Admin: Textbook, syllabus, homework, midterms, final, grading, office hours, ...

MOTIVATION FOR LINEAR ALGEBRA

Theory: Linear transformations are everywhere!

~ Signals: Fourier transform is linear
~ Physics: Quantum time evolution is linear
~ Calculus: Integration and differentiation are linear

Applications: countless...

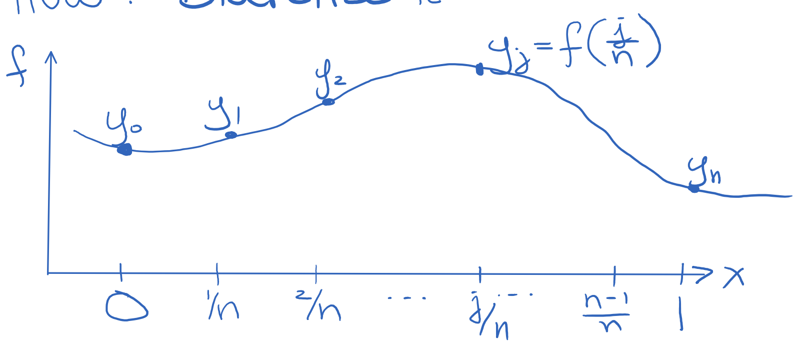
Application: Solving differential equations

example:

Solve $f'(x) = g(x) \quad \forall x \in [0,1]$

$f(0) = f_0$

How? **Discretize** it



$n+1$ variables: $y_j \quad j = 0, \dots, n$

equations: $y_0 = f_0$

$$f'(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta} = g(x)$$

$$\approx \frac{f(x+\frac{1}{n}) - f(x)}{1/n}$$

$$\Rightarrow n(y_{j+1} - y_j) \approx g(j/n), \quad j = 0, \dots, n-1$$

$\Bigg\}$ $n+1$ equations

Aside:

$$f'(x) \approx \frac{1}{1/n}\left(f(x+\tfrac{1}{n}) - f(x)\right)$$

$$f'(x) \approx \frac{1}{2/n}\left(f(x+\tfrac{1}{n}) - f(x-\tfrac{1}{n})\right) \leftarrow \text{better discretization}$$

Example: ECMWF 10-day weather forecasts

9km horizontal,

Example: ECMWF 10 day weather forecasts

9km horizontal,
137 vertical levels
$\Downarrow$
10^9 grid points
(~100? vars at each point)

https://www.ecmwf.int/en/about/media-centre/news/2016/new-forecast-model-cycle-brings-highest-ever-resolution

Application: Solving linear equations

e.g.,
$$\left.\begin{aligned} 2x - y &= 3 \\ -x + y &= -2 \end{aligned}\right\}$$
adding gives
$$(2-1)x + (-1+1)y = 3-2$$
$$\Rightarrow x = 1$$
$$\Rightarrow y = -1$$

Matlab https://matlab.mathworks.com/

```
>> A = [2 -1; -1 1]

A =

     2    -1
    -1     1

>> b = [3; -2]

b =

     3
    -2

>> A \ b

ans =

    1.0000
   -1.0000
```

Python https://colab.research.google.com/

```
[1] import numpy as np

    A = [[2,-1], [-1,1]]
    b = [3,-2]
    np.linalg.solve(A, b)

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

- but most applications are for large systems
  - we need fast, approximate solutions
  - often we solve the same system repeatedly

e.g., $f'(x) = g(x)$ $\qquad\qquad$ $f'(x) = h(x)$

$$\Rightarrow n(y_{j+1} - y_j) = g(j/n) \qquad \Rightarrow n(y_{j+1} - y_j) = h(j/n)$$

same coefficients of $y_0, \ldots, y_n$

same coefficients of $y_0, \dots, y_n$

- what if #equations > #variables?

e.g., $\begin{cases} x = 1 \\ x = 2 \end{cases}$ ← no solution!

e.g., $\begin{cases} 2x - y = 3 \\ -x + y = -2 \\ x + y = 4 \end{cases}$ ← no solution!

Matlab

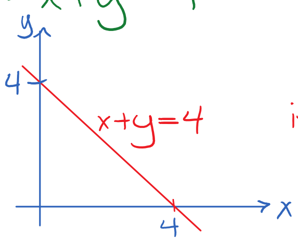
```
>> A = [1;
        1];
>> b = [1;2];
>> A \ b

ans =

    1.5000
```

```
>> A = [2 -1;
       -1  1;
        1  1];
>> b = [3; -2; 4];
>> A \ b

ans =

    2.4286
    1.2857
```

It tries to get as close as possible to a solution. We'll see more examples below.

- what if #equations < #variables?

Example: $x + y = 4$



infinitely many solutions!

Matlab https://matlab.mathworks.com/

