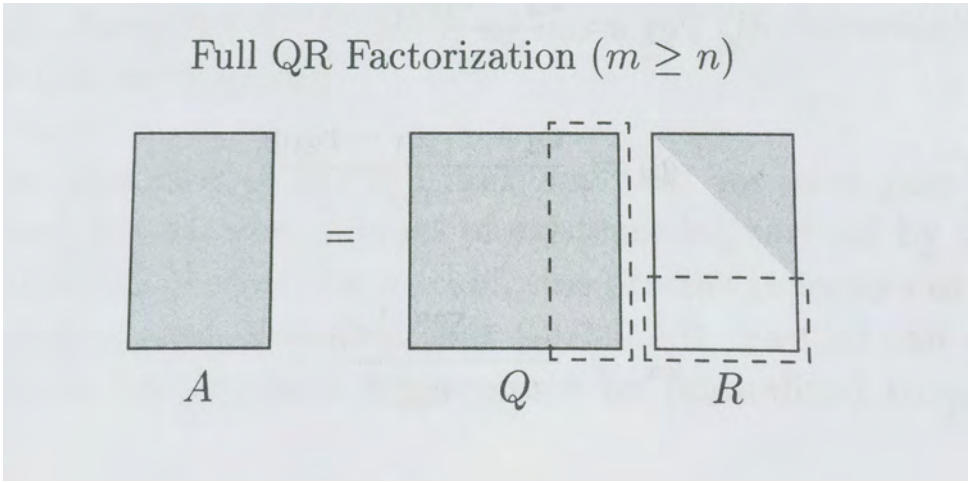
Diagrams from Trefethen:

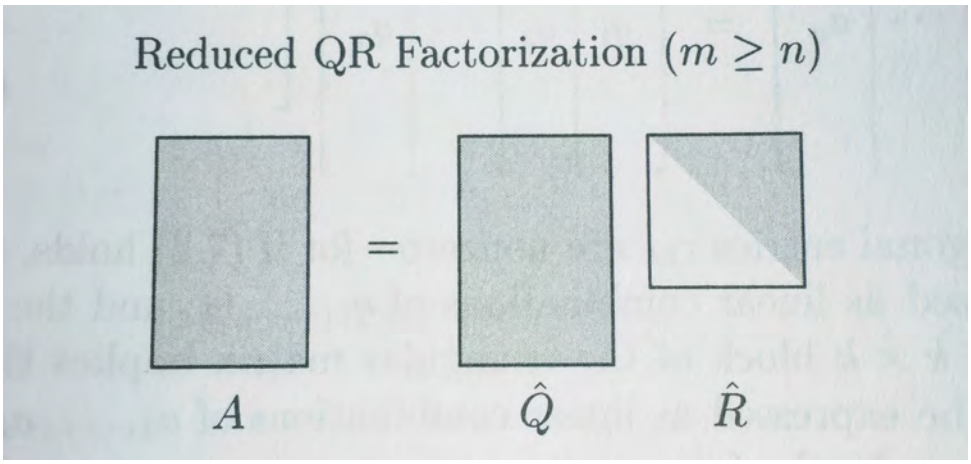




QR Factorization exists for ALL matrices

Just like with SVD, there are full and reduced versions of the QR factorization.





Matrix Inversion is Unstable ¶

In [197]:

```
from scipy.linalg import hilbert
```

In [229]:

```
n = 14
A = hilbert(n)
x = np.random.uniform(-10,10,n)
b = A @ x
```

In [230]:

```
A_inv = np.linalg.inv(A)
```

In [233]:

```
np.linalg.norm(np.eye(n) - A @ A_inv)
```

Out[233]:

```
5.0516495470543212
```

In [231]:

```
np.linalg.cond(A)
```

Out[231]:

```
2.2271635826494112e+17
```

In [232]:

```
A @ A_inv
```

Out[232]:

```
array([[ 1.    ,  0.    , -0.0001,  0.0005, -0.0006,  0.0105, -0.0243,
         0.1862, -0.6351,  2.2005, -0.8729,  0.8925, -0.0032, -0.0106],
       [ 0.    ,  1.    , -0.    ,  0.    ,  0.0035,  0.0097, -0.0408,
         0.0773, -0.0524,  1.6926, -0.7776, -0.111 , -0.0403, -0.0184],
       [ 0.    ,  0.    ,  1.    ,  0.0002,  0.0017,  0.0127, -0.0273,
         0.    ,  0.    ,  1.4688, -0.5312,  0.2812,  0.0117,  0.0264],
       [ 0.    ,  0.    , -0.    ,  1.0005,  0.0013,  0.0098, -0.0225,
         0.1555, -0.0168,  1.1571, -0.9656, -0.0391,  0.018 , -0.0259],
       [-0.    ,  0.    , -0.    ,  0.0007,  1.0001,  0.0154,  0.011 ,
        -0.2319,  0.5651, -0.2017,  0.2933, -0.6565,  0.2835, -0.0482],
       [ 0.    , -0.    ,  0.    , -0.0004,  0.0059,  0.9945, -0.0078,
        -0.0018, -0.0066,  1.1839, -0.9919,  0.2144, -0.1866,  0.0187],
       [-0.    ,  0.    , -0.    ,  0.0009, -0.002 ,  0.0266,  0.974 ,
        -0.146 ,  0.1883, -0.2966,  0.4267, -0.8857,  0.2265, -0.0453],
       [ 0.    ,  0.    , -0.    ,  0.0002,  0.0009,  0.0197, -0.0435,
         1.1372, -0.0692,  0.7691, -1.233 ,  0.1159, -0.1766, -0.0033],
       [ 0.    ,  0.    , -0.    ,  0.0002,  0.    , -0.0018, -0.0136,
         0.1332,  0.945 ,  0.3652, -0.2478, -0.1682,  0.0756, -0.0212],
       [ 0.    , -0.    , -0.    ,  0.0003,  0.0038, -0.0007,  0.0318,
        -0.0738,  0.2245,  1.2023, -0.2623, -0.2783,  0.0486, -0.0093],
       [-0.    ,  0.    , -0.    ,  0.0004, -0.0006,  0.013 , -0.0415,
         0.0292, -0.0371,  0.169 ,  1.0715, -0.09  ,  0.1668, -0.0197],
       [ 0.    , -0.    ,  0.    ,  0.    ,  0.0016,  0.0062, -0.0504,
         0.1476, -0.2341,  0.8454, -0.7907,  1.4812, -0.15  ,  0.0186],
       [ 0.    , -0.    ,  0.    , -0.0001,  0.0022,  0.0034, -0.0296,
         0.0944, -0.1833,  0.6901, -0.6526,  0.2556,  0.8563,  0.0128],
       [ 0.    ,  0.    ,  0.    , -0.0001,  0.0018, -0.0041, -0.0057,
        -0.0374, -0.165 ,  0.3968, -0.2264, -0.1538, -0.0076,  1.005 ]])
```

In [237]:

```
row_names = ['Normal Eqns- Naive',
             'QR Factorization',
             'SVD',
             'Scipy lstsq']

name2func = {'Normal Eqns- Naive': 'ls_naive',
             'QR Factorization': 'ls_qr',
             'SVD': 'ls_svd',
             'Scipy lstsq': 'scipylstq'}
```

In [238]:

```
pd.options.display.float_format = '{:,.9f}'.format
df = pd.DataFrame(index=row_names, columns=['Time', 'Error'])
```

In [239]:

```
for name in row_names:
    fcn = name2func[name]
```

```
    t = timeit.timeit(fcn + '(A,b)', number=5, globals=globals())
    coeffs = locals()[fcn](A, b)
    df.set_value(name, 'Time', t)
    df.set_value(name, 'Error', regr_metrics(b, A @ coeffs)[0])
```

SVD is best here! ¶

DO NOT RERUN

In [240]:

```
df
```

Out[240]:

	Time	Error
Normal Eqns- Naive	0.001334339	3.598901966
QR Factorization	0.002166139	0.000000000
SVD	0.001556937	0.000000000
Scipy lstsq	0.001871590	0.000000000

In [240]:

```
df
```

Out[240]:

	Time	Error
Normal Eqns- Naive	0.001334339	3.598901966
QR Factorization	0.002166139	0.000000000
SVD	0.001556937	0.000000000
Scipy lstsq	0.001871590	0.000000000

Another reason not to take inverse

Even if A is incredibly sparse, A^{-1} is generally dense. For large matrices, A^{-1} could be so dense as to not fit in memory.

Runtime ¶

Matrix Inversion: $2n^3$

Matrix Multiplication: n^3

Cholesky: $\frac{1}{3}n^3$

QR, Gram Schmidt: $2mn^2$, $m \geq n$ (chapter 8 of Trefethen)

QR, Householder: $2mn^2 - \frac{2}{3}n^3$ (chapter 10 of Trefethen)

Solving a triangular system: n^2

Why Cholesky Factorization is Fast:

Cholesky factorization

every positive definite A can be factored as

$$A = LL^T$$

with L lower triangular

cost: $(1/3)n^3$ flops

Solving linear equations by Cholesky factorization.

given a set of linear equations $Ax = b$, with $A \in \mathbf{S}^n_{++}$.

1. *Cholesky factorization.* Factor A as $A = LL^T$ ($(1/3)n^3$ flops).
2. *Forward substitution.* Solve $Lz_1 = b$ (n^2 flops).
3. *Backward substitution.* Solve $L^T x = z_1$ (n^2 flops).

---

cost: $(1/3)n^3 + 2n^2 \approx (1/3)n^3$ for large n

(source: Stanford Convex Optimization: Numerical Linear Algebra Background Slides)

A Case Where QR is the Best ¶

In [65]:

```
m=100
n=15
t=np.linspace(0, 1, m)
```

In [66]:

```
# Vandermonde matrix
A=np.stack([t**i for i in range(n)], 1)
```

In [67]:

```
b=np.exp(np.sin(4*t))

# This will turn out to normalize the solution to be 1
b /= 2006.787453080206
```

In [68]:

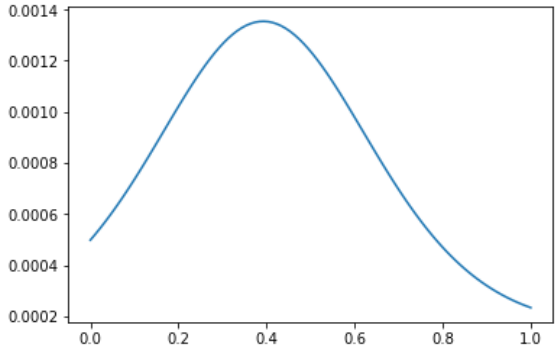
```
from matplotlib import pyplot as plt
%matplotlib inline
```

In [69]:

```
plt.plot(t, b)
```

Out[69]:

```
[<matplotlib.lines.Line2D at 0x7fdfc1fa7eb8>]
```



Check that we get 1:

In [58]:

```
1 - ls_qr(A, b)[14]
```

Out[58]:

```
1.4137685733217609e-07
```

Bad condition number:

In [60]:

```
kappa = np.linalg.cond(A); kappa
```

Out[60]:

```
5.827807196683593e+17
```

In [181]:

```
row_names = ['Normal Eqns- Naive',
             'QR Factorization',
             'SVD',
             'Scipy lstsq']

name2func = {'Normal Eqns- Naive': 'ls_naive',
             'QR Factorization': 'ls_qr',
             'SVD': 'ls_svd',
             'Scipy lstsq': 'scipylstq'}
```

In [74]:

```
pd.options.display.float_format = '{:,.9f}'.format
df = pd.DataFrame(index=row_names, columns=['Time', 'Error'])
```

In [75]:

```
for name in row_names:
    fcn = name2func[name]
    t = timeit.timeit(fcn + '(A,b)', number=5, globals=globals())
    coeffs = locals()[fcn](A, b)
    df.set_value(name, 'Time', t)
    df.set_value(name, 'Error', np.abs(1 - coeffs[-1]))
```

In [76]:

```
df
```

Out[76]:

	Time	Error
Normal Eqns- Naive	0.001565099	1.357066025
QR Factorization	0.002632104	0.000000116

	Time	Error
SVD	0.003503785	0.000000116
Scipy lstsq	0.002763502	0.000000116

The solution for least squares via the normal equations is unstable in general, although stable for problems with small condition numbers.

Low-rank ¶

In [258]:

```
m = 100
n = 10
x = np.random.uniform(-10,10,n)
A2 = np.random.uniform(-40,40, [m, int(n/2)])   # removed np.asfortranarray
A = np.hstack([A2, A2])
```

In [259]:

```
A.shape, A2.shape
```

Out[259]:

```
((100, 10), (100, 5))
```

In [260]:

```
b = A @ x + np.random.normal(0,1,m)
```

In [263]:

```
row_names = ['Normal Eqns- Naive',
             'QR Factorization',
             'SVD',
             'Scipy lstsq']

name2func = {'Normal Eqns- Naive': 'ls_naive',
             'QR Factorization': 'ls_qr',
             'SVD': 'ls_svd',
             'Scipy lstsq': 'scipylstq'}
```

In [264]:

```
pd.options.display.float_format = '{:,.9f}'.format
df = pd.DataFrame(index=row_names, columns=['Time', 'Error'])
```

In [265]:

```
for name in row_names:
    fcn = name2func[name]
    t = timeit.timeit(fcn + '(A,b)', number=5, globals=globals())
    coeffs = locals()[fcn](A, b)
    df.set_value(name, 'Time', t)
    df.set_value(name, 'Error', regr_metrics(b, A @ coeffs)[0])
```

In [266]:

```
df
```

Out[266]:

	Time	Error
Normal Eqns- Naive	0.001227640	300.658979382
QR Factorization	0.002315920	0.876019803
SVD	0.001745647	1.584746056
Scipy lstsq	0.002067989	0.804750398

Comparison ¶

Our results from above:

In [257]:

```
df
```

Out[257]:

# rows	100			1000			10000		
# cols	20	100	1000	20	100	1000	20	100	1000
Normal Eqns- Naive	0.001276	0.003634	NaN	0.000960	0.005172	0.293126	0.002226	0.021248	1.164655
Normal Eqns- Cholesky	0.001660	0.003958	NaN	0.001665	0.004007	0.093696	0.001928	0.010456	0.399464
QR Factorization	0.002174	0.006486	NaN	0.004235	0.017773	0.213232	0.019229	0.116122	2.208129
SVD	0.003880	0.021737	NaN	0.004672	0.026950	1.280490	0.018138	0.130652	3.433003
Scipy lstsq	0.004338	0.020198	NaN	0.004320	0.021199	1.083804	0.012200	0.088467	2.134780

From Trefethen (page 84):

Normal equations/Cholesky is fastest when it works. Cholesky can only be used on symmetric, positive definite matrices. Also, normal equations/Cholesky is unstable for matrices with high condition numbers or with low-rank.

Linear regression via QR has been recommended by numerical analysts as the standard method for years. It is natural, elegant, and good for "daily use".

You can read an overview of this Numerical Linear Algebra course in this blog post. The course was originally taught in the University of San Francisco MS in Analytics graduate program. Course lecture videos are available on YouTube (note that the notebook numbers and video numbers do not line up, since some notebooks took longer than 1 video to cover).

You can ask questions about the course on our fast.ai forums.

7. PageRank with Eigen Decompositions ¶

Two Handy T ricks ¶

Here are two tools that we'll be using today, which are useful in general.

1. Psutil is a great way to check on your memory usage. This will be useful here since we are using a larger data set.

In [462]:

```
import psutil
```

In [447]:

```
process = psutil.Process(os.getpid())
t = process.memory_info()
```

In [449]:

```
t.vms, t.rss
```

Out[449]:

```
(19475513344, 17856520192)
```

In [450]:

```
def mem_usage():
    process = psutil.Process(os.getpid())
    return process.memory_info().rss / psutil.virtual_memory().total
```

In [451]:

```
mem_usage()
```

Out[451]:

```
0.13217061955758594
```

2. TQDM gives you progress bars.

In [364]:

```
from time import sleep
```

In [463]:

```
# Without TQDM
s = 0
for i in range(10):
    s += i
    sleep(0.2)
print(s)
```

```
45
```

In [465]:

```
# With TQDM
from tqdm import tqdm

s = 0
for i in tqdm(range(10)):
    s += i
    sleep(0.2)
print(s)
```

```
100%|██████████| 10/10 [00:02<00:00,  4.96it/s]
```

```
45
```

Motivation ¶

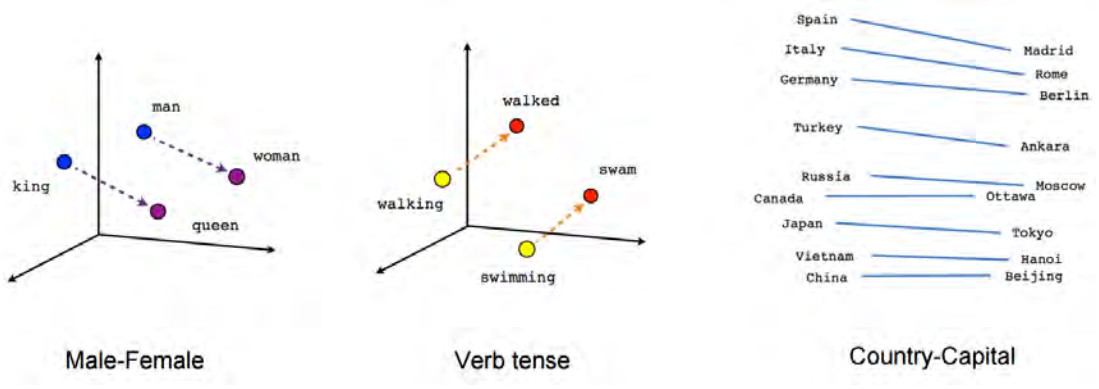
Review

- What is SVD?
- What are some applications of SVD?

Additional SVD Application ¶

An interesting use of SVD that I recently came across was as a step in the de-biasing of Word2Vec word embeddings, from Quantifying and Reducing Stereotypes in Word Embeddings(Bolukbasi, et al).

Word2Vec is a useful library released by Google that represents words as vectors. The similarity of vectors captures semantic meaning, and analogies can be found, such as Paris:France :: Tokyo: Japan.



(source: Vector Representations of Words)

However, these embeddings can implicitly encode bias, such as father:doctor :: mother:nurse and man:computer programmer :: woman:homemaker.

One approach for de-biasing the space involves using SVD to reduce the dimensionality (Bolukbasi paper).

You can read more about bias in word embeddings:

- How Vector Space Mathematics Reveals the Hidden Sexism in Language(MIT Tech Review)
- ConceptNet: better, less-stereotyped word vectors
- Semantics derived automatically from language corpora necessarily contain human biases (excellent and very interesting paper!)

Ways to think about SVD ¶

- Data compression
- SVD trades a large number of features for a smaller set of better features
- All matrices are diagonal (if you use change of bases on the domain and range)

Perspectives on SVD ¶

We usually talk about SVD in terms of matrices,

$$A = U\Sigma V^T$$

but we can also think about it in terms of vectors. SVD gives us sets of orthonormal vectors v_j and u_j such that

$$Av_j = \sigma_j u_j$$

σ_j are scalars, called singular values

Q: Does this remind you of anything?

Answer ¶

Relationship between SVD and Eigen Decomposition : the left-singular vectors of A are the eigenvectors of AA^T. The right-singular vectors of A are the eigenvectors of A^TA. The non-zero singular values of A are the square roots of the eigenvalues of A^TA (and AA^T).

SVD is a generalization of eigen decomposition. Not all matrices have eigen values, but ALL matrices have singular values.

Let's forget SVD for a bit and talk about how to find the eigenvalues of a symmetric positive definite matrix...

Further resources on SVD ¶

- SVD: image compression and least squares
- Image Compression with SVD
- The Extraordinary SVD
- A Singularly Valuable Decomposition: The SVD of a Matrix

Today: Eigen Decomposition ¶

The best classical methods for computing the SVD are variants on methods for computing eigenvalues. In addition to their links to SVD, Eigen decompositions are useful on their own as well. Here are a few practical applications of eigen decomposition:

- rapid matrix powers
- nth Fibonacci number
- Behavior of ODEs
- Markov Chains (health care economics, Page Rank)

- Linear Discriminant Analysis on Iris dataset

Check out the 3 Blue 1 Brown videos on Change of basis and Eigenvalues and eigenvectors

"Eigenvalues are a way to see into the heart of a matrix... All the difficulties of matrices are swept away" -Strang

Vocab : A Hermitian matrix is one that is equal to it's own conjugate transpose. In the case of real-valued matrices (which is all we are considering in this course), Hermitian means the same as Symmetric .

Relevant Theorems:

- If A is symmetric, then eigenvalues of A are real and $A = Q\Lambda Q^T$
- If A is triangular, then its eigenvalues are equal to its diagonal entries

DBpedia Dataset ¶

Let's start with the Power Method , which finds one eigenvector. What good is just one eigenvector? you may be wondering. This is actually the basis for PageRank (read The $25,000,000,000 Eigenvector: the Linear Algebra Behind Google for more info)

Instead of trying to rank the importance of all websites on the internet, we are going to use a dataset of Wikipedia links from DBpedia. DBpedia provides structured Wikipedia data available in 125 languages.

"The full DBpedia data set features 38 million labels and abstracts in 125 different languages, 25.2 million links to images and 29.8 million links to external web pages; 80.9 million links to Wikipedia categories, and 41.2 million links to YAGO categories" --about DBpedia

Today's lesson is inspired by this SciKit Learn Example

Imports ¶

In [361]:

```
import os, numpy as np, pickle
from bz2 import BZ2File
from datetime import datetime
from pprint import pprint
from time import time
from tqdm import tqdm_notebook
from scipy import sparse

from sklearn.decomposition import randomized_svd
from sklearn.externals.joblib import Memory
from urllib.request import urlopen
```

Downloading the data ¶

The data we have are:

- redirects : URLs that redirect to other URLs
- links : which pages link to which other pages

Note: this takes a while

In [367]:

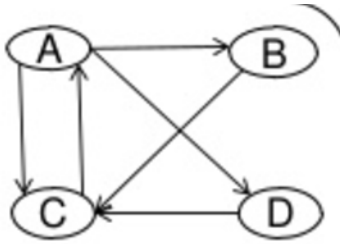
```
PATH = 'data/dbpedia/'
URL_BASE = 'http://downloads.dbpedia.org/3.5.1/en/'
filenames = ["redirects_en.nt.bz2", "page_links_en.nt.bz2"]

for filename in filenames:
    if not os.path.exists(PATH+filename):
        print("Downloading '%s', please wait..." % filename)
        open(PATH+filename, 'wb').write(urlopen(URL_BASE+filename).read())
```

In [368]:

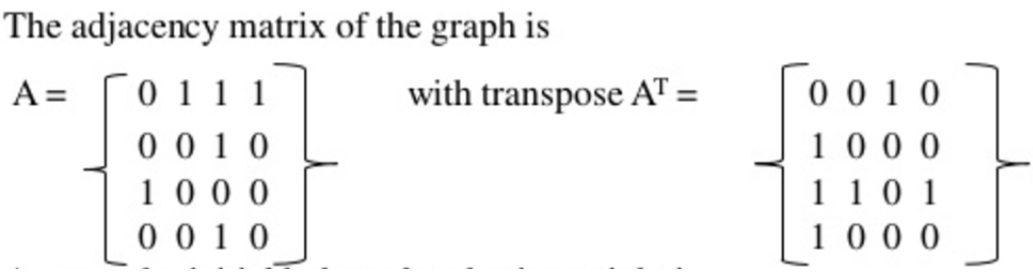
```
redirects_filename = PATH+filenames[0]
page_links_filename = PATH+filenames[1]
```

Graph Adjacency Matrix ¶

We will construct a graph adjacency matrix , of which pages point to which.



(source: PageRank and HyperLink Induced Topic Search)



(source: PageRank and HyperLink Induced Topic Search)

The power A^2 will give you how many ways there are to get from one page to another in 2 steps. You can see a more detailed example, as applied to airline travel, worked out in these notes.

We want to keep track of which pages point to which pages. We will store this in a square matrix, with a 1 in position (r, c) indicating that the topic in row r points to the topic in column c

You can read more about graphs here.

Data Format ¶

One line of the file looks like:

- `<http://dbpedia.org/resource/AfghanistanHistory> <http://dbpedia.org/property/redirect> <http://dbpedia.org/resource/History_of_Afghanistan> .`

In the below slice, the plus 1, -1 are to remove the <>

In [369]:

```
DBPEDIA_RESOURCE_PREFIX_LEN = len("http://dbpedia.org/resource/")
SLICE = slice(DBPEDIA_RESOURCE_PREFIX_LEN + 1, -1)
```

In [370]:

```
def get_lines(filename): return (line.split() for line in BZ2File(filename))
```

Loop through redirections and create dictionary of source to final destination

In [371]:

```
def get_redirect(targ, redirects):
    seen = set()
    while True:
        transitive_targ = targ
        targ = redirects.get(targ)
        if targ is None or targ in seen: break
        seen.add(targ)
    return transitive_targ
```

In [372]:

```
def get_redirects(redirects_filename):
    redirects={}
    lines = get_lines(redirects_filename)
    return {src[SLICE]:get_redirect(targ[SLICE], redirects)
                for src,_,targ,_ in tqdm_notebook(lines, leave=False)}
```

In [373]:

```
redirects = get_redirects(redirects_filename)
```

In [374]:

```
mem_usage()
```

Out[374]:

```
13.766303744
```

In [375]:

```
def add_item(lst, redirects, index_map, item):
    k = item[SLICE]
    lst.append(index_map.setdefault(redirects.get(k, k), len(index_map)))
```

In [376]:

```
limit=119077682 #5000000
```

In [377]:

```
# Computing the integer index map
index_map = dict() # links->IDs
lines = get_lines(page_links_filename)
source, destination, data = [],[],[]
for l, split in tqdm_notebook(enumerate(lines), total=limit):
    if l >= limit: break
    add_item(source, redirects, index_map, split[0])
    add_item(destination, redirects, index_map, split[2])
    data.append(1)
```

In [379]:

```
n=len(data); n
```

Out[379]:

```
119077682
```

Looking at our data ¶

The below steps are just to illustrate what info is in our data and how it is structured. They are not efficient.

Let's see what type of items are in index_map:

In [425]:

```
index_map.popitem()
```

Out[425]:

```
(b'1940_Cincinnati_Reds_Team_Issue', 9991173)
```

Let's look at one item in our index map:

1940_Cincinnati_Reds_Team_Issue has index 9991173. This only shows up once in the destination list:

In [427]:

```
[i for i,x in enumerate(source) if x == 9991173]
```

Out[427]:

```
[119077649]
```

In [428]:

```
source[119077649], destination[119077649]
```

Out[428]:

```
(9991173, 9991050)
```

Now, we want to check which page is the source (has index 9991050). Note: usually you should not access a dictionary by searching for its values. This is inefficient and not how dictionaries are intended to be used.

In [429]:

```
for page_name, index in index_map.items():
    if index == 9991050:
        print(page_name)
```

```
b'W711-2'
```

We can see on Wikipedia that the Cincinati Red Teams Issue has redirected to W711-2:

In [431]:

```
test_inds = [i for i,x in enumerate(source) if x == 9991050]
```

In [466]:

```
len(test_inds)
```

Out[466]:

```
47
```

In [433]:

```
test_inds[:5]
```

Out[433]:

```
[119076756, 119076757, 119076758, 119076759, 119076760]
```

In [434]:

```
test_dests = [destination[i] for i in test_inds]
```

Now, we want to check which page is the source (has index 9991174):

In [435]:

```
for page_name, index in index_map.items():
    if index in test_dests:
        print(page_name)
```

```
b'Baseball'
b'Ohio'
b'Cincinnati'
b'Flash_Thompson'
b'1940'
b'1938'
b'Lonny_Frey'
b'Cincinnati_Reds'
```

```
b'Ernie_Lombardi'
b'Baseball_card'
b'James_Wilson'
b'Trading_card'
b'Detroit_Tigers'
b'Baseball_culture'
b'Frank_McCormick'
b'Bucky_Walters'
b'1940_World_Series'
b'Billy_Werber'
b'Ival_Goodman'
b'Harry_Craft'
b'Paul_Derringer'
b'Johnny_Vander_Meer'
b'Cigarette_card'
b'Eddie_Joost'
b'Myron_McCormick'
b'Beckett_Media'
b'Icarus_affair'
b'Ephemera'
b'Sports_card'
b'James_Turner'
b'Jimmy_Ripple'
b'Lewis_Riggs'
b'The_American_Card_Catalog'
b'Rookie_card'
b'Willard_Hershberger'
b'Elmer_Riddle'
b'Joseph_Beggs'
b'Witt_Guise'
b'Milburn_Shoffner'
```

We can see that the items in the list appear in the wikipedia page:

W711-2 is also commonly known as the 1940 Cincinnati Reds Team Issue instead of The American Card Catalog reference number. This is a baseball card set of 35 total unnumbered, black and white cards measuring 2-1/8" × 2-5/8". The complete set was issued by the Cincinnati Reds baseball club, sold at the ballpark. 29 cards feature a player's portrait photo on the front while name, position and biographical information and five years worth of statistics are on the back. The balance of the 6 cards were reference cards describing the World Series win in 1940 against the Detroit Tigers and an order form in which to get more of these sets from Harry Hartman Publishing Co. in Cincinnati, Ohio.

This set had few known baseball stars on the team with the exception of Johnny Vandermeer who still holds the record for two consecutive no-hitter games in 1938, Eddie Joost and Ernie Lombardi. A complete list of players in this set are shown below.

Player/Card List [edit]

-Morrie Arnovich
-William (Bill) Baker
-Joe Beggs
-Harry Craft
-Paul Derringer
-Lonny Frey
-Ival Goodman
-Harry (Hank) Gowdy
-Witt Guise
-Harry (Socko) Hartman
-Willard Hershberger
-John Hutchings
-Eddie Joost
-Ernie Lombardi

(Source: Wikipedia)

Create Matrix ¶

Below we create a sparse matrix using Scipy's COO format, and that convert it to CSR.

Questions : What are COO and CSR? Why would we create it with COO and then convert it right away?

In [341]:

```
X = sparse.coo_matrix((data, (destination,source)), shape=(n,n), dtype=np.float32)
X = X.tocsr()
```

In [347]:

```
del(data,destination, source)
```

In [342]:

```
X
```

Out[342]:

```
<119077682x119077682 sparse matrix of type '<class 'numpy.float32'>'
        with 93985520 stored elements in Compressed Sparse Row format>
```

In [343]:

```
names = {i: name for name, i in index_map.items()}
```

In [349]:

```
mem_usage()
```

Out[349]:

```
12.903882752
```

Save matrix so we don't have to recompute ¶

In [350]:

```
pickle.dump(X, open(PATH+'X.pkl', 'wb'))
pickle.dump(index_map, open(PATH+'index_map.pkl', 'wb'))
```

In [316]:

```
X = pickle.load(open(PATH+'X.pkl', 'rb'))
index_map = pickle.load(open(PATH+'index_map.pkl', 'rb'))
```

In [317]:

```
names = {i: name for name, i in index_map.items()}
```

In [351]:

```
X
```

Out[351]:

```
<119077682x119077682 sparse matrix of type '<class 'numpy.float32'>'
        with 93985520 stored elements in Compressed Sparse Row format>
```

Power method ¶

Motivation ¶

An $n \times n$ matrix A is diagonalizable if it has n linearly independent eigenvectors $v_1, \ldots v_n$.

Then any w can be expressed $w = \sum_{j=1}^{n} c_j v_j$, for some scalars c_j.

Exercise: Show that

$$A^k w = \sum_{j=1}^{n} c_j \lambda_j^k v_j$$

Question : How will this behave for large k?

This is inspiration for the power method .

Code¶

In [281]:

```
def show_ex(v):
    print(', '.join(names[i].decode() for i in np.abs(v.squeeze()).argsort()[-1:-10:-1]))
```

In [453]:

```
?np.squeeze
```

How to normalize a sparse matrix:

In [336]:

```
S = sparse.csr_matrix(np.array([[1,2],[3,4]]))
Sr = S.sum(axis=0).A1; Sr
```

Out[336]:

```
array([4, 6], dtype=int64)
```

numpy.matrix.A1

In [337]:

```
S.indices
```

Out[337]:

```
array([0, 1, 0, 1], dtype=int32)
```

In [338]:

```
S.data
```

Out[338]:

```
array([1, 2, 3, 4], dtype=int64)
```

In [339]:

```
S.data / np.take(Sr, S.indices)
```

Out[339]:

```
array([ 0.25   ,  0.33333,  0.75   ,  0.66667])
```

In [328]:

```
def power_method(A, max_iter=100):
    n = A.shape[1]
    A = np.copy(A)
    A.data /= np.take(A.sum(axis=0).A1, A.indices)

    scores = np.ones(n, dtype=np.float32) * np.sqrt(A.sum()/(n*n)) # initial guess
    for i in range(max_iter):
        scores = A @ scores
        nrm = np.linalg.norm(scores)
        scores /= nrm
        print(nrm)

    return scores
```

Question : Why normalize the scores on each iteration?

In [345]:

```
scores = power_method(X, max_iter=10)
```

```
0.621209
0.856139
1.02793
1.02029
1.02645
1.02504
1.02364
1.02126
1.019
1.01679
```

In [346]:

```
show_ex(scores)
```

```
Living_people, Year_of_birth_missing_%28living_people%29, United_States, United_Kingdom, Race_and_ethnicity_in_the_United_States_Cen
```

In [327]:

```
mem_usage()
```

Out[327]:

```
11.692331008
```

Comments ¶

Many advanced eigenvalue algorithms that are used in practice are variations on the power method.

In Lesson 3: Background Removal, we used Facebook's fast randomized pca/svd library, fbpca. Check out the source code for the pca method we used. It uses the power method!

Further Study

- Check out Google Page Rank, Power Iteration and the Second EigenValue of the Google Matrix for animations of the distribution as it converges.
- The convergence rate of the power method is the ratio of the largest eigenvalue to the 2nd largest eigenvalue. It can be speeded up by adding shifts. To find eigenvalues other than the largest, a method called deflation can be used. See Chapter 12.1 of Greenbaum & Chartier for more details.
- Krylov Subspaces: In the Power Iteration, notice that we multiply by our matrix A each time, effectively calculating

  $$Ab,\ A^2b,\ A^3b,\ A^4b\ \ldots$$

  The matrix with those vectors as columns is called the Krylov matrix, and the space spanned by those vectors is the Krylov subspace. Keep this in mind for later.

Compare to SVD ¶

In [27]:

```
%time U, s, V = randomized_svd(X, 3, n_iter=3)
```

```
CPU times: user 8min 40s, sys: 1min 20s, total: 10min 1s
Wall time: 5min 56s
```

In [28]:

```
mem_usage()
```

Out[28]:

```
28.353073152
```

In [30]:

```
# Top wikipedia pages according to principal singular vectors
show_ex(U.T[0])
```

```
List_of_World_War_II_air_aces, List_of_animated_feature-length_films, List_of_animated_feature_films:2000s, International_Swimming_H
```

In [31]:

```
show_ex(U.T[1])
```

```
List_of_former_United_States_senators, List_of_United_States_Representatives_from_New_York, List_of_United_States_Representatives_fr
```

In [32]:

```
show_ex(V[0])
```

```
United_States, Japan, United_Kingdom, Germany, Race_and_ethnicity_in_the_United_States_Census, France, United_States_Army_Air_Forces
```

In [33]:

```
show_ex(V[1])
```

```
Democratic_Party_%28United_States%29, Republican_Party_%28United_States%29, Democratic-Republican_Party, United_States, Whig_Party_%
```

Exercise: Normalize the data in various ways. Don't overwrite the adjacency matrix, but instead create a new one. See how your results differ.

Eigen Decomposition vs SVD:

- SVD involves 2 bases, eigen decomposition involves 1 basis
- SVD bases are orthonormal, eigen basis generally not orthogonal
- All matrices have an SVD, not all matrices (not even all square) have an eigen decomposition

QR Algorithm ¶

We used the power method to find the eigenvector corresponding to the largest eigenvalue of our matrix of Wikipedia links. This eigenvector gave us the relative importance of each Wikipedia page (like a simplified PageRank).

Next, let's look at a method for finding all eigenvalues of a symmetric, positive definite matrix. This method includes 2 fundamental algorithms in numerical linear algebra, and is a basis for many more complex methods.

The Second Eigenvalue of the Google Matrix: has "implications for the convergence rate of the standard PageRank algorithm as the web scales, for the stability of PageRank to perturbations to the link structure of the web, for the detection of Google spammers, and for the design of algorithms to speed up PageRank".

Avoiding Confusion: QR Algorithm vs QR Decomposition ¶

The QR algorithm uses something called the QR decomposition . Both are important, so don't get them confused. The QR decomposition decomposes a matrix $A = QR$ into a set of orthonormal columns Q and a triangular matrix R. We will look at several ways to calculate the QR decomposition in a future lesson. For now, just know that it is giving us an orthogonal matrix and a triangular matrix.

Linear Algebra ¶

Two matrices A and B are similar if there exists a non-singular matrix X such that

$$B = X^{-1}AX$$

Watch this: Change of Basis

Theorem : If X is non-singular, then A and $X^{-1}AX$ have the same eigenvalues.

More Linear Algebra ¶

A Schur factorization of a matrix A is a factorization:

$$A = QTQ^*$$

where Q is unitary and T is upper-triangular.

Question : What can you say about the eigenvalues of A?

Theorem: Every square matrix has a Schur factorization.

Other resources ¶

Review: Linear combinations, span, and basis vectors

See Lecture 24 for proofs of above theorems (and more!)

