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

## MOTIVATION FOR LINEAR ALGEBRA

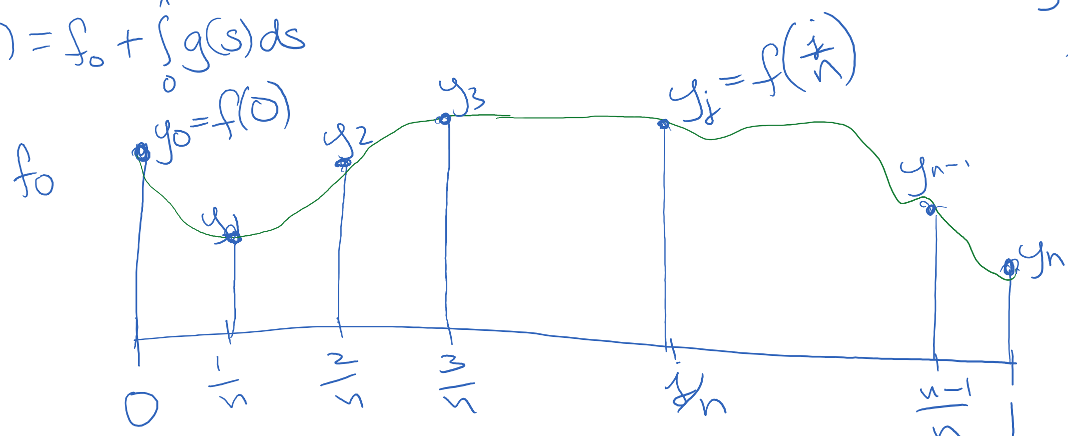
**Theory**: Linear transformations are everywhere!
- ~ Signals: Fourier transform is linear
- ~ Physics: Quantum time evolution is linear
- ~ Calculus: Integration and differentiation are linear

**Applications**: countless...

**Application**: Solving differential equations

example:
Solve $\boxed{\begin{array}{l} f'(x) = g(x) \\ f(0) = f_0 \end{array}}$ $\quad \underline{\forall x \in [0,1]}$ $\quad \xrightarrow{\text{discretize}}$ $\quad f'\left(\frac{j}{n}\right) = g\left(\frac{j}{n}\right)$
$y_0 = f_0$ ✓

$$f(x) = f_0 + \int_0^x g(s)\,ds$$




n+1 variables $y_j$

$$f'(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta} \approx \frac{f\left(x+\frac{1}{n}\right) - f(x)}{1/n}$$

$$f'\left(\frac{j}{n}\right) \approx n\left(y_{j+1} - y_j\right) = g\left(\frac{j}{n}\right) \qquad j = 0, 1, \ldots, n-1 \ ✓$$

...,

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

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

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

n+1 equations

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

## Example: ECMWF 10-day weather forecasts

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

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

## Application: Solving linear equations

e.g., $\begin{cases} 2x - y = 3 \\ -x + y = -2 \end{cases}$

$\begin{cases} a(2x-y) \\ +b(-x+y) \end{cases} = 3a - 2b$

$a = b = 1$: $x = 1 \Rightarrow y = -1$

Gaussian elimination

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

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

same coefficients of $y_0, \dots, y_n$

- what if #equations > #variables?

e.g., $\begin{cases} x = 1 \\ x = 2 \end{cases}$ $\qquad$ e.g., $\begin{cases} 2x - y = 3 \\ -x + y = -2 \end{cases}$ $\qquad$ $\begin{cases} x + y = 1 \\ 2x + 2y = 0 \end{cases}$

eg.
$\begin{cases} \boxed{x = 1 \\ x = 2} \end{cases}$

no solution!

eg.
$\begin{cases} 2x - y = 3 \\ -x + y = -2 \\ x + y = 4 \end{cases}$

no solution!

$\{ \; 2x + 2y = 0$

Matlab

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

ans =

    1.5000
```

$A*x-b$

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

ans =

    2.4286
    1.2857
```

np.linalg.lstsq

Matlab automatically returns $\arg\min_x \|Ax - b\|$

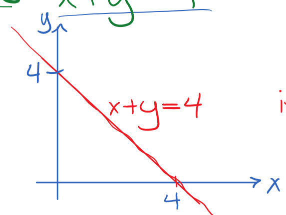
It tries to get as close as possible to a solution.
We'll see more examples below.

ex: variables $x, y, z$

$x + y = 5$
$2x + 2y = 10$

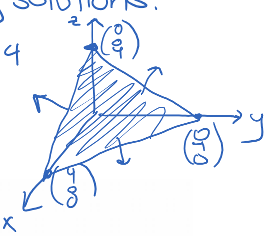
- what if # indep. equations < # variables?