```
>> A = [1 1];
>> b = [4];
>> A \ b

ans =

     4
     0
```

Python https://colab.research.google.com/

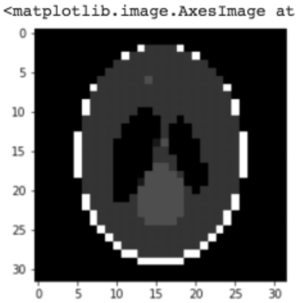
```
import numpy as np

A = [[1,1]]
b = [4]
np.linalg.solve(A, b)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most
<ipython-input-4-bc599599071c> in <module>()
      3 A = [[1,1]]
      4 b = [4]
----> 5 np.linalg.solve(A, b)

<__array_function__ internals> in solve(*args, **kwargs)

                    1 frames
/usr/local/lib/python3.6/dist-packages/numpy/linalg/linalg
    211         m, n = a.shape[-2:]
    212         if m != n:
--> 213             raise LinAlgError('Last 2 dimensions o
    214
    215 def _assert_finite(*arrays):

LinAlgError: Last 2 dimensions of the array must be square
```

Example: "Compressed Sensing"

```python
import matplotlib.pyplot as plt

w = 32
I = phantom(n=w)

imgplot = plt.imshow(I, cmap='gray')
imgplot
```

```
<matplotlib.image.AxesImage at 0x7f4(
```



https://en.wikipedia.org/wiki/Shepp–Logan_phantom

Shepp–Logan phantom

From Wikipedia, the free encyclopedia

The **Shepp–Logan phantom** is a standard test image created by Larry Shepp and Benjamin F. Logan for their 1974 paper *The Fourier Reconstruction of a Head Section*.[1] It serves as the model of a human head in the development and testing of image reconstruction algorithms.[2][3][4]

Image of the Shepp–Logan Phantom

```python
import numpy as np

x = np.ndarray.flatten(I)

n = len(x)
m = int(.7 * n)
print(m, n)

A = np.random.rand(m, n)

b = A.dot(x)
```

```
716 1024
```

716 equations
on $32^2=1024$ variables

```python
y = np.linalg.solve(A, b)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most :
<ipython-input-4-9087bf01a13c> in <module>()
----> 1 y = np.linalg.solve(A, b)

<__array_function__ internals> in solve(*args, **kwargs)

                              1 frames
/usr/local/lib/python3.6/dist-packages/numpy/linalg/linalg.
    211         m, n = a.shape[-2:]
    212         if m != n:
--> 213             raise LinAlgError('Last 2 dimensions of
    214
    215 def _assert_finite(*arrays):

LinAlgError: Last 2 dimensions of the array must be square
```

https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html

numpy.linalg.lstsq

numpy.linalg.**lstsq**(*a, b, rcond='warn'*) [source]

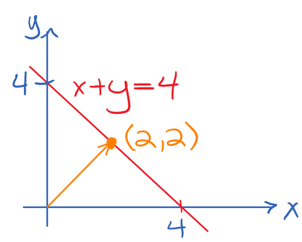
Return the least-squares solution to a linear matrix equation.

Computes the vector x that approximatively solves the equation `a @ x = b`. The equation may be under-, well-, or over-determined (i.e., the number of linearly independent rows of *a* can be less than, equal to, or greater than its number of linearly independent columns). If *a* is square and of full rank, then *x* (but for round-off error) is the "exact" solution of the equation. Else, *x* minimizes the Euclidean 2-norm $||b - ax||$.

example:

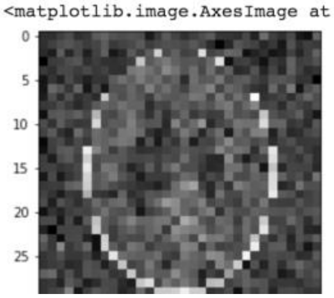
```python
import numpy as np

A = [[1,1]]
b = [[4]]
np.linalg.lstsq(A, b, rcond=None)[0]
```

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




```python
y = np.linalg.lstsq(A, b, rcond=None)[0]

y = np.resize(y, (w,w))

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

```
<matplotlib.image.AxesImage at 0x7f4c922(
```



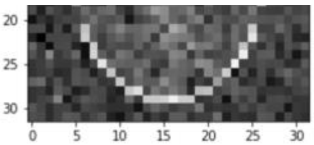
What is CVXPY?

https://www.cvxpy.org/

CVXPY is a Python-embedded modeling language for convex optimization problem automatically transforms the problem into standard form, calls a solver, and unpacks the results.