Algorithm ¶

The most basic version of the QR algorithm:

```
for k=1,2,...
    Q, R = A
    A = R @ Q
```

Under suitable assumptions, this algorithm converges to the Schur form of A!

Why it works ¶

Written again, only with subscripts:

$A_0 = A$

for $k = 1, 2, \ldots$

Q_k, R_k = A_{k-1}

A_k = $R_k Q_k$

We can think of this as constructing sequences of A_k, Q_k, and R_k.

$$A_k = Q_k \, R_k$$
$$Q_k^{-1} \, A_k = R_k$$

Thus,

$$R_k Q_k = Q_k^{-1} \, A_k \, Q_k$$
$$A_k = Q_k^{-1} Q_2^{-1} Q_1^{-1} A Q_1 Q_2 \ldots Q_k$$

Trefethen proves the following on page 216-217:

$$A^k = Q_1 Q_2 \ldots Q_k R_k R_{k-1} \ldots R_1$$

Key: The QR algorithm constructs orthonormal bases for successive powers A^k. And remember the close relationship between powers of A and the eigen decomposition.

To learn more, read up on Rayleigh quotients.

Pure QR¶

In [467]:

```
n = 6
A = np.random.rand(n,n)
AT = A @ A.T
```

In [474]:

```
def pure_qr(A, max_iter=50000):
    Ak = np.copy(A)
    n = A.shape[0]
    QQ = np.eye(n)
    for k in range(max_iter):
        Q, R = np.linalg.qr(Ak)
        Ak = R @ Q
        QQ = QQ @ Q
        if k % 100 == 0:
            print(Ak)
            print("\n")
    return Ak, QQ
```