Example: $x + y = 4$




infinitely many solutions!

$x + y + z = 4$




Matlab
https://matlab.mathworks.com/

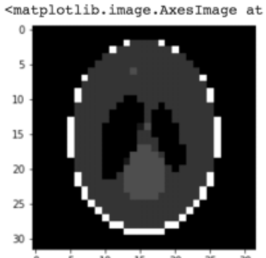
```
>> A = [1 1];
>> b = [4];
>> A \ b

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"

```
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 r
<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]

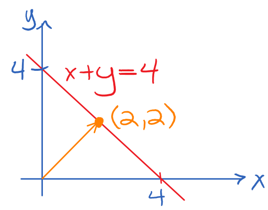
Return the least-squares solution to a linear matrix equation.

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:

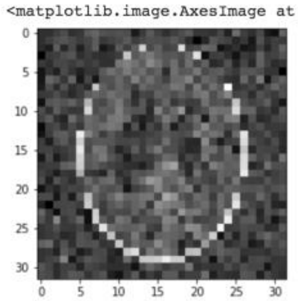
```python
import numpy as np

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
```



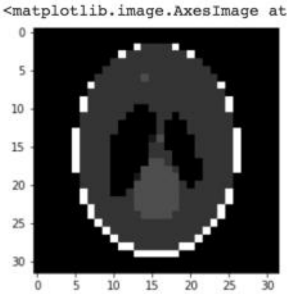
# What is CVXPY?

https://www.cvxpy.org/

CVXPY is a Python-embedded modeling language for convex optimization problems. It 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>
```



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**

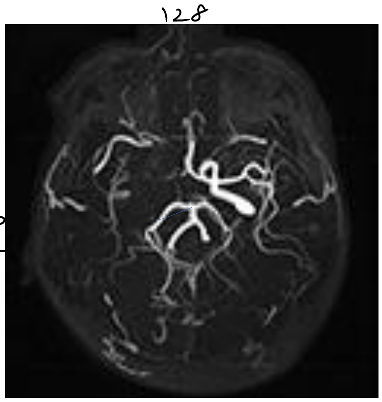
Compressed sensing has been used[36][37] to shorten magnetic resonance imaging scanning sessions on conventional hardware.[38][39][40] Reconstruction methods include

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

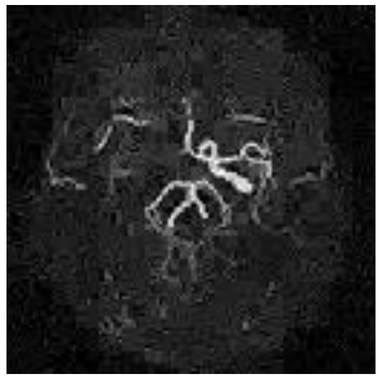
Compressed sensing addresses the issue of high scan time by enabling faster acquisition by measuring fewer Fourier coefficients. This produces a high-quality image with relatively lower scan time. Another application (also discussed ahead) is for CT reconstruction with fewer X-ray projections. Compressed 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]

128

128



$\longrightarrow$



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

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

15 minutes
in $\ell_1$ magic

# Themes

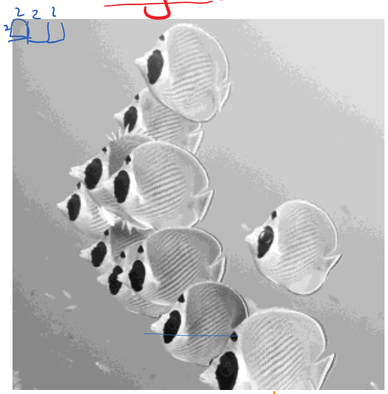
- Geometry
  - linear transformations $A$
  - hyperplanes
- High dimensions
  - sparse matrices ✓
  - dimension reduction ✓
- Systems of linear equations $A\vec{x} = \vec{b}$
- Computer-assisted linear algebra
  - Matlab
  - python/numpy

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

## Application: Image compression

Original

2 2 1
2



not sparse!

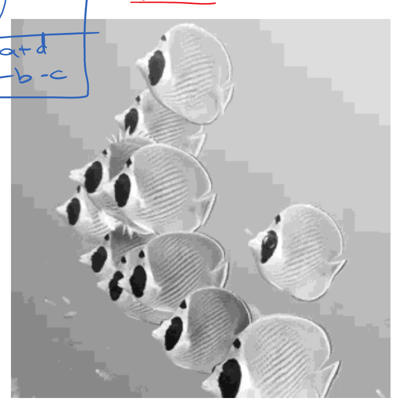
| a | b |
|---|---|
| c | d |

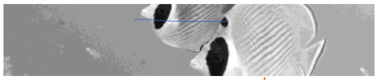
$\longrightarrow$

| $\frac{a+b+c+d}{4}$ | $b+d-a-c$ |
|---|---|
| $c+d-a-b$ | $a+d-b-c$ |

Keep the largest 10% of coefficients in the Hadamard basis

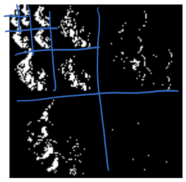
Result





not sparse!

Hadamard basis



sparse (in H basis)

## Theme:
## Sparse matrices

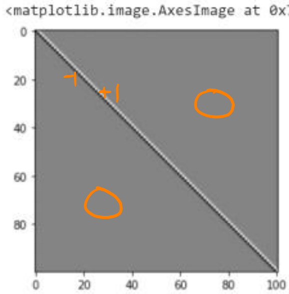
$$f'(x) = g(x)$$
$$\Downarrow$$
$$y_{j+1} - y_j = \frac{1}{n} g(j/n), \quad j = 0, \ldots, n-1$$

only 2 nonzero coefficients per equation

$$0 \cdot y_1 + 0 \cdot y_2 + \cdots - 1 y_j + 1 y_{j+1} + 0 \cdot y_{j+2} + \cdots$$