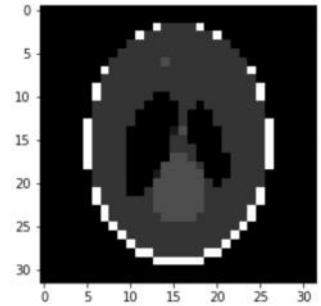
```python
import cvxpy as cp

# Construct the problem
z = cp.Variable(n)
objective = cp.Minimize(cp.sum_squares(A*z - b))
constraints = [0 <= z]  # can also try [0 <= z, z <= 1]
problem = cp.Problem(objective, constraints)

# Solve it
result = problem.solve()
# The optimal value for z is stored in z.value
plt.imshow(np.resize(z.value, (w,w)), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f4c82b9f710>
```



```
# The optimal value for z is stored in z.value
plt.imshow(np.resize(z.value, (w,w)), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f4c82b9f710>
```



Moral: For some applications we can solve for n variables with $< n$ equations!

"Compressed Sensing" for sparse solutions (in some basis)

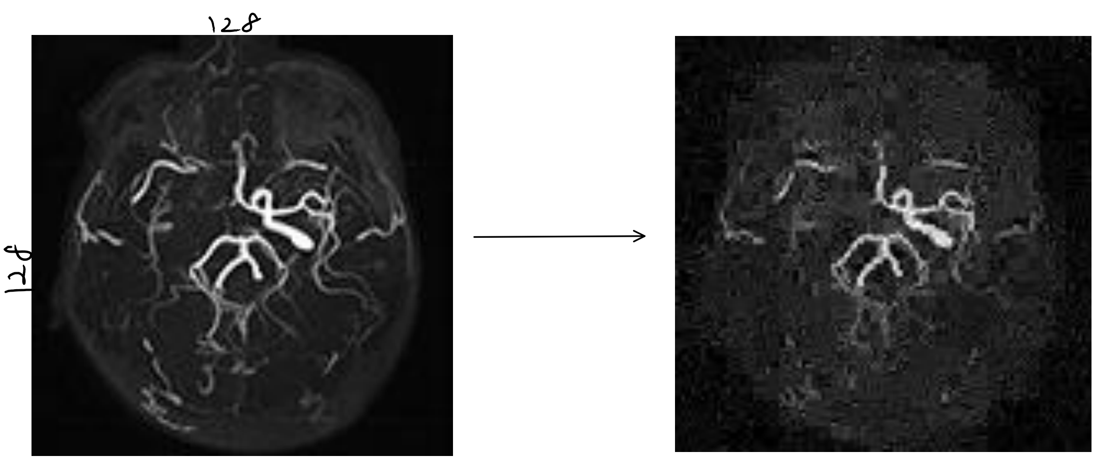
https://en.wikipedia.org/wiki/Compressed_sensing

Magnetic resonance imaging

Compressed sensing has been used[36][37] to shorten magnetic resonance imaging scanning sessions on conventional hardware.[38][39][40] Reconstruction methods include

- ISTA
- FISTA
- SISTA
- ePRESS[41]
- EWISTA[42]
- EWISTARS[43] etc.

Compressed sensing addresses the issue of high scan time by enabling faster acquisition by measuring fewer Fourier coefficients. This produces a high-quality image with relatively lower scan time. Another application (also discussed ahead) is for CT reconstruction with fewer X-ray projections. Compressed sensing, in this case, removes the high spatial gradient parts – mainly, image noise and artifacts. This holds tremendous potential as one can obtain high-resolution CT images at low radiation doses (through lower current-mA settings).[44]



\# of variables
$n = 128^2 = 16384$

\# of observations
$m = 4480 = 0.27n$

15 minutes in ℓ_1 magic

Themes

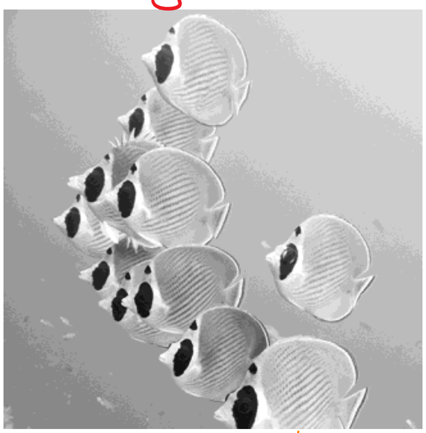
- Geometry
  - linear transformations
  - hyperplanes
- High dimensions
  - sparse matrices
  - dimension reduction

- Systems of linear equations
  - Computer-assisted linear algebra
    - Matlab   - python/numpy

$\underbrace{\text{- Matlab} \quad \text{- python/numpy}}_{\text{DIY: how they work}}$

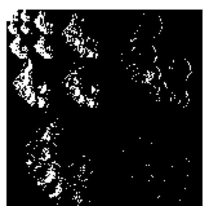
Application: Image compression

Original

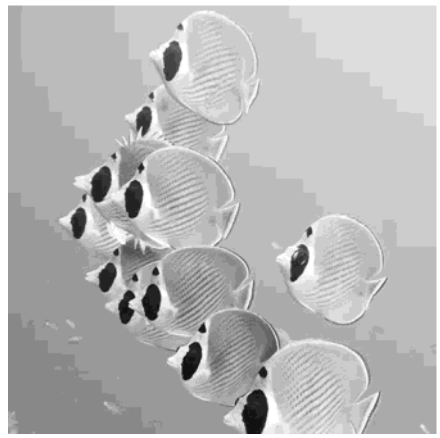


not sparse!