Pure QR¶

In [475]:

```
Ak, Q = pure_qr(A)
```

```
[[ 2.65646  0.21386  0.16765  0.75256  0.61251  0.93518]
 [ 0.52744  0.47579  0.17052 -0.41086 -0.21182 -0.01876]
 [ 0.29923  0.06964  0.11173  0.1879  -0.29101  0.60032]
 [ 0.2274   0.46162 -0.26654  0.08899  0.24631  0.26447]
 [-0.06093  0.02892  0.34162  0.07533  0.02393 -0.05456]
 [-0.06025  0.02694 -0.11675 -0.00927 -0.11939 -0.00767]]


[[ 2.78023  0.52642  0.0395  -0.11135  0.1569   1.15184]
 [ 0.       0.18624 -0.297   -0.07256 -0.04537  0.27907]
 [ 0.       0.69328  0.34105 -0.12198  0.11029  0.0992 ]
 [-0.      -0.0494  -0.02057  0.09461  0.59466  0.09115]
 [-0.       0.00008 -0.02659 -0.40372  0.06542  0.38612]
 [-0.       0.       0.       0.      -0.      -0.11832]]


[[ 2.78023 -0.12185 -0.51401  0.17625 -0.07467  1.15184]
 [ 0.       0.2117  -0.70351  0.09974 -0.02986  0.00172]
 [ 0.       0.28284  0.32635 -0.0847  -0.08488 -0.29104]
 [-0.      -0.00068 -0.00088 -0.01282  0.54836  0.13447]
 [-0.       0.      -0.00102 -0.45718  0.16208 -0.37726]
 [-0.       0.       0.       0.      -0.      -0.11832]]


[[ 2.78023 -0.33997  0.4049   0.17949  0.06291  1.15184]
 [ 0.       0.48719 -0.48788 -0.05831 -0.12286 -0.23486]
 [ 0.       0.49874  0.05104 -0.07191  0.03638  0.17261]
 [ 0.       0.00002  0.       0.02128  0.41958  0.3531 ]
 [ 0.      -0.       0.00002 -0.58571  0.1278  -0.18838]
 [ 0.       0.       0.      -0.       0.      -0.11832]]


[[ 2.78023  0.35761  0.38941  0.07462  0.17493  1.15184]
 [ 0.       0.0597  -0.55441 -0.0681  -0.04456  0.14084]
 [ 0.       0.43221  0.47853 -0.06068  0.12117  0.25519]
 [-0.      -0.      -0.       0.16206  0.45708  0.37724]
 [-0.       0.      -0.      -0.54821 -0.01298  0.1336 ]
 [ 0.       0.       0.       0.      -0.      -0.11832]]


[[ 2.78023  0.06853 -0.52424  0.05224 -0.18287  1.15184]
 [ 0.       0.36572 -0.6889   0.07864 -0.09263  0.105  ]
 [ 0.       0.29772  0.17251 -0.09836 -0.02347 -0.27191]
 [ 0.      -0.      -0.       0.13719  0.57888 -0.20884]
 [ 0.       0.      -0.      -0.42642  0.01189 -0.34139]
 [ 0.       0.       0.       0.      -0.      -0.11832]]


[[ 2.78023 -0.52782  0.03045 -0.14389 -0.12436  1.15184]
 [ 0.       0.25091 -0.27593  0.08994 -0.06581 -0.28672]
 [ 0.       0.7107   0.28732  0.10154  0.04751 -0.05245]
 [ 0.      -0.      -0.       0.0297  -0.59054 -0.01234]
 [ 0.      -0.       0.       0.41475  0.11938 -0.40001]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.1533   0.50599  0.18983  0.01158  1.15184]
 [ 0.       0.18627 -0.69511 -0.0991  -0.00189  0.01621]
 [ 0.       0.29151  0.35196  0.05638 -0.10949  0.29102]
 [ 0.      -0.      -0.      -0.02207 -0.48261  0.25246]
 [ 0.      -0.       0.       0.52268  0.17116  0.31053]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.29683 -0.43751 -0.13852  0.13032  1.15184]
 [ 0.       0.48375 -0.53231 -0.01164  0.13482  0.216  ]
 [ 0.       0.45431  0.05448 -0.07972 -0.01795 -0.19571]
 [ 0.      -0.       0.       0.10042 -0.40743 -0.39915]
 [ 0.      -0.       0.       0.59786  0.04867 -0.02893]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.39373 -0.35284  0.00838 -0.19     1.15184]
 [ 0.       0.05184 -0.51278  0.05752 -0.0564  -0.16496]
 [ 0.       0.47384  0.48639  0.09426  0.09806 -0.24031]
 [ 0.      -0.      -0.       0.17043 -0.52593  0.30622]
 [ 0.      -0.       0.       0.47936 -0.02134 -0.25766]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.0355   0.52751  0.10215  0.16042  1.15184]
 [ 0.       0.34047 -0.69946 -0.09606 -0.06944 -0.08773]
 [ 0.       0.28717  0.19776  0.09382 -0.04624  0.27796]
 [ 0.      -0.      -0.       0.08542 -0.60072 -0.10298]
 [ 0.      -0.       0.       0.40458  0.06367  0.38672]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51706 -0.11032 -0.17442 -0.07581  1.15184]
 [ 0.       0.31706 -0.2805  -0.08235  0.0863   0.29136]
 [ 0.       0.70612  0.22117 -0.09973 -0.02853  0.00826]
 [ 0.      -0.      -0.      -0.01235 -0.5494  -0.13103]
 [ 0.      -0.       0.       0.45589  0.16144 -0.37814]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18529 -0.49517 -0.18006  0.06123  1.15184]
 [ 0.       0.16123 -0.68291 -0.09155  0.0279  -0.03476]
 [ 0.       0.30371  0.377    0.01655 -0.12472 -0.2894 ]
 [ 0.       0.       0.       0.01974 -0.42059 -0.35131]
 [ 0.       0.      -0.       0.5847   0.12935 -0.19168]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25477  0.46327 -0.07615  0.17427  1.15184]
 [ 0.       0.47278 -0.57146 -0.03461 -0.12937 -0.19683]
 [ 0.       0.41516  0.06545  0.08395 -0.00278  0.21498]
 [ 0.      -0.       0.       0.16125 -0.45555 -0.3784 ]
 [ 0.      -0.       0.       0.54974 -0.01216  0.13028]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42908  0.30889  0.05099  0.18322  1.15184]
 [ 0.       0.05265 -0.46631  0.04976 -0.06424  0.18962]
 [ 0.       0.52032  0.48558  0.11539  0.07122  0.22136]
 [ 0.       0.       0.       0.13823 -0.57801 -0.21116]
 [ 0.       0.      -0.       0.42728  0.01086  0.33995]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00334 -0.52869 -0.14306 -0.12531  1.15184]
 [ 0.       0.3149  -0.7066  -0.10679 -0.04392  0.07066]
 [ 0.       0.28002  0.22333  0.08372 -0.06816 -0.28278]
 [ 0.       0.       0.       0.03087 -0.59113 -0.00969]
 [ 0.       0.      -0.       0.41416  0.11821 -0.40008]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49486  0.18611  0.18973  0.01317  1.15184]
 [ 0.       0.37763 -0.30407 -0.06365  0.1086  -0.28934]
 [ 0.       0.68255  0.1606  -0.09511 -0.00985  0.03524]
 [ 0.       0.       0.      -0.02239 -0.48423  0.24985]
 [ 0.       0.      -0.       0.52107  0.17148  0.31262]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21799  0.48167 -0.13976  0.12898  1.15184]
 [ 0.       0.13689 -0.66682  0.07458 -0.05442  0.05403]
 [ 0.       0.31981  0.40134  0.03053  0.12464  0.28642]
 [ 0.       0.       0.       0.09859 -0.40695 -0.39886]
 [ 0.       0.      -0.       0.59834  0.0505  -0.03275]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21427 -0.48333 -0.00984  0.18993  1.15184]
 [ 0.       0.45651 -0.60498  0.07108  0.11122  0.17774]
 [ 0.       0.38164  0.08172 -0.08395  0.02278 -0.23101]
```

```
 [ 0.      -0.       0.       0.17077 -0.52445 -0.30819]
 [ 0.      -0.       0.       0.48084 -0.02169  0.2553 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.46223 -0.25664  0.10111  0.16108  1.15184]
 [ 0.       0.06482 -0.41681 -0.04447  0.07111 -0.21419]
 [ 0.       0.56981  0.47341 -0.12657 -0.04398 -0.19769]
 [ 0.       0.       0.       0.08669 -0.60057 -0.10548]
 [ 0.       0.      -0.       0.40472  0.0624   0.38605]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02821  0.52795  0.17386  0.07707  1.15184]
 [ 0.       0.28917 -0.71053 -0.11126 -0.01551 -0.05366]
 [ 0.       0.27609  0.24906  0.06641 -0.08922  0.28649]
 [ 0.      -0.      -0.      -0.01166 -0.55066  0.12827]
 [ 0.      -0.       0.       0.45463  0.16075  0.37909]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46334 -0.25463  0.18062 -0.05954  1.15184]
 [ 0.       0.42654 -0.3423   0.03075 -0.12716  0.28138]
 [ 0.       0.64432  0.11169  0.0883  -0.00883 -0.07605]
 [ 0.       0.       0.       0.01821 -0.42163  0.3495 ]
 [ 0.       0.      -0.       0.58366  0.13087  0.19495]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25153 -0.46503  0.07768 -0.17359  1.15184]
 [ 0.       0.11372 -0.64641  0.05247 -0.07193 -0.07417]
 [ 0.       0.34021  0.42451  0.0736   0.10791 -0.28188]
 [ 0.      -0.      -0.       0.16041 -0.45403  0.37953]
 [ 0.      -0.       0.       0.55126 -0.01132 -0.12694]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17551  0.49872 -0.04973 -0.18356  1.15184]
 [ 0.       0.43663 -0.63305  0.0961   0.08709 -0.15898]
 [ 0.       0.35358  0.1016  -0.08053  0.04103  0.2443 ]
 [ 0.       0.      -0.       0.13926 -0.57713  0.21349]
 [ 0.       0.      -0.       0.42816  0.00983 -0.3385 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.49117  0.19565 -0.14223 -0.12625  1.15184]
 [ 0.       0.0908  -0.36765 -0.03961  0.07922  0.23763]
 [ 0.       0.61897  0.44743 -0.12984 -0.01694  0.16879]
 [ 0.      -0.      -0.       0.03205 -0.5917  -0.00704]
 [ 0.      -0.       0.       0.41359  0.11704 -0.40014]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05938 -0.52535 -0.18961 -0.01475  1.15184]
 [ 0.       0.26338 -0.71138 -0.10793  0.01612  0.03662]
 [ 0.       0.27524  0.27485  0.03935 -0.10739 -0.28917]
 [ 0.       0.       0.      -0.02269 -0.48584 -0.24724]
 [ 0.       0.      -0.       0.51945  0.17177 -0.31469]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42532  0.31405 -0.14099  0.12764  1.15184]
 [ 0.       0.46081 -0.3892  -0.01295 -0.1334  -0.26875]
 [ 0.       0.59743  0.07742  0.07957 -0.02634  0.11283]
 [ 0.      -0.      -0.       0.09675 -0.40651 -0.39852]
 [ 0.      -0.       0.       0.59879  0.05233 -0.03657]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28599  0.44467  0.0113  -0.18985  1.15184]
 [ 0.       0.0924  -0.62121 -0.03168  0.07992  0.09528]
 [ 0.       0.36541  0.44583 -0.1049  -0.0812   0.27546]
 [ 0.      -0.      -0.       0.1711  -0.52297  0.31015]
 [ 0.      -0.       0.       0.48232 -0.02201 -0.25291]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.13845 -0.51025 -0.10006 -0.16173  1.15184]
 [ 0.       0.41438 -0.65605 -0.11185 -0.06046  0.14063]
 [ 0.       0.33057  0.12385  0.07387 -0.05803 -0.2553 ]
 [ 0.       0.       0.       0.08796 -0.6004   0.10798]
 [ 0.       0.      -0.       0.40489  0.06113 -0.38536]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.5134  -0.12629 -0.1733  -0.07834  1.15184]
 [ 0.       0.13174 -0.32385  0.0319  -0.08965 -0.25857]
 [ 0.       0.66277  0.40649  0.12582 -0.01014 -0.13453]
 [ 0.      -0.      -0.      -0.01095 -0.55191 -0.12551]
 [ 0.      -0.       0.       0.45338  0.16004 -0.38001]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.09043  0.52091 -0.18117  0.05785  1.15184]
 [ 0.       0.2376  -0.70917  0.09396 -0.04876 -0.01941]
 [ 0.       0.27745  0.30063 -0.00151  0.11739  0.29083]
 [ 0.       0.       0.       0.01671 -0.4227  -0.34767]
 [ 0.       0.      -0.       0.58259  0.13238 -0.19821]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38356 -0.36387  0.07921 -0.1729   1.15184]
 [ 0.       0.4804  -0.43903 -0.05633 -0.12341  0.25287]
 [ 0.       0.54759  0.05783  0.07018 -0.04075 -0.14496]
 [ 0.       0.      -0.       0.15953 -0.45252  0.38064]
 [ 0.       0.      -0.       0.55277 -0.01045 -0.12358]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.32133 -0.41985  0.04847  0.1839   1.15184]
 [ 0.       0.07395 -0.59076 -0.01536  0.0819  -0.11744]
 [ 0.       0.39586  0.46428 -0.12416 -0.05129 -0.26677]
 [ 0.       0.       0.       0.14027 -0.57624 -0.21581]
 [ 0.       0.      -0.       0.42905  0.00881  0.33703]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10295  0.51858 -0.14139 -0.12719  1.15184]
 [ 0.       0.39062 -0.67448  0.12025  0.03188 -0.1227 ]
 [ 0.       0.31214  0.14761 -0.06308  0.07439  0.26439]
 [ 0.       0.       0.       0.03323 -0.59225 -0.00439]
 [ 0.       0.      -0.       0.41304  0.11586 -0.40018]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52631  0.05017 -0.18948 -0.01632  1.15184]
 [ 0.       0.18641 -0.29145 -0.0171   0.10162  0.27541]
 [ 0.       0.69517  0.35182 -0.11404  0.03696  0.09543]
 [ 0.      -0.      -0.      -0.02295 -0.48746 -0.24462]
 [ 0.      -0.       0.       0.51783  0.17204 -0.31674]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.12158 -0.51453 -0.14221  0.12628  1.15184]
 [ 0.       0.21193 -0.70383 -0.06881  0.076    0.0019 ]
 [ 0.       0.2828   0.3263  -0.04211 -0.11272 -0.29147]
 [ 0.       0.       0.       0.09491 -0.4061  -0.39816]
 [ 0.       0.      -0.       0.59919  0.05418 -0.04039]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34032  0.4046   0.01277 -0.18975  1.15184]
 [ 0.       0.48718 -0.4875   0.09008  0.10189 -0.23501]
 [ 0.       0.49912  0.05105 -0.06166  0.05188  0.17241]
 [ 0.       0.       0.       0.1714  -0.52147  0.3121 ]
 [ 0.       0.      -0.       0.48382 -0.02232 -0.25051]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35731  0.38968  0.09901  0.16238  1.15184]
 [ 0.       0.0598  -0.55473  0.0032  -0.08133  0.14065]
 [ 0.       0.43189  0.47843  0.13388  0.02096  0.2553 ]
```

```
 [ 0.       0.       0.       0.08922 -0.60022 -0.11048]
 [ 0.       0.      -0.       0.40507  0.05986  0.38465]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.06881 -0.5242  -0.17273 -0.07959  1.15184]
 [ 0.       0.36593 -0.6888  -0.12153 -0.0007   0.10514]
 [ 0.       0.29783  0.1723   0.04626 -0.08989 -0.27185]
 [ 0.       0.       0.      -0.01023 -0.55314 -0.12275]
 [ 0.       0.      -0.       0.45215  0.15932 -0.38091]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.52786  0.02978  0.1817  -0.05616  1.15184]
 [ 0.       0.25036 -0.27597  0.00829  0.11107 -0.28665]
 [ 0.       0.71065  0.28787 -0.09408  0.06109 -0.05281]
 [ 0.       0.       0.       0.01523 -0.42379  0.34581]
 [ 0.       0.      -0.       0.5815   0.13386  0.20144]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.15304  0.50607 -0.08074  0.17219  1.15184]
 [ 0.       0.18648 -0.6952   0.03825 -0.09147  0.01606]
 [ 0.       0.29142  0.35175  0.08101  0.09272  0.29103]
 [ 0.       0.       0.       0.15863 -0.45102 -0.38172]
 [ 0.       0.      -0.       0.55427 -0.00955  0.1202 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.29718 -0.43727 -0.04721 -0.18423  1.15184]
 [ 0.       0.48381 -0.53197 -0.11247 -0.07526  0.21616]
 [ 0.       0.45466  0.05442  0.05413 -0.06119 -0.19553]
 [ 0.       0.      -0.       0.14128 -0.57533  0.21812]
 [ 0.       0.      -0.       0.42996  0.00781 -0.33554]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39344 -0.35318  0.14055  0.12812  1.15184]
 [ 0.       0.05187 -0.51315  0.00659  0.08029 -0.16476]
 [ 0.       0.47347  0.48636 -0.13567  0.00965 -0.24044]
 [ 0.       0.       0.       0.03442 -0.59279  0.00175]
 [ 0.       0.      -0.       0.4125   0.11467  0.4002 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03577  0.52749  0.18934  0.01789  1.15184]
 [ 0.       0.34068 -0.69939 -0.1138   0.03323 -0.08788]
 [ 0.       0.28724  0.19755  0.02106 -0.10242  0.27791]
 [ 0.      -0.      -0.      -0.02319 -0.48908  0.24199]
 [ 0.      -0.       0.       0.51622  0.17228  0.31875]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.5172  -0.10967 -0.14341  0.12491  1.15184]
 [ 0.       0.31653 -0.28038  0.04262  0.11135  0.29135]
 [ 0.       0.70624  0.2217  -0.06857  0.07793  0.00862]
 [ 0.      -0.      -0.       0.09305 -0.40572 -0.39775]
 [ 0.      -0.       0.       0.59957  0.05604 -0.04421]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18503 -0.49527 -0.01424  0.18965  1.15184]
 [ 0.       0.16143 -0.68303 -0.01015  0.09519 -0.0346 ]
 [ 0.       0.3036   0.3768  -0.10826 -0.06406 -0.28941]
 [ 0.       0.       0.       0.17168 -0.51997 -0.31403]
 [ 0.       0.      -0.       0.48532 -0.0226   0.24808]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25511  0.46308  0.09796  0.16302  1.15184]
 [ 0.       0.4729  -0.57116 -0.12554 -0.04666 -0.19698]
 [ 0.       0.41546  0.06533  0.04626 -0.07008  0.21484]
 [ 0.      -0.      -0.       0.09049 -0.60002 -0.11297]
 [ 0.      -0.       0.       0.40527  0.0586   0.38392]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.4288   0.30928 -0.17215 -0.08084  1.15184]
 [ 0.       0.0526  -0.46671  0.01637  0.07958  0.18942]
 [ 0.       0.51992  0.48563 -0.12923  0.0411   0.22154]
 [ 0.      -0.      -0.      -0.00949 -0.55436 -0.12   ]
 [ 0.      -0.       0.       0.45093  0.15858 -0.38179]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.0036  -0.52869  0.18221 -0.05447  1.15184]
 [ 0.       0.31511 -0.70655  0.094   -0.06711  0.0708 ]
 [ 0.       0.28007  0.22312  0.01322  0.10712 -0.28275]
 [ 0.      -0.      -0.       0.01378 -0.42491  0.34392]
 [ 0.      -0.       0.       0.58038  0.13531  0.20465]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49508  0.18551 -0.08227  0.17147  1.15184]
 [ 0.       0.37717 -0.30381 -0.07749 -0.09914 -0.28938]
 [ 0.       0.68281  0.16106  0.04417 -0.08488  0.03489]
 [ 0.      -0.      -0.       0.1577  -0.44953 -0.38277]
 [ 0.      -0.       0.       0.55576 -0.00862  0.11681]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21772  0.48179  0.04594  0.18455  1.15184]
 [ 0.       0.13709 -0.66697 -0.01194 -0.09157  0.05387]
 [ 0.       0.31966  0.40114  0.12396  0.03308  0.28645]
 [ 0.       0.       0.       0.14227 -0.5744  -0.22042]
 [ 0.       0.      -0.       0.43089  0.00681  0.33402]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.2146  -0.48319 -0.1397  -0.12905  1.15184]
 [ 0.       0.45666 -0.60473 -0.13098 -0.01645  0.1779 ]
 [ 0.       0.38189  0.08157  0.0361  -0.07911 -0.23089]
 [ 0.       0.       0.       0.03561 -0.59331  0.0009 ]
 [ 0.       0.      -0.       0.41198  0.11347 -0.4002 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.46197 -0.25711 -0.18919 -0.01945  1.15184]
 [ 0.       0.06467 -0.41722 -0.02879 -0.07874 -0.21399]
 [ 0.       0.5694   0.47356  0.11253 -0.07277 -0.19791]
 [ 0.      -0.      -0.      -0.0234  -0.49069 -0.23935]
 [ 0.      -0.       0.       0.5146   0.17249 -0.32073]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02795  0.52796  0.1446  -0.12353  1.15184]
 [ 0.       0.28938 -0.71051 -0.06202  0.0937  -0.0538 ]
 [ 0.       0.27611  0.24885 -0.05182 -0.09838  0.28647]
 [ 0.      -0.      -0.       0.09118 -0.40539  0.39731]
 [ 0.      -0.       0.       0.5999   0.0579   0.04803]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46363 -0.25411  0.01571 -0.18953  1.15184]
 [ 0.       0.4262  -0.34194 -0.10508 -0.07786  0.28147]
 [ 0.       0.64468  0.11203  0.02623 -0.08483 -0.07573]
 [ 0.       0.      -0.       0.17194 -0.51845  0.31595]
 [ 0.       0.      -0.       0.48684 -0.02285 -0.24563]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25125 -0.46518 -0.0969  -0.16365  1.15184]
 [ 0.       0.1139  -0.6466  -0.02852 -0.08437 -0.074  ]
 [ 0.       0.34003  0.42433  0.13059  0.00215 -0.28193]
 [ 0.      -0.      -0.       0.09175 -0.59981  0.11546]
 [ 0.      -0.       0.       0.40549  0.05734 -0.38318]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17582  0.49861 -0.17156 -0.08208  1.15184]
 [ 0.       0.43681 -0.63284  0.12872 -0.01606 -0.15914]
 [ 0.       0.35379  0.10142 -0.02137  0.08779  0.2442 ]
```

```
 [ 0.       0.       0.      -0.00874 -0.55557 -0.11724]
 [ 0.       0.      -0.       0.44972  0.15782 -0.38264]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.49095  0.19619  0.18271 -0.05278  1.15184]
 [ 0.       0.09052 -0.36805 -0.04589 -0.07569  0.23745]
 [ 0.       0.61858  0.44771  0.08364 -0.10079  0.16905]
 [ 0.       0.       0.       0.01234 -0.42605  0.342  ]
 [ 0.       0.      -0.       0.57924  0.13674  0.20783]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05913 -0.52538 -0.08379  0.17073  1.15184]
 [ 0.       0.26359 -0.71139 -0.02488  0.10628  0.03676]
 [ 0.       0.27524  0.27464 -0.08525 -0.0762  -0.28915]
 [ 0.       0.       0.       0.15674 -0.44806 -0.3838 ]
 [ 0.       0.      -0.       0.55723 -0.00766  0.11339]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42565  0.3136   0.04466  0.18486  1.15184]
 [ 0.       0.46059 -0.38879 -0.12336 -0.05234 -0.26887]
 [ 0.       0.59783  0.07764  0.0148  -0.08254  0.11255]
 [ 0.      -0.       0.       0.14326 -0.57346 -0.22273]
 [ 0.      -0.       0.       0.43183  0.00583  0.33249]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28571  0.44486  0.13885  0.12997  1.15184]
 [ 0.       0.09256 -0.62144 -0.04133 -0.07541  0.0951 ]
 [ 0.       0.36518  0.44567  0.12946 -0.02889  0.27552]
 [ 0.       0.       0.       0.03681 -0.59382 -0.00353]
 [ 0.       0.      -0.       0.41147  0.11228  0.40018]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.13875 -0.51017  0.18902  0.021    1.15184]
 [ 0.       0.41457 -0.65588  0.11658 -0.0508   0.14078]
 [ 0.       0.33074  0.12366  0.00017  0.09391 -0.25522]
 [ 0.      -0.      -0.      -0.02358 -0.4923   0.2367 ]
 [ 0.      -0.       0.       0.51299  0.17267  0.32269]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51325 -0.12689 -0.14578  0.12213  1.15184]
 [ 0.       0.13134 -0.32418 -0.06697 -0.06751 -0.25841]
 [ 0.       0.66245  0.40688  0.04539 -0.11784 -0.13483]
 [ 0.      -0.      -0.       0.08931 -0.40509 -0.39683]
 [ 0.      -0.       0.       0.60021  0.05977 -0.05184]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.09017  0.52095  0.01719 -0.1894   1.15184]
 [ 0.       0.23781 -0.7092   0.00841  0.10555 -0.01955]
 [ 0.       0.27742  0.30042 -0.10755 -0.04702  0.29082]
 [ 0.      -0.      -0.       0.17218 -0.51693  0.31786]
 [ 0.      -0.       0.       0.48836 -0.02309 -0.24315]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38392 -0.3635   0.09583  0.16427  1.15184]
 [ 0.       0.48029 -0.43862  0.13335  0.02486  0.25301]
 [ 0.       0.548    0.05794 -0.00692  0.08087 -0.14472]
 [ 0.      -0.       0.       0.09301 -0.59957 -0.11794]
 [ 0.      -0.       0.       0.40572  0.05608  0.38242]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.32103 -0.42007  0.17097  0.08331  1.15184]
 [ 0.       0.07408 -0.59104  0.05211  0.06505 -0.11726]
 [ 0.       0.39559  0.46415 -0.11986  0.06063 -0.26685]
 [ 0.       0.       0.      -0.00797 -0.55676  0.11449]
 [ 0.       0.      -0.       0.44853  0.15705  0.38347]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.10324  0.51852  0.18319 -0.05108  1.15184]
 [ 0.       0.39082 -0.67435 -0.0914   0.08443 -0.12285]
 [ 0.       0.31227  0.14741 -0.02899 -0.0931   0.26432]
 [ 0.      -0.      -0.       0.01093 -0.42721  0.34006]
 [ 0.      -0.       0.       0.57808  0.13815  0.21099]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52625  0.05082 -0.08532  0.16997  1.15184]
 [ 0.       0.18591 -0.29166  0.08808  0.05336  0.27529]
 [ 0.       0.69497  0.35232 -0.00778  0.11968  0.09577]
 [ 0.      -0.      -0.       0.15575 -0.4466  -0.3848 ]
 [ 0.      -0.       0.       0.55869 -0.00667  0.10996]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.12132 -0.51459  0.04338  0.18517  1.15184]
 [ 0.       0.21214 -0.70388  0.03415  0.0967   0.00204]
 [ 0.       0.28274  0.32609 -0.11915 -0.01656 -0.29147]
 [ 0.       0.       0.       0.14423 -0.5725  -0.22502]
 [ 0.       0.      -0.       0.43279  0.00486  0.33095]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34068  0.4043  -0.13799 -0.13088  1.15184]
 [ 0.       0.48717 -0.48711  0.13592 -0.00445 -0.23517]
 [ 0.       0.49951  0.05106  0.00074  0.08058  0.1722 ]
 [ 0.       0.       0.       0.03801 -0.59431  0.00617]
 [ 0.       0.      -0.       0.41098  0.11107 -0.40015]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35701  0.38996 -0.18884 -0.02255  1.15184]
 [ 0.       0.05989 -0.55505  0.06203  0.05273  0.14045]
 [ 0.       0.43157  0.47834 -0.09919  0.09232  0.2554 ]
 [ 0.      -0.      -0.      -0.02374 -0.49391 -0.23405]
 [ 0.      -0.       0.       0.51138  0.17283 -0.32462]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.06908 -0.52417  0.14695 -0.12073  1.15184]
 [ 0.       0.36613 -0.6887   0.0536  -0.1091   0.10529]
 [ 0.       0.29793  0.1721   0.06074  0.08077 -0.27179]
 [ 0.      -0.      -0.       0.08744 -0.40482  0.39631]
 [ 0.      -0.       0.       0.60047  0.06165  0.05565]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.5279   0.02911 -0.01868  0.18926  1.15184]
 [ 0.       0.24981 -0.27602 -0.10557 -0.03528 -0.28658]
 [ 0.       0.7106   0.28842 -0.01997 -0.11045 -0.05317]
 [ 0.      -0.      -0.       0.17239 -0.5154  -0.31976]
 [ 0.      -0.       0.       0.48989 -0.0233   0.24066]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.15278  0.50615 -0.09476 -0.16489  1.15184]
 [ 0.       0.18668 -0.69528  0.05306  0.08378  0.01591]
 [ 0.       0.29134  0.35154 -0.12239  0.01324  0.29104]
 [ 0.      -0.      -0.       0.09427 -0.59933  0.12043]
 [ 0.      -0.       0.       0.40597  0.05482 -0.38165]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.29754 -0.43703 -0.17036 -0.08453  1.15184]
 [ 0.       0.48387 -0.53162 -0.13043  0.03612  0.21632]
 [ 0.       0.455    0.05436 -0.01111 -0.08092 -0.19536]
 [ 0.       0.       0.      -0.00718 -0.55794 -0.11174]
 [ 0.       0.      -0.       0.44735  0.15627 -0.38428]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39314 -0.35351 -0.18366  0.04939  1.15184]
 [ 0.       0.05191 -0.51351 -0.07124 -0.0376  -0.16455]
 [ 0.       0.47311  0.48632  0.0649  -0.11953 -0.24058]
```

```
 [ 0.      -0.      -0.       0.00955 -0.4284  -0.3381 ]
 [ 0.      -0.       0.       0.57689  0.13954 -0.21412]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03604  0.52747  0.08683 -0.1692   1.15184]
 [ 0.       0.34089 -0.69931 -0.01131  0.11804 -0.08802]
 [ 0.       0.28731  0.19734 -0.08731 -0.05748  0.27787]
 [ 0.      -0.      -0.       0.15474 -0.44516  0.38577]
 [ 0.      -0.       0.       0.56014 -0.00565 -0.10651]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51734 -0.10903 -0.04209 -0.18547  1.15184]
 [ 0.       0.316   -0.28026 -0.11821 -0.01508  0.29134]
 [ 0.       0.70636  0.22223 -0.03631 -0.09732  0.00898]
 [ 0.       0.       0.       0.14519 -0.57153  0.22731]
 [ 0.       0.      -0.       0.43376  0.00389 -0.32938]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18476 -0.49537  0.13712  0.13178  1.15184]
 [ 0.       0.16164 -0.68314  0.06698  0.06844 -0.03445]
 [ 0.       0.30348  0.37659 -0.11831  0.04269 -0.28943]
 [ 0.       0.       0.       0.03922 -0.59478 -0.0088 ]
 [ 0.       0.      -0.       0.41051  0.10986  0.4001 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25545  0.46289 -0.18865 -0.0241   1.15184]
 [ 0.       0.47301 -0.57086  0.11438 -0.0697  -0.19714]
 [ 0.       0.41576  0.06522  0.02643  0.07968  0.21469]
 [ 0.       0.       0.      -0.02387 -0.49552 -0.23139]
 [ 0.       0.      -0.       0.50977  0.17296 -0.32652]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42851  0.30968  0.1481  -0.11931  1.15184]
 [ 0.       0.05255 -0.46711 -0.07872 -0.02004  0.18921]
 [ 0.       0.51952  0.48568  0.01962 -0.13418  0.22171]
 [ 0.       0.       0.       0.08555 -0.40459  0.39576]
 [ 0.       0.      -0.       0.6007   0.06353  0.05946]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00387 -0.52869 -0.02016  0.18911  1.15184]
 [ 0.       0.31532 -0.70651  0.02563  0.11264  0.07094]
 [ 0.       0.28012  0.22291 -0.1039  -0.02914 -0.28271]
 [ 0.       0.       0.       0.17258 -0.51386 -0.32164]
 [ 0.       0.      -0.       0.49143 -0.02349  0.23814]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49531  0.18491 -0.09369 -0.1655   1.15184]
 [ 0.       0.37671 -0.30355  0.12559 -0.00703 -0.28942]
 [ 0.       0.68307  0.16152  0.04456  0.08475  0.03454]
 [ 0.       0.       0.       0.09552 -0.59906  0.1229 ]
 [ 0.       0.      -0.       0.40623  0.05356 -0.38086]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21744  0.48192 -0.16976 -0.08575  1.15184]
 [ 0.       0.13729 -0.66712  0.0772   0.05072  0.05371]
 [ 0.       0.31951  0.40094 -0.10597  0.07229  0.28648]
 [ 0.      -0.      -0.      -0.00638 -0.5591  -0.10898]
 [ 0.      -0.       0.       0.44619  0.15547 -0.38507]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21493 -0.48304  0.18411 -0.04769  1.15184]
 [ 0.       0.45681 -0.60448  0.08465 -0.10131  0.17806]
 [ 0.       0.38215  0.08142  0.04707  0.07309 -0.23077]
 [ 0.      -0.      -0.       0.00819 -0.42961  0.33611]
 [ 0.      -0.       0.       0.57568  0.1409   0.21723]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.46171 -0.25757  0.08835 -0.16841  1.15184]
 [ 0.       0.06452 -0.41764  0.08378  0.0024  -0.21379]
 [ 0.       0.56899  0.47371  0.02583  0.13152 -0.19812]
 [ 0.       0.       0.       0.15369 -0.44373  0.38671]
 [ 0.       0.      -0.       0.56157 -0.00461 -0.10304]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02769  0.52798 -0.0408  -0.18575  1.15184]
 [ 0.       0.28959 -0.71049  0.05347  0.09885 -0.05394]
 [ 0.       0.27613  0.24864 -0.11116 -0.00061  0.28644]
 [ 0.      -0.      -0.       0.14614 -0.57054  0.2296 ]
 [ 0.      -0.       0.       0.43475  0.00294 -0.32779]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46392 -0.25357  0.13626  0.13268  1.15184]
 [ 0.       0.42585 -0.34158  0.12687 -0.03165  0.28155]
 [ 0.       0.64504  0.11238  0.04892  0.07416 -0.07541]
 [ 0.      -0.      -0.       0.04044 -0.59524 -0.01143]
 [ 0.      -0.       0.       0.41006  0.10865  0.40004]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25097 -0.46534  0.18845  0.02563  1.15184]
 [ 0.       0.11409 -0.64678  0.0839   0.02996 -0.07383]
 [ 0.       0.33984  0.42414 -0.0827   0.10106 -0.28197]
 [ 0.       0.       0.      -0.02397 -0.49713  0.22872]
 [ 0.       0.      -0.       0.50817  0.17306  0.3284 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17613  0.4985  -0.14925  0.11788  1.15184]
 [ 0.       0.43698 -0.63263  0.04223 -0.12267 -0.15929]
 [ 0.       0.354    0.10125  0.06943  0.05777  0.2441 ]
 [ 0.       0.       0.       0.08366 -0.4044  -0.39517]
 [ 0.       0.      -0.       0.6009   0.06542 -0.06326]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.49074  0.19672 -0.02165  0.18895  1.15184]
 [ 0.       0.09025 -0.36844  0.08746 -0.01333  0.23726]
 [ 0.       0.61819  0.44798  0.0609   0.11599  0.16931]
 [ 0.      -0.      -0.       0.17274 -0.51232 -0.3235 ]
 [ 0.      -0.       0.       0.49298 -0.02365  0.2356 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05887 -0.52541  0.09261  0.16611  1.15184]
 [ 0.       0.2638  -0.71139  0.07324  0.08097  0.0369 ]
 [ 0.       0.27523  0.27443 -0.11115  0.02671 -0.28913]
 [ 0.       0.       0.       0.09677 -0.59878 -0.12538]
 [ 0.       0.      -0.       0.40651  0.05231  0.38005]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42598  0.31315  0.16914  0.08696  1.15184]
 [ 0.       0.46037 -0.38839 -0.12016  0.05927 -0.26899]
 [ 0.       0.59824  0.07786 -0.05295 -0.06506  0.11226]
 [ 0.      -0.      -0.      -0.00557 -0.56025  0.10623]
 [ 0.      -0.       0.       0.44504  0.15466  0.38584]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28542  0.44504 -0.18454  0.04599  1.15184]
 [ 0.       0.09273 -0.62167  0.08579  0.00617  0.09493]
 [ 0.       0.36495  0.4455  -0.046    0.1244   0.27558]
 [ 0.      -0.      -0.       0.00686 -0.43085 -0.33409]
 [ 0.      -0.       0.       0.57444  0.14223 -0.22032]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.13905 -0.51009 -0.08986  0.16761  1.15184]
 [ 0.       0.41476 -0.65571  0.0038   0.12713  0.14093]
 [ 0.       0.33091  0.12347 -0.08722 -0.03472 -0.25514]
```

```
 [ 0.       0.       0.       0.15262 -0.44231 -0.38762]
 [ 0.       0.      -0.       0.56298 -0.00353  0.09956]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.5131  -0.12749  0.03951  0.18603  1.15184]
 [ 0.       0.13095 -0.3245  -0.09093  0.02761 -0.25826]
 [ 0.       0.66212  0.40728 -0.0829  -0.09532 -0.13514]
 [ 0.      -0.      -0.       0.14708 -0.56953 -0.23187]
 [ 0.      -0.       0.       0.43576  0.002    0.32618]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.08992  0.521   -0.13538 -0.13357  1.15184]
 [ 0.       0.23803 -0.70923  0.08692  0.06051 -0.0197 ]
 [ 0.       0.27739  0.3002  -0.10458  0.05324  0.29081]
 [ 0.      -0.      -0.       0.04165 -0.59567  0.01405]
 [ 0.      -0.       0.       0.40962  0.10743 -0.39995]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38427 -0.36313 -0.18823 -0.02716  1.15184]
 [ 0.       0.48019 -0.43821 -0.10237  0.089    0.25315]
 [ 0.       0.54841  0.05804 -0.05911 -0.05564 -0.14447]
 [ 0.       0.       0.      -0.02405 -0.49873 -0.22605]
 [ 0.       0.      -0.       0.50656  0.17314 -0.33025]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.32074 -0.4203  -0.15037  0.11644  1.15184]
 [ 0.       0.07422 -0.59131 -0.08139  0.01803 -0.11707]
 [ 0.       0.39531  0.46401 -0.00138 -0.1343  -0.26693]
 [ 0.      -0.      -0.       0.08177 -0.40424 -0.39454]
 [ 0.      -0.       0.       0.60105  0.06732 -0.06706]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10353  0.51847 -0.02314  0.18877  1.15184]
 [ 0.       0.39102 -0.67421 -0.04293 -0.11681 -0.123  ]
 [ 0.       0.31241  0.14721  0.09707  0.0089   0.26425]
 [ 0.       0.       0.       0.17288 -0.51076 -0.32535]
 [ 0.       0.      -0.       0.49453 -0.02379  0.23303]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52619  0.05147  0.09153  0.16671  1.15184]
 [ 0.       0.18541 -0.29186  0.09381 -0.04232  0.27517]
 [ 0.       0.69476  0.35282  0.09424  0.07428  0.09611]
 [ 0.      -0.      -0.       0.09802 -0.59848 -0.12785]
 [ 0.      -0.       0.       0.40681  0.05106  0.37923]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.12106 -0.51465  0.16852  0.08816  1.15184]
 [ 0.       0.21235 -0.70394  0.09556  0.03729  0.00219]
 [ 0.       0.28268  0.32588 -0.09035  0.07939 -0.29147]
 [ 0.       0.       0.      -0.00474 -0.56138  0.10349]
 [ 0.       0.      -0.       0.44391  0.15383  0.38659]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34104  0.404    0.18495 -0.0443   1.15184]
 [ 0.       0.48716 -0.48673 -0.07033  0.1164  -0.23532]
 [ 0.       0.4999   0.05107 -0.06801 -0.04322  0.172  ]
 [ 0.      -0.      -0.       0.00555 -0.4321   0.33206]
 [ 0.      -0.       0.       0.57319  0.14354  0.22338]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35671  0.39023 -0.09137  0.1668   1.15184]
 [ 0.       0.05998 -0.55537  0.07202 -0.03798  0.14026]
 [ 0.       0.43125  0.47825  0.0487   0.12644  0.25551]
 [ 0.      -0.      -0.       0.15152 -0.44091 -0.3885 ]
 [ 0.      -0.       0.       0.56438 -0.00243  0.09605]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.06936 -0.52413 -0.03821 -0.1863   1.15184]
 [ 0.       0.36634 -0.68859 -0.07154 -0.09831  0.10543]
 [ 0.       0.29803  0.17189  0.09974 -0.01609 -0.27174]
 [ 0.      -0.      -0.       0.14801 -0.56851  0.23415]
 [ 0.      -0.       0.       0.43678  0.00108 -0.32455]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.52793  0.02845  0.1345   0.13445  1.15184]
 [ 0.       0.24926 -0.27607 -0.0941   0.05933 -0.28652]
 [ 0.       0.71055  0.28897 -0.09819 -0.05452 -0.05353]
 [ 0.      -0.      -0.       0.04288 -0.5961  -0.01667]
 [ 0.      -0.       0.       0.40919  0.10621  0.39985]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.15252  0.50622  0.18801  0.02868  1.15184]
 [ 0.       0.18689 -0.69537 -0.09862 -0.01068  0.01576]
 [ 0.       0.29125  0.35134  0.06588 -0.10396  0.29105]
 [ 0.       0.       0.      -0.0241  -0.50032  0.22337]
 [ 0.       0.      -0.       0.50497  0.17319  0.33207]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.29789 -0.43679  0.15149 -0.11498  1.15184]
 [ 0.       0.48393 -0.53127  0.02561 -0.1329   0.21648]
 [ 0.       0.45535  0.0543   0.07744  0.02593 -0.19518]
 [ 0.      -0.      -0.       0.07988 -0.40412  0.39388]
 [ 0.      -0.       0.       0.60117  0.06921  0.07086]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39284 -0.35383 -0.02464  0.18858  1.15184]
 [ 0.       0.05194 -0.51388 -0.06193  0.05152 -0.16435]
 [ 0.       0.47275  0.48629 -0.08566 -0.10565 -0.24072]
 [ 0.      -0.      -0.       0.173   -0.5092  -0.32719]
 [ 0.      -0.       0.       0.49609 -0.02391  0.23045]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03631  0.52745 -0.09044 -0.1673   1.15184]
 [ 0.       0.3411  -0.69924  0.09101  0.07606 -0.08816]
 [ 0.       0.28738  0.19713 -0.09674  0.03954  0.27782]
 [ 0.      -0.      -0.       0.09927 -0.59817  0.13032]
 [ 0.      -0.       0.       0.40712  0.04982 -0.37839]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51747 -0.10838  0.16789  0.08935  1.15184]
 [ 0.       0.31546 -0.28015  0.08854 -0.07966  0.29133]
 [ 0.       0.70648  0.22277  0.0975   0.03602  0.00935]
 [ 0.      -0.      -0.      -0.0039  -0.5625   0.10074]
 [ 0.      -0.       0.       0.44279  0.15299  0.38731]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18449 -0.49547 -0.18535  0.0426   1.15184]
 [ 0.       0.16184 -0.68326 -0.09396  0.01864 -0.03429]
 [ 0.       0.30337  0.37639  0.029   -0.12236 -0.28945]
 [ 0.      -0.      -0.       0.00427 -0.43338 -0.33   ]
 [ 0.      -0.       0.       0.57191  0.14482 -0.22641]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25579  0.4627   0.09288 -0.16596  1.15184]
 [ 0.       0.47312 -0.57056  0.02194  0.13215 -0.1973 ]
 [ 0.       0.41606  0.06511 -0.08377 -0.00517  0.21455]
 [ 0.      -0.       0.       0.15039 -0.43953  0.38935]
 [ 0.      -0.       0.       0.56576 -0.0013  -0.09254]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42823  0.31007  0.0369   0.18657  1.15184]
 [ 0.       0.0525  -0.4675   0.0542  -0.06049  0.18901]
 [ 0.       0.51912  0.48573  0.10978  0.07962  0.22189]
```

```
 [ 0.      -0.      -0.       0.14893 -0.56748 -0.23641]
 [ 0.      -0.       0.       0.43781  0.00016  0.32291]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00413 -0.52868 -0.13362 -0.13533  1.15184]
 [ 0.       0.31553 -0.70646 -0.10347 -0.05144  0.07108]
 [ 0.       0.28016  0.2227   0.08828 -0.062   -0.28268]
 [ 0.      -0.      -0.       0.0441  -0.5965   0.01929]
 [ 0.      -0.       0.       0.40879  0.10498 -0.39973]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49553  0.18431  0.18777  0.0302   1.15184]
 [ 0.       0.37625 -0.30329 -0.07284  0.10249 -0.28946]
 [ 0.       0.68334  0.16198 -0.0941  -0.01801  0.03419]
 [ 0.      -0.      -0.      -0.02413 -0.50192  0.22068]
 [ 0.      -0.       0.       0.50338  0.17321  0.33386]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21717  0.48204 -0.15259  0.11352  1.15184]
 [ 0.       0.13748 -0.66727  0.07987 -0.04646  0.05355]
 [ 0.       0.31936  0.40075  0.01734  0.12708  0.28651]
 [ 0.      -0.      -0.       0.07798 -0.40403 -0.39318]
 [ 0.      -0.       0.       0.60126  0.07111 -0.07464]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21525 -0.4829  -0.02614  0.18838  1.15184]
 [ 0.       0.45696 -0.60422  0.06142  0.11689  0.17821]
 [ 0.       0.3824   0.08127 -0.08548  0.01571 -0.23065]
 [ 0.       0.      -0.       0.17309 -0.50763 -0.32901]
 [ 0.       0.      -0.       0.49766 -0.024    0.22784]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.46145 -0.25804 -0.08935 -0.16789  1.15184]
 [ 0.       0.06436 -0.41805  0.04906 -0.06791 -0.21359]
 [ 0.       0.56857  0.47387  0.12326  0.0527  -0.19834]
 [ 0.       0.       0.       0.10051 -0.59784  0.13278]
 [ 0.       0.      -0.       0.40745  0.04857 -0.37753]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02743  0.52799  0.16725  0.09054  1.15184]
 [ 0.       0.28981 -0.71047 -0.1098  -0.02411 -0.05408]
 [ 0.       0.27615  0.24842  0.07307 -0.08374  0.28641]
 [ 0.       0.       0.      -0.00304 -0.56361  0.098  ]
 [ 0.       0.      -0.       0.44168  0.15213  0.38802]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46421 -0.25304 -0.18573  0.0409   1.15184]
 [ 0.       0.4255  -0.34123 -0.04319  0.12338  0.28164]
 [ 0.       0.6454   0.11273 -0.08889  0.00025 -0.07508]
 [ 0.       0.       0.       0.00301 -0.43468 -0.32791]
 [ 0.       0.      -0.       0.57061  0.14607 -0.22942]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25069 -0.46549 -0.09438  0.16511  1.15184]
 [ 0.       0.11427 -0.64697 -0.05918  0.06663 -0.07366]
 [ 0.       0.33965  0.42396 -0.06274 -0.1145  -0.28201]
 [ 0.      -0.      -0.       0.14923 -0.43816 -0.39018]
 [ 0.      -0.       0.       0.56713 -0.00014  0.089  ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17645  0.49839  0.03559  0.18682  1.15184]
 [ 0.       0.43716 -0.63241 -0.08934 -0.0941  -0.15944]
 [ 0.       0.35421  0.10107  0.08325 -0.03494  0.244  ]
 [ 0.       0.      -0.       0.14983 -0.56643 -0.23867]
 [ 0.       0.      -0.       0.43886 -0.00074  0.32124]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.49052  0.19726  0.13273  0.1362   1.15184]
 [ 0.       0.08997 -0.36883  0.04481 -0.07623  0.23707]
 [ 0.       0.61779  0.44826  0.12842  0.02603  0.16957]
 [ 0.      -0.      -0.       0.04533 -0.59689 -0.0219 ]
 [ 0.      -0.       0.       0.4084   0.10375  0.3996 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05861 -0.52544 -0.18752 -0.03171  1.15184]
 [ 0.       0.26401 -0.7114  -0.10901  0.00651  0.03705]
 [ 0.       0.27523  0.27422  0.04867 -0.10341 -0.28911]
 [ 0.      -0.      -0.      -0.02412 -0.5035  -0.21798]
 [ 0.      -0.       0.       0.50179  0.17321 -0.33562]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42632  0.3127   0.15367 -0.11205  1.15184]
 [ 0.       0.46015 -0.38798 -0.00094  0.13396 -0.26911]
 [ 0.       0.59864  0.07808 -0.08191  0.01822  0.11198]
 [ 0.      -0.      -0.       0.07608 -0.40399  0.39244]
 [ 0.      -0.       0.       0.60131  0.07301  0.07843]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28513  0.44522  0.02764 -0.18816  1.15184]
 [ 0.       0.0929  -0.6219  -0.03827  0.07706  0.09475]
 [ 0.       0.36473  0.44533 -0.09758 -0.0898   0.27565]
 [ 0.       0.       0.       0.17316 -0.50606  0.33082]
 [ 0.       0.      -0.       0.49923 -0.02407 -0.2252 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.13935 -0.51001  0.08825  0.16847  1.15184]
 [ 0.       0.41495 -0.65554  0.10738  0.0682   0.14108]
 [ 0.       0.33108  0.12328 -0.07764  0.05272 -0.25506]
 [ 0.       0.       0.       0.10175 -0.59749 -0.13524]
 [ 0.       0.      -0.       0.4078   0.04733  0.37665]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51295 -0.1281   0.16661  0.09172  1.15184]
 [ 0.       0.13055 -0.32482 -0.0384   0.08686 -0.2581 ]
 [ 0.       0.6618   0.40768 -0.12637  0.00053 -0.13544]
 [ 0.      -0.      -0.      -0.00218 -0.5647   0.09525]
 [ 0.      -0.       0.       0.44059  0.15126  0.3887 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.08966  0.52104  0.1861  -0.03921  1.15184]
 [ 0.       0.23824 -0.70926 -0.09843  0.03916 -0.01984]
 [ 0.       0.27736  0.29999  0.01325 -0.11658  0.2908 ]
 [ 0.       0.       0.       0.00179 -0.43599  0.3258 ]
 [ 0.       0.      -0.       0.5693   0.1473   0.2324 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38462 -0.36275 -0.09587  0.16425  1.15184]
 [ 0.       0.48008 -0.4378   0.0441   0.12826  0.25329]
 [ 0.       0.54882  0.05815 -0.07373  0.03401 -0.14422]
 [ 0.       0.      -0.       0.14805 -0.43682 -0.39097]
 [ 0.       0.      -0.       0.56847  0.00104  0.08545]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.32045 -0.42052  0.03428  0.18707  1.15184]
 [ 0.       0.07436 -0.59158 -0.02132  0.08061 -0.11688]
 [ 0.       0.39504  0.46387 -0.11992 -0.06046 -0.26701]
 [ 0.      -0.      -0.       0.15072 -0.56536 -0.24093]
 [ 0.      -0.       0.       0.43993 -0.00163  0.31955]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10382  0.51841  0.13184  0.13707  1.15184]
 [ 0.       0.39122 -0.67408 -0.11775 -0.04036 -0.12314]
 [ 0.       0.31254  0.14701  0.06809 -0.06971  0.26418]
```

```
 [ 0.       0.       0.       0.04657 -0.59727 -0.02451]
 [ 0.       0.      -0.       0.40802  0.10252  0.39945]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52613  0.05211 -0.18726 -0.03321  1.15184]
 [ 0.       0.18492 -0.29207 -0.02569  0.09958  0.27505]
 [ 0.       0.69455  0.35331 -0.11698  0.02697  0.09645]
 [ 0.       0.       0.      -0.0241  -0.50509 -0.21529]
 [ 0.       0.      -0.       0.50021  0.17319 -0.33736]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.1208  -0.51471 -0.15474  0.11056  1.15184]
 [ 0.       0.21256 -0.704   -0.0764   0.06849  0.00234]
 [ 0.       0.28262  0.32567 -0.03009 -0.11643 -0.29147]
 [ 0.      -0.      -0.       0.07418 -0.40397 -0.39167]
 [ 0.      -0.       0.       0.60132  0.07491 -0.0822 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.3414   0.4037   0.02915 -0.18794  1.15184]
 [ 0.       0.48715 -0.48634  0.08109  0.10918 -0.23547]
 [ 0.       0.50028  0.05108 -0.06571  0.04664  0.17179]
 [ 0.      -0.       0.       0.1732  -0.50448  0.33261]
 [ 0.      -0.       0.       0.50081 -0.02411 -0.22255]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35642  0.3905   0.08715  0.16904  1.15184]
 [ 0.       0.06008 -0.55569  0.00871 -0.08097  0.14006]
 [ 0.       0.43093  0.47815  0.13206  0.0303   0.25562]
 [ 0.      -0.      -0.       0.10299 -0.59713 -0.1377 ]
 [ 0.      -0.       0.       0.40816  0.0461   0.37576]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.06964 -0.52409  0.16596  0.09289  1.15184]
 [ 0.       0.36655 -0.68849  0.12118  0.01009  0.10557]
 [ 0.       0.29813  0.17168 -0.05297  0.086   -0.27168]
 [ 0.       0.       0.      -0.00129 -0.56577  0.09251]
 [ 0.       0.      -0.       0.43952  0.15038  0.38936]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.52797  0.02778  0.18645 -0.03751  1.15184]
 [ 0.       0.24872 -0.27612 -0.0026   0.11115 -0.28645]
 [ 0.       0.7105   0.28951 -0.09978  0.05169 -0.05389]
 [ 0.      -0.      -0.       0.00058 -0.43733  0.32367]
 [ 0.      -0.       0.       0.56796  0.1485   0.23536]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.15225  0.5063  -0.09737  0.16337  1.15184]
 [ 0.       0.1871  -0.69545  0.04699 -0.08739  0.01561]
 [ 0.       0.29117  0.35113  0.07153  0.10014  0.29106]
 [ 0.      -0.      -0.       0.14683 -0.43549 -0.39173]
 [ 0.      -0.       0.       0.5698   0.00225  0.08189]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.29824 -0.43655 -0.03296 -0.1873   1.15184]
 [ 0.       0.48399 -0.53093 -0.10652 -0.08352  0.21663]
 [ 0.       0.4557   0.05424  0.0584  -0.05707 -0.19501]
 [ 0.      -0.       0.       0.1516  -0.56428  0.24317]
 [ 0.      -0.       0.       0.44101 -0.00251 -0.31785]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39255 -0.35416 -0.13094 -0.13793  1.15184]
 [ 0.       0.05198 -0.51424 -0.00112 -0.08055 -0.16415]
 [ 0.       0.47238  0.48625  0.13601 -0.00004 -0.24086]
 [ 0.       0.       0.       0.04781 -0.59763  0.02712]
 [ 0.       0.      -0.       0.40767  0.10128 -0.39928]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.03658  0.52743  0.18699  0.03471  1.15184]
 [ 0.       0.34131 -0.69917 -0.11636  0.02309 -0.0883 ]
 [ 0.       0.28746  0.19692  0.02993 -0.1001   0.27778]
 [ 0.       0.       0.      -0.02405 -0.50666  0.21258]
 [ 0.       0.      -0.       0.49863  0.17313  0.33907]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51761 -0.10773  0.1558  -0.10907  1.15184]
 [ 0.       0.31493 -0.28003 -0.03091 -0.11496  0.29131]
 [ 0.       0.70659  0.2233   0.0763  -0.07068  0.00971]
 [ 0.      -0.      -0.       0.07228 -0.404    0.39086]
 [ 0.      -0.       0.       0.60129  0.07681  0.08597]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18423 -0.49557 -0.03066  0.1877   1.15184]
 [ 0.       0.16204 -0.68337 -0.01822  0.09407 -0.03414]
 [ 0.       0.30325  0.37618 -0.10232 -0.07307 -0.28947]
 [ 0.      -0.      -0.       0.17321 -0.5029  -0.33439]
 [ 0.      -0.       0.       0.50239 -0.02413  0.21987]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25613  0.46251 -0.08604 -0.16961  1.15184]
 [ 0.       0.47324 -0.57026  0.12199  0.05538 -0.19746]
 [ 0.       0.41636  0.06499 -0.05089  0.06671  0.2144 ]
 [ 0.      -0.       0.       0.10422 -0.59675  0.14015]
 [ 0.      -0.       0.       0.40854  0.04486 -0.37485]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42794  0.31047 -0.1653  -0.09405  1.15184]
 [ 0.       0.05245 -0.4679   0.01046  0.08053  0.1888 ]
 [ 0.       0.51872  0.48578 -0.13202  0.03109  0.22206]
 [ 0.       0.       0.      -0.0004  -0.56683 -0.08977]
 [ 0.       0.      -0.       0.43846  0.14949 -0.39   ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00439 -0.52868 -0.18678  0.03582  1.15184]
 [ 0.       0.31575 -0.70641 -0.10028  0.05745  0.07122]
 [ 0.       0.28021  0.22248 -0.0025  -0.10782 -0.28264]
 [ 0.      -0.      -0.      -0.00059 -0.43868 -0.32152]
 [ 0.      -0.       0.       0.56661  0.14968 -0.23829]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49576  0.18371  0.09886 -0.16247  1.15184]
 [ 0.       0.37579 -0.30303  0.06747  0.10604 -0.2895 ]
 [ 0.       0.68359  0.16244 -0.05212  0.08047  0.03384]
 [ 0.      -0.      -0.       0.1456  -0.43418  0.39246]
 [ 0.      -0.       0.       0.57111  0.00349 -0.07831]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.2169   0.48216  0.03163  0.18753  1.15184]
 [ 0.       0.13768 -0.66742 -0.00508 -0.09229  0.05338]
 [ 0.       0.31921  0.40055  0.12105  0.04234  0.28654]
 [ 0.      -0.      -0.       0.15246 -0.56319 -0.24541]
 [ 0.      -0.       0.       0.4421  -0.00337  0.31612]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21558 -0.48275 -0.13004 -0.13878  1.15184]
 [ 0.       0.45711 -0.60397 -0.12954 -0.02569  0.17837]
 [ 0.       0.38266  0.08112  0.04145 -0.07636 -0.23053]
 [ 0.      -0.      -0.       0.04905 -0.59797  0.02973]
 [ 0.      -0.       0.       0.40733  0.10004 -0.39909]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.46119 -0.2585   0.18671  0.0362   1.15184]
 [ 0.       0.06421 -0.41846  0.02203  0.0808  -0.21339]
 [ 0.       0.56816  0.47402 -0.11849  0.06272 -0.19856]
```

```
 [ 0.      -0.      -0.      -0.02397 -0.50823  0.20987]
 [ 0.      -0.       0.       0.49706  0.17306  0.34075]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02717  0.528    0.15684 -0.10756  1.15184]
 [ 0.       0.29002 -0.71045 -0.07151  0.08677 -0.05422]
 [ 0.       0.27617  0.24821 -0.04124 -0.10318  0.28639]
 [ 0.       0.       0.       0.07038 -0.40406  0.39001]
 [ 0.       0.      -0.       0.60123  0.0787   0.08972]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.4645  -0.25251  0.03217 -0.18744  1.15184]
 [ 0.       0.42516 -0.34087 -0.09802 -0.08644  0.28172]
 [ 0.       0.64575  0.11307  0.03318 -0.08252 -0.07476]
 [ 0.      -0.       0.       0.17321 -0.50131  0.33615]
 [ 0.      -0.       0.       0.50398 -0.02412 -0.21717]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25041 -0.46564  0.08493  0.17017  1.15184]
 [ 0.       0.11446 -0.64716  0.02264  0.08622 -0.07349]
 [ 0.       0.33947  0.42377 -0.13006 -0.01134 -0.28206]
 [ 0.      -0.      -0.       0.10545 -0.59635 -0.1426 ]
 [ 0.      -0.       0.       0.40894  0.04363  0.37393]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17676  0.49828 -0.16464 -0.09521  1.15184]
 [ 0.       0.43733 -0.6322   0.12963 -0.00613 -0.1596 ]
 [ 0.       0.35442  0.1009  -0.02791  0.08584  0.2439 ]
 [ 0.      -0.      -0.       0.00051 -0.56788 -0.08704]
 [ 0.      -0.       0.       0.43742  0.14858 -0.39062]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.49031  0.1978   0.1871  -0.03412  1.15184]
 [ 0.       0.0897  -0.36922 -0.03835 -0.07962  0.23689]
 [ 0.       0.6174   0.44853  0.09323 -0.09212  0.16983]
 [ 0.      -0.      -0.      -0.00174 -0.44006  0.31935]
 [ 0.      -0.       0.       0.56524  0.15082  0.24119]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05836 -0.52547 -0.10034  0.16156  1.15184]
 [ 0.       0.26423 -0.7114  -0.0352   0.10341  0.03719]
 [ 0.       0.27522  0.274   -0.07731 -0.08414 -0.28909]
 [ 0.      -0.      -0.       0.14433 -0.43289 -0.39316]
 [ 0.      -0.       0.       0.5724   0.00476  0.07471]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42665  0.31224 -0.0303  -0.18775  1.15184]
 [ 0.       0.45992 -0.38757  0.11903  0.06142 -0.26923]
 [ 0.       0.59905  0.07831 -0.02073  0.08135  0.11169]
 [ 0.      -0.       0.       0.15331 -0.56208  0.24765]
 [ 0.      -0.       0.       0.44321 -0.00422 -0.31437]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28484  0.44541 -0.12913 -0.13962  1.15184]
 [ 0.       0.09306 -0.62213  0.03605  0.07815  0.09457]
 [ 0.       0.3645   0.44517 -0.13113  0.01965  0.27571]
 [ 0.       0.       0.       0.05029 -0.59829  0.03233]
 [ 0.       0.      -0.       0.407    0.0988  -0.39889]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.13965 -0.50992  0.18641  0.03768  1.15184]
 [ 0.       0.41514 -0.65537  0.12063 -0.04042  0.14123]
 [ 0.       0.33125  0.12308 -0.00794  0.09348 -0.25497]
 [ 0.       0.       0.      -0.02387 -0.5098   0.20716]
 [ 0.       0.      -0.       0.49549  0.17296  0.34241]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.5128  -0.1287   0.15787 -0.10605  1.15184]
 [ 0.       0.13016 -0.32515  0.05967  0.07381 -0.25794]
 [ 0.       0.66147  0.40807 -0.05737  0.11265 -0.13574]
 [ 0.      -0.      -0.       0.06849 -0.40416  0.38913]
 [ 0.      -0.       0.       0.60113  0.0806   0.09347]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.08941  0.52109  0.03368 -0.18718  1.15184]
 [ 0.       0.23845 -0.70929 -0.00069  0.10596 -0.01998]
 [ 0.       0.27733  0.29978 -0.10303 -0.05608  0.29079]
 [ 0.       0.       0.       0.17317 -0.49971  0.33789]
 [ 0.       0.      -0.       0.50558 -0.02408 -0.21445]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38497 -0.36238 -0.08381 -0.17072  1.15184]
 [ 0.       0.47997 -0.43739 -0.13126 -0.03412  0.25343]
 [ 0.       0.54923  0.05826  0.01232 -0.08027 -0.14398]
 [ 0.      -0.       0.       0.10668 -0.59594  0.14505]
 [ 0.      -0.       0.       0.40935  0.04241 -0.37299]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.32015 -0.42075  0.16397  0.09636  1.15184]
 [ 0.       0.0745  -0.59185  0.04715  0.0688  -0.1167 ]
 [ 0.       0.39477  0.46373 -0.12411  0.05127 -0.2671 ]
 [ 0.      -0.      -0.       0.00143 -0.56891  0.0843 ]
 [ 0.      -0.       0.       0.43639  0.14766  0.39122]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.1041   0.51835 -0.1874   0.03243  1.15184]
 [ 0.       0.39142 -0.67394  0.09938 -0.07498 -0.12329]
 [ 0.       0.31268  0.14681  0.01966  0.09541  0.26412]
 [ 0.      -0.      -0.      -0.00285 -0.44144 -0.31716]
 [ 0.      -0.       0.       0.56385  0.15194 -0.24407]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52606  0.05276  0.10182 -0.16063  1.15184]
 [ 0.       0.18442 -0.29228 -0.0824  -0.06143  0.27494]
 [ 0.       0.69434  0.35381  0.01936 -0.11854  0.09679]
 [ 0.      -0.      -0.       0.14304 -0.43162  0.39383]
 [ 0.      -0.       0.       0.57367  0.00605 -0.0711 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.12054 -0.51478  0.02897  0.18796  1.15184]
 [ 0.       0.21277 -0.70406  0.02677  0.09908  0.00248]
 [ 0.       0.28257  0.32546 -0.11748 -0.02556 -0.29146]
 [ 0.      -0.      -0.       0.15415 -0.56095 -0.24988]
 [ 0.      -0.       0.       0.44434 -0.00506  0.31261]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34176  0.4034   0.12822  0.14046  1.15184]
 [ 0.       0.48714 -0.48595 -0.1359  -0.00512 -0.23562]
 [ 0.       0.50067  0.05109  0.00469 -0.08044  0.17158]
 [ 0.       0.       0.       0.05154 -0.5986  -0.03492]
 [ 0.       0.      -0.       0.40669  0.09755  0.39867]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35612  0.39077  0.18611  0.03916  1.15184]
 [ 0.       0.06017 -0.55601 -0.05737 -0.05781  0.13987]
 [ 0.       0.43061  0.47805  0.10683 -0.08332  0.25572]
 [ 0.      -0.      -0.      -0.02374 -0.51136  0.20444]
 [ 0.      -0.       0.       0.49393  0.17283  0.34404]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.06991 -0.52406  0.15888 -0.10452  1.15184]
 [ 0.       0.36675 -0.68839  0.06474 -0.10296  0.10572]
 [ 0.       0.29824  0.17148  0.05197  0.08657 -0.27163]
```

```
 [ 0.       0.       0.       0.06659 -0.40429  0.38821]
 [ 0.       0.      -0.       0.601    0.08249  0.09721]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.528    0.02712  0.0352  -0.1869   1.15184]
 [ 0.       0.24817 -0.27617  0.10203  0.04398 -0.28638]
 [ 0.       0.71045  0.29006  0.0106   0.11194 -0.05425]
 [ 0.      -0.      -0.       0.17311 -0.49812  0.33962]
 [ 0.      -0.       0.       0.50718 -0.02402 -0.2117 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.15199  0.50638 -0.08269 -0.17127  1.15184]
 [ 0.       0.18731 -0.69554  0.04711  0.08736  0.01546]
 [ 0.       0.29109  0.35092 -0.12295  0.00456  0.29106]
 [ 0.       0.       0.       0.1079  -0.59551  0.14749]
 [ 0.       0.      -0.       0.40978  0.04119 -0.37203]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.29859 -0.43631 -0.16329 -0.0975   1.15184]
 [ 0.       0.48405 -0.53058 -0.13283  0.02608  0.21679]
 [ 0.       0.45604  0.05418 -0.00512 -0.08148 -0.19483]
 [ 0.      -0.      -0.       0.00236 -0.56992 -0.08157]
 [ 0.      -0.       0.       0.43537  0.14673 -0.3918 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39225 -0.35449 -0.18768  0.03074  1.15184]
 [ 0.       0.05201 -0.5146  -0.06731 -0.04427 -0.16395]
 [ 0.       0.47202  0.48622  0.07638 -0.11254 -0.24099]
 [ 0.       0.       0.      -0.00395 -0.44285 -0.31495]
 [ 0.       0.      -0.       0.56244  0.15303 -0.24692]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03685  0.52741 -0.10329  0.15969  1.15184]
 [ 0.       0.34152 -0.69909  0.02291 -0.11642 -0.08844]
 [ 0.       0.28753  0.19671  0.08117  0.06573  0.27773]
 [ 0.      -0.      -0.       0.14172 -0.43037 -0.39447]
 [ 0.      -0.       0.       0.57492  0.00736  0.06748]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51774 -0.10708 -0.02763 -0.18816  1.15184]
 [ 0.       0.3144  -0.27992 -0.11658 -0.02379  0.2913 ]
 [ 0.       0.70671  0.22383 -0.02911 -0.09992  0.01008]
 [ 0.      -0.      -0.       0.15497 -0.55981  0.2521 ]
 [ 0.      -0.       0.       0.44548 -0.00588 -0.31082]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18396 -0.49566 -0.1273  -0.14129  1.15184]
 [ 0.       0.16225 -0.68349 -0.06208 -0.07302 -0.03398]
 [ 0.       0.30313  0.37598  0.12097 -0.0342  -0.28949]
 [ 0.       0.       0.       0.05279 -0.59889  0.03752]
 [ 0.       0.      -0.       0.4064   0.0963  -0.39844]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25648  0.46232 -0.18579 -0.04062  1.15184]
 [ 0.       0.47335 -0.56996  0.12004 -0.05953 -0.19762]
 [ 0.       0.41666  0.06488  0.01958  0.08156  0.21426]
 [ 0.      -0.      -0.      -0.02359 -0.51291 -0.20172]
 [ 0.      -0.       0.       0.49238  0.17268 -0.34564]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42766  0.31086 -0.15988  0.10299  1.15184]
 [ 0.       0.05241 -0.4683   0.07625  0.02788  0.1886 ]
 [ 0.       0.51832  0.48582 -0.03348  0.13144  0.22223]
 [ 0.       0.       0.       0.0647  -0.40447 -0.38726]
 [ 0.       0.      -0.       0.60083  0.08439 -0.10094]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.00466 -0.52868  0.03671 -0.1866   1.15184]
 [ 0.       0.31596 -0.70637 -0.01578 -0.11452  0.07136]
 [ 0.       0.28025  0.22227  0.10091  0.03799 -0.2826 ]
 [ 0.       0.       0.       0.17302 -0.49651  0.34133]
 [ 0.       0.      -0.       0.50878 -0.02394 -0.20893]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49598  0.18311  0.08156  0.1718   1.15184]
 [ 0.       0.37533 -0.30277 -0.12563 -0.00169 -0.28955]
 [ 0.       0.68385  0.1629  -0.03881 -0.08773  0.03348]
 [ 0.       0.       0.       0.10911 -0.59506 -0.14993]
 [ 0.       0.      -0.       0.41023  0.03997  0.37105]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21663  0.48228 -0.16261 -0.09863  1.15184]
 [ 0.       0.13788 -0.66756  0.07323  0.05644  0.05322]
 [ 0.       0.31906  0.40035 -0.11112  0.06397  0.28657]
 [ 0.       0.       0.       0.0033  -0.57092 -0.07884]
 [ 0.       0.      -0.       0.43437  0.14578 -0.39236]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21591 -0.48261 -0.18795  0.02905  1.15184]
 [ 0.       0.45726 -0.60371 -0.09428  0.0925   0.17853]
 [ 0.       0.38291  0.08097 -0.03972 -0.07724 -0.2304 ]
 [ 0.      -0.      -0.      -0.00501 -0.44427 -0.31271]
 [ 0.      -0.       0.       0.56102  0.1541  -0.24974]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.46093 -0.25897 -0.10476  0.15873  1.15184]
 [ 0.       0.06406 -0.41887 -0.08308 -0.01036 -0.21319]
 [ 0.       0.56775  0.47417 -0.0128  -0.13348 -0.19877]
 [ 0.       0.       0.       0.14038 -0.42915 -0.39507]
 [ 0.       0.      -0.       0.57614  0.00871  0.06384]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02691  0.52802 -0.02629 -0.18836  1.15184]
 [ 0.       0.29023 -0.71043  0.04581  0.10271 -0.05436]
 [ 0.       0.27619  0.248   -0.11072 -0.00908  0.28636]
 [ 0.       0.       0.       0.15578 -0.55866  0.25431]
 [ 0.       0.      -0.       0.44663 -0.00669 -0.30901]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46479 -0.25198 -0.12638 -0.14212  1.15184]
 [ 0.       0.42481 -0.34051 -0.12866  0.02273  0.28181]
 [ 0.       0.64611  0.11342 -0.04391 -0.0774  -0.07444]
 [ 0.      -0.      -0.       0.05404 -0.59916  0.04011]
 [ 0.      -0.       0.       0.40613  0.09504 -0.39818]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25013 -0.46579  0.18547  0.04209  1.15184]
 [ 0.       0.11464 -0.64734  0.0811   0.03704 -0.07332]
 [ 0.       0.33928  0.42359 -0.09111  0.09347 -0.2821 ]
 [ 0.      -0.      -0.      -0.02342 -0.51445  0.19899]
 [ 0.      -0.       0.       0.49084  0.17251  0.34722]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17707  0.49817 -0.16086  0.10145  1.15184]
 [ 0.       0.43751 -0.63199  0.05483 -0.11765 -0.15975]
 [ 0.       0.35463  0.10072  0.06304  0.06456  0.2438 ]
 [ 0.      -0.      -0.       0.06281 -0.40467 -0.38627]
 [ 0.      -0.       0.       0.60062  0.08627 -0.10466]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.49009  0.19833  0.03824 -0.1863   1.15184]
 [ 0.       0.08943 -0.36962 -0.08813  0.00596  0.2367 ]
 [ 0.       0.61701  0.4488  -0.05072 -0.12089  0.17009]
```

```
 [ 0.      -0.      -0.       0.17291 -0.49491  0.34302]
 [ 0.      -0.       0.       0.51038 -0.02382 -0.20614]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.0581  -0.5255   0.08043  0.17234  1.15184]
 [ 0.       0.26444 -0.71141  0.0674   0.086    0.03733]
 [ 0.       0.27522  0.27379 -0.11269  0.01878 -0.28908]
 [ 0.      -0.      -0.       0.11033 -0.5946  -0.15237]
 [ 0.      -0.       0.       0.41069  0.03876  0.37006]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42698  0.31179 -0.16192 -0.09976  1.15184]
 [ 0.       0.4597  -0.38717  0.12423 -0.05003 -0.26935]
 [ 0.       0.59945  0.07853  0.04816  0.0688   0.1114 ]
 [ 0.      -0.      -0.       0.00426 -0.5719  -0.07611]
 [ 0.      -0.       0.       0.43339  0.14483 -0.39289]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28455  0.44559  0.1882  -0.02736  1.15184]
 [ 0.       0.09323 -0.62235 -0.08486 -0.01446  0.09439]
 [ 0.       0.36427  0.445    0.05801 -0.11922  0.27577]
 [ 0.      -0.      -0.      -0.00604 -0.44571  0.31046]
 [ 0.      -0.       0.       0.55958  0.15513  0.25253]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.13995 -0.50984 -0.10622  0.15776  1.15184]
 [ 0.       0.41534 -0.6552  -0.00882  0.12694  0.14138]
 [ 0.       0.33143  0.12289 -0.08331 -0.04307 -0.25489]
 [ 0.      -0.      -0.       0.13901 -0.42795 -0.39565]
 [ 0.      -0.       0.       0.57735  0.01008  0.06019]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51265 -0.1293   0.02494  0.18854  1.15184]
 [ 0.       0.12976 -0.32548 -0.09254  0.02082 -0.25778]
 [ 0.       0.66115  0.40847 -0.07557 -0.1014  -0.13604]
 [ 0.       0.       0.       0.15657 -0.55749 -0.25651]
 [ 0.       0.      -0.       0.4478  -0.00748  0.30718]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.08915  0.52113  0.12546  0.14293  1.15184]
 [ 0.       0.23866 -0.70932 -0.08252 -0.06652 -0.02013]
 [ 0.       0.2773   0.29957  0.10801 -0.04571  0.29078]
 [ 0.      -0.       0.       0.0553  -0.59942 -0.0427 ]
 [ 0.       0.       0.       0.40587  0.09379  0.39792]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38533 -0.362   -0.18513 -0.04354  1.15184]
 [ 0.       0.47986 -0.43698 -0.10961  0.07987  0.25357]
 [ 0.       0.54965  0.05837 -0.05433 -0.06037 -0.14373]
 [ 0.       0.       0.      -0.02322 -0.51599 -0.19626]
 [ 0.       0.       0.       0.48931  0.17231 -0.34877]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.31986 -0.42097  0.16183 -0.0999   1.15184]
 [ 0.       0.07463 -0.59213  0.08287 -0.00961 -0.11651]
 [ 0.       0.3945   0.4636  -0.01264  0.13368 -0.26718]
 [ 0.       0.       0.       0.06093 -0.40492  0.38525]
 [ 0.       0.       0.       0.60037  0.08815  0.10837]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10439  0.51829 -0.03976  0.18598  1.15184]
 [ 0.       0.39162 -0.67381 -0.03259 -0.12018 -0.12344]
 [ 0.       0.31282  0.14661  0.09585  0.01726  0.26405]
 [ 0.       0.       0.       0.17277 -0.4933  -0.3447 ]
 [ 0.       0.       0.       0.51199 -0.02368  0.20333]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.526    0.05341  0.0793   0.17286  1.15184]
 [ 0.       0.18393 -0.29249  0.09628 -0.03577  0.27482]
 [ 0.       0.69414  0.3543   0.08901  0.08073  0.09713]
 [ 0.       0.      -0.       0.11153 -0.59412 -0.1548 ]
 [ 0.       0.      -0.       0.41117  0.03755  0.36905]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.12029 -0.51484  0.16122  0.10088  1.15184]
 [ 0.       0.21298 -0.70411  0.09255  0.04444  0.00263]
 [ 0.       0.28251  0.32525 -0.09606  0.07226 -0.29146]
 [ 0.       0.       0.       0.00523 -0.57287  0.07339]
 [ 0.       0.       0.       0.43242  0.14386  0.39341]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34211  0.40309  0.18844 -0.02568  1.15184]
 [ 0.       0.48712 -0.48557 -0.08139  0.10896 -0.23578]
 [ 0.       0.50106  0.05111 -0.06359 -0.04948  0.17137]
 [ 0.       0.      -0.      -0.00705 -0.44716  0.30819]
 [ 0.       0.      -0.       0.55813  0.15614  0.2553 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35582  0.39105 -0.10767  0.15677  1.15184]
 [ 0.       0.06027 -0.55633  0.07539 -0.03085  0.13967]
 [ 0.       0.43029  0.47796  0.03592  0.13063  0.25583]
 [ 0.       0.      -0.       0.13762 -0.42677 -0.39619]
 [ 0.       0.      -0.       0.57853  0.01147  0.05653]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.07019 -0.52402  0.02358  0.18871  1.15184]
 [ 0.       0.36696 -0.68828  0.06382  0.10357  0.10586]
 [ 0.       0.29834  0.17127 -0.10059  0.00844 -0.27157]
 [ 0.       0.       0.       0.15735 -0.55631 -0.25871]
 [ 0.       0.       0.       0.44899 -0.00826  0.30533]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.52804  0.02645  0.12452  0.14375  1.15184]
 [ 0.       0.24762 -0.27623 -0.09775  0.05267 -0.28631]
 [ 0.       0.7104   0.29061 -0.0944  -0.06122 -0.05461]
 [ 0.       0.      -0.       0.05656 -0.59966 -0.04528]
 [ 0.       0.      -0.       0.40563  0.09253  0.39763]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.15173  0.50646  0.18479  0.04499  1.15184]
 [ 0.       0.18752 -0.69562 -0.09743 -0.01909  0.01531]
 [ 0.       0.291    0.35071  0.07456 -0.09784  0.29107]
 [ 0.       0.       0.      -0.023   -0.51751  0.19353]
 [ 0.       0.       0.       0.48778  0.17209  0.35029]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.29895 -0.43607 -0.16278  0.09834  1.15184]
 [ 0.       0.48411 -0.53023 -0.03932  0.12954  0.21695]
 [ 0.       0.45639  0.05412 -0.07436 -0.03366 -0.19466]
 [ 0.       0.       0.       0.05906 -0.4052  -0.38419]
 [ 0.       0.       0.       0.60009  0.09003 -0.11207]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39195 -0.35482  0.04128 -0.18565  1.15184]
 [ 0.       0.05205 -0.51497  0.06607 -0.04611 -0.16375]
 [ 0.       0.47166  0.48618  0.07611  0.11272 -0.24113]
 [ 0.       0.       0.       0.1726  -0.49169  0.34636]
 [ 0.       0.       0.       0.5136  -0.02352 -0.20049]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03712  0.5274  -0.07816 -0.17338  1.15184]
 [ 0.       0.34173 -0.69902  0.08545  0.08236 -0.08858]
 [ 0.       0.28761  0.1965  -0.0992   0.0326   0.27769]
```

```
 [ 0.       0.      -0.       0.11273 -0.59363  0.15722]
 [ 0.       0.      -0.       0.41166  0.03635 -0.36802]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51788 -0.10643 -0.16052 -0.10199  1.15184]
 [ 0.       0.31386 -0.2798  -0.09405  0.07278  0.29129]
 [ 0.       0.70682  0.22437 -0.09479 -0.04314  0.01044]
 [ 0.       0.       0.       0.0062  -0.57382 -0.07067]
 [ 0.       0.       0.       0.43147  0.14288 -0.39391]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18369 -0.49576 -0.18866  0.024    1.15184]
 [ 0.       0.16245 -0.6836  -0.09541  0.00942 -0.03382]
 [ 0.       0.30302  0.37578  0.04084 -0.11886 -0.28951]
 [ 0.       0.      -0.      -0.00803 -0.44862 -0.3059 ]
 [ 0.       0.      -0.       0.55667  0.15711 -0.25804]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25682  0.46214 -0.10912  0.15576  1.15184]
 [ 0.       0.47346 -0.56966 -0.0087  -0.13372 -0.19777]
 [ 0.       0.41697  0.06476  0.0828   0.01329  0.21411]
 [ 0.       0.       0.       0.1362  -0.42561 -0.39669]
 [ 0.       0.       0.       0.57968  0.01289  0.05285]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42737  0.31125 -0.02222 -0.18888  1.15184]
 [ 0.       0.05236 -0.4687  -0.05847  0.05632  0.1884 ]
 [ 0.       0.51792  0.48587 -0.10339 -0.0878   0.22241]
 [ 0.       0.       0.       0.15811 -0.55511  0.2609 ]
 [ 0.       0.       0.       0.45018 -0.00903 -0.30346]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00492 -0.52868  0.12359  0.14455  1.15184]
 [ 0.       0.31617 -0.70632  0.09965  0.05863  0.0715 ]
 [ 0.       0.2803   0.22206 -0.09235  0.05561 -0.28257]
 [ 0.       0.       0.       0.05782 -0.59989 -0.04786]
 [ 0.       0.       0.       0.4054   0.09127  0.39733]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.4962   0.18251  0.18443  0.04642  1.15184]
 [ 0.       0.37487 -0.30251 -0.08114  0.09586 -0.28959]
 [ 0.       0.68411  0.16336 -0.09247 -0.02578  0.03313]
 [ 0.       0.      -0.      -0.02276 -0.51903  0.19079]
 [ 0.       0.      -0.       0.48626  0.17184  0.35179]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21635  0.48241  0.16372 -0.09677  1.15184]
 [ 0.       0.13807 -0.66771 -0.08433  0.03798  0.05306]
 [ 0.       0.31891  0.40016 -0.00397 -0.12814  0.28661]
 [ 0.       0.       0.       0.05719 -0.40551  0.38309]
 [ 0.       0.       0.       0.59978  0.0919   0.11575]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21624 -0.48246 -0.04281  0.1853   1.15184]
 [ 0.       0.45741 -0.60346  0.05095  0.12188  0.17868]
 [ 0.       0.38317  0.08082 -0.08643  0.00828 -0.23028]
 [ 0.       0.       0.       0.17241 -0.49007 -0.348  ]
 [ 0.       0.       0.       0.51522 -0.02332  0.19763]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.46067 -0.25943  0.07701  0.17389  1.15184]
 [ 0.       0.06391 -0.41929 -0.05347  0.06438 -0.21299]
 [ 0.       0.56734  0.47432 -0.1193  -0.06125 -0.19898]
 [ 0.       0.      -0.       0.11393 -0.59312 -0.15965]
 [ 0.       0.      -0.       0.41218  0.03516  0.36698]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.02665  0.52803 -0.15981 -0.1031   1.15184]
 [ 0.       0.29044 -0.71041  0.10774  0.03235 -0.0545 ]
 [ 0.       0.27621  0.24779 -0.07912  0.07794  0.28633]
 [ 0.       0.      -0.       0.00719 -0.57476 -0.06795]
 [ 0.       0.      -0.       0.43054  0.14189 -0.39439]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46508 -0.25145  0.18887 -0.02232  1.15184]
 [ 0.       0.42445 -0.34016  0.05489 -0.11853  0.28189]
 [ 0.       0.64647  0.11377  0.08867  0.00814 -0.07412]
 [ 0.       0.      -0.      -0.00897 -0.4501   0.30359]
 [ 0.       0.      -0.       0.55519  0.15806  0.26075]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.24985 -0.46594 -0.11056  0.15474  1.15184]
 [ 0.       0.11483 -0.64753 -0.06551  0.06053 -0.07315]
 [ 0.       0.33909  0.4234  -0.05098 -0.12014 -0.28215]
 [ 0.       0.      -0.       0.13476 -0.42448 -0.39717]
 [ 0.       0.      -0.       0.58081  0.01433  0.04916]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17739  0.49805 -0.02086 -0.18903  1.15184]
 [ 0.       0.43768 -0.63178  0.08186  0.10075 -0.1599 ]
 [ 0.       0.35484  0.10055 -0.08558  0.02851  0.2437 ]
 [ 0.       0.      -0.       0.15886 -0.5539   0.26309]
 [ 0.       0.      -0.       0.45139 -0.00977 -0.30157]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.48987  0.19887  0.12265  0.14535  1.15184]
 [ 0.       0.08916 -0.37001  0.04973 -0.07295  0.23652]
 [ 0.       0.61661  0.44907  0.12639  0.03491  0.17035]
 [ 0.       0.      -0.       0.05908 -0.6001  -0.05044]
 [ 0.       0.      -0.       0.40519  0.09     0.39701]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05785 -0.52553  0.18406  0.04786  1.15184]
 [ 0.       0.26465 -0.71141  0.10925  0.00282  0.03747]
 [ 0.       0.27521  0.27358 -0.0573   0.0988  -0.28906]
 [ 0.       0.       0.      -0.02249 -0.52054  0.18805]
 [ 0.       0.       0.       0.48475  0.17158  0.35327]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42731  0.31134 -0.16464  0.09519  1.15184]
 [ 0.       0.45947 -0.38676  0.01482 -0.13308 -0.26946]
 [ 0.       0.59986  0.07876  0.08343 -0.00989  0.11112]
 [ 0.       0.       0.       0.05533 -0.40586 -0.38197]
 [ 0.       0.       0.       0.59943  0.09376 -0.11942]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28427  0.44578  0.04434 -0.18494  1.15184]
 [ 0.       0.0934  -0.62258 -0.04483  0.07352  0.09421]
 [ 0.       0.36404  0.44483 -0.08925 -0.09802  0.27583]
 [ 0.       0.       0.       0.17219 -0.48846  0.34962]
 [ 0.       0.       0.       0.51683 -0.0231  -0.19475]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.14025 -0.50976 -0.07586 -0.1744   1.15184]
 [ 0.       0.41553 -0.65503 -0.10232 -0.07569  0.14153]
 [ 0.       0.3316   0.1227   0.08106 -0.04711 -0.25481]
 [ 0.       0.      -0.       0.11512 -0.59259  0.16207]
 [ 0.       0.      -0.       0.4127   0.03396 -0.36591]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51249 -0.1299   0.1591   0.1042   1.15184]
 [ 0.       0.12937 -0.3258  -0.04449  0.0837  -0.25762]
 [ 0.       0.66082  0.40886 -0.1262  -0.00885 -0.13635]
```

```
 [ 0.       0.      -0.       0.00819 -0.57568  0.06523]
 [ 0.       0.      -0.       0.42961  0.1409   0.39485]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.08889  0.52117 -0.18906  0.02064  1.15184]
 [ 0.       0.23887 -0.70935  0.10186 -0.02942 -0.02027]
 [ 0.       0.27727  0.29935 -0.02455  0.11466  0.29077]
 [ 0.       0.      -0.      -0.00989 -0.45159 -0.30126]
 [ 0.       0.      -0.       0.5537   0.15898 -0.26344]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38568 -0.36163  0.112   -0.15371  1.15184]
 [ 0.       0.47975 -0.43657 -0.03113 -0.13199  0.25371]
 [ 0.       0.55006  0.05848  0.0767  -0.02676 -0.14348]
 [ 0.       0.      -0.       0.13329 -0.42337  0.39761]
 [ 0.       0.      -0.       0.58192  0.01579 -0.04547]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.31956 -0.42119  0.01949  0.18918  1.15184]
 [ 0.       0.07477 -0.5924  -0.02734  0.07884 -0.11632]
 [ 0.       0.39422  0.46346 -0.11486 -0.06952 -0.26726]
 [ 0.       0.      -0.       0.15959 -0.55267 -0.26526]
 [ 0.       0.      -0.       0.45262 -0.01051  0.29966]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10468  0.51823 -0.1217  -0.14614  1.15184]
 [ 0.       0.39182 -0.67367  0.11468  0.04857 -0.12358]
 [ 0.       0.31295  0.14641 -0.07271  0.06474  0.26398]
 [ 0.       0.      -0.       0.06035 -0.60029  0.05301]
 [ 0.       0.      -0.       0.405    0.08874 -0.39667]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52593  0.05406 -0.18369 -0.04928  1.15184]
 [ 0.       0.18343 -0.2927  -0.03377  0.09692  0.2747 ]
 [ 0.       0.69393  0.3548  -0.119    0.01711  0.09747]
 [ 0.       0.       0.      -0.0222  -0.52205 -0.18531]
 [ 0.       0.       0.       0.48325  0.17129 -0.35471]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.12003 -0.5149   0.16555 -0.09361  1.15184]
 [ 0.       0.21319 -0.70417  0.08318 -0.06022  0.00277]
 [ 0.       0.28246  0.32504  0.01776  0.11886 -0.29146]
 [ 0.       0.       0.       0.05347 -0.40625  0.3808 ]
 [ 0.       0.       0.       0.59904  0.09561  0.12308]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34247  0.40279  0.04587 -0.18457  1.15184]
 [ 0.       0.48711 -0.48518  0.07117  0.1159  -0.23593]
 [ 0.       0.50144  0.05112 -0.06944  0.04087  0.17116]
 [ 0.       0.      -0.       0.17194 -0.48684  0.35122]
 [ 0.       0.      -0.       0.51845 -0.02285 -0.19184]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35552  0.39132 -0.07471 -0.17489  1.15184]
 [ 0.       0.06037 -0.55665 -0.01424  0.08022  0.13948]
 [ 0.       0.42997  0.47786 -0.12956 -0.03957  0.25594]
 [ 0.       0.       0.       0.11631 -0.59204  0.16448]
 [ 0.       0.       0.       0.41325  0.03278 -0.36484]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.07047 -0.52398  0.15838  0.10529  1.15184]
 [ 0.       0.36717 -0.68818  0.12015  0.01918  0.10601]
 [ 0.       0.29844  0.17106 -0.05919  0.08173 -0.27152]
 [ 0.       0.       0.       0.0092  -0.57658  0.06252]
 [ 0.       0.       0.       0.42871  0.13989  0.39529]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.52807  0.02579  0.18923 -0.01897  1.15184]
 [ 0.       0.24707 -0.27628 -0.01314  0.11019 -0.28624]
 [ 0.       0.71034  0.29116 -0.10444  0.04203 -0.05497]
 [ 0.       0.      -0.      -0.01078 -0.45309  0.29892]
 [ 0.       0.      -0.       0.5522   0.15987  0.26609]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.15147  0.50654 -0.11343  0.15266  1.15184]
 [ 0.       0.18773 -0.69571  0.05551 -0.08235  0.01516]
 [ 0.       0.29092  0.3505   0.06109  0.10674  0.29108]
 [ 0.       0.      -0.       0.1318  -0.42229 -0.39801]
 [ 0.       0.      -0.       0.583    0.01728  0.04176]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.2993  -0.43583  0.01812  0.18932  1.15184]
 [ 0.       0.48417 -0.52988  0.09977  0.09151  0.21711]
 [ 0.       0.45674  0.05406 -0.06247  0.05252 -0.19448]
 [ 0.       0.       0.       0.16031 -0.55143 -0.26743]
 [ 0.       0.       0.       0.45386 -0.01122  0.29773]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39166 -0.35515  0.12075  0.14693  1.15184]
 [ 0.       0.05208 -0.51533 -0.0043   0.08045 -0.16355]
 [ 0.       0.47129  0.48615 -0.13568 -0.00947 -0.24127]
 [ 0.       0.      -0.       0.06162 -0.60047 -0.05558]
 [ 0.       0.      -0.       0.40482  0.08747  0.39632]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03739  0.52738  0.1833   0.0507   1.15184]
 [ 0.       0.34194 -0.69894 -0.11798  0.01313 -0.08873]
 [ 0.       0.28768  0.19629  0.03827 -0.09713  0.27764]
 [ 0.       0.       0.      -0.02189 -0.52354  0.18256]