```
from scipy import sparse

n = 1000
A = sparse.diags([1,-1], offsets=[1,0], shape=(n,n+1))
                                              100 101
print(A)

  (0, 1)      1.0        (0,0)   -1.0
  (1, 2)      1.0        (1,1)   -1.0
  (2, 3)      1.0
  (3, 4)      1.0
  (4, 5)      1.0
  (5, 6)      1.0
  (6, 7)      1.0

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

200 10100
2000  10^6
```

```
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.]]

import matplotlib.pyplot as plt

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

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

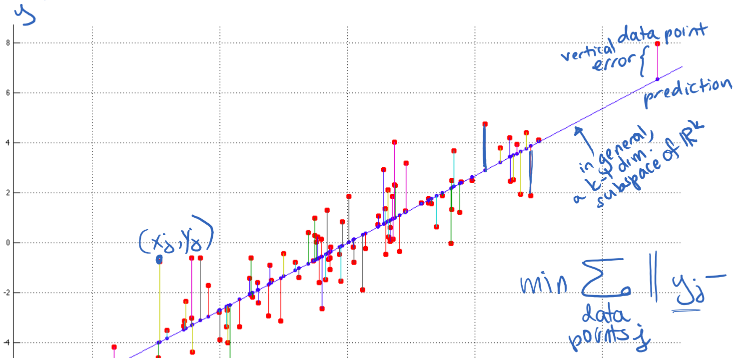



## Theme:
## Dimension reduction

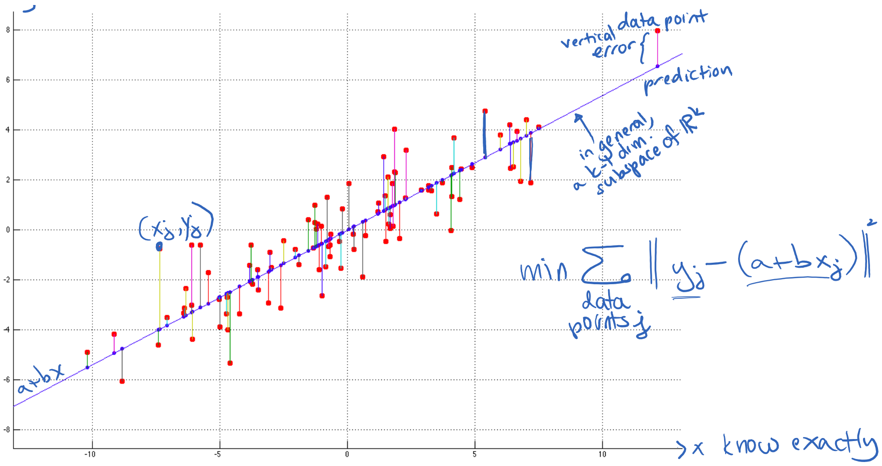
Applications: Least-squares fitting,
Principal component analysis (PCA) –