Keep the largest 10% of coefficients in the Hadamard basis



Result



sparse (in H basis)

Theme:
Sparse matrices

$$f'(x) = g(x)$$

$$\Downarrow$$

$$y_{j+1} - y_j = \frac{1}{n} g(j/n), \quad j = 0, \dots, n-1$$

only 2 nonzero coefficients per equation

```
from scipy import sparse

n = 100
A = sparse.diags([1,-1], offsets=[1,0], shape=(n,n+1))

print(A)
```

```
  (0, 1)	1.0
  (1, 2)	1.0
  (2, 3)	1.0
  (3, 4)	1.0
  (4, 5)	1.0
  (5, 6)	1.0
```

```
A_denserep = A.toarray()

print(A_denserep)
```

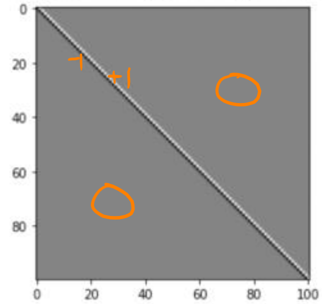
```
[[-1.  1.  0. ...  0.  0.  0.]
 [ 0. -1.  1. ...  0.  0.  0.]
 [ 0.  0. -1. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ...  1.  0.  0.]
 [ 0.  0.  0. ... -1.  1.  0.]
 [ 0.  0.  0. ...  0. -1.  1.]]
```

```
# Sparse representations use less memory (and are faster to use)
print( A.size, A_denserep.size )
```

```
200 10100
```

```
import matplotlib.pyplot as plt

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

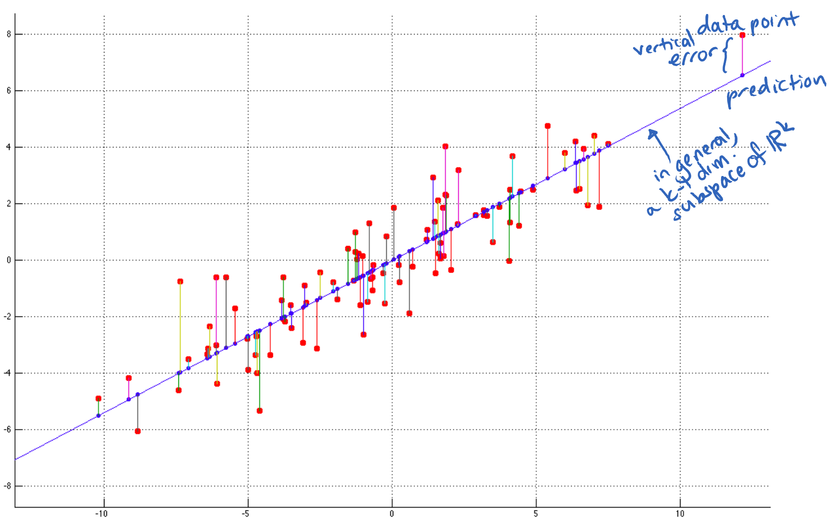
```
<matplotlib.image.AxesImage at 0x7f1
```



Theme:
Dimension reduction

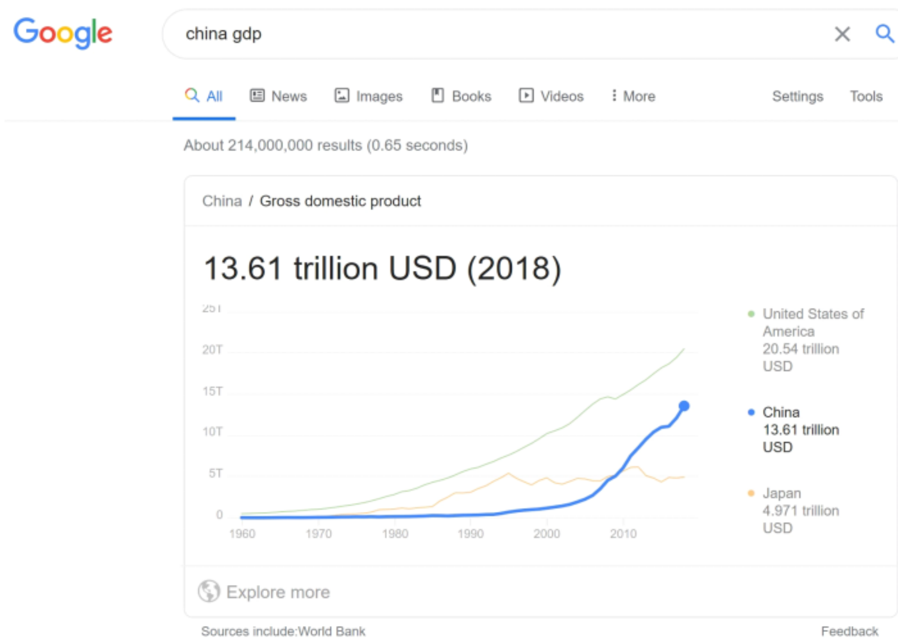
Applications: Least-squares fitting
Principal component analysis (PCA)

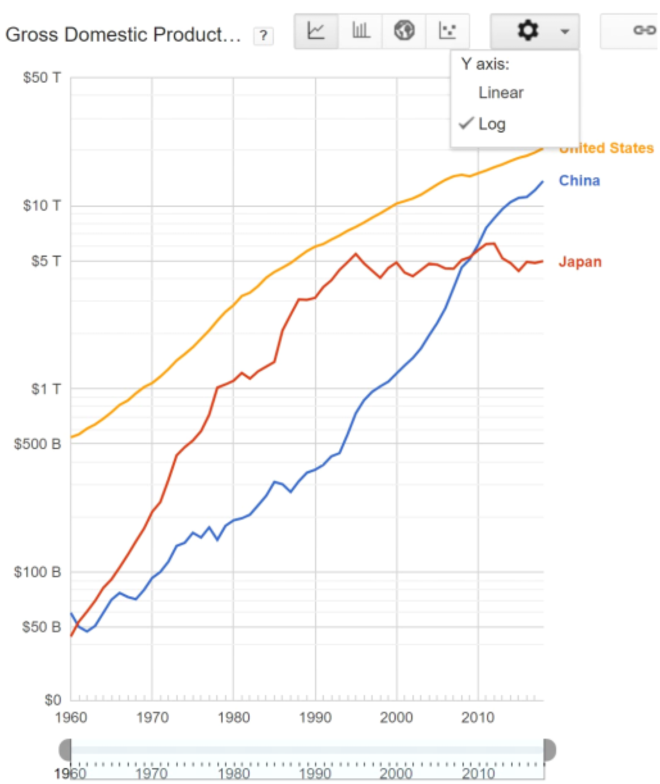
https://en.wikipedia.org/wiki/Principal_component_analysis



Example: Predict China's gross domestic product (GDP) in 2030.

Answer:





$$G_t = G_0 e^{rt}$$

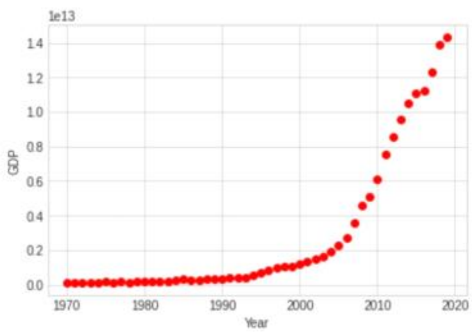
$$\Downarrow$$

$$\log G_t = (\log G_0) + rt$$

straight line

An exponential should fit the data better