 [ 0.       0.       0.       0.48175  0.17098  0.35613]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51801 -0.10578 -0.16644  0.09202  1.15184]
 [ 0.       0.31332 -0.27969  0.01892  0.11734  0.29127]
 [ 0.       0.70693  0.2249  -0.08326  0.06268  0.01081]
 [ 0.       0.       0.       0.05163 -0.40667 -0.3796 ]
 [ 0.       0.       0.       0.59862  0.09746 -0.12673]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18343 -0.49586  0.0474  -0.18418  1.15184]
 [ 0.       0.16266 -0.68372  0.02645 -0.09218 -0.03367]
 [ 0.       0.30291  0.37557  0.09538  0.08182 -0.28952]
 [ 0.       0.       0.       0.17167 -0.48523  0.3528 ]
 [ 0.       0.       0.       0.52007 -0.02258 -0.18892]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25716  0.46194 -0.07355 -0.17539  1.15184]
 [ 0.       0.47358 -0.56935  0.11779  0.06392 -0.19793]
 [ 0.       0.41727  0.06465 -0.05532  0.063    0.21396]
 [ 0.       0.      -0.       0.11749 -0.59148  0.16689]
 [ 0.       0.      -0.       0.41381  0.0316  -0.36374]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42709  0.31164  0.15765  0.10637  1.15184]
 [ 0.       0.05232 -0.4691  -0.00466 -0.08103  0.18819]
 [ 0.       0.51752  0.48591  0.13399 -0.02116  0.22258]
 [ 0.       0.      -0.       0.01022 -0.57747  0.05981]
 [ 0.       0.      -0.       0.42782  0.13887  0.39571]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00518 -0.52868  0.18939 -0.01729  1.15184]
 [ 0.       0.31638 -0.70627  0.10545 -0.04748  0.07164]
 [ 0.       0.28035  0.22185 -0.00795  0.10748 -0.28253]
```

```
 [ 0.       0.       0.      -0.01165 -0.45461  0.29656]
 [ 0.       0.       0.       0.55068  0.16073  0.26872]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49642  0.18191  0.11485 -0.15159  1.15184]
 [ 0.       0.3744  -0.30226  0.05657  0.11207 -0.28963]
 [ 0.       0.68437  0.16383 -0.05982  0.07517  0.03278]
 [ 0.       0.      -0.       0.13029 -0.42123  0.39839]
 [ 0.       0.      -0.       0.58406  0.01879 -0.03804]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21608  0.48253 -0.01674 -0.18944  1.15184]
 [ 0.       0.13827 -0.66786 -0.00201  0.09249  0.0529 ]
 [ 0.       0.31876  0.39996 -0.11733 -0.0516   0.28663]
 [ 0.       0.       0.       0.16101 -0.55018  0.26959]
 [ 0.       0.       0.       0.45511 -0.01192 -0.29577]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21656 -0.48231 -0.1198  -0.14771  1.15184]
 [ 0.       0.45756 -0.6032  -0.12747 -0.03472  0.17884]
 [ 0.       0.38342  0.08067  0.04653 -0.07328 -0.23016]
 [ 0.       0.      -0.       0.06288 -0.60063  0.05815]
 [ 0.       0.      -0.       0.40467  0.0862  -0.39595]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.46041 -0.2599  -0.1829  -0.05211  1.15184]
 [ 0.       0.06376 -0.4197  -0.01538 -0.08224 -0.21279]
 [ 0.       0.56692  0.47447  0.1234  -0.05255 -0.1992 ]
 [ 0.       0.       0.      -0.02156 -0.52502 -0.17982]
 [ 0.       0.       0.       0.48027  0.17065 -0.35753]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.0264   0.52804  0.16731 -0.09042  1.15184]
 [ 0.       0.29066 -0.71039 -0.08021  0.07891 -0.05464]
 [ 0.       0.27623  0.24757 -0.03025 -0.10684  0.28631]
 [ 0.       0.       0.       0.0498  -0.40713  0.37837]
 [ 0.       0.       0.       0.59816  0.09929  0.13036]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46537 -0.25091 -0.04894  0.18378  1.15184]
 [ 0.       0.4241  -0.3398   0.08996  0.09465  0.28198]
 [ 0.       0.64682  0.11413 -0.04018  0.07952 -0.07379]
 [ 0.       0.       0.       0.17136 -0.48361 -0.35437]
 [ 0.       0.       0.       0.52168 -0.02227  0.18597]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.24957 -0.46609 -0.07238 -0.17587  1.15184]
 [ 0.       0.11501 -0.64771 -0.01657 -0.08766 -0.07298]
 [ 0.       0.33891  0.42322  0.12887  0.02057 -0.28219]
 [ 0.       0.       0.       0.11866 -0.59091  0.1693 ]
 [ 0.       0.       0.       0.41439  0.03042 -0.36262]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.1777   0.49794  0.15692  0.10745  1.15184]
 [ 0.       0.43785 -0.63157 -0.12979 -0.00358 -0.16006]
 [ 0.       0.35506  0.10038  0.03411 -0.08348  0.2436 ]
 [ 0.       0.       0.       0.01125 -0.57834  0.0571 ]
 [ 0.       0.       0.       0.42695  0.13784  0.39611]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.48965  0.19941  0.18954 -0.01563  1.15184]
 [ 0.       0.08889 -0.3704  -0.0307  -0.08273  0.23633]
 [ 0.       0.61622  0.44934  0.10171 -0.08281  0.1706 ]
 [ 0.       0.      -0.      -0.01248 -0.45613  0.29418]
 [ 0.       0.      -0.       0.54916  0.16156  0.27132]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.05759 -0.52555  0.11626 -0.15051  1.15184]
 [ 0.       0.26487 -0.71142  0.04544 -0.09942  0.03761]
 [ 0.       0.27521  0.27336  0.06841  0.09143 -0.28904]
 [ 0.       0.       0.       0.12876 -0.4202   0.39873]
 [ 0.       0.       0.       0.58509  0.02033 -0.03431]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42764  0.31088  0.01536  0.18956  1.15184]
 [ 0.       0.45924 -0.38636 -0.11389 -0.07039 -0.26958]
 [ 0.       0.60026  0.07899  0.02673 -0.07968  0.11083]
 [ 0.       0.       0.       0.1617  -0.54891 -0.27174]
 [ 0.       0.       0.       0.45638 -0.01261  0.2938 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28398  0.44596 -0.11884 -0.14848  1.15184]
 [ 0.       0.09356 -0.62281  0.03064  0.0805   0.09403]
 [ 0.       0.36382  0.44467 -0.13214  0.0104   0.27589]
 [ 0.       0.       0.       0.06415 -0.60077  0.06071]
 [ 0.       0.       0.       0.40452  0.08493 -0.39557]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.14055 -0.50968  0.1825   0.05351  1.15184]
 [ 0.       0.41572 -0.65485  0.12368 -0.0301   0.14168]
 [ 0.       0.33177  0.12251 -0.0157   0.09241 -0.25472]
 [ 0.       0.       0.      -0.0212  -0.52649  0.17707]
 [ 0.       0.       0.       0.4788   0.17029  0.3589 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51234 -0.1305  -0.16817  0.08882  1.15184]
 [ 0.       0.12898 -0.32613 -0.0518  -0.07931 -0.25746]
 [ 0.       0.66049  0.40925  0.06875 -0.10625 -0.13665]
 [ 0.       0.       0.       0.04797 -0.40762 -0.37711]
 [ 0.       0.       0.       0.59767  0.10111 -0.13397]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.08864  0.52122  0.05047 -0.18336  1.15184]
 [ 0.       0.23909 -0.70938 -0.01013  0.10556 -0.02041]
 [ 0.       0.27724  0.29914 -0.09755 -0.06501  0.29076]
 [ 0.       0.       0.       0.17103 -0.48199  0.35591]
 [ 0.       0.       0.       0.5233  -0.02194 -0.183  ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38603 -0.36125  0.07122  0.17635  1.15184]
 [ 0.       0.47964 -0.43616  0.1285   0.04332  0.25385]
 [ 0.       0.55047  0.05859 -0.01773  0.07929 -0.14324]
 [ 0.       0.       0.       0.11983 -0.59031 -0.17171]
 [ 0.       0.       0.       0.41498  0.02926  0.36149]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.31927 -0.42142 -0.15619 -0.10851  1.15184]
 [ 0.       0.07491 -0.59267 -0.04206 -0.07209 -0.11613]
 [ 0.       0.39395  0.46332  0.12755 -0.04186 -0.26734]
 [ 0.       0.       0.       0.01229 -0.5792  -0.05439]
 [ 0.       0.       0.       0.42609  0.1368  -0.39649]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10497  0.51818  0.18967 -0.01396  1.15184]
 [ 0.       0.39202 -0.67354 -0.10622  0.06507 -0.12373]
 [ 0.       0.31309  0.14621 -0.01042 -0.09677  0.26391]
 [ 0.       0.       0.      -0.01328 -0.45767  0.29179]
 [ 0.       0.       0.       0.54762  0.16237  0.2739 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52586  0.0547  -0.11767  0.14941  1.15184]
 [ 0.       0.18294 -0.29291  0.07582  0.06908  0.27458]
 [ 0.       0.69371  0.35529 -0.03106  0.1162   0.0978 ]
```

```
 [ 0.       0.       0.       0.1272  -0.4192  -0.39903]
 [ 0.       0.       0.       0.58609  0.02188  0.03057]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.11977 -0.51496  0.01397  0.18967  1.15184]
 [ 0.       0.21341 -0.70422  0.019    0.10095  0.00292]
 [ 0.       0.2824   0.32482 -0.11506 -0.03464 -0.29146]
 [ 0.       0.      -0.       0.16236 -0.54763 -0.27388]
 [ 0.       0.      -0.       0.45766 -0.01328  0.2918 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34283  0.40248 -0.11788 -0.14925  1.15184]
 [ 0.       0.48709 -0.48479  0.13522  0.01459 -0.23608]
 [ 0.       0.50183  0.05114 -0.01006  0.07994  0.17095]
 [ 0.       0.      -0.       0.06543 -0.6009   0.06327]
 [ 0.       0.      -0.       0.4044   0.08366 -0.39517]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35522  0.39159  0.18208  0.05491  1.15184]
 [ 0.       0.06046 -0.55697 -0.05248 -0.06233  0.13928]
 [ 0.       0.42965  0.47777  0.11344 -0.07403  0.25604]
 [ 0.       0.      -0.      -0.02083 -0.52796  0.17431]
 [ 0.       0.      -0.       0.47734  0.16991  0.36024]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.07075 -0.52395 -0.16901  0.0872   1.15184]
 [ 0.       0.36737 -0.68808 -0.07516  0.09571  0.10615]
 [ 0.       0.29855  0.17086 -0.04268 -0.09141 -0.27146]
 [ 0.       0.      -0.       0.04616 -0.40814 -0.37581]
 [ 0.       0.      -0.       0.59715  0.10293 -0.13757]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.5281   0.02512  0.05201 -0.18293  1.15184]
 [ 0.       0.24653 -0.27634  0.09762  0.05263 -0.28617]
 [ 0.       0.71028  0.2917   0.00076  0.11264 -0.05533]
 [ 0.       0.      -0.       0.17067 -0.48037  0.35743]
 [ 0.       0.      -0.       0.52492 -0.02158 -0.18001]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.15121  0.50662 -0.07004 -0.17682  1.15184]
 [ 0.       0.18794 -0.69579  0.04083  0.09056  0.01501]
 [ 0.       0.29083  0.35029 -0.12289 -0.00423  0.29109]
 [ 0.       0.       0.       0.12099 -0.5897   0.17411]
 [ 0.       0.       0.       0.41559  0.0281  -0.36034]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.29965 -0.43558 -0.15544 -0.10958  1.15184]
 [ 0.       0.48423 -0.52953 -0.13442  0.01615  0.21726]
 [ 0.       0.45709  0.054    0.00073 -0.08159 -0.19431]
 [ 0.       0.      -0.       0.01334 -0.58004 -0.05169]
 [ 0.       0.      -0.       0.42525  0.13575 -0.39685]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39136 -0.35547  0.18978 -0.0123   1.15184]
 [ 0.       0.05212 -0.51569  0.06289  0.05036 -0.16334]
 [ 0.       0.47093  0.48611 -0.08682  0.10469 -0.24141]
 [ 0.       0.      -0.      -0.01406 -0.45921  0.28938]
 [ 0.       0.      -0.       0.54608  0.16314  0.27644]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03766  0.52736 -0.11906  0.1483   1.15184]
 [ 0.       0.34215 -0.69887  0.03457 -0.11359 -0.08887]
 [ 0.       0.28775  0.19608  0.07409  0.07351  0.2776 ]
 [ 0.       0.      -0.       0.12563 -0.41823 -0.3993 ]
 [ 0.       0.      -0.       0.58706  0.02346  0.02682]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.51814 -0.10513  0.01258  0.18977  1.15184]
 [ 0.       0.31279 -0.27958  0.11424  0.03261  0.29126]
 [ 0.       0.70704  0.22544  0.0215   0.10205  0.01117]
 [ 0.       0.       0.       0.16301 -0.54634 -0.27601]
 [ 0.       0.       0.       0.45895 -0.01393  0.28979]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18316 -0.49596  0.11691  0.15001  1.15184]
 [ 0.       0.16286 -0.68383  0.05692  0.07722 -0.03351]
 [ 0.       0.30279  0.37537 -0.123    0.02561 -0.28954]
 [ 0.       0.      -0.       0.0667  -0.60101 -0.06583]
 [ 0.       0.      -0.       0.40429  0.08239  0.39475]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.2575   0.46175  0.18166  0.05629  1.15184]
 [ 0.       0.47369 -0.56905 -0.12464  0.04928 -0.19809]
 [ 0.       0.41757  0.06454 -0.01284 -0.08283  0.21382]
 [ 0.       0.       0.      -0.02043 -0.52941  0.17156]
 [ 0.       0.       0.       0.47588  0.16952  0.36156]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.4268   0.31204  0.16984 -0.08558  1.15184]
 [ 0.       0.05227 -0.4695  -0.07302 -0.03541  0.18799]
 [ 0.       0.51713  0.48596  0.04693 -0.12728  0.22275]
 [ 0.       0.      -0.       0.04436 -0.4087   0.37448]
 [ 0.       0.      -0.       0.59659  0.10473  0.14116]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00544 -0.52867 -0.05355  0.18249  1.15184]
 [ 0.       0.31659 -0.70623  0.00543  0.11555  0.07178]
 [ 0.       0.2804   0.22164 -0.09702 -0.04685 -0.2825 ]
 [ 0.       0.      -0.       0.17028 -0.47876 -0.35893]
 [ 0.       0.      -0.       0.52653 -0.02119  0.17699]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49664  0.18131  0.06886  0.17728  1.15184]
 [ 0.       0.37394 -0.302   -0.12505 -0.01049 -0.28967]
 [ 0.       0.68462  0.16429 -0.03279 -0.09036  0.03243]
 [ 0.       0.       0.       0.12215 -0.58908 -0.1765 ]
 [ 0.       0.       0.       0.41622  0.02694  0.35918]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21581  0.48265  0.15469  0.11063  1.15184]
 [ 0.       0.13847 -0.66801 -0.06896 -0.06171  0.05274]
 [ 0.       0.31861  0.39976  0.11552 -0.0555   0.28666]
 [ 0.       0.      -0.       0.0144  -0.58087  0.04899]
 [ 0.       0.      -0.       0.42442  0.13469  0.39719]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21689 -0.48216 -0.18988  0.01064  1.15184]
 [ 0.       0.45771 -0.60294 -0.10276  0.08307  0.179  ]
 [ 0.       0.38368  0.08052 -0.03223 -0.08057 -0.23004]
 [ 0.       0.      -0.      -0.0148  -0.46077 -0.28696]
 [ 0.       0.      -0.       0.54453  0.16389 -0.27896]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.46015 -0.26036 -0.12045  0.14717  1.15184]
 [ 0.       0.06362 -0.42011 -0.08158 -0.01841 -0.21258]
 [ 0.       0.56651  0.47461  0.00069 -0.13414 -0.19941]
 [ 0.       0.       0.       0.12403 -0.41728 -0.39953]
 [ 0.       0.       0.       0.58801  0.02506  0.02306]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02614  0.52805 -0.01118 -0.18985  1.15184]
 [ 0.       0.29087 -0.71037  0.03763  0.10607 -0.05478]
 [ 0.       0.27625  0.24736 -0.10958 -0.01772  0.28628]
```

```
 [ 0.       0.       0.       0.16365 -0.54503  0.27814]
 [ 0.       0.       0.       0.46026 -0.01456 -0.28775]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46565 -0.25038  0.11593  0.15076  1.15184]
 [ 0.       0.42375 -0.33944  0.12982 -0.01378  0.28206]
 [ 0.       0.64718  0.11448  0.03873  0.08029 -0.07347]
 [ 0.       0.       0.       0.06797 -0.6011  -0.06838]
 [ 0.       0.       0.       0.40419  0.08111  0.39431]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.24929 -0.46624  0.18123  0.05767  1.15184]
 [ 0.       0.1152  -0.6479   0.07785  0.04364 -0.07281]
 [ 0.       0.33872  0.42303 -0.09858  0.08548 -0.28223]
 [ 0.       0.      -0.      -0.02001 -0.53085  0.1688 ]
 [ 0.       0.      -0.       0.47444  0.1691   0.36286]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17801  0.49783  0.17065 -0.08396  1.15184]
 [ 0.       0.43803 -0.63135 -0.0668   0.11136 -0.16021]
 [ 0.       0.35527  0.1002  -0.05602 -0.07063  0.2435 ]
 [ 0.       0.       0.       0.04257 -0.4093   0.37311]
 [ 0.       0.       0.       0.59599  0.10652  0.14473]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.48944  0.19994 -0.05509  0.18203  1.15184]
 [ 0.       0.08862 -0.3708   0.08818  0.0017   0.23614]
 [ 0.       0.61583  0.44961  0.03976  0.12502  0.17086]
 [ 0.       0.       0.       0.16986 -0.47715 -0.36041]
 [ 0.       0.       0.       0.52814 -0.02078  0.17396]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05733 -0.52558 -0.06768 -0.17773  1.15184]
 [ 0.       0.26508 -0.71142 -0.06113 -0.09066  0.03775]
 [ 0.       0.2752   0.27315  0.11366 -0.01066 -0.28902]
 [ 0.       0.       0.       0.12329 -0.58843  0.17889]
 [ 0.       0.       0.       0.41686  0.02579 -0.35799]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42797  0.31043 -0.15394 -0.11168  1.15184]
 [ 0.       0.45901 -0.38595  0.12751 -0.04076 -0.2697 ]
 [ 0.       0.60067  0.07922  0.04325  0.0721   0.11054]
 [ 0.       0.      -0.       0.01547 -0.58168 -0.04629]
 [ 0.       0.      -0.       0.42361  0.13362 -0.39751]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28369  0.44614  0.18997 -0.00899  1.15184]
 [ 0.       0.09373 -0.62303 -0.0832  -0.0224   0.09386]
 [ 0.       0.36359  0.4445   0.06913 -0.11307  0.27595]
 [ 0.       0.      -0.      -0.01552 -0.46233  0.28452]
 [ 0.       0.      -0.       0.54296  0.1646   0.28144]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.14085 -0.50959  0.12183 -0.14603  1.15184]
 [ 0.       0.41591 -0.65468  0.02166 -0.12546  0.14183]
 [ 0.       0.33194  0.12232  0.0785   0.05117 -0.25464]
 [ 0.       0.       0.       0.12241 -0.41636  0.39973]
 [ 0.       0.       0.       0.58893  0.02667 -0.0193 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51219 -0.1311   0.00978  0.18993  1.15184]
 [ 0.       0.12859 -0.32646 -0.09367  0.01373 -0.2573 ]
 [ 0.       0.66016  0.40964 -0.06755 -0.10707 -0.13695]
 [ 0.       0.       0.       0.16426 -0.54371 -0.28025]
 [ 0.       0.       0.       0.46158 -0.01518  0.28569]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.08838  0.52126 -0.11496 -0.15151  1.15184]
 [ 0.       0.2393  -0.70941  0.07774  0.07217 -0.02056]
 [ 0.       0.27721  0.29893 -0.11087  0.038    0.29075]
 [ 0.       0.       0.       0.06925 -0.60118  0.07093]
 [ 0.       0.       0.       0.40412  0.07984 -0.39386]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38638 -0.36088  0.18078  0.05905  1.15184]
 [ 0.       0.47953 -0.43575  0.11583 -0.0705   0.25399]
 [ 0.       0.55088  0.0587   0.04937  0.06454 -0.14299]
 [ 0.       0.       0.      -0.01957 -0.53228  0.16605]
 [ 0.       0.       0.       0.47301  0.16866  0.36413]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.31897 -0.42164 -0.17144  0.08233  1.15184]
 [ 0.       0.07505 -0.59294 -0.08348  0.00112 -0.11595]
 [ 0.       0.39368  0.46318  0.02645 -0.1316  -0.26742]
 [ 0.       0.       0.       0.0408  -0.40992 -0.37172]
 [ 0.       0.       0.       0.59537  0.10829 -0.14828]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10525  0.51812 -0.05663  0.18156  1.15184]
 [ 0.       0.39222 -0.6734  -0.0216  -0.1227  -0.12387]
 [ 0.       0.31322  0.14601  0.09383  0.02576  0.26384]
 [ 0.       0.      -0.       0.16942 -0.47554 -0.36187]
 [ 0.       0.      -0.       0.52976 -0.02033  0.1709 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.5258   0.05535 -0.06649 -0.17818  1.15184]
 [ 0.       0.18245 -0.29312 -0.09832  0.02898  0.27446]
 [ 0.       0.6935   0.35578 -0.08325 -0.0869   0.09814]
 [ 0.       0.      -0.       0.12444 -0.58777  0.18128]
 [ 0.       0.      -0.       0.41752  0.02465 -0.35679]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.11951 -0.51502  0.15318  0.11272  1.15184]
 [ 0.       0.21362 -0.70428  0.0891   0.05117  0.00307]
 [ 0.       0.28234  0.32461 -0.10109  0.06491 -0.29146]
 [ 0.       0.      -0.       0.01654 -0.58247  0.0436 ]
 [ 0.       0.      -0.       0.42282  0.13254  0.39782]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34319  0.40218 -0.19004  0.00734  1.15184]
 [ 0.       0.48708 -0.4844   0.09139 -0.10072 -0.23623]
 [ 0.       0.50222  0.05115  0.05876  0.05513  0.17074]
 [ 0.       0.      -0.      -0.0162  -0.4639  -0.28206]
 [ 0.       0.      -0.       0.54139  0.16529 -0.2839 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35492  0.39186 -0.12321  0.14488  1.15184]
 [ 0.       0.06056 -0.55729  0.07811 -0.02324  0.13909]
 [ 0.       0.42934  0.47767  0.02249  0.13357  0.25615]
 [ 0.       0.       0.       0.12078 -0.41548 -0.3999 ]
 [ 0.       0.       0.       0.58982  0.02831  0.01552]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.07102 -0.52391  0.00838  0.19     1.15184]
 [ 0.       0.36758 -0.68797  0.05547  0.10835  0.10629]
 [ 0.       0.29865  0.17065 -0.10086  0.00052 -0.2714 ]
 [ 0.       0.      -0.       0.16486 -0.54238 -0.28236]
 [ 0.       0.      -0.       0.46291 -0.01577  0.28361]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.52813  0.02446  0.11397  0.15225  1.15184]
 [ 0.       0.24598 -0.2764  -0.10092  0.04583 -0.2861 ]
 [ 0.       0.71023  0.29225 -0.09018 -0.06761 -0.05569]
```

```
 [ 0.       0.       0.       0.07052 -0.60124 -0.07348]
 [ 0.       0.       0.       0.40406  0.07856  0.3934 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.15095  0.50669  0.18033  0.06041  1.15184]
 [ 0.       0.18815 -0.69587 -0.0956  -0.0271   0.01486]
 [ 0.       0.29075  0.35008  0.0824  -0.09125  0.2911 ]
 [ 0.       0.      -0.      -0.01911 -0.5337   0.16329]
 [ 0.       0.      -0.       0.47159  0.1682   0.36537]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.30001 -0.43534 -0.17222  0.08069  1.15184]
 [ 0.       0.48429 -0.52919 -0.05254  0.12479  0.21742]
 [ 0.       0.45744  0.05394 -0.07054 -0.04098 -0.19413]
 [ 0.       0.      -0.       0.03904 -0.41058 -0.37029]
 [ 0.       0.      -0.       0.59471  0.11005 -0.15181]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39106 -0.3558  -0.05817  0.18107  1.15184]
 [ 0.       0.05216 -0.51606 -0.06986  0.04014 -0.16314]
 [ 0.       0.47057  0.48607 -0.0656  -0.11914 -0.24154]
 [ 0.       0.       0.       0.16894 -0.47393 -0.36331]
 [ 0.       0.       0.       0.53136 -0.01986  0.16782]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03792  0.52734 -0.0653  -0.17862  1.15184]
 [ 0.       0.34236 -0.69879  0.07936  0.08834 -0.08901]
 [ 0.       0.28783  0.19587 -0.1012   0.02541  0.27755]
 [ 0.       0.       0.       0.12557 -0.5871   0.18366]
 [ 0.       0.       0.       0.41819  0.02352 -0.35557]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51827 -0.10448 -0.15241 -0.11375  1.15184]
 [ 0.       0.31225 -0.27947 -0.0989   0.0657   0.29125]
 [ 0.       0.70715  0.22598 -0.09166 -0.04988  0.01154]
 [ 0.       0.      -0.       0.01763 -0.58325 -0.04091]
 [ 0.       0.      -0.       0.42204  0.13146 -0.3981 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18289 -0.49606  0.1901  -0.0057   1.15184]
 [ 0.       0.16307 -0.68395  0.09596 -0.00036 -0.03336]
 [ 0.       0.30268  0.37516 -0.05196  0.11437 -0.28956]
 [ 0.       0.      -0.      -0.01686 -0.46547  0.2796 ]
 [ 0.       0.      -0.       0.53982  0.16595  0.28633]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25784  0.46156  0.12457 -0.14371  1.15184]
 [ 0.       0.4738  -0.56875 -0.00492  0.13395 -0.19825]
 [ 0.       0.41787  0.06443 -0.081   -0.02144  0.21367]
 [ 0.       0.       0.       0.11912 -0.41462  0.40003]
 [ 0.       0.       0.       0.59067  0.02996 -0.01175]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42651  0.31243 -0.00697 -0.19005  1.15184]
 [ 0.       0.05223 -0.46989 -0.06254  0.0517   0.18778]
 [ 0.       0.51673  0.486   -0.09617 -0.09569  0.22292]
 [ 0.       0.      -0.       0.16544 -0.54104  0.28446]
 [ 0.       0.      -0.       0.46426 -0.01635 -0.2815 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00571 -0.52867 -0.11299 -0.15298  1.15184]
 [ 0.       0.3168  -0.70618 -0.09537 -0.06551  0.07192]
 [ 0.       0.28044  0.22143  0.09593 -0.04899 -0.28246]
 [ 0.       0.       0.       0.0718  -0.60128  0.07602]
 [ 0.       0.       0.       0.40401  0.07729 -0.39291]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.49686  0.1807  -0.17987 -0.06177  1.15184]
 [ 0.       0.37348 -0.30175  0.08859 -0.08882 -0.2897 ]
 [ 0.       0.68487  0.16475  0.0903   0.03314  0.03208]
 [ 0.       0.      -0.      -0.01864 -0.53511 -0.16053]
 [ 0.       0.      -0.       0.47018  0.16772 -0.36659]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21553  0.48277  0.17298 -0.07905  1.15184]
 [ 0.       0.13867 -0.66816 -0.08786  0.02914  0.05257]
 [ 0.       0.31847  0.39956  0.00936 -0.1278   0.28669]
 [ 0.       0.      -0.       0.03729 -0.41128  0.36883]
 [ 0.       0.      -0.       0.59402  0.1118   0.15533]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21722 -0.48202  0.05971 -0.18057  1.15184]
 [ 0.       0.45786 -0.60269 -0.03967 -0.12605  0.17915]
 [ 0.       0.38394  0.08037  0.08675 -0.00054 -0.22992]
 [ 0.       0.       0.       0.16844 -0.47232  0.36473]
 [ 0.       0.       0.       0.53297 -0.01935 -0.16472]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.45989 -0.26082  0.0641   0.17906  1.15184]
 [ 0.       0.06347 -0.42053 -0.05769  0.06051 -0.21238]
 [ 0.       0.5661   0.47476 -0.11468 -0.06963 -0.19963]
 [ 0.       0.       0.       0.1267  -0.58641 -0.18604]
 [ 0.       0.       0.       0.41888  0.02239  0.35433]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02588  0.52807  0.15164  0.11478  1.15184]
 [ 0.       0.29108 -0.71035 -0.10514 -0.04022 -0.05492]
 [ 0.       0.27628  0.24715  0.08458 -0.07185  0.28625]
 [ 0.       0.      -0.       0.01872 -0.58401  0.03822]
 [ 0.       0.      -0.       0.42128  0.13037  0.39837]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46594 -0.24985  0.19014 -0.00406  1.15184]
 [ 0.       0.4234  -0.33909  0.06573 -0.11276  0.28215]
 [ 0.       0.64753  0.11483  0.0877   0.01624 -0.07315]
 [ 0.       0.       0.      -0.01749 -0.46706  0.27711]
 [ 0.       0.       0.       0.53823  0.16657  0.28873]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.24901 -0.46639  0.12592 -0.14252  1.15184]
 [ 0.       0.11539 -0.64808  0.07133 -0.05368 -0.07264]
 [ 0.       0.33854  0.42284  0.03846  0.12466 -0.28228]
 [ 0.       0.      -0.       0.11745 -0.41379  0.40012]
 [ 0.       0.      -0.       0.5915   0.03164 -0.00796]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17833  0.49772 -0.00555 -0.1901   1.15184]
 [ 0.       0.4382  -0.63114  0.07364  0.10698 -0.16036]
 [ 0.       0.35548  0.10003 -0.08746  0.02172  0.2434 ]
 [ 0.       0.       0.       0.166   -0.53968  0.28655]
 [ 0.       0.       0.       0.46561 -0.01692 -0.27938]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.48922  0.20047  0.11199  0.15371  1.15184]
 [ 0.       0.08836 -0.37119  0.05439 -0.06936  0.23596]
 [ 0.       0.61543  0.44987  0.12377  0.04359  0.17112]
 [ 0.       0.       0.       0.07307 -0.60131 -0.07856]
 [ 0.       0.       0.       0.40398  0.07601  0.39241]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05708 -0.52561 -0.1794  -0.06312  1.15184]
 [ 0.       0.26529 -0.71142 -0.10873 -0.01184  0.03789]
 [ 0.       0.2752   0.27294  0.06522 -0.09367 -0.289  ]
```

```
 [ 0.       0.       0.      -0.01814 -0.53651 -0.15777]
 [ 0.       0.       0.       0.46879  0.16722 -0.36779]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.4283   0.30997 -0.17372  0.0774   1.15184]
 [ 0.       0.45879 -0.38555  0.02842 -0.13079 -0.26982]
 [ 0.       0.60107  0.07944  0.0841  -0.00151  0.11026]
 [ 0.       0.      -0.       0.03556 -0.412   -0.36733]
 [ 0.       0.      -0.       0.59329  0.11353 -0.15883]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.2834   0.44633  0.06125 -0.18005  1.15184]
 [ 0.       0.0939  -0.62326 -0.05128  0.06927  0.09368]
 [ 0.       0.36336  0.44433 -0.07991 -0.10571  0.27601]
 [ 0.       0.      -0.       0.16791 -0.47072  0.36612]
 [ 0.       0.      -0.       0.53457 -0.01882 -0.16159]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.14115 -0.50951  0.0629   0.17948  1.15184]
 [ 0.       0.4161  -0.65451  0.09665  0.0829   0.14198]
 [ 0.       0.33212  0.12213 -0.08412  0.0412  -0.25456]
 [ 0.       0.      -0.       0.12782 -0.5857  -0.18841]
 [ 0.       0.      -0.       0.41959  0.02127  0.35307]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51203 -0.1317   0.15087  0.1158   1.15184]
 [ 0.       0.1282  -0.32679 -0.05019  0.0802  -0.25714]
 [ 0.       0.65983  0.41003 -0.12536 -0.01798 -0.13725]
 [ 0.       0.       0.       0.01982 -0.58475  0.03553]
 [ 0.       0.       0.       0.42054  0.12927  0.39862]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.08812  0.5213  -0.19017  0.00242  1.15184]
 [ 0.       0.23951 -0.70944  0.10426 -0.01969 -0.0207 ]
 [ 0.       0.27718  0.29872 -0.03527  0.11175  0.29074]
 [ 0.       0.       0.      -0.01809 -0.46865 -0.27462]
 [ 0.       0.       0.       0.53665  0.16717 -0.29111]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38674 -0.3605  -0.12727  0.14132  1.15184]
 [ 0.       0.47941 -0.43533  0.01756  0.13444  0.25413]
 [ 0.       0.55129  0.05882 -0.079    0.01909 -0.14274]
 [ 0.       0.      -0.       0.11576 -0.41299 -0.40018]
 [ 0.       0.      -0.       0.5923   0.03333  0.00417]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.31868 -0.42186  0.00413  0.19014  1.15184]
 [ 0.       0.07519 -0.59321 -0.0334   0.07653 -0.11576]
 [ 0.       0.39341  0.46304 -0.10894 -0.07841 -0.2675 ]
 [ 0.       0.       0.       0.16655 -0.53831 -0.28862]
 [ 0.       0.       0.       0.46698 -0.01746  0.27723]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10554  0.51806 -0.111   -0.15443  1.15184]
 [ 0.       0.39242 -0.67326  0.11106  0.0565  -0.12402]
 [ 0.       0.31336  0.14581 -0.07694  0.0595   0.26377]
 [ 0.       0.       0.       0.07435 -0.60132  0.08109]
 [ 0.       0.       0.       0.40397  0.07474 -0.3919 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52573  0.056    0.17892  0.06446  1.15184]
 [ 0.       0.18195 -0.29334  0.04131 -0.09374  0.27433]
 [ 0.       0.69329  0.35628  0.12017 -0.00745  0.09848]
 [ 0.       0.       0.      -0.01762 -0.53789  0.15501]
 [ 0.       0.       0.       0.4674   0.16671  0.36896]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.11925 -0.51508  0.17445 -0.07575  1.15184]
 [ 0.       0.21383 -0.70433  0.08902 -0.05135  0.00321]
 [ 0.       0.28229  0.3244   0.00534  0.11999 -0.29146]
 [ 0.       0.      -0.       0.03385 -0.41276  0.36581]
 [ 0.       0.      -0.       0.59253  0.11524  0.16231]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34355  0.40187 -0.06279  0.17952  1.15184]
 [ 0.       0.48706 -0.48401 -0.06031 -0.12191 -0.23638]
 [ 0.       0.50261  0.05117  0.07277 -0.03459  0.17053]
 [ 0.       0.      -0.       0.16735 -0.46913 -0.3675 ]
 [ 0.       0.      -0.       0.53616 -0.01826  0.15845]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35463  0.39213 -0.06169 -0.1799   1.15184]
 [ 0.       0.06066 -0.5576  -0.01979  0.07907  0.13889]
 [ 0.       0.42902  0.47757 -0.12635 -0.04878  0.25626]
 [ 0.       0.      -0.       0.12893 -0.58498  0.19078]
 [ 0.       0.      -0.       0.42032  0.02016 -0.3518 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.0713  -0.52387 -0.15008 -0.11681  1.15184]
 [ 0.       0.36778 -0.68787 -0.11849 -0.02797  0.10644]
 [ 0.       0.29876  0.17045  0.06493 -0.07714 -0.27135]
 [ 0.       0.       0.       0.02093 -0.58548 -0.03285]
 [ 0.       0.       0.       0.41981  0.12816 -0.39885]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.52816  0.02379  0.19018 -0.00079  1.15184]
 [ 0.       0.24544 -0.27646 -0.0232   0.10831 -0.28603]
 [ 0.       0.71017  0.29279 -0.10807  0.03226 -0.05605]
 [ 0.       0.       0.      -0.01866 -0.47024  0.27211]
 [ 0.       0.       0.       0.53505  0.16774  0.29345]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.15069  0.50677 -0.1286   0.14011  1.15184]
 [ 0.       0.18835 -0.69596  0.06365 -0.07634  0.01471]
 [ 0.       0.29067  0.34987  0.04982  0.11237  0.2911 ]
 [ 0.       0.       0.       0.11405 -0.41223 -0.4002 ]
 [ 0.       0.       0.       0.59306  0.03504  0.00037]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.30036 -0.4351   0.00271  0.19016  1.15184]
 [ 0.       0.48435 -0.52884  0.09218  0.09918  0.21758]
 [ 0.       0.45779  0.05388 -0.06629  0.04752 -0.19395]
 [ 0.       0.      -0.       0.16707 -0.53693 -0.29069]
 [ 0.       0.      -0.       0.46836 -0.01798  0.27506]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39077 -0.35613 -0.11    -0.15514  1.15184]
 [ 0.       0.0522  -0.51642  0.00967 -0.07999 -0.16294]
 [ 0.       0.47021  0.48603  0.13468  0.0189  -0.24168]
 [ 0.       0.      -0.       0.07563 -0.60131  0.08363]
 [ 0.       0.      -0.       0.40398  0.07346 -0.39136]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03819  0.52732  0.17844  0.0658   1.15184]
 [ 0.       0.34256 -0.69872 -0.11873  0.00339 -0.08915]
 [ 0.       0.2879   0.19566  0.04605 -0.09359  0.27751]
 [ 0.       0.      -0.      -0.01708 -0.53926  0.15225]
 [ 0.       0.      -0.       0.46603  0.16617  0.37011]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.5184  -0.10383 -0.17516  0.07409  1.15184]
 [ 0.       0.31172 -0.27937  0.00687  0.11848  0.29123]
 [ 0.       0.70726  0.22651 -0.08934  0.05406  0.01191]
```

```
 [ 0.       0.      -0.       0.03215 -0.41354 -0.36425]
 [ 0.       0.      -0.       0.59175  0.11694 -0.16577]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18263 -0.49616 -0.06433  0.17897  1.15184]
 [ 0.       0.16327 -0.68406 -0.03477  0.08947 -0.0332 ]
 [ 0.       0.30256  0.37496 -0.08741 -0.09019 -0.28958]
 [ 0.       0.       0.       0.16676 -0.46754 -0.36884]
 [ 0.       0.       0.       0.53775 -0.01767  0.15529]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25819  0.46137  0.06047  0.18031  1.15184]
 [ 0.       0.47391 -0.56844 -0.11291 -0.07226 -0.19841]
 [ 0.       0.41818  0.06432  0.05954 -0.05893  0.21352]
 [ 0.       0.      -0.       0.13003 -0.58424 -0.19315]
 [ 0.       0.      -0.       0.42106  0.01905  0.3505 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42623  0.31282  0.1493   0.11781  1.15184]
 [ 0.       0.05219 -0.47029  0.00101 -0.08113  0.18758]
 [ 0.       0.51633  0.48604  0.1352  -0.01135  0.2231 ]
 [ 0.       0.       0.       0.02205 -0.58619  0.03017]
 [ 0.       0.       0.       0.4191   0.12704  0.39906]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00597 -0.52867 -0.19018 -0.00084  1.15184]
 [ 0.       0.31702 -0.70613 -0.10953  0.03737  0.07206]
 [ 0.       0.28049  0.22121  0.01799 -0.10617 -0.28243]
 [ 0.       0.       0.      -0.0192  -0.47184 -0.26959]
 [ 0.       0.       0.       0.53345  0.16828 -0.29577]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49708  0.1801  -0.12993  0.13888  1.15184]
 [ 0.       0.37301 -0.3015  -0.0449  -0.11708 -0.28974]
 [ 0.       0.68513  0.16522  0.06712 -0.06899  0.03173]
 [ 0.       0.      -0.       0.11232 -0.41149 -0.40018]
 [ 0.       0.      -0.       0.5938   0.03676 -0.00343]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21526  0.48289  0.00128  0.19018  1.15184]
 [ 0.       0.13886 -0.6683   0.00934 -0.09213  0.05241]
 [ 0.       0.31832  0.39937  0.11276  0.06082  0.28672]
 [ 0.       0.       0.       0.16757 -0.53553 -0.29275]
 [ 0.       0.       0.       0.46976 -0.01849  0.27287]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21755 -0.48187  0.10899  0.15585  1.15184]
 [ 0.       0.45801 -0.60243  0.12478  0.04356  0.17931]
 [ 0.       0.38419  0.08022 -0.05137  0.06987 -0.2298 ]
 [ 0.       0.      -0.       0.0769  -0.60129 -0.08616]
 [ 0.       0.      -0.       0.404    0.07218  0.39082]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.45963 -0.26128  0.17794  0.06713  1.15184]
 [ 0.       0.06332 -0.42094  0.00888  0.0831  -0.21218]
 [ 0.       0.56568  0.4749  -0.12731  0.04237 -0.19984]
 [ 0.       0.       0.      -0.01653 -0.54062  0.14949]
 [ 0.       0.       0.       0.46467  0.16561  0.37123]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02562  0.52808  0.17585 -0.07243  1.15184]
 [ 0.       0.29129 -0.71033 -0.088    0.07025 -0.05506]
 [ 0.       0.2763   0.24694 -0.01904 -0.10931  0.28623]
 [ 0.       0.      -0.       0.03047 -0.41436  0.36267]
 [ 0.       0.      -0.       0.59093  0.11862  0.16921]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.46623 -0.24931  0.06587 -0.17841  1.15184]
 [ 0.       0.42304 -0.33874 -0.0809  -0.10237  0.28223]
 [ 0.       0.64789  0.11519  0.04714 -0.07577 -0.07282]
 [ 0.       0.       0.       0.16614 -0.46595  0.37017]
 [ 0.       0.       0.       0.53934 -0.01705 -0.1521 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.24873 -0.46654 -0.05926 -0.18072  1.15184]
 [ 0.       0.11557 -0.64827 -0.01031 -0.0887  -0.07247]
 [ 0.       0.33835  0.42266  0.12699  0.02983 -0.28232]
 [ 0.       0.      -0.       0.13113 -0.58348  0.19551]
 [ 0.       0.      -0.       0.42181  0.01796 -0.34919]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17864  0.49761 -0.1485  -0.11881  1.15184]
 [ 0.       0.43838 -0.63093  0.12924  0.01306 -0.16052]
 [ 0.       0.35569  0.09985 -0.03997  0.08074  0.24329]
 [ 0.       0.       0.       0.02317 -0.58689 -0.0275 ]
 [ 0.       0.       0.       0.4184   0.12591 -0.39925]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.489    0.20101  0.19017  0.00246  1.15184]
 [ 0.       0.08809 -0.37159 -0.02305 -0.08503  0.23577]
 [ 0.       0.61504  0.45014  0.10905 -0.07305  0.17138]
 [ 0.       0.       0.      -0.01971 -0.47344  0.26706]
 [ 0.       0.       0.       0.53185  0.16879  0.29806]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05682 -0.52564  0.13124 -0.13764  1.15184]
 [ 0.       0.2655  -0.71143  0.05543 -0.09432  0.03803]
 [ 0.       0.2752   0.27273  0.05864  0.09789 -0.28898]
 [ 0.       0.      -0.       0.11058 -0.41079  0.40013]
 [ 0.       0.      -0.       0.5945   0.0385   0.00724]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42863  0.30951 -0.00015  0.19018  1.15184]
 [ 0.       0.45856 -0.38515 -0.10787 -0.0792  -0.26993]
 [ 0.       0.60148  0.07967  0.0328  -0.07749  0.10997]
 [ 0.       0.      -0.       0.16806 -0.53413 -0.29479]
 [ 0.       0.      -0.       0.47116 -0.01897  0.27066]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28312  0.44651  0.10798  0.15655  1.15184]
 [ 0.       0.09406 -0.62348 -0.0251  -0.08247  0.0935 ]
 [ 0.       0.36314  0.44416  0.1325  -0.00113  0.27607]
 [ 0.       0.       0.       0.07818 -0.60125 -0.08868]
 [ 0.       0.       0.       0.40404  0.07091  0.39025]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.14145 -0.50943  0.17744  0.06844  1.15184]
 [ 0.       0.41629 -0.65433  0.12579 -0.01992  0.14213]
 [ 0.       0.33229  0.12194 -0.02308  0.09075 -0.25447]
 [ 0.       0.      -0.      -0.01595 -0.54197  0.14673]
 [ 0.       0.      -0.       0.46332  0.16504  0.37233]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51188 -0.1323   0.17653 -0.07077  1.15184]
 [ 0.       0.12781 -0.32712  0.04351  0.08394 -0.25698]
 [ 0.       0.6595   0.41042 -0.07932  0.09878 -0.13755]
 [ 0.       0.       0.       0.02881 -0.41521  0.36105]
 [ 0.       0.       0.       0.59008  0.12028  0.17263]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.08787  0.52135 -0.06741  0.17783  1.15184]
 [ 0.       0.23972 -0.70947  0.01983 -0.10426 -0.02084]
 [ 0.       0.27716  0.2985   0.09107  0.0737   0.29073]
```

```
 [ 0.       0.       0.       0.16549 -0.46438 -0.37147]
 [ 0.       0.       0.       0.54092 -0.0164   0.14889]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38709 -0.36012 -0.05803 -0.18111  1.15184]
 [ 0.       0.4793  -0.43492 -0.12503 -0.05243  0.25427]
 [ 0.       0.5517   0.05893  0.02315 -0.07792 -0.14249]
 [ 0.       0.       0.       0.13222 -0.58271  0.19786]
 [ 0.       0.       0.       0.42258  0.01687 -0.34786]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.31838 -0.42208  0.1477   0.1198   1.15184]
 [ 0.       0.07533 -0.59348  0.03687  0.07494 -0.11558]
 [ 0.       0.39314  0.4629  -0.13023  0.03244 -0.26758]
 [ 0.       0.       0.       0.02431 -0.58757  0.02482]
 [ 0.       0.       0.       0.41772  0.12478  0.39943]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10583  0.518   -0.19014 -0.00408  1.15184]
 [ 0.       0.39262 -0.67312  0.1119  -0.05487 -0.12417]
 [ 0.       0.3135   0.14561  0.00139  0.09723  0.2637 ]
 [ 0.       0.       0.      -0.02019 -0.47505 -0.26452]
 [ 0.       0.       0.       0.53024  0.16928 -0.30032]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52566  0.05664 -0.13255  0.13638  1.15184]
 [ 0.       0.18146 -0.29355  0.06839  0.07617  0.27421]
 [ 0.       0.69307  0.35677 -0.04271  0.11263  0.09882]
 [ 0.       0.      -0.       0.10883 -0.41012 -0.40005]
 [ 0.       0.      -0.       0.59517  0.04026 -0.01105]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.119   -0.51514  0.00159 -0.19018  1.15184]
 [ 0.       0.21404 -0.70439 -0.01085 -0.10223  0.00336]
 [ 0.       0.28223  0.32419  0.11182  0.04378 -0.29146]
 [ 0.       0.      -0.       0.16852 -0.53271  0.29683]
 [ 0.       0.      -0.       0.47258 -0.01944 -0.26842]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.3439   0.40156 -0.10697 -0.15725  1.15184]
 [ 0.       0.48705 -0.48363  0.13388  0.02396 -0.23654]
 [ 0.       0.503    0.05118 -0.01536  0.07909  0.17032]
 [ 0.       0.       0.       0.07945 -0.6012   0.0912 ]
 [ 0.       0.       0.       0.4041   0.06963 -0.38967]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35433  0.3924   0.17693  0.06976  1.15184]
 [ 0.       0.06076 -0.55792 -0.04743 -0.06631  0.1387 ]
 [ 0.       0.4287   0.47747  0.11906 -0.06456  0.25636]
 [ 0.       0.       0.      -0.01536 -0.54331  0.14397]
 [ 0.       0.       0.       0.46198  0.16445  0.37341]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.07158 -0.52383 -0.17719  0.0691   1.15184]
 [ 0.       0.36799 -0.68776 -0.08469  0.0875   0.10658]
 [ 0.       0.29886  0.17024 -0.03305 -0.09523 -0.27129]
 [ 0.       0.       0.       0.02717 -0.41609 -0.35941]
 [ 0.       0.       0.       0.5892   0.12192 -0.17603]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.52819  0.02313 -0.06895  0.17724  1.15184]
 [ 0.       0.24489 -0.27652 -0.09229 -0.06113 -0.28596]
 [ 0.       0.71011  0.29334  0.00946 -0.11245 -0.05641]
 [ 0.       0.      -0.       0.16481 -0.4628  -0.37275]
 [ 0.       0.      -0.       0.54249 -0.01573  0.14567]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.15043  0.50685 -0.05681 -0.1815   1.15184]
 [ 0.       0.18856 -0.69604  0.03423  0.09334  0.01456]
 [ 0.       0.29058  0.34967 -0.12219 -0.01314  0.29111]
 [ 0.       0.      -0.       0.1333  -0.58192  0.20021]
 [ 0.       0.      -0.       0.42337  0.01579 -0.34652]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.30071 -0.43485  0.1469   0.12079  1.15184]
 [ 0.       0.48441 -0.52849  0.13527 -0.00635  0.21774]
 [ 0.       0.45814  0.05382 -0.00645  0.0813  -0.19378]
 [ 0.       0.      -0.       0.02544 -0.58823  0.02215]
 [ 0.       0.      -0.       0.41706  0.12364  0.39959]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39047 -0.35645 -0.1901  -0.00569  1.15184]
 [ 0.       0.05224 -0.51678 -0.05809 -0.05583 -0.16274]
 [ 0.       0.46984  0.48599  0.09618 -0.09615 -0.24181]
 [ 0.       0.      -0.      -0.02064 -0.47666 -0.26196]
 [ 0.       0.      -0.       0.52863  0.16973 -0.30255]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03846  0.5273   0.13384 -0.13511  1.15184]
 [ 0.       0.34277 -0.69864 -0.04609  0.1095  -0.08929]
 [ 0.       0.28798  0.19546 -0.06611 -0.08065  0.27746]
 [ 0.       0.      -0.       0.10706 -0.40948  0.39992]
 [ 0.       0.      -0.       0.59581  0.04203  0.01486]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51853 -0.10318  0.00303 -0.19016  1.15184]
 [ 0.       0.31118 -0.27926 -0.11113 -0.04148  0.29122]
 [ 0.       0.70736  0.22705 -0.01348 -0.10362  0.01227]
 [ 0.       0.       0.       0.16897 -0.53128  0.29886]
 [ 0.       0.       0.       0.47401 -0.01988 -0.26617]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18236 -0.49625 -0.10595 -0.15794  1.15184]
 [ 0.       0.16348 -0.68417 -0.05149 -0.08105 -0.03305]
 [ 0.       0.30245  0.37475  0.12443 -0.01693 -0.2896 ]
 [ 0.       0.      -0.       0.08073 -0.60112  0.09372]
 [ 0.       0.      -0.       0.40417  0.06836 -0.38907]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25853  0.46118  0.17641  0.07106  1.15184]
 [ 0.       0.47402 -0.56814 -0.12826  0.03903 -0.19857]
 [ 0.       0.41848  0.06421 -0.00626 -0.08351  0.21338]
 [ 0.       0.      -0.      -0.01475 -0.54463  0.1412 ]
 [ 0.       0.      -0.       0.46066  0.16384  0.37446]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42594  0.31321 -0.17783  0.06742  1.15184]
 [ 0.       0.05215 -0.47069  0.0691   0.04249  0.18737]
 [ 0.       0.51594  0.48608 -0.05975  0.12181  0.22327]
 [ 0.       0.      -0.       0.02554 -0.417   -0.35773]
 [ 0.       0.      -0.       0.58829  0.12354 -0.17941]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00623 -0.52866 -0.07049  0.17664  1.15184]
 [ 0.       0.31723 -0.70609 -0.00533  0.11563  0.0722 ]
 [ 0.       0.28054  0.221   -0.09219 -0.0556  -0.28239]
 [ 0.       0.       0.       0.16411 -0.46124 -0.374  ]
 [ 0.       0.       0.       0.54405 -0.01502  0.14242]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.4973   0.1795   0.05557  0.18188  1.15184]
 [ 0.       0.37254 -0.30124 -0.12384 -0.01936 -0.28978]
 [ 0.       0.68538  0.16569 -0.02648 -0.09261  0.03138]
```

```
 [ 0.       0.      -0.       0.13437 -0.58112 -0.20256]
 [ 0.       0.      -0.       0.42418  0.01472  0.34515]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21499  0.48302  0.14609  0.12177  1.15184]
 [ 0.       0.13906 -0.66845 -0.06441 -0.06656  0.05225]
 [ 0.       0.31817  0.39917  0.1192  -0.04692  0.28675]
 [ 0.       0.       0.       0.02659 -0.58888  0.01949]
 [ 0.       0.       0.       0.41641  0.1225   0.39972]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21788 -0.48172  0.19004  0.00729  1.15184]
 [ 0.       0.45816 -0.60217  0.11007 -0.07319  0.17947]
 [ 0.       0.38445  0.08007  0.02472  0.0831  -0.22967]
 [ 0.       0.      -0.      -0.02107 -0.47827  0.2594 ]
 [ 0.       0.      -0.       0.52702  0.17016  0.30475]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.45936 -0.26174 -0.13513  0.13383  1.15184]
 [ 0.       0.06318 -0.42135 -0.07927 -0.0264  -0.21198]
 [ 0.       0.56527  0.47505  0.01441 -0.13342 -0.20005]
 [ 0.       0.      -0.       0.10527 -0.40888 -0.39976]
 [ 0.       0.      -0.       0.59641  0.04381 -0.01868]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02536  0.52809 -0.00448  0.19013  1.15184]
 [ 0.       0.29151 -0.7103  -0.02894 -0.10884 -0.0552 ]
 [ 0.       0.27632  0.24672  0.10771  0.02652  0.2862 ]
 [ 0.       0.       0.       0.16939 -0.52985 -0.30087]
 [ 0.       0.       0.       0.47545 -0.02031  0.26389]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46651 -0.24877  0.10492  0.15862  1.15184]
 [ 0.       0.42269 -0.33838  0.13036 -0.0048   0.28232]
 [ 0.       0.64824  0.11554  0.03338  0.08282 -0.0725 ]
 [ 0.       0.      -0.       0.082   -0.60104 -0.09624]
 [ 0.       0.      -0.       0.40426  0.06709  0.38846]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.24845 -0.46669 -0.17588 -0.07236  1.15184]
 [ 0.       0.11576 -0.64845 -0.07419 -0.04974 -0.0723 ]
 [ 0.       0.33817  0.42247  0.10513 -0.07719 -0.28237]
 [ 0.       0.      -0.      -0.01412 -0.54594 -0.13844]
 [ 0.       0.      -0.       0.45935  0.16321 -0.37549]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17896  0.49749 -0.17846  0.06575  1.15184]
 [ 0.       0.43855 -0.63071  0.07793 -0.10394 -0.16067]
 [ 0.       0.35591  0.09968  0.04852  0.07588  0.24319]
 [ 0.       0.       0.       0.02394 -0.41794 -0.35603]
 [ 0.       0.       0.       0.58735  0.12515 -0.18277]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.48878  0.20154  0.07203 -0.17602  1.15184]
 [ 0.       0.08782 -0.37198 -0.08753 -0.0096   0.23558]
 [ 0.       0.61464  0.4504  -0.02807 -0.12825  0.17163]
 [ 0.       0.       0.       0.16337 -0.45968  0.37523]
 [ 0.       0.       0.       0.54561 -0.01428 -0.13915]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05656 -0.52567  0.05434  0.18226  1.15184]
 [ 0.       0.26572 -0.71143  0.05442  0.09493  0.03817]
 [ 0.       0.27519  0.27251 -0.11406  0.00236 -0.28896]
 [ 0.       0.       0.       0.13543 -0.58029 -0.2049 ]
 [ 0.       0.       0.       0.425    0.01366  0.34377]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.42896  0.30905  0.14528  0.12274  1.15184]
 [ 0.       0.45832 -0.38474 -0.13005  0.03148 -0.27005]
 [ 0.       0.60188  0.07991 -0.03822 -0.075    0.10968]
 [ 0.       0.      -0.       0.02774 -0.58951  0.01682]
 [ 0.       0.      -0.       0.41578  0.12134  0.39985]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28283  0.44669  0.18997  0.00889  1.15184]
 [ 0.       0.09423 -0.62371 -0.08088 -0.02991  0.09332]
 [ 0.       0.36291  0.444    0.07929 -0.10614  0.27613]
 [ 0.       0.       0.      -0.02147 -0.47988  0.25682]
 [ 0.       0.       0.       0.52541  0.17055  0.30692]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.14175 -0.50934 -0.1364   0.13253  1.15184]
 [ 0.       0.41647 -0.65416 -0.03449  0.12262  0.14228]
 [ 0.       0.33246  0.12175 -0.07279 -0.05886 -0.25439]
 [ 0.       0.       0.       0.10348 -0.40831 -0.39957]
 [ 0.       0.       0.       0.59698  0.04561 -0.02249]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51173 -0.1329   0.00592 -0.19009  1.15184]
 [ 0.       0.12742 -0.32745  0.09428 -0.00635 -0.25682]
 [ 0.       0.65917  0.41081  0.05882  0.11225 -0.13785]
 [ 0.       0.       0.       0.1698  -0.5284   0.30287]
 [ 0.       0.       0.       0.47689 -0.02071 -0.26159]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.08761  0.52139 -0.10389 -0.1593   1.15184]
 [ 0.       0.23994 -0.7095   0.07258  0.07747 -0.02098]
 [ 0.       0.27713  0.29829 -0.11319  0.03014  0.29072]
 [ 0.       0.      -0.       0.08327 -0.60093  0.09875]
 [ 0.       0.      -0.       0.40436  0.06581 -0.38782]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38744 -0.35974  0.17534  0.07365  1.15184]
 [ 0.       0.47919 -0.43451  0.12108 -0.06098  0.25441]
 [ 0.       0.55211  0.05904  0.04428  0.06819 -0.14224]
 [ 0.       0.      -0.      -0.01348 -0.54724  0.13568]
 [ 0.       0.      -0.       0.45805  0.16256  0.3765 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.31809 -0.42231  0.17907 -0.06407  1.15184]
 [ 0.       0.07547 -0.59375  0.08322  0.00729 -0.11539]
 [ 0.       0.39287  0.46276 -0.03981  0.12816 -0.26766]
 [ 0.       0.       0.       0.02236 -0.4189   0.3543 ]
 [ 0.       0.       0.       0.58639  0.12673  0.1861 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10612  0.51794 -0.07356  0.17538  1.15184]
 [ 0.       0.39282 -0.67299 -0.01003 -0.12425 -0.12431]
 [ 0.       0.31364  0.14541  0.09095  0.03432  0.26364]
 [ 0.       0.       0.       0.16261 -0.45814 -0.37643]
 [ 0.       0.       0.       0.54716 -0.01352  0.13587]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52559  0.05729 -0.05309 -0.18262  1.15184]
 [ 0.       0.18097 -0.29377 -0.09992  0.02195  0.27409]
 [ 0.       0.69286  0.35726 -0.07695 -0.09275  0.09915]
 [ 0.       0.       0.       0.13648 -0.57946  0.20724]
 [ 0.       0.       0.       0.42583  0.01261 -0.34236]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.11874 -0.51519  0.14446  0.1237   1.15184]
 [ 0.       0.21425 -0.70445  0.08524  0.05751  0.0035 ]
 [ 0.       0.28218  0.32398 -0.10547  0.05737 -0.29145]
```

```
 [ 0.       0.       0.       0.0289  -0.59013  0.01416]
 [ 0.       0.       0.       0.41516  0.12018  0.39995]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34426  0.40126  0.18989  0.01049  1.15184]
 [ 0.       0.48703 -0.48324 -0.10028  0.09188 -0.23669]
 [ 0.       0.50339  0.0512  -0.0536  -0.06014  0.17011]
 [ 0.       0.      -0.      -0.02184 -0.4815   0.25423]
 [ 0.       0.      -0.       0.52379  0.17092  0.30907]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35403  0.39267 -0.13766  0.13122  1.15184]
 [ 0.       0.06086 -0.55824  0.08009 -0.01527  0.1385 ]
 [ 0.       0.42839  0.47737  0.00861  0.13516  0.25647]
 [ 0.       0.      -0.       0.10167 -0.40777 -0.39933]
 [ 0.       0.      -0.       0.59752  0.04742 -0.02631]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.07186 -0.52379  0.00738 -0.19004  1.15184]
 [ 0.       0.3682  -0.68766 -0.04648 -0.11258  0.10672]
 [ 0.       0.29897  0.17003  0.10048  0.00765 -0.27123]
 [ 0.       0.      -0.       0.17018 -0.52694  0.30486]
 [ 0.       0.      -0.       0.47835 -0.02109 -0.25926]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.52822  0.02246  0.10286  0.15997  1.15184]
 [ 0.       0.24434 -0.27658 -0.1036   0.03882 -0.28589]
 [ 0.       0.71005  0.29389 -0.08554 -0.07371 -0.05677]
 [ 0.       0.      -0.       0.08455 -0.60081 -0.10126]
 [ 0.       0.      -0.       0.40448  0.06454  0.38718]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.15016  0.50693 -0.1748  -0.07493  1.15184]
 [ 0.       0.18877 -0.69612  0.09321  0.03468  0.01441]
 [ 0.       0.2905   0.34946 -0.08943  0.08427  0.29112]
 [ 0.       0.      -0.      -0.01281 -0.54852 -0.13292]
 [ 0.       0.      -0.       0.45677  0.1619  -0.37748]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.30107 -0.43461 -0.17966  0.06238  1.15184]
 [ 0.       0.48446 -0.52814 -0.06505  0.11878  0.21789]
 [ 0.       0.45849  0.05377 -0.06607 -0.04779 -0.1936 ]
 [ 0.       0.       0.       0.0208  -0.4199  -0.35253]
 [ 0.       0.       0.       0.5854   0.12829 -0.18942]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.39017 -0.35678 -0.0751   0.17473  1.15184]
 [ 0.       0.05227 -0.51714 -0.07322  0.03364 -0.16254]
 [ 0.       0.46948  0.48595 -0.05418 -0.12474 -0.24195]
 [ 0.       0.      -0.       0.16181 -0.4566  -0.37761]
 [ 0.       0.      -0.       0.54869 -0.01273  0.13256]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03873  0.52728  0.05185  0.18298  1.15184]
 [ 0.       0.34298 -0.69857 -0.07274 -0.09397 -0.08943]
 [ 0.       0.28805  0.19525  0.10269 -0.01796  0.27742]
 [ 0.       0.       0.       0.13752 -0.57861 -0.20957]
 [ 0.       0.       0.       0.42669  0.01157  0.34094]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51866 -0.10253 -0.14363 -0.12466  1.15184]
 [ 0.       0.31064 -0.27916 -0.10314  0.05846  0.2912 ]
 [ 0.       0.70747  0.22759 -0.08813 -0.05627  0.01264]
 [ 0.       0.       0.       0.03007 -0.59073 -0.01151]
 [ 0.       0.       0.       0.41457  0.11902 -0.40003]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.1821  -0.49635  0.1898   0.01208  1.15184]
 [ 0.       0.16368 -0.68429  0.09567  0.00843 -0.03289]
 [ 0.       0.30234  0.37455 -0.06226  0.10903 -0.28961]
 [ 0.       0.       0.      -0.02218 -0.48312  0.25164]
 [ 0.       0.       0.       0.52217  0.17126  0.31119]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25887  0.46099 -0.13891  0.1299   1.15184]
 [ 0.       0.47414 -0.56784  0.01872 -0.13277 -0.19872]
 [ 0.       0.41879  0.06409  0.07836  0.02949  0.21323]
 [ 0.       0.       0.       0.09985 -0.40727 -0.39906]
 [ 0.       0.       0.       0.59802  0.04924 -0.03013]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42565  0.3136   0.00883 -0.18998  1.15184]
 [ 0.       0.0521  -0.47108 -0.06638  0.04661  0.18717]
 [ 0.       0.51554  0.48612 -0.08808 -0.10321  0.22344]
 [ 0.       0.       0.       0.17054 -0.52547  0.30684]
 [ 0.       0.       0.       0.47982 -0.02145 -0.25692]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.0065  -0.52866 -0.10182 -0.16063  1.15184]
 [ 0.       0.31744 -0.70604 -0.09062 -0.07206  0.07234]
 [ 0.       0.28059  0.22079  0.09903 -0.04216 -0.28235]
 [ 0.       0.      -0.       0.08582 -0.60067  0.10377]
 [ 0.       0.      -0.       0.40462  0.06327 -0.38651]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49752  0.17889  0.17425  0.0762   1.15184]
 [ 0.       0.37208 -0.30099 -0.0952   0.08146 -0.28982]
 [ 0.       0.68563  0.16615 -0.08765 -0.04008  0.03102]
 [ 0.       0.      -0.      -0.01213 -0.5498   0.13016]
 [ 0.       0.      -0.       0.4555   0.16122  0.37844]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21472  0.48314  0.18024 -0.0607   1.15184]
 [ 0.       0.13926 -0.6686  -0.09045  0.02009  0.05209]
 [ 0.       0.31803  0.39897  0.0224  -0.12611  0.28678]
 [ 0.       0.       0.       0.01926 -0.42092  0.35075]
 [ 0.       0.       0.       0.58437  0.12983  0.19271]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21821 -0.48157  0.07663 -0.17406  1.15184]
 [ 0.       0.4583  -0.60191 -0.02766 -0.12927  0.17962]
 [ 0.       0.38471  0.07993  0.08635  0.00746 -0.22955]
 [ 0.       0.      -0.       0.16099 -0.45507  0.37876]
 [ 0.       0.      -0.       0.55022 -0.0119  -0.12923]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.4591  -0.2622  -0.05059 -0.18333  1.15184]
 [ 0.       0.06304 -0.42176  0.0617  -0.05629 -0.21178]
 [ 0.       0.56486  0.47519  0.10937  0.0778  -0.20026]
 [ 0.       0.       0.       0.13855 -0.57774  0.2119 ]
 [ 0.       0.       0.       0.42755  0.01053 -0.3395 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.0251   0.5281   0.1428   0.12561  1.15184]
 [ 0.       0.29172 -0.71028 -0.10203 -0.04774 -0.05534]
 [ 0.       0.27634  0.24651  0.08948 -0.06551  0.28617]
 [ 0.       0.       0.       0.03124 -0.59131  0.00885]
 [ 0.       0.       0.       0.41398  0.11784  0.4001 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.4668  -0.24824 -0.18969 -0.01366  1.15184]
 [ 0.       0.42233 -0.33803 -0.07564  0.10623  0.2824 ]
 [ 0.       0.64859  0.1159  -0.08607 -0.02397 -0.07217]
```

```
 [ 0.       0.       0.      -0.02249 -0.48473 -0.24903]
 [ 0.       0.       0.       0.52056  0.17158 -0.31328]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.24817 -0.46683  0.14015 -0.12856  1.15184]
 [ 0.       0.11594 -0.64864  0.07651 -0.04615 -0.07213]
 [ 0.       0.33799  0.42229  0.02534  0.12792 -0.28241]
 [ 0.       0.       0.       0.09801 -0.40681  0.39876]
 [ 0.       0.       0.       0.59849  0.05107  0.03395]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17927  0.49738 -0.0103   0.1899   1.15184]
 [ 0.       0.43872 -0.6305  -0.06464 -0.11271 -0.16082]
 [ 0.       0.35612  0.09951  0.08884 -0.01458  0.24309]
 [ 0.       0.       0.       0.17088 -0.52399 -0.30881]
 [ 0.       0.       0.       0.4813  -0.02179  0.25455]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.48856  0.20207  0.10078  0.16129  1.15184]
 [ 0.       0.08756 -0.37238  0.05879 -0.06549  0.2354 ]
 [ 0.       0.61425  0.45067  0.12055  0.05207  0.17189]
 [ 0.       0.      -0.       0.08709 -0.60052 -0.10627]
 [ 0.       0.      -0.       0.40477  0.062    0.38583]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05631 -0.52569  0.17369  0.07747  1.15184]
 [ 0.       0.26593 -0.71143  0.10751  0.02049  0.03831]
 [ 0.       0.27519  0.2723  -0.07245  0.0881  -0.28895]
 [ 0.       0.       0.      -0.01144 -0.55105  0.1274 ]
 [ 0.       0.       0.       0.45424  0.16053  0.37938]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42929  0.3086   0.1808  -0.05901  1.15184]
 [ 0.       0.45809 -0.38434 -0.04153  0.12717 -0.27017]
 [ 0.       0.60228  0.08014 -0.08393 -0.00679  0.10939]
 [ 0.       0.      -0.       0.01774 -0.42196  0.34893]
 [ 0.       0.      -0.       0.58333  0.13135  0.19598]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28254  0.44687  0.07817 -0.17338  1.15184]
 [ 0.       0.0944  -0.62393 -0.05753  0.06427  0.09315]
 [ 0.       0.36269  0.44383 -0.06956 -0.11274  0.27619]
 [ 0.       0.       0.       0.16014 -0.45355  0.37988]
 [ 0.       0.       0.       0.55174 -0.01105 -0.12589]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.14205 -0.50926 -0.04934 -0.18367  1.15184]
 [ 0.       0.41666 -0.65399 -0.09036 -0.08982  0.14243]
 [ 0.       0.33264  0.12157  0.0868  -0.03497 -0.2543 ]
 [ 0.       0.      -0.       0.13958 -0.57685  0.21422]
 [ 0.       0.      -0.       0.42844  0.00951 -0.33804]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51157 -0.13349 -0.14197 -0.12655  1.15184]
 [ 0.       0.12703 -0.32778  0.0555  -0.0764  -0.25666]
 [ 0.       0.65884  0.4112   0.1239   0.02685 -0.13815]
 [ 0.       0.       0.       0.03242 -0.59187 -0.0062 ]
 [ 0.       0.       0.       0.41342  0.11667 -0.40015]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.08736  0.52143 -0.18957 -0.01524  1.15184]
 [ 0.       0.24015 -0.70953  0.1057  -0.01008 -0.02113]
 [ 0.       0.2771   0.29808 -0.04534  0.10798  0.29071]
 [ 0.       0.      -0.      -0.02277 -0.48635 -0.24642]
 [ 0.       0.      -0.       0.51894  0.17186 -0.31534]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.38779 -0.35936 -0.14137  0.12721  1.15184]
 [ 0.       0.47907 -0.4341   0.00362  0.13551  0.25455]
 [ 0.       0.55252  0.05916 -0.08055  0.01111 -0.14199]
 [ 0.       0.       0.       0.09617 -0.40637 -0.39841]
 [ 0.       0.       0.       0.59892  0.05291 -0.03777]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.3178  -0.42253  0.01176 -0.18982  1.15184]
 [ 0.       0.07561 -0.59402  0.03946 -0.07366 -0.1152 ]
 [ 0.       0.3926   0.46262  0.10209  0.08708 -0.26774]
 [ 0.       0.       0.       0.1712  -0.5225   0.31077]
 [ 0.       0.       0.       0.48279 -0.02211 -0.25216]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.1064   0.51788 -0.09973 -0.16193  1.15184]
 [ 0.       0.39302 -0.67285  0.1069   0.06417 -0.12446]
 [ 0.       0.31377  0.14521 -0.0808   0.05399  0.26357]
 [ 0.       0.      -0.       0.08835 -0.60035  0.10877]
 [ 0.       0.      -0.       0.40495  0.06073 -0.38514]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52552  0.05793  0.17312  0.07873  1.15184]
 [ 0.       0.18048 -0.29398  0.04832 -0.09009  0.27397]
 [ 0.       0.69264  0.35775  0.12056  0.00196  0.09949]
 [ 0.       0.      -0.      -0.01073 -0.5523   0.12464]
 [ 0.       0.      -0.       0.453    0.15981  0.38029]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.11848 -0.51525  0.18134 -0.05732  1.15184]
 [ 0.       0.21446 -0.7045   0.09386 -0.04206  0.00365]
 [ 0.       0.28212  0.32377 -0.00695  0.11984 -0.29145]
 [ 0.       0.       0.       0.01624 -0.42304  0.34709]
 [ 0.       0.       0.       0.58225  0.13284  0.19923]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34462  0.40095 -0.0797   0.17268  1.15184]
 [ 0.       0.48701 -0.48285 -0.04856 -0.12705 -0.23684]
 [ 0.       0.50378  0.05122  0.07561 -0.0278   0.1699 ]
 [ 0.       0.       0.       0.15925 -0.45205 -0.38098]
 [ 0.       0.       0.       0.55324 -0.01017  0.12252]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35373  0.39294 -0.04807 -0.18401  1.15184]
 [ 0.       0.06095 -0.55855 -0.02536  0.0775   0.13831]
 [ 0.       0.42807  0.47728 -0.12241 -0.05791  0.25657]
 [ 0.       0.       0.       0.14059 -0.57595  0.21653]
 [ 0.       0.       0.       0.42934  0.0085  -0.33656]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.07213 -0.52376  0.14113  0.12748  1.15184]
 [ 0.       0.3684  -0.68755  0.11624  0.03646  0.10687]
 [ 0.       0.29907  0.16983 -0.07022  0.07223 -0.27118]
 [ 0.       0.       0.       0.0336  -0.59242  0.00356]
 [ 0.       0.       0.       0.41287  0.11548  0.40018]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.52825  0.0218   0.18944  0.01681  1.15184]
 [ 0.       0.2438  -0.27664 -0.03271  0.10561 -0.28582]
 [ 0.       0.70998  0.29443 -0.11071  0.02252 -0.05713]
 [ 0.       0.      -0.      -0.02303 -0.48797  0.24379]
 [ 0.       0.      -0.       0.51732  0.17212  0.31737]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.1499   0.507    0.14259 -0.12585  1.15184]
 [ 0.       0.18898 -0.69621 -0.07125  0.06942  0.01426]
 [ 0.       0.29042  0.34925 -0.03788 -0.11687  0.29113]
```

```
 [ 0.       0.       0.       0.09432 -0.40598  0.39803]
 [ 0.       0.       0.       0.59931  0.05476  0.04159]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.30142 -0.43436 -0.01323  0.18972  1.15184]
 [ 0.       0.48452 -0.52779  0.08372  0.10646  0.21805]
 [ 0.       0.45884  0.05371 -0.06984  0.04206 -0.19342]
 [ 0.       0.       0.       0.17149 -0.521   -0.31271]
 [ 0.       0.       0.       0.48429 -0.02241  0.24975]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.38987 -0.3571   0.09868  0.16258  1.15184]
 [ 0.       0.05231 -0.5175  -0.01501  0.07917 -0.16234]
 [ 0.       0.46912  0.48591 -0.13304 -0.02823 -0.24208]
 [ 0.       0.      -0.       0.08962 -0.60016 -0.11126]
 [ 0.       0.      -0.       0.40513  0.05947  0.38442]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.039    0.52726 -0.17255 -0.07998  1.15184]
 [ 0.       0.34319 -0.69849  0.1187   0.00606 -0.08958]
 [ 0.       0.28813  0.19504 -0.05327  0.08958  0.27737]
 [ 0.       0.      -0.      -0.01    -0.55353 -0.12189]
 [ 0.       0.      -0.       0.45177  0.15909 -0.38119]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51879 -0.10188  0.18186 -0.05563  1.15184]
 [ 0.       0.31011 -0.27905  0.00503 -0.11839  0.29118]
 [ 0.       0.70757  0.22812  0.09446 -0.04501  0.013  ]
 [ 0.       0.      -0.       0.01477 -0.42414  0.34521]
 [ 0.       0.      -0.       0.58115  0.13432  0.20245]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18183 -0.49645 -0.08122  0.17197  1.15184]
 [ 0.       0.16389 -0.6844  -0.04307  0.08588 -0.03274]
 [ 0.       0.30222  0.37434 -0.07841 -0.09803 -0.28963]
 [ 0.       0.      -0.       0.15834 -0.45055 -0.38205]
 [ 0.       0.      -0.       0.55474 -0.00926  0.11914]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25921  0.4608  -0.04681 -0.18433  1.15184]
 [ 0.       0.47425 -0.56753  0.10733  0.08039 -0.19888]
 [ 0.       0.41909  0.06398 -0.06355  0.05448  0.21308]
 [ 0.       0.      -0.       0.14159 -0.57504  0.21884]
 [ 0.       0.      -0.       0.43025  0.00749 -0.33506]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42537  0.31399  0.14028  0.12841  1.15184]
 [ 0.       0.05206 -0.47148  0.00657 -0.08083  0.18697]
 [ 0.       0.51514  0.48617  0.13568 -0.00166  0.22361]
 [ 0.       0.      -0.       0.03479 -0.59296  0.00091]
 [ 0.       0.      -0.       0.41233  0.11429  0.4002 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00676 -0.52866 -0.18929 -0.01838  1.15184]
 [ 0.       0.31765 -0.70599 -0.11255  0.02726  0.07248]
 [ 0.       0.28063  0.22058  0.02754 -0.10402 -0.28232]
 [ 0.       0.      -0.      -0.02326 -0.48958 -0.24116]
 [ 0.       0.      -0.       0.51571  0.17235 -0.31938]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49773  0.17829  0.14379 -0.12448  1.15184]
 [ 0.       0.37161 -0.30074  0.03264  0.12092 -0.28986]
 [ 0.       0.68588  0.16662 -0.07387  0.06201  0.03067]
 [ 0.       0.      -0.       0.09246 -0.40561  0.39761]
 [ 0.       0.      -0.       0.59968  0.05662  0.04541]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023  0.21444  0.48326  0.0147  -0.18961  1.15184]
 [ 0.       0.13946 -0.66874 -0.01686  0.09113  0.05193]
 [ 0.       0.31788  0.39877 -0.10728 -0.06992  0.28681]
 [ 0.       0.       0.       0.17177 -0.51949  0.31464]
 [ 0.       0.       0.       0.4858  -0.02268 -0.24731]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21853 -0.48142 -0.09762 -0.16321  1.15184]
 [ 0.       0.45845 -0.60166 -0.12148 -0.05219  0.17978]
 [ 0.       0.38497  0.07978  0.05597 -0.06614 -0.22943]
 [ 0.       0.      -0.       0.09088 -0.59996  0.11375]
 [ 0.       0.      -0.       0.40534  0.0582  -0.38369]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.45884 -0.26266  0.17196  0.08123  1.15184]
 [ 0.       0.06289 -0.42218  0.00255  0.08345 -0.21158]
 [ 0.       0.56445  0.47534 -0.1303   0.03224 -0.20048]
 [ 0.       0.      -0.      -0.00925 -0.55474  0.11913]
 [ 0.       0.      -0.       0.45055  0.15834  0.38206]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02484  0.52812 -0.18237  0.05394  1.15184]
 [ 0.       0.29193 -0.71026  0.09475 -0.06097 -0.05549]
 [ 0.       0.27636  0.2463   0.00782  0.1106   0.28615]
 [ 0.       0.      -0.       0.01332 -0.42526 -0.34332]
 [ 0.       0.      -0.       0.58003  0.13576 -0.20565]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46709 -0.2477  -0.08275  0.17124  1.15184]
 [ 0.       0.42197 -0.33768  0.07085  0.10944  0.28248]
 [ 0.       0.64895  0.11626 -0.05398  0.07126 -0.07185]
 [ 0.       0.       0.       0.1574  -0.44907 -0.3831 ]
 [ 0.       0.       0.       0.55622 -0.00832  0.11574]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.24789 -0.46698  0.04554  0.18465  1.15184]
 [ 0.       0.11613 -0.64882  0.00385  0.08929 -0.07196]
 [ 0.       0.3378   0.4221  -0.12438 -0.03911 -0.28245]
 [ 0.       0.      -0.       0.14258 -0.57411 -0.22115]
 [ 0.       0.      -0.       0.43119  0.0065   0.33355]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.17958  0.49727  0.13943  0.12934  1.15184]
 [ 0.       0.4389  -0.63029 -0.12802 -0.02232 -0.16098]
 [ 0.       0.35634  0.09933  0.04552 -0.07764  0.24299]
 [ 0.       0.       0.       0.03599 -0.59347 -0.00173]
 [ 0.       0.       0.       0.41182  0.1131   0.4002 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.48834  0.20261  0.18913  0.01994  1.15184]
 [ 0.       0.0873  -0.37277 -0.01551 -0.08659  0.23521]
 [ 0.       0.61385  0.45093  0.11525 -0.063    0.17215]
 [ 0.       0.      -0.      -0.02346 -0.4912   0.23852]
 [ 0.       0.      -0.       0.51409  0.17255  0.32135]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05605 -0.52572 -0.14498  0.12309  1.15184]
 [ 0.       0.26614 -0.71144 -0.06496  0.08812  0.03846]
 [ 0.       0.27519  0.27209 -0.04812 -0.10338 -0.28893]
 [ 0.       0.      -0.       0.0906  -0.40529 -0.39716]
 [ 0.       0.      -0.       0.6      0.05849 -0.04923]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42962  0.30814 -0.01618  0.18949  1.15184]
 [ 0.       0.45786 -0.38393 -0.10094 -0.08777 -0.27029]
 [ 0.       0.60269  0.08037  0.03889 -0.07473  0.1091 ]
```

```
 [ 0.       0.       0.       0.17202 -0.51798 -0.31655]
 [ 0.       0.       0.       0.48732 -0.02293  0.24485]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28225  0.44705  0.09656  0.16385  1.15184]
 [ 0.       0.09457 -0.62416 -0.01942 -0.08407  0.09297]
 [ 0.       0.36246  0.44366  0.13221  0.00814  0.27625]
 [ 0.       0.      -0.       0.09215 -0.59974 -0.11624]
 [ 0.       0.      -0.       0.40556  0.05694  0.38295]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.14235 -0.50918  0.17137  0.08246  1.15184]
 [ 0.       0.41685 -0.65381  0.12704 -0.00991  0.14258]
 [ 0.       0.33281  0.12138 -0.03006  0.08859 -0.25422]
 [ 0.       0.       0.      -0.0085  -0.55594  0.11638]
 [ 0.       0.       0.       0.44935  0.15758  0.3829 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51141 -0.13409  0.18287 -0.05225  1.15184]
 [ 0.       0.12665 -0.32811  0.03496  0.08766 -0.25649]
 [ 0.       0.65851  0.41158 -0.08893  0.09041 -0.13845]
 [ 0.       0.      -0.       0.0119  -0.42641  0.3414 ]
 [ 0.       0.      -0.       0.57888  0.13719  0.20883]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.0871   0.52148 -0.08427  0.17049  1.15184]
 [ 0.       0.24036 -0.70955  0.02969 -0.10198 -0.02127]
 [ 0.       0.27707  0.29787  0.08357  0.08201  0.2907 ]
 [ 0.       0.      -0.       0.15643 -0.4476  -0.38412]
 [ 0.       0.      -0.       0.55769 -0.00735  0.11232]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.38815 -0.35898  0.04426  0.18496  1.15184]
 [ 0.       0.47895 -0.43369  0.12082  0.06146  0.25469]
 [ 0.       0.55293  0.05928 -0.02859  0.07614 -0.14174]
 [ 0.       0.       0.       0.14357 -0.57316 -0.22345]
 [ 0.       0.       0.       0.43213  0.00552  0.33201]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.3175  -0.42275  0.13858  0.13025  1.15184]
 [ 0.       0.07575 -0.59429  0.03159  0.07739 -0.11502]
 [ 0.       0.39233  0.46248 -0.13218  0.02302 -0.26782]
 [ 0.       0.      -0.       0.03719 -0.59397 -0.00436]
 [ 0.       0.      -0.       0.41132  0.1119   0.40018]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10669  0.51782  0.18896  0.02149  1.15184]
 [ 0.       0.39322 -0.67271 -0.11648  0.04455 -0.12461]
 [ 0.       0.31391  0.14501  0.00734 -0.09687  0.2635 ]
 [ 0.       0.       0.      -0.02364 -0.49281  0.23587]
 [ 0.       0.       0.       0.51248  0.17272  0.3233 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52545  0.05858  0.14615 -0.12169  1.15184]
 [ 0.       0.17999 -0.2942  -0.0602  -0.08254  0.27385]
 [ 0.       0.69242  0.35824  0.0541  -0.10782  0.09983]
 [ 0.       0.      -0.       0.08872 -0.405    0.39667]
 [ 0.       0.      -0.       0.60029  0.06036  0.05304]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.11822 -0.51531 -0.01766  0.18936  1.15184]
 [ 0.       0.21467 -0.70455  0.00233  0.10286  0.00379]
 [ 0.       0.28207  0.32356 -0.10772 -0.05292 -0.29145]
 [ 0.       0.      -0.       0.17224 -0.51645 -0.31846]
 [ 0.       0.      -0.       0.48884 -0.02316  0.24237]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.34498  0.40064 -0.0955  -0.16447  1.15184]
 [ 0.       0.48699 -0.48246  0.13189  0.03322 -0.23699]
 [ 0.       0.50416  0.05124 -0.0206   0.07788  0.16968]
 [ 0.       0.      -0.       0.0934  -0.5995   0.11872]
 [ 0.       0.      -0.       0.40579  0.05568 -0.38218]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35343  0.3932  -0.17078 -0.08369  1.15184]
 [ 0.       0.06105 -0.55887  0.04225  0.06976  0.13811]
 [ 0.       0.42775  0.47718 -0.12375  0.05499  0.25668]
 [ 0.       0.       0.      -0.00772 -0.55713 -0.11362]
 [ 0.       0.       0.       0.44816  0.15681 -0.38373]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.07241 -0.52372 -0.18334  0.05055  1.15184]
 [ 0.       0.36861 -0.68745 -0.09318  0.07851  0.10701]
 [ 0.       0.29918  0.16962 -0.02325 -0.09799 -0.27112]
 [ 0.       0.      -0.       0.0105  -0.42758 -0.33945]
 [ 0.       0.      -0.       0.57771  0.13859 -0.21198]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.52828  0.02113  0.08579 -0.16973  1.15184]
 [ 0.       0.24325 -0.2767   0.08601  0.06937 -0.28575]
 [ 0.       0.70992  0.29498 -0.01999  0.11127 -0.05749]
 [ 0.       0.      -0.       0.15544 -0.44615  0.3851 ]
 [ 0.       0.      -0.       0.55915 -0.00635 -0.10888]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.14964  0.50708 -0.04298 -0.18526  1.15184]
 [ 0.       0.18919 -0.69629  0.02729  0.09569  0.01411]
 [ 0.       0.29033  0.34904 -0.12082 -0.02216  0.29113]
 [ 0.       0.       0.       0.14454 -0.5722   0.22574]
 [ 0.       0.       0.       0.4331   0.00455 -0.33045]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.30177 -0.43412  0.13772  0.13116  1.15184]
 [ 0.       0.48457 -0.52744  0.1354   0.00332  0.21821]
 [ 0.       0.45919  0.05366 -0.01202  0.08062 -0.19324]
 [ 0.       0.       0.       0.03839 -0.59446 -0.007  ]
 [ 0.       0.       0.       0.41083  0.11069  0.40014]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.38958 -0.35743 -0.18878 -0.02304  1.15184]
 [ 0.       0.05236 -0.51786 -0.05301 -0.06069 -0.16214]
 [ 0.       0.46876  0.48587  0.10444 -0.08711 -0.24222]
 [ 0.       0.       0.      -0.02378 -0.49442 -0.23321]
 [ 0.       0.       0.       0.51087  0.17287 -0.32522]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.03927  0.52724  0.14732 -0.12028  1.15184]
 [ 0.       0.3434  -0.69842 -0.05727  0.10417 -0.08972]
 [ 0.       0.28821  0.19483 -0.05735 -0.08699  0.27732]
 [ 0.       0.       0.       0.08684 -0.40474  0.39614]
 [ 0.       0.       0.       0.60055  0.06224  0.05685]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.51892 -0.10123 -0.01914  0.18922  1.15184]
 [ 0.       0.30957 -0.27895  0.10719  0.05036  0.29117]
 [ 0.       0.70767  0.22866  0.00505  0.10458  0.01337]
 [ 0.       0.       0.       0.17245 -0.51492 -0.32035]
 [ 0.       0.       0.       0.49037 -0.02336  0.23987]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.18156 -0.49655 -0.09442 -0.16509  1.15184]
 [ 0.       0.16409 -0.68451 -0.04578 -0.08449 -0.03258]
 [ 0.       0.30211  0.37414  0.12525 -0.00816 -0.28965]
```

```
 [ 0.       0.       0.       0.09466 -0.59924  0.12121]
 [ 0.       0.       0.       0.40605  0.05442 -0.3814 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.25955  0.4606  -0.17017 -0.08491  1.15184]
 [ 0.       0.47436 -0.56723  0.13097 -0.02886 -0.19904]
 [ 0.       0.4194   0.06387 -0.00013  0.08368  0.21293]
 [ 0.       0.      -0.      -0.00693 -0.5583  -0.11087]
 [ 0.       0.      -0.       0.44699  0.15602 -0.38454]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.42508  0.31437  0.1838  -0.04885  1.15184]
 [ 0.       0.05202 -0.47188 -0.06458 -0.04903  0.18676]
 [ 0.       0.51475  0.4862   0.07173 -0.11519  0.22378]
 [ 0.       0.      -0.       0.00912 -0.42878  0.33748]
 [ 0.       0.      -0.       0.57651  0.13997  0.2151 ]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.00702 -0.52865 -0.08731  0.16896  1.15184]
 [ 0.       0.31786 -0.70594 -0.01641  0.11466  0.07263]
 [ 0.       0.28068  0.22037 -0.08638 -0.06412 -0.28228]
 [ 0.       0.      -0.       0.15441 -0.44471 -0.38606]
 [ 0.       0.      -0.       0.56059 -0.00533  0.10542]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.49795  0.17768  0.04169  0.18556  1.15184]
 [ 0.       0.37114 -0.30049 -0.12196 -0.02831 -0.28989]
 [ 0.       0.68613  0.16709 -0.01986 -0.09445  0.03032]
 [ 0.       0.       0.       0.14549 -0.57122 -0.22803]
 [ 0.       0.       0.       0.43407  0.00359  0.32888]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21417  0.48338 -0.13685 -0.13206  1.15184]
 [ 0.       0.13965 -0.66889  0.05961  0.07101  0.05176]
 [ 0.       0.31773  0.39858 -0.12219  0.03825  0.28684]
 [ 0.       0.       0.       0.0396  -0.59492  0.00963]
 [ 0.       0.       0.       0.41037  0.10948 -0.40008]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.21886 -0.48127  0.18859  0.02458  1.15184]
 [ 0.       0.4586  -0.6014   0.11625 -0.06303  0.17994]
 [ 0.       0.38523  0.07963  0.01729  0.08488 -0.22931]
 [ 0.       0.       0.      -0.02391 -0.49603  0.23055]
 [ 0.       0.       0.       0.50927  0.17299  0.32712]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.45858 -0.26312 -0.14846  0.11886  1.15184]
 [ 0.       0.06275 -0.42259 -0.07613 -0.03421 -0.21138]
 [ 0.       0.56403  0.47548  0.02815 -0.13126 -0.20069]
 [ 0.       0.       0.       0.08496 -0.40453 -0.39558]
 [ 0.       0.       0.       0.60077  0.06413 -0.06066]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.02458  0.52813 -0.02063  0.18906  1.15184]
 [ 0.       0.29215 -0.71024 -0.01973 -0.11096 -0.05563]
 [ 0.       0.27639  0.24608  0.10504  0.03542  0.28612]
 [ 0.       0.      -0.       0.17263 -0.51338 -0.32222]
 [ 0.       0.      -0.       0.49191 -0.02354  0.23734]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.46737 -0.24716 -0.09335 -0.1657   1.15184]
 [ 0.       0.42161 -0.33733 -0.13027 -0.00421  0.28256]
 [ 0.       0.6493   0.11662 -0.02784 -0.085   -0.07152]
 [ 0.       0.      -0.       0.09592 -0.59898  0.12368]
 [ 0.       0.      -0.       0.40632  0.05317 -0.38061]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

```
[[ 2.78023 -0.24761 -0.46713 -0.16956 -0.08613  1.15184]
 [ 0.       0.11632 -0.649   -0.07019 -0.05537 -0.0718 ]
 [ 0.       0.33762  0.42191  0.11081 -0.06868 -0.28249]
 [ 0.       0.       0.      -0.00613 -0.55946 -0.10812]
 [ 0.       0.       0.       0.44583  0.15521 -0.38532]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.1799   0.49715  0.18424 -0.04716  1.15184]
 [ 0.       0.43907 -0.63007 -0.08807  0.09558 -0.16113]
 [ 0.       0.35655  0.09916 -0.04067 -0.08026  0.24289]
 [ 0.       0.       0.       0.00777 -0.43     0.33548]
 [ 0.       0.       0.       0.57529  0.14132  0.21821]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.48812  0.20314  0.08883 -0.16816  1.15184]
 [ 0.       0.08704 -0.37317 -0.08613 -0.01765  0.23502]
 [ 0.       0.61346  0.45119 -0.01572 -0.13043  0.1724 ]
 [ 0.       0.      -0.       0.15336 -0.44328  0.387  ]
 [ 0.       0.      -0.       0.56201 -0.00427 -0.10195]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.05579 -0.52575 -0.0404  -0.18584  1.15184]
 [ 0.       0.26636 -0.71144 -0.04725 -0.09878  0.0386 ]
 [ 0.       0.27518  0.27187  0.11384  0.00613 -0.28891]
 [ 0.       0.       0.       0.14644 -0.57022  0.23031]
 [ 0.       0.       0.       0.43507  0.00265 -0.32729]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.42995  0.30768  0.13598  0.13296  1.15184]
 [ 0.       0.45762 -0.38353 -0.13189  0.0222  -0.2704 ]
 [ 0.       0.60309  0.08061 -0.0331  -0.0775   0.10882]
 [ 0.       0.       0.       0.04082 -0.59538 -0.01225]
 [ 0.       0.       0.       0.40992  0.10827  0.40001]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.28197  0.44723  0.18838  0.02611  1.15184]
 [ 0.       0.09474 -0.62438 -0.078   -0.03695  0.09279]
 [ 0.       0.36224  0.44349  0.08846 -0.09857  0.27631]
 [ 0.       0.      -0.      -0.024   -0.49763  0.22788]
 [ 0.       0.      -0.       0.50766  0.17309  0.32898]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.14265 -0.50909  0.1496  -0.11743  1.15184]
 [ 0.       0.41704 -0.65364  0.0471  -0.11842  0.14273]
 [ 0.       0.33298  0.12119  0.06627  0.06599 -0.25414]
 [ 0.       0.       0.       0.08307 -0.40434  0.39497]
 [ 0.       0.       0.       0.60095  0.06602  0.06446]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.51126 -0.13469  0.02212 -0.18889  1.15184]
 [ 0.       0.12626 -0.32845  0.0943   0.00131 -0.25633]
 [ 0.       0.65818  0.41197  0.04936  0.11687 -0.13875]
 [ 0.       0.      -0.       0.17279 -0.51183  0.32409]
 [ 0.       0.      -0.       0.49346 -0.0237  -0.23479]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.08684  0.52152 -0.09227 -0.1663   1.15184]
 [ 0.       0.24057 -0.70958  0.06704  0.08242 -0.02141]
 [ 0.       0.27704  0.29766 -0.11495  0.02211  0.29069]
 [ 0.       0.       0.       0.09717 -0.59869  0.12616]
 [ 0.       0.       0.       0.4066   0.05192 -0.37979]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.3885  -0.3586  -0.16894 -0.08733  1.15184]
 [ 0.       0.47884 -0.43328 -0.12542  0.05138  0.25483]
 [ 0.       0.55335  0.05939 -0.03909 -0.07134 -0.14149]
```