```python
# Download the GDP data from the World Bank using pandas

import numpy as np
import pandas as pd

country = 'chn'
download_url = 'http://api.worldbank.org/v2/sources/2/country/' \
               + country + '/series/NY.GDP.MKTP.CD?format=json'
data = pd.read_json(download_url)
data = data['source']['data']

years = np.array([int(term['variable'][2]['value']) for term in data])
values = np.array([float(term['value']) for term in data])


# Plot the data using matplotlib

import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')

plt.scatter(years, values, color='red')
plt.xlabel('Year')
plt.ylabel('GDP')
plt.show()
```




```python
import numpy as np

A = np.vstack(( np.ones(len(years)), years )).transpose()
print(A.shape)

fit = np.linalg.pinv(A).dot( np.log(values) )
print(fit)

prediction = np.exp( fit.dot([1, 2030]) )
print("Prediction for 2030:", prediction)
```

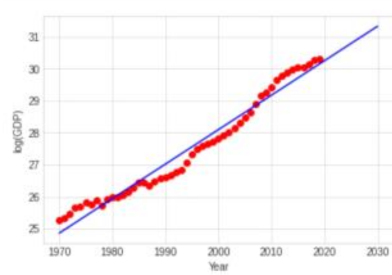
```
(50, 2)
[-1.87647466e+02  1.07871187e-01]
Prediction for 2030: 40448200246701.36
```

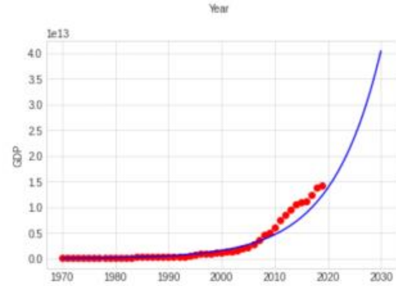
= $40 trillion