```
 [ 0.       0.      -0.      -0.00531 -0.56061 -0.10537]
 [ 0.       0.      -0.       0.44468  0.1544  -0.38608]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.31721 -0.42297  0.18467 -0.04546  1.15184]
 [ 0.       0.07589 -0.59456  0.08216  0.01549 -0.11483]
 [ 0.       0.39206  0.46234 -0.05251  0.12346 -0.2679 ]
 [ 0.       0.      -0.       0.00644 -0.43124  0.33346]
 [ 0.       0.      -0.       0.57405  0.14264  0.22128]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.10698  0.51776 -0.09034  0.16736  1.15184]
 [ 0.       0.39342 -0.67257  0.002   -0.12471 -0.12475]
 [ 0.       0.31405  0.14481  0.08717  0.04283  0.26343]
 [ 0.       0.      -0.       0.15228 -0.44187 -0.3879 ]
 [ 0.       0.      -0.       0.56342 -0.00319  0.09846]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.52537  0.05922  0.0391   0.18612  1.15184]
 [ 0.       0.1795  -0.29442  0.10104 -0.01467  0.27372]
 [ 0.       0.6922   0.35873  0.07007  0.09826  0.10016]
 [ 0.       0.       0.       0.14738 -0.56921 -0.23259]
 [ 0.       0.       0.       0.43608  0.00171  0.32567]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.11797 -0.51537 -0.13511 -0.13385  1.15184]
 [ 0.       0.21488 -0.70461 -0.08101 -0.06347  0.00394]
 [ 0.       0.28201  0.32335  0.10923 -0.04967 -0.29145]
 [ 0.       0.       0.       0.04204 -0.59581  0.01488]
 [ 0.       0.       0.       0.40948  0.10705 -0.39992]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023 -0.34534  0.40033 -0.18816 -0.02764  1.15184]
 [ 0.       0.48697 -0.48207  0.10806 -0.0826  -0.23714]
 [ 0.       0.50455  0.05126  0.04823  0.06453  0.16947]
 [ 0.       0.      -0.      -0.02407 -0.49923 -0.2252 ]
 [ 0.       0.      -0.       0.50606  0.17316 -0.33082]
 [ 0.       0.       0.       0.       0.      -0.11832]]