```python
# Can also use the "built-in" least-squares function

np.linalg.lstsq(A, np.log(values), rcond=None)[0]
```

```
array([-1.87647466e+02,  1.07871187e-01])
```

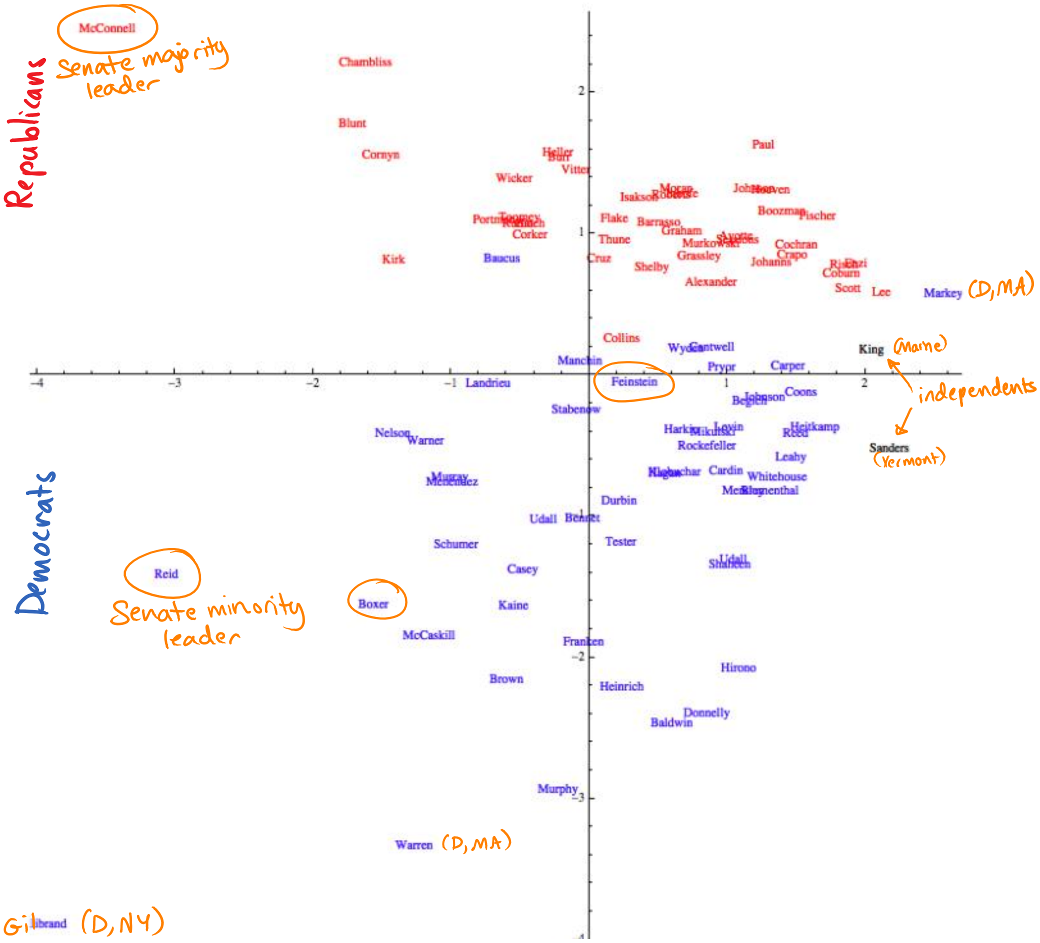
```python
pred_years = np.arange(1970, 2031)
y = np.exp( [fit.dot([1, year]) for year in pred_years] )

plt.scatter(years, np.log(values), color='red')
plt.plot(pred_years, np.log(y), color='blue')
plt.xlabel('Year')
plt.ylabel('log(GDP)')
plt.show()

plt.scatter(years, values, color='red')
plt.plot(pred_years, y, color='blue')
plt.xlabel('Year')
plt.ylabel('GDP')
plt.show()
```





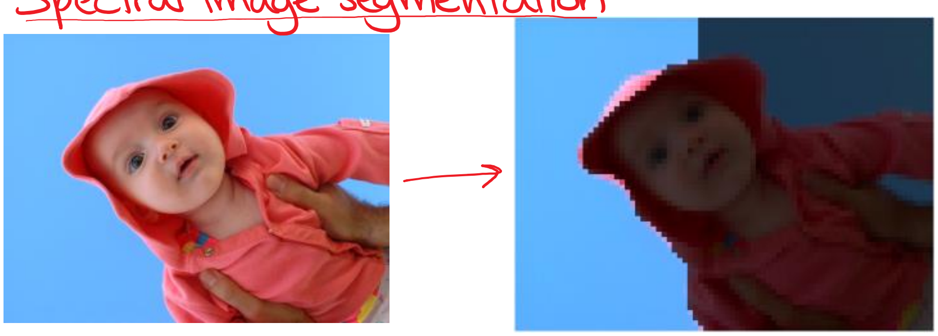
Principal component analysis (PCA)