[[ 2.78023  0.35313  0.39347 -0.15072  0.11598  1.15184]
 [ 0.       0.06115 -0.55918  0.08127 -0.00705  0.13792]
 [ 0.       0.42744  0.47708 -0.0055   0.13529  0.25678]
 [ 0.       0.       0.       0.08118 -0.4042  -0.39434]
 [ 0.       0.       0.       0.60109  0.06791 -0.06826]
 [ 0.       0.       0.       0.       0.      -0.11832]]
```

Let's compare to the eigenvalues:

In [469]:

```
np.linalg.eigvals(A)
```

Out[469]:

```
array([ 2.78023+0.j     , -0.11832+0.j     ,  0.26911+0.44246j,
        0.26911-0.44246j,  0.07454+0.49287j,  0.07454-0.49287j])
```

Check that Q is orthogonal:

In [476]:

```
np.allclose(np.eye(n), Q @ Q.T), np.allclose(np.eye(n), Q.T @ Q)
```

Out[476]:

```
(True, True)
```

This is really really slow.

Practical QR (QR with shifts) ¶

Idea: Instead of factoring A_k as $Q_k R_k$,

1. Get the QR factorization

$$A_k - s_k I = Q_k R_k$$

2. Set

$$A_{k+1} = R_k Q_k + s_k I$$

Choose s_k to approximate an eigenvalue of A. We'll use $s_k = A_k(m, m)$.

The idea of adding shifts to speed up convergence shows up in many algorithms in numerical linear algebra (including the power method, inverse iteration, and Rayleigh quotient iteration).

Homework: Add shifts to the QR algorithm ¶

In [460]:

```
#Exercise: Add shifts to the QR algorithm
#Exercise: def practical_qr(A, iters=10):
#Exercise:     return Ak, Q
```

Practical QR ¶

In [204]:

```
Ak, Q = practical_qr(A, 10)
```

```
[ 5.16264  2.53603  0.31923  0.35315  0.97569  0.43615]
[ 7.99381  0.05922  0.34478  0.29482  0.79026  0.29999]
[ 8.00491  0.04358  0.89735  0.26386  0.26182  0.31135]
[ 8.00493  0.13648  0.91881  0.14839  0.24313  0.33115]
[ 8.00493  0.43377  0.62809  0.13429  0.24592  0.33589]
[ 8.00493  0.81058  0.25128  0.13297  0.24722  0.3359 ]
[ 8.00493  0.98945  0.07221  0.13292  0.24747  0.3359 ]
[ 8.00493  1.0366   0.02497  0.13296  0.24751  0.3359 ]
[ 8.00493  1.04688  0.01465  0.13299  0.24752  0.3359 ]
[ 8.00493  1.04902  0.0125   0.13301  0.24753  0.3359 ]
```

Check that Q is orthogonal:

In [201]:

```
np.allclose(np.eye(n), Q @ Q.T), np.allclose(np.eye(n), Q.T @ Q)
```

Out[201]:

```
(True, True)
```

Let's compare to the eigenvalues:

In [196]:

```
np.linalg.eigvals(A)
```

Out[196]:

```
array([ 2.68500+0.j     ,  0.19274+0.41647j,  0.19274-0.41647j,
       -0.35988+0.43753j, -0.35988-0.43753j, -0.18346+0.j     ])
```

Problem : This is better than the unshifted version (which wasn't even guaranteed to converge), but is still really slow! In fact, it is $\mathcal{O}(n^4)$, which is awful.

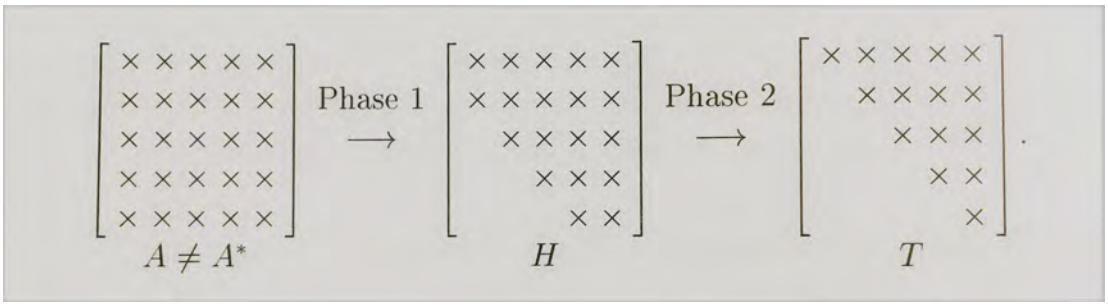
In the case of symmetric matrices, it's $\mathcal{O}(n^3)$

However, if you start with a Hessenberg matrix (zeros below the first subdiagonal), it's faster: $\mathcal{O}(n^3)$, and $\mathcal{O}(n^2)$ if symmetric.

A Two-Phase Approach ¶

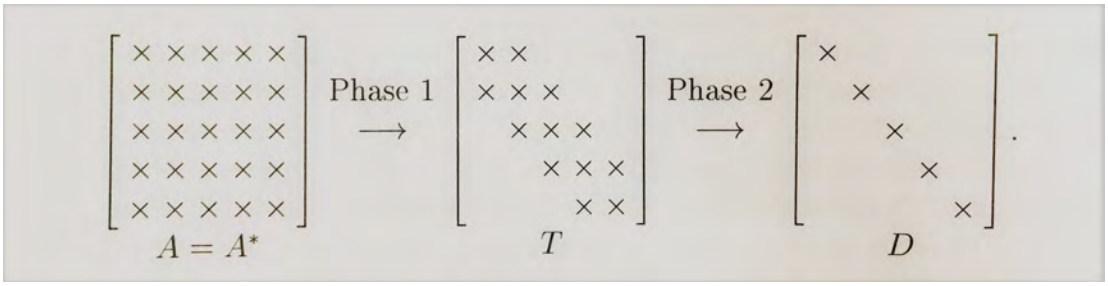
In practice, a two phase approach is used to find eigenvalues:

1. Reduce the matrix to Hessenberg form (zeros below the first subdiagonal)
2. Iterative process that causes Hessenberg to converge to a triangular matrix. The eigenvalues of a triangular matrix are the values on the diagonal, so we are finished!



(source: Trefethen, Lecture 25)

In the case of a Hermitian matrix, this approach is even faster, since the intermediate step is also Hermitian (and a Hermitian Hessenberg is tridiagonal).



(source: Trefethen, Lecture 25)

Phase 1 reaches an exact solution in a finite number of steps, whereas Phase 2 theoretically never reaches the exact solution.

We've already done step 2: the QR algorithm. Remember that it would be possible to just use the QR algorithm, but ridiculously slow.

Arnoldi Iteration ¶

We can use the Arnoldi iteration for phase 1 (and the QR algorithm for phase 2).

Initializations ¶

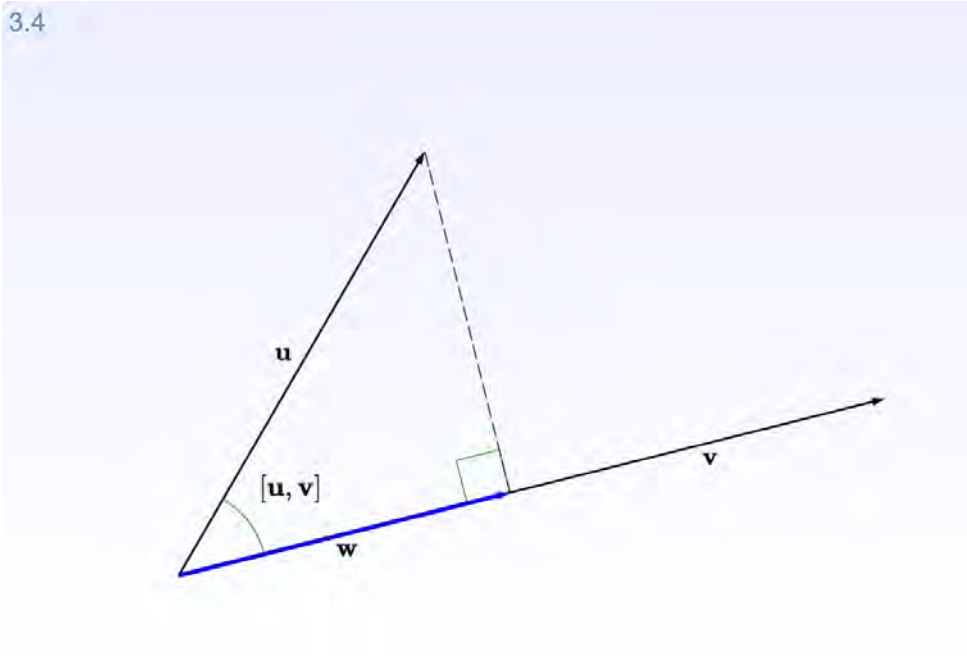
In [459]:

```
import numpy as np
n = 5
A0 = np.random.rand(n,n)  #.astype(np.float64)
A = A0 @ A0.T

np.set_printoptions(precision=5, suppress=True)
```

Linear Algebra Review: Projections ¶

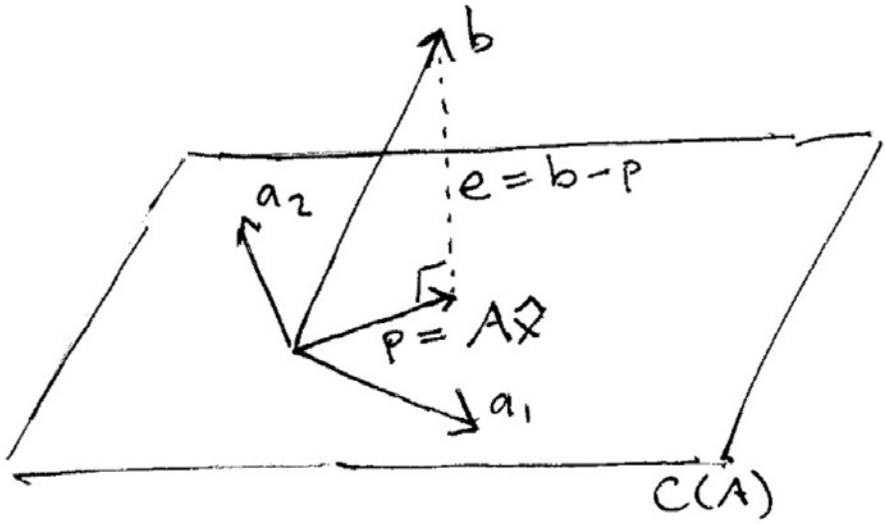
When vector $\mathbf{b}$ is projected onto a line $\mathbf{a}$, its projection $\mathbf{p}$ is the part of $\mathbf{b}$ along that line $\mathbf{a}$.

Let's look at interactive graphic (3.4) for section 3.2.2: Projections of the Immersive Linear Algebra online book.



(source: Immersive Math)

And here is what it looks like to project a vector onto a plane:



(source: The Linear Algebra View of Least-Squares Regression)

When vector $\mathbf{b}$ is projected onto a line $\mathbf{a}$, its projection $\mathbf{p}$ is the part of $\mathbf{b}$ along that line $\mathbf{a}$. So $\mathbf{p}$ is some multiple of $\mathbf{a}$. Let $\mathbf{p} = \hat{x}\mathbf{a}$ where $\hat{x}$ is a scalar.

Orthogonality ¶

The key to projection is orthogonality: The line from $\mathbf{b}$ to $\mathbf{p}$ (which can be written $\mathbf{b} - \hat{x}\mathbf{a}$) is perpendicular to $\mathbf{a}$.

This means that

$$\mathbf{a} \cdot (\mathbf{b} - \hat{x}\mathbf{a}) = 0$$

and so

$$\hat{x} = \frac{\mathbf{a} \cdot \mathbf{b}}{\mathbf{a} \cdot \mathbf{a}}$$

The Algorithm ¶

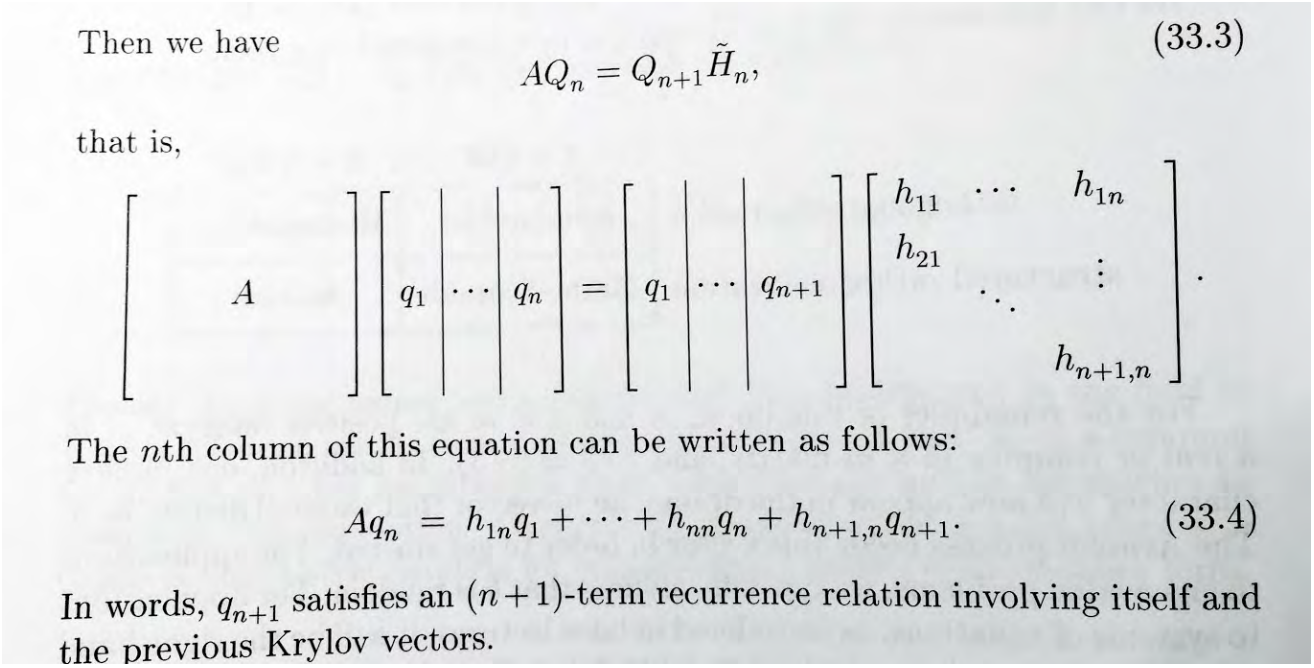
Motivation :

We want orthonormal columns in Q and a Hessenberg H such that $AQ = QH$.

Thinking about it iteratively,

$$AQ_n = Q_{n+1}H_n$$

where Q_{n+1} is $n \times n+1$ and H_n is $n+1 \times n$. This creates a solvable recurrence relation.



Then we have

$$AQ_n = Q_{n+1}\tilde{H}_n, \tag{33.3}$$

that is,

$$\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} \left[\begin{array}{c|c|c} q_1 & \cdots & q_n \end{array}\right] = \left[\begin{array}{c|c|c} q_1 & \cdots & q_{n+1} \end{array}\right] \begin{bmatrix} h_{11} & \cdots & h_{1n} \\ h_{21} & \ddots & \vdots \\ & & h_{n+1,n} \end{bmatrix}.$$

The nth column of this equation can be written as follows:

$$Aq_n = h_{1n}q_1 + \cdots + h_{nn}q_n + h_{n+1,n}q_{n+1}. \tag{33.4}$$

In words, q_{n+1} satisfies an $(n+1)$-term recurrence relation involving itself and the previous Krylov vectors.

(source: Trefethen, Lecture 33)

Pseudo-code for Arnoldi Algorithm

```
Start with an arbitrary vector (normalized to have norm 1) for first col of Q
for n=1,2,3...
    v = A @ nth col of Q
    for j=1,...n
        project v onto q_j, and subtract the projection off of v
        want to capture part of v that isn't already spanned by prev columns of Q
        store coefficients in H
    normalize v, and then make it the (n+1)th column of Q
```

Notice that we are multiplying A by the previous vector in Q and removing the components that are not orthogonal to the existing columns of Q.

Question: Repeated multiplications of A? Does this remind you of anything?

Answer: ¶

In [ ]:

```
#Exercise Answer
The *Power Method* involved iterative multiplications by A as well!
```

About how the Arnoldi Iteration works ¶

- With the Arnoldi Iteration, we are finding an orthonormal basis for the Krylov subspace. The Krylov matrix

  $$K = \begin{bmatrix} b & Ab & A^2b & \dots & A^{n-1}b \end{bmatrix}$$

  has a QR factorization

  $$K = QR$$

  and that is the same Q that is being found in the Arnoldi Iteration. Note that the Arnoldi Iteration does not explicity calculate K or R.

- Intuition: K contains good information about the largest eigenvalues of A, and the QR factorization reveals this information by peeling off one approximate eigenvector at a time.

The Arnoldi Iteration is two things:

1. the basis of many of the iterative algorithms of numerical linear algebra
2. a technique for finding eigenvalues of nonhermitian matrices (Trefethen, page 257)

How Arnoldi Locates Eigenvalues

1. Carry out Arnoldi iteration
2. Periodically calculate the eigenvalues (called Arnoldi estimates or Ritz values) of the Hessenberg H, using the QR algorithm
3. Check at whether these values are converging. If they are, they're probably eigenvalues of A.

Implementation ¶

In [246]:

```
# Decompose square matrix A @ Q ~= Q @ H
def arnoldi(A):
    m, n = A.shape
    assert(n <= m)

    # Hessenberg matrix
    H = np.zeros([n+1,n]) #, dtype=np.float64)
    # Orthonormal columns
    Q = np.zeros([m,n+1]) #, dtype=np.float64)
    # 1st col of Q is a random column with unit norm
    b = np.random.rand(m)
    Q[:,0] = b / np.linalg.norm(b)
    for j in range(n):
        v = A @ Q[:,j]
        for i in range(j+1):
            #This comes from the formula for projection of v onto q.
            #Since columns q are orthonormal, q dot q = 1
            H[i,j] = np.dot(Q[:,i], v)
            v = v - (H[i,j] * Q[:,i])
        H[j+1,j] = np.linalg.norm(v)
        Q[:,j+1] = v / H[j+1,j]

        # printing this to see convergence, would be slow to use in practice
        print(np.linalg.norm(A @ Q[:,:-1] - Q @ H))
    return Q[:,:-1], H[:-1,:]
```

In [86]:

```
Q, H = arnoldi(A)
```

```
8.59112969133
4.45398729097
0.935693639985
3.36613943339
0.817740180293
```

Check that H is tri-diagonal:

In [70]:

```
H
```

Out[70]:

```
array([[ 5.62746,  4.05085, -0.     ,  0.     , -0.     ],
       [ 4.05085,  3.07109,  0.33036,  0.     , -0.     ],
       [ 0.     ,  0.33036,  0.98297,  0.11172, -0.     ],
       [ 0.     ,  0.     ,  0.11172,  0.29777,  0.07923],
       [ 0.     ,  0.     ,  0.     ,  0.07923,  0.06034]])
```

Exercise ¶

Write code to confirm that:

1. AQ = QH
2. Q is orthonormal

Answer: ¶

In [71]:

```
#Exercise:
np.allclose(A @ Q, Q @ H)
```

Out[71]:

```
True
```

In [456]:

```
#Exercise:
np.allclose(np.eye(len(Q)), Q.T @ Q)
```

Out[456]:

```
True
```

General Case: ¶

General Matrix：Now we can do this on our general matrix A (not symmetric). In this case, we are getting a Hessenberg instead of a Tri-diagonal

In [74]:

```
Q0, H0 = arnoldi(A0)
```

```
1.44287067354
1.06234006889
0.689291414367
0.918098818651
4.7124490411e-16
```

Check that H is Hessenberg:

In [76]:

```
H0
```

Out[76]:

```
array([[ 1.67416,  0.83233, -0.39284,  0.10833,  0.63853],
       [ 1.64571,  1.16678, -0.54779,  0.50529,  0.28515],
       [ 0.     ,  0.16654, -0.22314,  0.08577, -0.02334],
       [ 0.     ,  0.     ,  0.79017,  0.11732,  0.58978],
       [ 0.     ,  0.     ,  0.     ,  0.43238, -0.07413]])
```

In [77]:

```
np.allclose(A0 @ Q0, Q0 @ H0)
```

Out[77]:

```
True
```

In [78]:

```
np.allclose(np.eye(len(Q0)), Q0.T @ Q0), np.allclose(np.eye(len(Q0)), Q0 @ Q0.T)
```

Out[78]:

```
(True, True)
```

Putting it all together ¶

In [215]:

```
def eigen(A, max_iter=20):
    Q, H = arnoldi(A)
    Ak, QQ = practical_qr(H, max_iter)
    U = Q @ QQ
    D = np.diag(Ak)
    return U, D
```

In [213]:

```
n = 10
A0 = np.random.rand(n,n)
A = A0 @ A0.T
```

In [242]:

```
U, D = eigen(A, 40)
```

```
14.897422908
1.57451192745
1.4820012435
0.668164424736
0.438450319682
0.674050723258
1.19470880942
0.217103444634
0.105443975462
3.8162597576e-15
[ 27.34799   1.22613   1.29671   0.70253   0.49651   0.56779   0.60974
   0.70123   0.19209   0.04905]
[ 27.34981   1.85544   1.04793   0.49607   0.44505   0.7106    1.00724
   0.07293   0.16058   0.04411]
[ 27.34981   2.01074   0.96045   0.54926   0.61117   0.8972    0.53424
   0.19564   0.03712   0.04414]
[ 27.34981   2.04342   0.94444   0.61517   0.89717   0.80888   0.25402
   0.19737   0.03535   0.04414]
[ 27.34981   2.04998   0.94362   0.72142   1.04674   0.58643   0.21495
   0.19735   0.03534   0.04414]
[ 27.34981   2.05129   0.94496   0.90506   0.95536   0.49632   0.21015
   0.19732   0.03534   0.04414]
[ 27.34981   2.05156   0.94657   1.09452   0.79382   0.46723   0.20948
   0.1973    0.03534   0.04414]
[ 27.34981   2.05161   0.94863   1.1919    0.70539   0.45628   0.20939
   0.19728   0.03534   0.04414]
[ 27.34981   2.05162   0.95178   1.22253   0.67616   0.45174   0.20939
   0.19727   0.03534   0.04414]
[ 27.34981   2.05162   0.95697   1.22715   0.66828   0.44981   0.2094
   0.19725   0.03534   0.04414]
[ 27.34981   2.05162   0.96563   1.22124   0.66635   0.44899   0.20941
   0.19724   0.03534   0.04414]
[ 27.34981   2.05162   0.97969   1.20796   0.66592   0.44864   0.20942
   0.19723   0.03534   0.04414]
[ 27.34981   2.05162   1.00135   1.18652   0.66585   0.44849   0.20943
   0.19722   0.03534   0.04414]
[ 27.34981   2.05162   1.03207   1.15586   0.66584   0.44843   0.20943
   0.19722   0.03534   0.04414]
[ 27.34981   2.05162   1.07082   1.11714   0.66584   0.4484    0.20944
   0.19721   0.03534   0.04414]
[ 27.34981   2.05162   1.11307   1.07489   0.66585   0.44839   0.20944
   0.1972    0.03534   0.04414]
[ 27.34981   2.05162   1.15241   1.03556   0.66585   0.44839   0.20945
   0.1972    0.03534   0.04414]
[ 27.34981   2.05162   1.18401   1.00396   0.66585   0.44839   0.20945
   0.1972    0.03534   0.04414]
[ 27.34981   2.05162   1.20652   0.98145   0.66585   0.44839   0.20946
   0.19719   0.03534   0.04414]
[ 27.34981   2.05162   1.22121   0.96676   0.66585   0.44839   0.20946
   0.19719   0.03534   0.04414]
[ 27.34981   2.05162   1.23026   0.95771   0.66585   0.44839   0.20946
   0.19719   0.03534   0.04414]
[ 27.34981   2.05162   1.23563   0.95234   0.66585   0.44839   0.20946
   0.19718   0.03534   0.04414]
[ 27.34981   2.05162   1.23876   0.94921   0.66585   0.44839   0.20947
   0.19718   0.03534   0.04414]
[ 27.34981   2.05162   1.24056   0.94741   0.66585   0.44839   0.20947
   0.19718   0.03534   0.04414]
[ 27.34981   2.05162   1.24158   0.94639   0.66585   0.44839   0.20947
   0.19718   0.03534   0.04414]
[ 27.34981   2.05162   1.24216   0.94581   0.66585   0.44839   0.20947
   0.19718   0.03534   0.04414]
[ 27.34981   2.05162   1.24249   0.94548   0.66585   0.44839   0.20947
   0.19718   0.03534   0.04414]
[ 27.34981   2.05162   1.24268   0.94529   0.66585   0.44839   0.20947
   0.19718   0.03534   0.04414]
[ 27.34981   2.05162   1.24278   0.94519   0.66585   0.44839   0.20947
   0.19718   0.03534   0.04414]
[ 27.34981   2.05162   1.24284   0.94513   0.66585   0.44839   0.20947
   0.19718   0.03534   0.04414]
[ 27.34981   2.05162   1.24288   0.94509   0.66585   0.44839   0.20947
```

```
   0.19718   0.03534   0.04414]
[ 27.34981   2.05162   1.2429    0.94507   0.66585   0.44839   0.20947
   0.19717   0.03534   0.04414]
[ 27.34981   2.05162   1.24291   0.94506   0.66585   0.44839   0.20947
   0.19717   0.03534   0.04414]
[ 27.34981   2.05162   1.24291   0.94506   0.66585   0.44839   0.20947
   0.19717   0.03534   0.04414]
[ 27.34981   2.05162   1.24292   0.94505   0.66585   0.44839   0.20947
   0.19717   0.03534   0.04414]
[ 27.34981   2.05162   1.24292   0.94505   0.66585   0.44839   0.20947
   0.19717   0.03534   0.04414]
[ 27.34981   2.05162   1.24292   0.94505   0.66585   0.44839   0.20948
   0.19717   0.03534   0.04414]
[ 27.34981   2.05162   1.24292   0.94505   0.66585   0.44839   0.20948
   0.19717   0.03534   0.04414]
[ 27.34981   2.05162   1.24292   0.94505   0.66585   0.44839   0.20948
   0.19717   0.03534   0.04414]
[ 27.34981   2.05162   1.24292   0.94505   0.66585   0.44839   0.20948
   0.19717   0.03534   0.04414]
```

In [240]:

```
D
```

Out[240]:

```
array([ 5.10503,  0.58805,  0.49071, -0.65174, -0.60231, -0.37664,
       -0.13165,  0.0778 , -0.10469, -0.29771])
```

In [241]:

```
np.linalg.eigvals(A)
```

Out[241]:

```
array([ 27.34981,   2.05162,   1.24292,   0.94505,   0.66585,   0.44839,
         0.20948,   0.19717,   0.04414,   0.03534])
```

In [236]:

```
np.linalg.norm(U @ np.diag(D) @ U.T - A)
```

Out[236]:

```
0.0008321887107978883
```

In [237]:

```
np.allclose(U @ np.diag(D) @ U.T, A, atol=1e-3)
```

Out[237]:

```
True
```

Further Reading ¶

Let's find some eigenvalues!

from Nonsymmetric Eigenvalue Problems chapter:

Note that "direct" methods must still iterate, since finding eigenvalues is mathematically equivalent to finding zeros of polynomials, for which no noniterative methods can exist. We call a method direct if experience shows that it (nearly) never fails to converge in a fixed number of iterations.

Iterative methods typically provide approximations only to a subset of the eigenvalues and eigenvectors and are usually run only long enough to get a few adequately accurate eigenvalues rather than a large number

our ultimate algorithm: the shifted Hessenberg QR algorithm

More reading:

- The Symmetric Eigenproblem and SVD

- Iterative Methods for Eigenvalue Problems Rayleigh-Ritz Method, Lanczos algorithm

Coming Up ¶

We will be coding our own QR decomposition (two different ways!) in the future, but first we are going to see another way that the QR decomposition can be used: to calculate linear regression.

End¶

Miscellaneous Notes ¶

Symmetric matrices come up naturally:

- distance matrices
- relationship matrices (Facebook or LinkedIn)
- ODEs

We will look at positive definite matrices, since that guarantees that all the eigenvalues are real.

Note: in the confusing language of NLA, the QR algorithm is direct, because you are making progress on all columns at once. In other math/CS language, the QR algorithm is iterative, because it iteratively converges and never reaches an exact solution.

structured orthogonalization. In the language of NLA, Arnoldi iteration is considered an iterative algorithm, because you could stop part way and have a few columns completed.

a Gram-Schmidt style iteration for transforming a matrix to Hessenberg form

You can read an overview of this Numerical Linear Algebra course in this blog post. The course was originally taught in the University of San Francisco MS in Analytics graduate program. Course lecture videos are available on YouTube (note that the notebook numbers and video numbers do not line up, since some notebooks took longer than 1 video to cover).

You can ask questions about the course on our fast.ai forums.

8. Implementing QR Factorization ¶

We used QR factorization in computing eigenvalues and to compute least squares regression. It is an important building block in numerical linear algebra.

"One algorithm in numerical linear algebra is more important than all the others: QR factorization." --Trefethen, page 48

Recall that for any matrix A, $A = QR$ where Q is orthogonal and R is upper-triangular.

Reminder : The QR algorithm, which we looked at in the last lesson, uses the QR decomposition, but don't confuse the two.

In Numpy ¶

In [5]:

```
import numpy as np

np.set_printoptions(suppress=True, precision=4)
```

In [6]:

```
n = 5
A = np.random.rand(n,n)
npQ, npR = np.linalg.qr(A)
```

Check that Q is orthogonal:

In [7]:

```
np.allclose(np.eye(n), npQ @ npQ.T), np.allclose(np.eye(n), npQ.T @ npQ)
```

Out[7]:

```
(True, True)
```

Check that R is triangular

In [8]:

```
npR
```

Out[8]:

```
array([[-0.8524, -0.7872, -1.1163, -1.2248, -0.7587],
       [ 0.    , -0.9363, -0.2958, -0.7666, -0.632 ],
       [ 0.    ,  0.    ,  0.4645, -0.1744, -0.3542],
       [ 0.    ,  0.    ,  0.    ,  0.4328, -0.2567],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.1111]])
```

Gram-Schmidt ¶

Classical Gram-Schmidt (unstable) ¶

For each j, calculate a single projection

$$v_j = P_j a_j$$

where P_j projects onto the space orthogonal to the span of $q_1, \ldots, q_{j-1}$.

In [10]:

```
def cgs(A):
    m, n = A.shape
    Q = np.zeros([m,n], dtype=np.float64)
    R = np.zeros([n,n], dtype=np.float64)
    for j in range(n):
        v = A[:,j]
        for i in range(j):
            R[i,j] = np.dot(Q[:,i], A[:,j])
            v = v - (R[i,j] * Q[:,i])
        R[j,j] = np.linalg.norm(v)
        Q[:, j] = v / R[j,j]
    return Q, R
```

In [11]:

```
Q, R = cgs(A)
```

In [12]:

```
np.allclose(A, Q @ R)
```

Out[12]:

```
True
```

Check if Q is unitary:

In [109]:

```
np.allclose(np.eye(len(Q)), Q.dot(Q.T))
```

Out[109]:

```
True
```

In [115]:

```
np.allclose(npQ, -Q)
```

Out[115]:

```
True
```

Gram-Schmidt should remind you a bit of the Arnoldi Iteration (used to transform a matrix to Hessenberg form) since it is also a structured orthogonalization.

Modified Gram-Schmidt ¶

Classical (unstable) Gram-Schmidt: for each j, calculate a single projection

$$v_j = P_j a_j$$

where P_j projects onto the space orthogonal to the span of $q_1, \ldots, q_{j-1}$.

Modified Gram-Schmidt: for each j, calculate $j-1$ projections

$$P_j = P_{\perp q_{j-1} \cdots \perp q_2 \perp q_1}$$

In [30]:

```
import numpy as np
n = 3
A = np.random.rand(n,n).astype(np.float64)
```

In [33]:

```
def mgs(A):
    V = A.copy()
    m, n = A.shape
    Q = np.zeros([m,n], dtype=np.float64)
    R = np.zeros([n,n], dtype=np.float64)
    for i in range(n):
        R[i,i] = np.linalg.norm(V[:,i])
        Q[:,i] = V[:,i] / R[i,i]
        for j in range(i, n):
            R[i,j] = np.dot(Q[:,i],V[:,j])
            V[:,j] = V[:,j] - R[i,j]*Q[:,i]
    return Q, R
```

In [34]:

```
Q, R = mgs(A)
```

In [35]:

```
np.allclose(np.eye(len(Q)), Q.dot(Q.T.conj()))
```

Out[35]:

```
True
```

In [36]:

```
np.allclose(A, np.matmul(Q,R))
```

Out[36]:

```
True
```

Householder ¶

Intro ¶

$Gram-Schmidt$	$Triangular\,Orthogonalization$	$AR_1R_2\cdots R_n = Q$
$Householder$	$Orthogonal\,Triangularization$	$Q_n\cdots Q_2Q_1A = R$

Householder reflections lead to a more nearly orthogonal matrix Q with rounding errors

Gram-Schmidt can be stopped part-way, leaving a reduced QR of 1st n columns of A

Initialization ¶

In [113]:

```
import numpy as np
n = 4
A = np.random.rand(n,n).astype(np.float64)

Q = np.zeros([n,n], dtype=np.float64)
R = np.zeros([n,n], dtype=np.float64)
```

In [114]:

```
A
```

Out[114]:

```
array([[ 0.5435,  0.6379,  0.4011,  0.5773],
       [ 0.0054,  0.8049,  0.6804,  0.0821],
       [ 0.2832,  0.2416,  0.8656,  0.8099],
       [ 0.1139,  0.9621,  0.7623,  0.5648]])
```

In [115]:

```
from scipy.linalg import block_diag
```

In [116]:

```
np.set_printoptions(5)
```

Algorithm ¶

I added more computations and more info than needed, since it's illustrative of how the algorithm works. This version returns the Householder Reflectors as well.

In [117]:

```
def householder_lots(A):
    m, n = A.shape
    R = np.copy(A)
    V = []
    Fs = []
    for k in range(n):
        v = np.copy(R[k:,k])
        v = np.reshape(v, (n-k, 1))
        v[0] += np.sign(v[0]) * np.linalg.norm(v)
        v /= np.linalg.norm(v)
        R[k:,k:] = R[k:,k:] - 2*np.matmul(v, np.matmul(v.T, R[k:,k:]))
        V.append(v)
        F = np.eye(n-k) - 2 * np.matmul(v, v.T)/np.matmul(v.T, v)
        Fs.append(F)
    return R, V, Fs
```

Check that R is upper triangular:

In [121]:

```
R
```

Out[121]:

```
array([[-0.62337, -0.84873, -0.88817, -0.97516],
       [ 0.     , -1.14818, -0.86417, -0.30109],
       [ 0.     ,  0.     , -0.64691, -0.45234],
       [-0.     ,  0.     ,  0.     , -0.26191]])
```

As a check, we will calculate Q^T and R using the block matrices F. The matrices F are the householder reflectors.

Note that this is not a computationally efficient way of working with Q. In most cases, you do not actually need Q. For instance, if you are using QR to solve least squares, you just need Q^*b.

- See page 74 of Trefethen for techniques to implicitly calculate the product Q^*b or Qx.
- See these lecture notes for a different implementation of Householder that calculates Q simultaneously as part of R.

In [175]:

```
QT = np.matmul(block_diag(np.eye(3), F[3]),
               np.matmul(block_diag(np.eye(2), F[2]),
                         np.matmul(block_diag(np.eye(1), F[1]), F[0])))
```

In [178]:

```
F[1]
```

Out[178]:

```
array([[-0.69502,  0.10379, -0.71146],
       [ 0.10379,  0.99364,  0.04356],
       [-0.71146,  0.04356,  0.70138]])
```

In [177]:

```
block_diag(np.eye(1), F[1])
```

Out[177]:

```
array([[ 1.     ,  0.     ,  0.     ,  0.     ],
       [ 0.     , -0.69502,  0.10379, -0.71146],
       [ 0.     ,  0.10379,  0.99364,  0.04356],
       [ 0.     , -0.71146,  0.04356,  0.70138]])
```

In [176]:

```
np.matmul(block_diag(np.eye(1), F[1]), F[0])
```

Out[176]:

```
array([[-0.87185, -0.00861, -0.45431, -0.18279],
       [ 0.08888, -0.69462,  0.12536, -0.70278],
       [-0.46028,  0.10167,  0.88193, -0.00138],
       [-0.14187, -0.71211,  0.00913,  0.68753]])
```

In [123]:

```
QT
```

Out[123]:

```
array([[-0.87185, -0.00861, -0.45431, -0.18279],
       [ 0.08888, -0.69462,  0.12536, -0.70278],
       [ 0.45817, -0.112  , -0.88171,  0.01136],
       [ 0.14854,  0.71056, -0.02193, -0.68743]])
```

In [124]:

```
R2 = np.matmul(block_diag(np.eye(3), F[3]),
               np.matmul(block_diag(np.eye(2), F[2]),
                         np.matmul(block_diag(np.eye(1), F[1]),
                                   np.matmul(F[0], A))))
```

In [125]:

```
np.allclose(A, np.matmul(np.transpose(QT), R2))
```

Out[125]:

```
True
```

In [126]:

```
np.allclose(R, R2)
```

Out[126]:

```
True
```

Here's a concise version of Householder (although I do create a new R, instead of overwriting A and computing it in place).

In [179]:

```
def householder(A):
    m, n = A.shape
    R = np.copy(A)
    Q = np.eye(m)
    V = []
    for k in range(n):
        v = np.copy(R[k:,k])
        v = np.reshape(v, (n-k, 1))
        v[0] += np.sign(v[0]) * np.linalg.norm(v)
        v /= np.linalg.norm(v)
        R[k:,k:] = R[k:,k:] - 2 * v @ v.T @ R[k:,k:]
        V.append(v)
    return R, V
```

In [157]:

```
RH, VH = householder(A)
```

Check that R is diagonal:

In [129]:

```
RH
```

Out[129]:

```
array([[-0.62337, -0.84873, -0.88817, -0.97516],
       [-0.     , -1.14818, -0.86417, -0.30109],
       [-0.     , -0.     , -0.64691, -0.45234],
       [-0.     ,  0.     ,  0.     , -0.26191]])
```

In [130]:

```
VH
```

Out[130]:

```
[array([[ 0.96743],
        [ 0.00445],
        [ 0.2348 ],
        [ 0.09447]]), array([[ 0.9206 ],
        [-0.05637],
        [ 0.38641]]), array([[ 0.99997],
        [-0.00726]]), array([[ 1.]])]
```

In [131]:

```
np.allclose(R, RH)
```

Out[131]:

```
True
```

In [132]:

```
def implicit_Qx(V,x):
    n = len(x)
    for k in range(n-1,-1,-1):
        x[k:n] -= 2*np.matmul(v[-k], np.matmul(v[-k], x[k:n]))
```

In [133]:

```
A
```

Out[133]:

```
array([[ 0.54348,  0.63791,  0.40114,  0.57728],
       [ 0.00537,  0.80485,  0.68037,  0.0821 ],
       [ 0.2832 ,  0.24164,  0.86556,  0.80986],
       [ 0.11395,  0.96205,  0.76232,  0.56475]])
```

Both classical and modified Gram-Schmidt require $2mn^2$ flops.

Gotchas ¶

Some things to be careful about:

- when you've copied values vs. when you have two variables pointing to the same memory location
- the difference between a vector of length n and a 1 x n matrix (np.matmul deals with them differently)

Analogy ¶

	$A = QR$	$A = QHQ^*$
orthogonal structuring	Householder	Householder
structured orthogonalization	Gram-Schmidt	Arnoldi

Gram-Schmidt and Arnoldi : succession of triangular operations, can be stopped part way and first n columns are correct

Householder : succession of orthogoal operations. Leads to more nearly orthogonal A in presence of rounding errors.

Note that to compute a Hessenberg reduction $A = QHQ^*$, Householder reflectors are applied to two sides of A, rather than just one.

Examples ¶

The following examples are taken from Lecture 9 of Trefethen and Bau, although translated from MATLAB into Python

Ex: Classical vs Modified Gram-Schmidt ¶

This example is experiment 2 from section 9 of Trefethen. We want to construct a square matrix A with random singular vectors and widely varying singular values spaced by factors of 2 between 2^{-1} and $2^{-(n+1)}$

In [68]:

```
import matplotlib.pyplot as plt
from matplotlib import rcParams
%matplotlib inline
```

In [183]:

```
n = 100
U, X = np.linalg.qr(np.random.randn(n,n))   # set U to a random orthogonal matrix
V, X = np.linalg.qr(np.random.randn(n,n))   # set V to a random orthogonal matrix
S = np.diag(np.power(2,np.arange(-1,-(n+1),-1), dtype=float))  # Set S to a diagonal matrix w/ exp
                                                               # values between 2^-1 and 2^-(n+1)
```

In [184]:

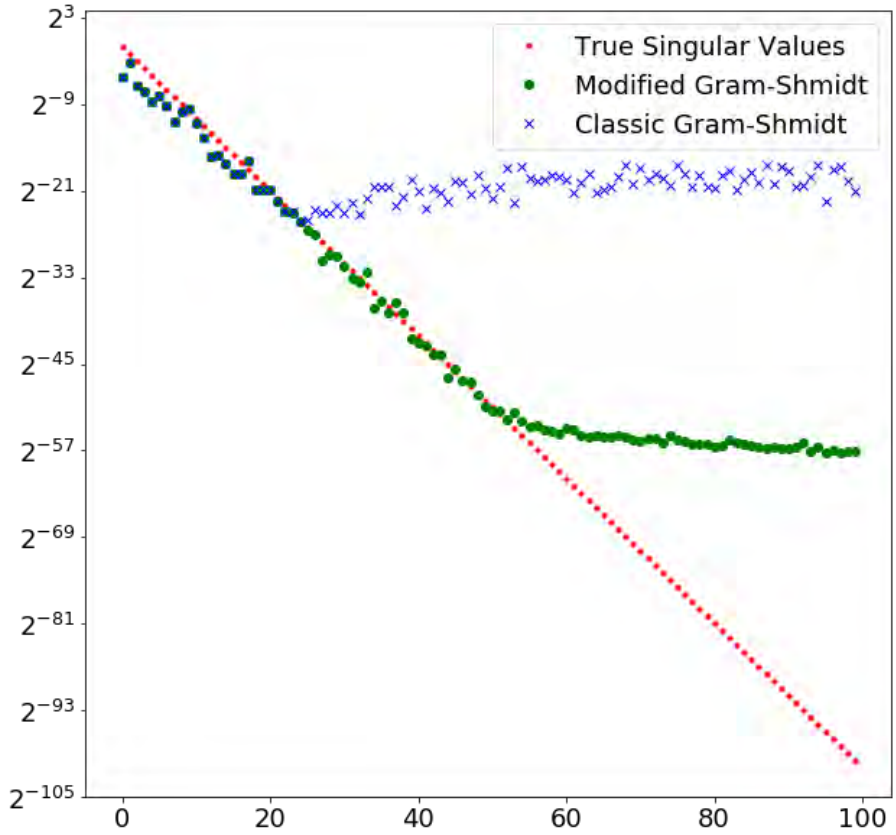
```
A = np.matmul(U,np.matmul(S,V))
```

In [188]:

```
QC, RC = cgs(A)
QM, RM = mgs(A)
```

In [189]:

```
plt.figure(figsize=(10,10))
plt.semilogy(np.diag(S), 'r.', basey=2, label="True Singular Values")
plt.semilogy(np.diag(RM), 'go', basey=2, label="Modified Gram-Shmidt")
plt.semilogy(np.diag(RC), 'bx', basey=2, label="Classic Gram-Shmidt")
plt.legend()
rcParams.update({'font.size': 18})
```



In [94]:

```
type(A[0,0]), type(RC[0,0]), type(S[0,0])
```

Out[94]:

```
(numpy.float64, numpy.float64, numpy.float64)
```

In [92]:

```
eps = np.finfo(np.float64).eps; eps
```

Out[92]:

```
2.2204460492503131e-16
```

In [93]:

```
np.log2(eps), np.log2(np.sqrt(eps))
```

Out[93]:

```
(-52.0, -26.0)
```

Ex 9.3: Numerical loss of orthogonality ¶

This example is experiment 3 from section 9 of Trefethen.

In [95]:

```
A = np.array([[0.70000, 0.70711], [0.70001, 0.70711]])
```

Gram-Schmidt:

In [96]:

```
Q1, R1 = mgs(A)
```

Householder:

In [105]:

```
R2, V, F = householder_lots(A)
Q2T = np.matmul(block_diag(np.eye(1), F[1]), F[0])
```

Numpy's Householder:

In [106]:

```
Q3, R3 = np.linalg.qr(A)
```

Check that all the QR factorizations work:

In [107]:

```
np.matmul(Q1, R1)
```

Out[107]:

```
array([[ 0.7   ,  0.7071],
       [ 0.7   ,  0.7071]])
```

In [108]:

```
np.matmul(Q2T.T, R2)
```

Out[108]:

```
array([[ 0.7   ,  0.7071],
       [ 0.7   ,  0.7071]])
```

In [109]:

```
np.matmul(Q3, R3)
```

Out[109]:

```
array([[ 0.7   ,  0.7071],
       [ 0.7   ,  0.7071]])
```

Check how close Q is to being perfectly orthonormal:

In [110]:

```
np.linalg.norm(np.matmul(Q1.T, Q1) - np.eye(2))  # Modified Gram-Schmidt
```

Out[110]:

```
3.2547268868202263e-11
```

In [111]:

```
np.linalg.norm(np.matmul(Q2T.T, Q2T) - np.eye(2))  # Our implementation of Householder
```

Out[111]:

```
1.1110522984689321e-16
```

In [112]:

```
np.linalg.norm(np.matmul(Q3.T, Q3) - np.eye(2))  # Numpy (which uses Householder)
```

Out[112]:

```
2.5020189909116529e-16
```

GS (Q1) is less stable than Householder (Q2T, Q3)

End¶

Intro to Convolutions ¶

Set up ¶

In [84]:

```
%matplotlib inline
import math,sys,os,numpy as np
from numpy.linalg import norm
from PIL import Image
from matplotlib import pyplot as plt, rcParams, rc
from scipy.ndimage import imread
from skimage.measure import block_reduce
import pickle as pickle
from scipy.ndimage.filters import correlate, convolve
rc('animation', html='html5')
rcParams['figure.figsize'] = 3, 6
%precision 4
np.set_printoptions(precision=4, linewidth=100)
```

In [85]:

```
def plots(ims, interp=False, titles=None):
    ims=np.array(ims)
    mn,mx=ims.min(),ims.max()
    f = plt.figure(figsize=(12,24))
    for i in range(len(ims)):
        sp=f.add_subplot(1, len(ims), i+1)
        if not titles is None: sp.set_title(titles[i], fontsize=18)
        plt.imshow(ims[i], interpolation=None if interp else 'none', vmin=mn,vmax=mx)

def plot(im, interp=False):
    f = plt.figure(figsize=(3,6), frameon=True)
    # plt.show(im)
    plt.imshow(im, interpolation=None if interp else 'none')

plt.gray()
plt.close()
```

MNIST Data ¶

In [4]:

```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
```

In [5]:

```
mnist.keys()
```

Out[5]:

```
dict_keys(['DESCR', 'COL_NAMES', 'target', 'data'])
```

In [6]:

```
mnist['data'].shape, mnist['target'].shape
```

Out[6]:

```
((70000, 784), (70000,))
```

In [7]:

```
images = np.reshape(mnist['data'], (70000, 28, 28))
labels = mnist['target'].astype(int)
n=len(images)
images.shape, labels.shape
```

Out[7]:

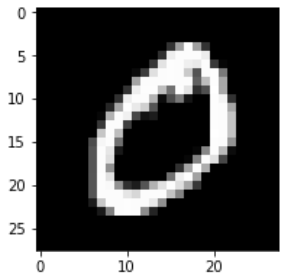
```
((70000, 28, 28), (70000,))
```

In [53]:

```
images = images/255
```

In [88]:

```
plot(images[0])
```



In [89]:

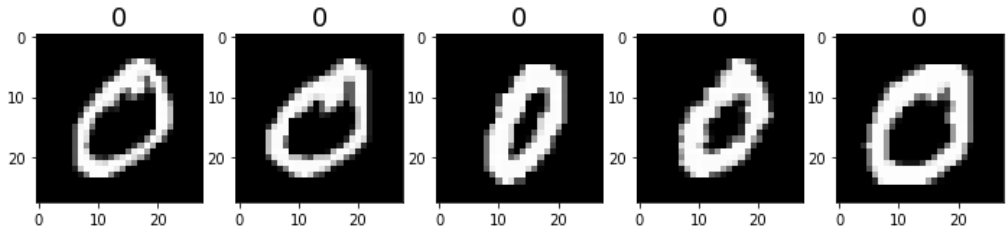
```
labels[0]
```

Out[89]:

```
0
```

In [56]:

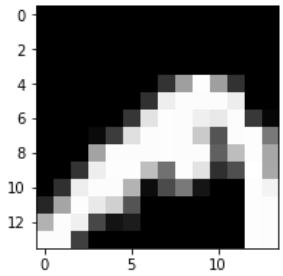
```
plots(images[:5], titles=labels[:5])
```



we can zoom in on part of the image

In [57]:

```
plot(images[0,0:14, 8:22])
```

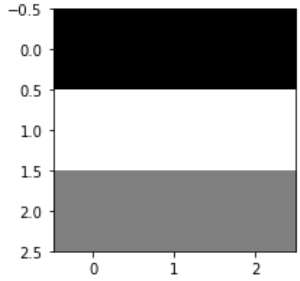


Edge Detection ¶

We will look at how to create an Edge detector:

In [44]:

```
top=[[-1,-1,-1],
     [ 1, 1, 1],
     [ 0, 0, 0]]

plot(top)
```



In [90]:

```
dims = np.index_exp[10:28:1,3:13]
images[0][dims]
```

Out[90]:

```
array([[ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.1882,  0.9333,  0.9882,  0.9882],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.149 ,  0.6471,  0.9922,  0.9137,  0.8157],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.0275,  0.698 ,  0.9882,  0.9412,  0.2784,  0.0745],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.2235,  0.9882,  0.9882,  0.2471,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.7765,  0.9922,  0.7451,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.298 ,  0.9647,  0.9882,  0.4392,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.3333,  0.9882,  0.902 ,  0.098 ,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.3333,  0.9882,  0.8745,  0.    ,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.3333,  0.9882,  0.5686,  0.    ,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.3373,  0.9922,  0.8824,  0.    ,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.3333,  0.9882,  0.9765,  0.5725,  0.1882,  0.1137,  0.3333],
       [ 0.    ,  0.    ,  0.    ,  0.3333,  0.9882,  0.9882,  0.9882,  0.898 ,  0.8431,  0.9882],
       [ 0.    ,  0.    ,  0.    ,  0.1098,  0.7804,  0.9882,  0.9882,  0.9922,  0.9882,  0.9882],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.098 ,  0.502 ,  0.9882,  0.9922,  0.9882,  0.5529],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ]])
```

In [61]:

```
corrtop = correlate(images[0], top)
```

In [62]:

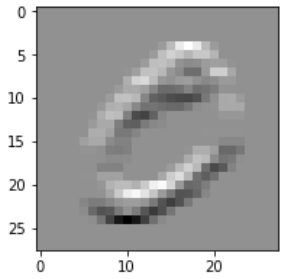
```
corrtop[dims]
```

Out[62]:

```
array([[ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.1882,  0.9216,  0.9765,  0.7843, -0.2392],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.149 ,  0.6078,  0.6667,  0.4431, -0.1882, -0.6196],
       [ 0.    ,  0.    ,  0.    ,  0.0275,  0.5765,  0.9176,  0.8392, -0.3451, -1.4275, -1.5961],
       [ 0.    ,  0.    ,  0.    ,  0.1961,  0.4863,  0.4863, -0.4039, -0.9725, -1.0471, -0.4627],
       [ 0.    ,  0.    ,  0.    ,  0.5529,  0.5569,  0.3137, -0.4863, -0.4902, -0.2471,  0.    ],
       [ 0.    ,  0.    ,  0.298 ,  0.4863,  0.4824, -0.1216, -0.3098, -0.3059,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.0353,  0.0588, -0.0275, -0.4039, -0.4275, -0.3412,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.    , -0.0275, -0.1255, -0.1255, -0.098 ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.    , -0.3059, -0.3059, -0.3059,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.0039,  0.0078,  0.3216,  0.3176,  0.3137,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    , -0.0039, -0.0078,  0.0863,  0.6627,  0.8549,  0.8745,  0.6353,  1.1451],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.0118,  0.4275,  1.1373,  1.8549,  2.0941,  1.6745],
       [ 0.    ,  0.    , -0.2235, -0.4314, -0.4314, -0.2078,  0.0941,  0.2392,  0.2392,  0.0706],
       [ 0.    ,  0.    , -0.1098, -0.7922, -1.2784, -1.1686, -0.4863,  0.    , -0.4353, -1.2039],
       [ 0.    ,  0.    ,  0.    , -0.098 , -0.6   , -1.5882, -2.4824, -2.9686, -2.5333, -1.6863],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ],
       [ 0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ,  0.    ]])
```

In [64]:

```
plot(corrtop)
```



In [65]:

```
np.rot90(top, 1)
```

Out[65]:

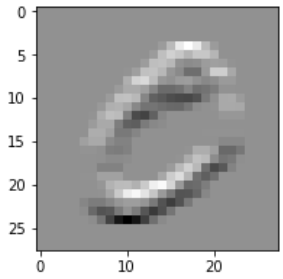
```
array([[-1,  1,  0],
       [-1,  1,  0],
       [-1,  1,  0]])
```

In [66]:

```
convtop = convolve(images[0], np.rot90(top,2))
plot(convtop)
np.allclose(convtop, corrtop)
```

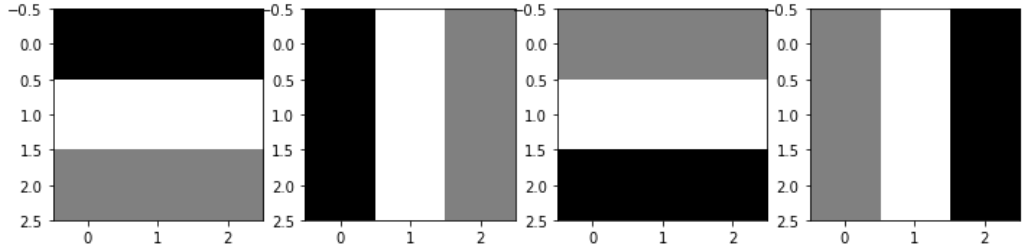
Out[66]:

```
True
```



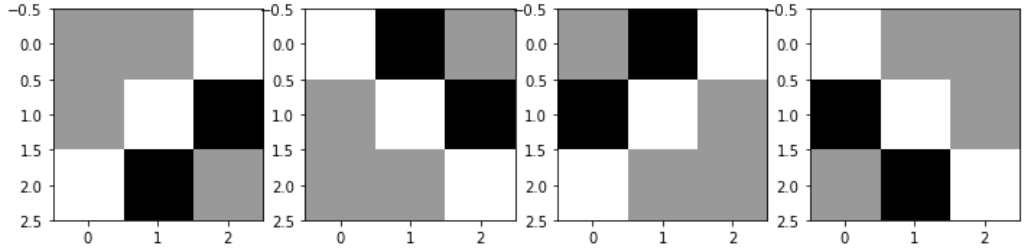
In [67]:

```
straights=[np.rot90(top,i) for i in range(4)]
plots(straights)
```



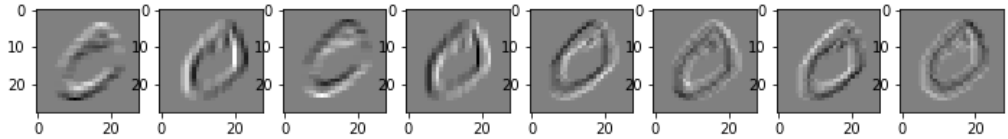
In [68]:

```
br=[[ 0, 0, 1],
    [ 0, 1,-1.5],
    [ 1,-1.5, 0]]

diags = [np.rot90(br,i) for i in range(4)]
plots(diags)
```



In [69]:

```
rots = straights + diags
corrs = [correlate(images[0], rot) for rot in rots]
plots(corrs)
```

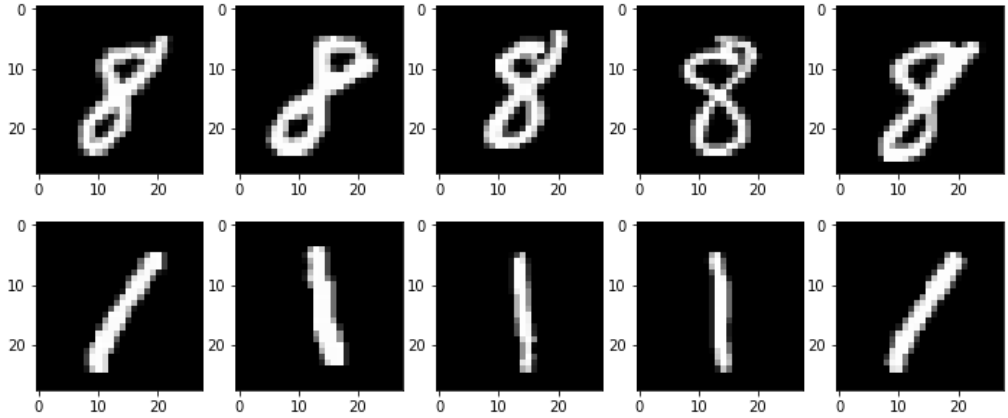


In [72]:

```
eights=[images[i] for i in range(n) if labels[i]==8]
ones=[images[i] for i in range(n) if labels[i]==1]
```

In [73]:

```
plots(eights[:5])
plots(ones[:5])
```



In [78]:

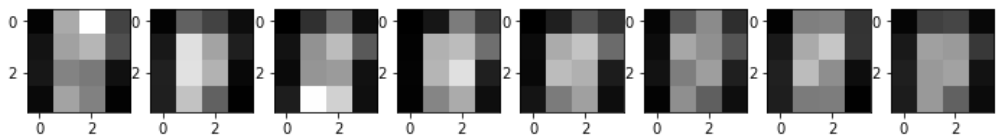
```
def normalize(arr): return (arr-arr.mean())/arr.std()
```

In [79]:

```
filts8 = np.array([ims.mean(axis=0) for ims in pool8])
filts8 = normalize(filts8)
```

In [80]:

```
plots(filts8)
```

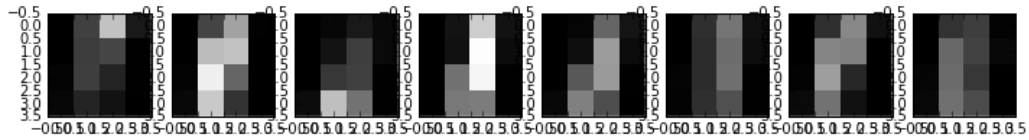


In [30]:

```
pool1 = [np.array([pool(correlate(im, rot)) for im in ones]) for rot in rots]
filts1 = np.array([ims.mean(axis=0) for ims in pool1])
filts1 = normalize(filts1)
```

In [31]:

```
plots(filts1)
```

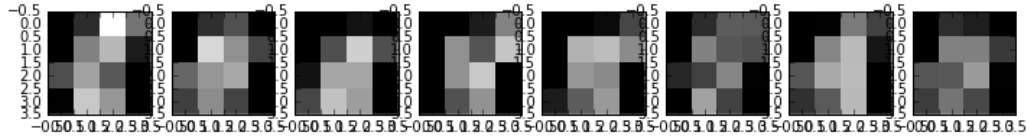


In [32]:

```
def pool_corr(im): return np.array([pool(correlate(im, rot)) for rot in rots])
```

In [33]:

```
plots(pool_corr(eights[0]))
```



In [35]:

```
def sse(a,b): return ((a-b)**2).sum()
def is8_n2(im): return 1 if sse(pool_corr(im),filts1) > sse(pool_corr(im),filts8) else 0
```

In [36]:

```
sse(pool_corr(eights[0]), filts8), sse(pool_corr(eights[0]), filts1)
```

Out[36]:

```
(126.77776, 181.26105)
```

In [37]:

```
[np.array([is8_n2(im) for im in ims]).sum() for ims in [eights,ones]]
```

Out[37]:

```
[5223, 287]
```

In [38]:

```
[np.array([(1-is8_n2(im)) for im in ims]).sum() for ims in [eights,ones]]
```

Out[38]:

```
[166, 5892]
```

In [ ]:

```
def n1(a,b): return (np.fabs(a-b)).sum()
def is8_n1(im): return 1 if n1(pool_corr(im),filts1) > n1(pool_corr(im),filts8) else 0
```

In [ ]:

```
[np.array([is8_n1(im) for im in ims]).sum() for ims in [eights,ones]]
```

In [ ]:

```
[np.array([(1-is8_n1(im)) for im in ims]).sum() for ims in [eights,ones]]
```

Gradient Descent Intro ¶

In [2]:

```
%matplotlib inline
import math,sys,os,numpy as np
from numpy.random import random
from matplotlib import pyplot as plt, rcParams, animation, rc
from __future__ import print_function, division
from ipywidgets import interact, interactive, fixed
from ipywidgets.widgets import *
rc('animation', html='html5')
rcParams['figure.figsize'] = 3, 3
%precision 4
np.set_printoptions(precision=4, linewidth=100)
```

In [1]:

```
def lin(a,b,x): return a*x+b
```

In [3]:

```
a=3.
b=8.
```

In [7]:

```
n=30
x = random(n)
y = lin(a,b,x)
```

In [8]:

```
x
```

Out[8]:

```
array([ 0.3046,  0.918 ,  0.7925,  0.8476,  0.2508,  0.3504,  0.8326,  0.6875,  0.4449,  0.4687,
        0.5901,  0.2757,  0.6629,  0.169 ,  0.8677,  0.6612,  0.112 ,  0.1669,  0.6226,  0.6174,
        0.3871,  0.4724,  0.3242,  0.7871,  0.0157,  0.8589,  0.7008,  0.2942,  0.3166,  0.5847])
```

In [9]:

```
y
```

Out[9]:

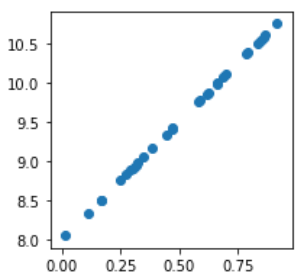
```
array([  8.9138,  10.7541,  10.3775,  10.5428,   8.7525,   9.0511,  10.4977,  10.0626,   9.3347,
         9.4062,   9.7704,   8.827 ,   9.9888,   8.507 ,  10.603 ,   9.9836,   8.336 ,   8.5006,
         9.8678,   9.8523,   9.1614,   9.4172,   8.9725,  10.3614,   8.0471,  10.5766,  10.1025,
         8.8827,   8.9497,   9.7542])
```

In [10]:

```
plt.scatter(x,y)
```

Out[10]:

```
<matplotlib.collections.PathCollection at 0x7ff89aea9668>
```



In [11]:

```
def sse(y,y_pred): return ((y-y_pred)**2).sum()
def loss(y,a,b,x): return sse(y, lin(a,b,x))
```

```
def avg_loss(y,a,b,x): return np.sqrt(loss(y,a,b,x)/n)
```

In [12]:

```
a_guess=-1.
b_guess=1.
avg_loss(y, a_guess, b_guess, x)
```

Out[12]:

```
9.1074
```

In [13]:

```
lr=0.01
# d[(y-(a*x+b))**2,b] = 2 (b + a x - y)      = 2 (y_pred - y)
# d[(y-(a*x+b))**2,a] = 2 x (b + a x - y)    = x * dy/db
```

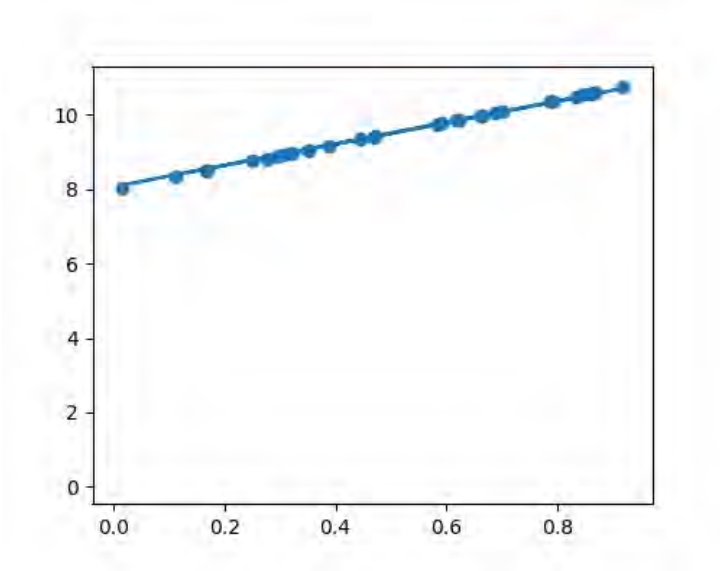
In [14]:

```
def upd():
    global a_guess, b_guess

    # make a prediction using the current weights
    y_pred = lin(a_guess, b_guess, x)

    # calculate the derivate of the loss
    dydb = 2 * (y_pred - y)
    dyda = x*dydb

    # update our weights by moving in direction of steepest descent
    a_guess -= lr*dyda.mean()
    b_guess -= lr*dydb.mean()
```

In [15]:

```
fig = plt.figure(dpi=100, figsize=(5, 4))
plt.scatter(x,y)
line, = plt.plot(x,lin(a_guess,b_guess,x))
plt.close()

def animate(i):
    line.set_ydata(lin(a_guess,b_guess,x))
    for i in range(10): upd()
    return line,

ani = animation.FuncAnimation(fig, animate, np.arange(0, 40), interval=100)
ani
```

Out[15]:



Homework 1 ¶

This is due on Thurs, 6/1

1. Consider the polynomial $p(x) = (x-2)^9 = x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512$

   a. Plot $p(x)$ for $x = 1.920,\ 1.921,\ 1.922, \ldots, 2.080$ evaluating p via its coefficients $1,\ ,-18,\ 144, \ldots$

   b. Plot the same plot again, now evaluating p via the expression $(x-2)^9$.

   c. Explain the difference.

   (The numpy method linspace will be useful for this)

2. How many different double-precision numbers are there? Express your answer using powers of 2

3. Using the updated Numbers Every Programmer Should Know, how much longer does a main memory reference take than an L1 cache look-up? How much longer does a disk seek take than a main memory reference?

4. From the Halide Video, what are 4 ways to traverse a 2d array?

5. Using the animations below (source), explain what the benefits and pitfalls of each approach. Green squares indicate that a value is being read; red indicates a value is being written. Your answers should be longer in length (give more detail) than just two words.

a. https://raw.githubusercontent.com/fastai/numerical-linear-algebra/master/nbs/images/Halide1.gif

b. https://raw.githubusercontent.com/fastai/numerical-linear-algebra/master/nbs/images/Halide2.gif

c. https://raw.githubusercontent.com/fastai/numerical-linear-algebra/master/nbs/images/Halide3.gif

6. Prove that if $A = QBQ^T$ for some orthnogonal matrix Q, the A and B have the same singular values.

7. What is the stochastic part of stochastic gradient descent?

Homework 2 ¶

This homework covers the material in Lessons 3 & 4. It is due Thurday , June 15 . Please submit your answers as a PDF .

1. Modify the LU method (without pivoting) to work in-place . That is, it should not allocate any new memory for L or U, but instead overwrite A.

In [ ]:

```
def LU(A):
    U = np.copy(A)
    m, n = A.shape
    L = np.eye(n)
    for k in range(n-1):
        for j in range(k+1,n):
            L[j,k] = U[j,k]/U[k,k]
            U[j,k:n] -= L[j,k] * U[k,k:n]
    return L, U
```

2. Modify our LU method from class to add pivoting, as described in the lesson. Hint: the swap method below will be useful

In [ ]:

```
def swap(a,b):
    temp = np.copy(a)
    a[:] = b
    b[:] = temp
```

3. For each of the following sets of dimensions, either

- give the dimensions of the output of an operation on A and B, or
- answer incompatible if the dimensions are incompatible according to the numpy rules of broadcasting.

```
a.  A      (2d array):  3 x 3
    B      (1d array):      1

b.  A      (2d array):      2 x 1
    B      (3d array):  6 x 4 x 2

c.  A      (2d array):  5 x 4
    B      (1d array):      4

d.  A      (3d array):  32 x 64 x 8
    B      (3d array):   32 x 1 x 8
```

```
e.  A      (3d array):       64 x 1
    B      (3d array):  32 x 1 x 16

f.  A      (3d array):  32 x 64 x 2
    B      (3d array):   32 x 1 x 8
```

4. Write how this matrix would be stored in compressed row format:

$$\begin{pmatrix} 1 & & & -2 & -3 \\ & 3 & & & -9 \\ & & -7 & 4 & \\ -1 & 2 & 7 & & \\ -3 & & 26 & & \end{pmatrix}$$

Homework 3 ¶

1. Add shifts to the QR algorithm

Instead of factoring A_k as $Q_k R_k$ (the way the pure QR algorithm without shifts does), the shifted QR algorithms:

i. Get the QR factorization

$$A_k - s_k I = Q_k R_k$$

ii. Set

$$A_{k+1} = R_k Q_k + s_k I$$

Choose $s_k = A_k(m, m)$, an approximation of an eigenvalue of A.

The idea of adding shifts to speed up convergence shows up in many algorithms in numerical linear algebra (including the power method, inverse iteration, and Rayleigh quotient iteration).

In [1]:

```
def practical_qr(A, iters=10):
    # Fill method in
    return Ak, Q
```