```
ListPlot[{#} & /@ dataprojectedmanually, AspectRatio → 1, PlotMarkers → data[[All, 1, 1]],
 PlotStyle → (data[[All, 1, 3]]) /. {"D" → Blue, "R" → Red, "I" → Black}]
{97, 17}
{17, 2}
```



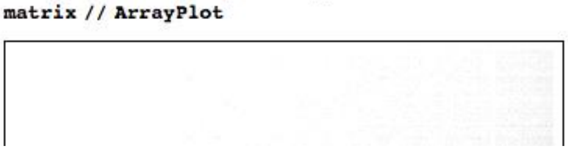
- Even purely combinatorial applications

Spectral image segmentation

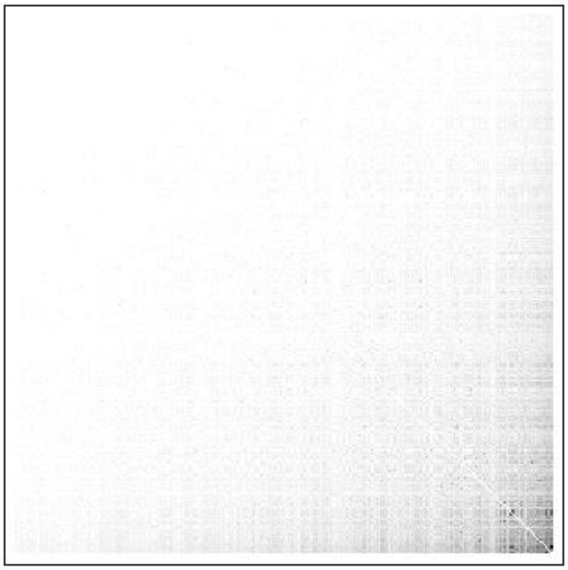


Spectral clustering

```
matrix = Import["/Users/breic/Desktop/adjacencymatrix.txt", "Table"];
matrix += Transpose[matrix];
matrix // ArrayPlot
```



```
matrix = Import["/Users/breic/Desktop/adjacencymatrix.txt", "Table"];
matrix += Transpose[matrix];
matrix // ArrayPlot
```



```
laplacian = DiagonalMatrix[Plus @@ # & /@ matrix] - matrix // N;
di = DiagonalMatrix[(1 / Plus @@ #) & /@ matrix // N];
evs = Eigensystem[√di .laplacian.√di] // Transpose // Reverse;

coordinates = evs[[2 ;; 9, 2]] // Transpose;

numclusters = 16;
```

embedding into $\mathbb{R}^d$

```
ClusteringComponents[coordinates, numclusters, 1, Method → "PAM"]
```

```
{Indiana Jones and the L        A Walk to Remember          Bend It Like Beckham        Con Air
 Lord of the Rings: The         Coyote Ugly                 Bridget Jones's Diary       Double Jeopardy
 Lord of the Rings: The         Dirty Dancing               Frida                       Gone in 60 Seconds
 Lord of the Rings: The         How to Lose a Guy in 10     Life Is Beautiful           Independence Day
 Raiders of the Lost Ark ,      Maid in Manhattan           Love Actually               Lethal Weapon 4
 Star Wars: Episode IV:         Pretty Woman              , Moulin Rouge              , Men in Black II            ,
 Star Wars: Episode VI:         Sister Act                  My Big Fat Greek Weddin     Pearl Harbor
 Star Wars: Episode V: T        The Princess Diaries 2:     Pride and Prejudice         The Fast and the Furiou
 The Lord of the Rings:         The Princess Diaries (W     Rabbit-Proof Fence          The Patriot
                                The Wedding Planner         Shakespeare in Love         Tomb Raider
                                What Women Want             Whale Rider                 Twister

                                                                                        2001: A Space Odyssey
 12 Angry Men                   A Bug's Life                Amelie                      All the President's Men
 Airplane!                      Breakfast at Tiffany's      American Beauty             Blade Runner
 American Pie                   City of Angels              Being John Malkovich        Gandhi
 American Pie 2                 Ever After: A Cinderell     Crouching Tiger             Jaws
 Austin Powers in Goldme        Finding Nemo (Widescree     Election                    L.A. Confidential
 Austin Powers: Internat        Grease                      Eternal Sunshine of the     Lawrence of Arabia
 Austin Powers: The Spy         Harry Potter and the Ch     High Fidelity               Lord of the Rings: The
 Interview with the Vamp ,      Harry Potter and the Pr ,   Lock                      , One Flew Over the Cucko ,
 Liar Liar                      Harry Potter and the So     Lost in Translation         Seven Samurai
 Meet the Parents               Runaway Bride               Magnolia                    The Aviator
 Ransom                         The Lion King: Special      Run Lola Run                The Exorcist
 Spaceballs                     The NeverEnding Story       Rushmore                    The Godfather
 Spider-Man                     The Princess Bride          Sideways                    The Godfather
 Wayne's World                  The Sound of Music          The Royal Tenenbaums        The Graduate
                                Willy Wonka & the Choco     Y Tu Mama Tambien           The Great Escape
                                                                                        The Maltese Falcon
```

Cold Mountain
Collateral
Crash
Fahrenheit 9/11
Finding Neverland
Hotel Rwanda
Man on Fire
Master and Commander: T
Million Dollar Baby
Ocean's Twelve
Ray
Road to Perdition
Runaway Jury
Seabiscuit
The Manchurian Candidat
The Notebook
The Phantom of the Oper
The Pianist

,

A Fish Called Wanda
Alien: Collector's Edit
Back to the Future
Back to the Future Part
Batman
Die Hard 2: Die Harder
Die Hard With a Vengean
Goldfinger
Groundhog Day
Indiana Jones and the T
Men in Black
Mission: Impossible
Predator: Collector's E
Rocky
Speed
Star Trek II: The Wrath
Terminator 2: Extreme E
The Hunt for Red Octobe
The Terminator
True Lies

,

A Knight's Tale
Ice Age
Jurassic Park
Lara Croft: Tomb Raider
Minority Report
Pirates of the Caribbea
Rush Hour
Rush Hour 2
Sleeping Beauty: Specia
Spider-Man 2
Star Wars: Episode II:
Star Wars: Episode I: T
Terminator 3: Rise of t
The Fifth Element
The Incredibles
The Matrix
The Matrix: Reloaded
The Matrix: Revolutions
The Mummy
The Mummy Returns
X2: X-Men United
X-Men

,

Adaptation
A Few Good Men
Air Force One
Armageddon
Clear and Present Dange
Crimson Tide
Enemy of the State
Entrapment
High Crimes
In the Line of Fire
Lethal Weapon
Lethal Weapon 2
Lethal Weapon 3
Patriot Games
Rules of Engagement
Swordfish
The Bone Collector
The Client
The Fugitive
The Negotiator
The Pelican Brief
The Rock
The Sum of All Fears

,

12 Monkeys
Almost Famous
American History X
Anchorman: The Legend o
Donnie Darko
Garden State
GoodFellas: Special Edi
Grosse Pointe Blank
Heat: Special Edition
Kill Bill: Vol. 1
Kill Bill: Vol. 2
Memento
Napoleon Dynamite
Office Space
Pulp Fiction
Requiem for a Dream
Reservoir Dogs
Seven
Sin City

,

Ace Ventura: Pet Detect
A League of Their Own
A River Runs Through It
Basic Instinct
Cheaper by the Dozen
Daddy Day Care
Erin Brockovich
Face/Off
Father of the Bride
Kindergarten Cop
Legally Blonde
Mrs. Doubtfire
Notting Hill
Pay It Forward
Phenomenon
Serendipity
Shall We Dance?
Sleepless in Seattle
Steel Magnolias

,

50 First Dates
Anger Management
Bad Boys II
Behind Enemy Lines
Bruce Almighty
Dodgeball: A True Under
Harold and Kumar Go to
Hero
Hidalgo
Hitch
Hostage
I Robot
Meet the Fockers
National Treasure
Ocean's Eleven
Sahara
Shrek 2
The Bourne Identity
The Bourne Supremacy
The Count of Monte Cris

,

Apollo 13
As Good as It Gets
Black Hawk Down
Boys Don't Cry
Cast Away
Chocolat
Dances With Wolves: Spe
Dead Man Walking
Driving Miss Daisy
Enemy at the Gates
E.T. the Extra-Terrestr
Field of Dreams
Forrest Gump
Fried Green Tomatoes
Gladiator
Glory
Good Will Hunting
Jerry Maguire
Moonstruck
My Cousin Vinny
October Sky
Philadelphia
Primal Fear
Rain Man
Remember the Titans

}

2. algorithms based on fast SDD solvers
 * max-flow & multi-commodity flow problems
   - for decades, best algorithms were deterministic, combinatorial
     [Goldberg-Rao '98]: $\tilde{O}(m\sqrt{n}/\varepsilon)$ for $(1-\varepsilon)$-approx max flow
     augmenting paths
     blocking flows...
   - recent breakthroughs based on numerical linear algebra, spectral graph theory
     [S-H Teng et al. '11]: $\tilde{O}(m n^{1/3}/\varepsilon^{11/3})$
     USC
     [Kelner et al., '13; Sherman '13]:
     $\tilde{O}(m/\varepsilon^2)$ or $\tilde{O}(km/\varepsilon^2)$ for k flows
 * generating random spanning trees
 * graph sparsification
 * sparsest cut
 * distributed routing