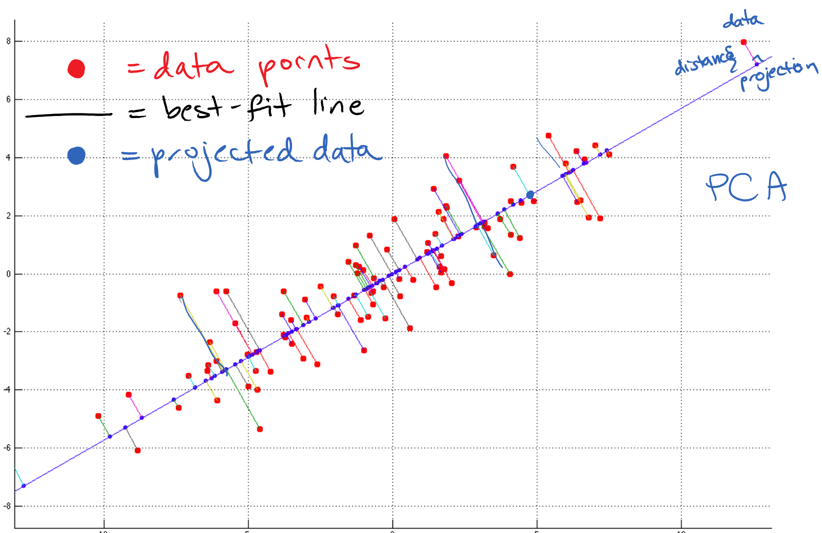
https://en.wikipedia.org/wiki/Principal_component_analysis

y noisy



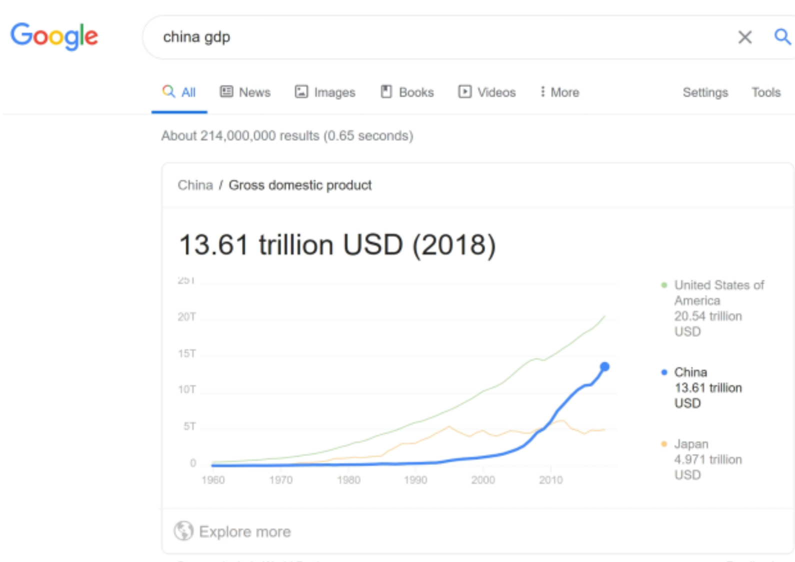

$$\min \sum_{\substack{\text{data}\\\text{points } j}} \| y_j - (a + b x_j) \|^2$$

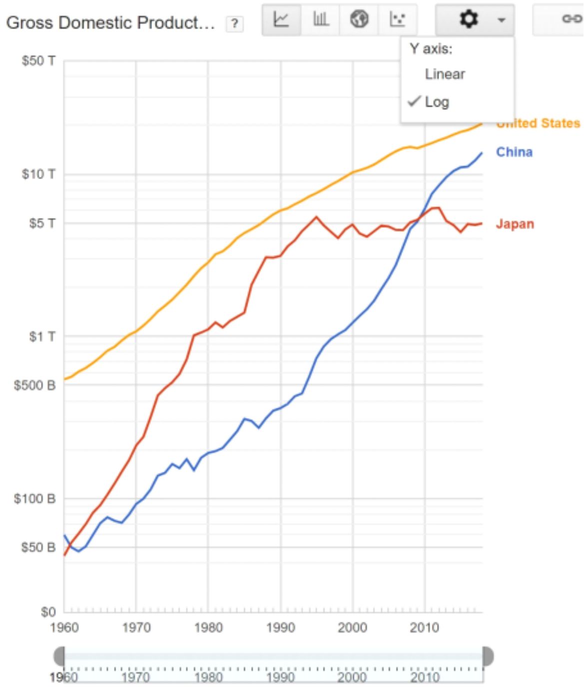



**Example**: Predict China's gross domestic product (GDP) in 2030.

Answer:






$G_t = G_0 e^{rt}$

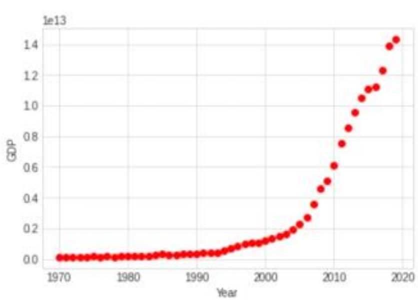
$\Downarrow$

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

straight line

An exponential should fit the data better

```
# 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)
```

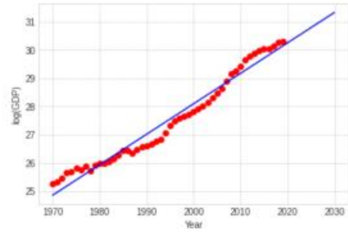
```
(50, 2)
[-1.87647466e+02  1.07871187e-01]
Prediction for 2030: 40448200246701.36
```

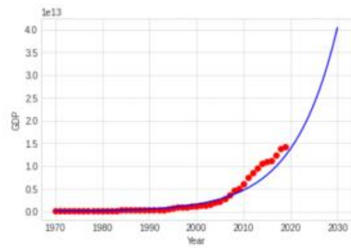
= $40 trillion

```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])
```

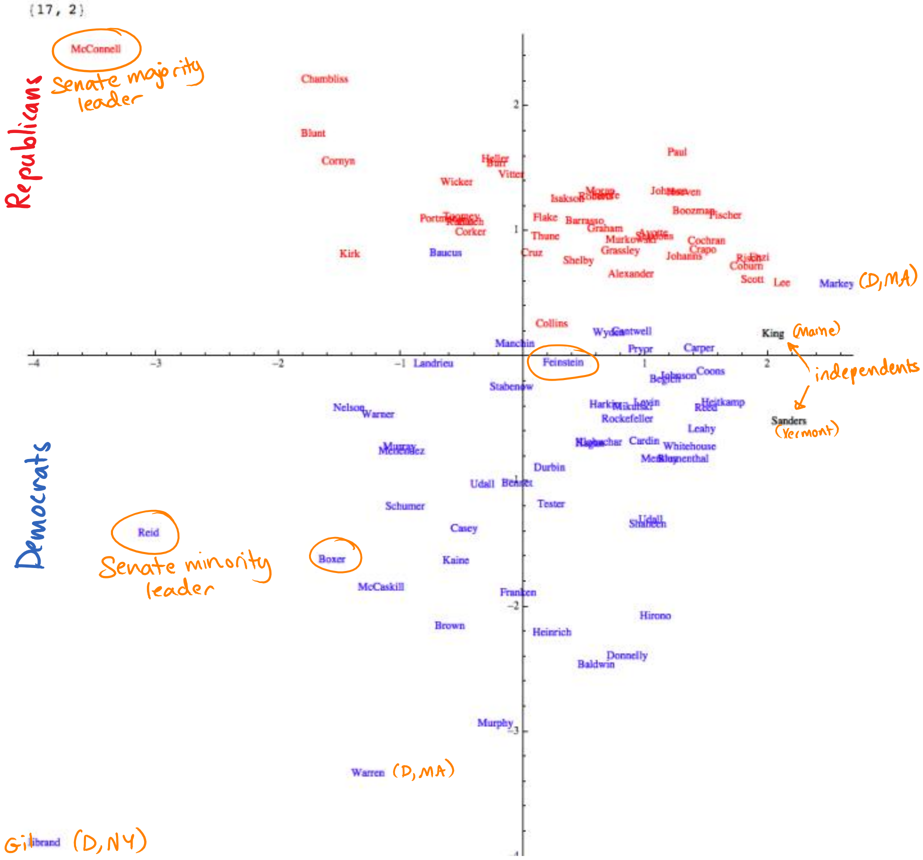
```python
pred_years = np.arange(1970, 2031)
y = np.exp( [fit.dot([1, year]) for year in pred_years] )

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()
```





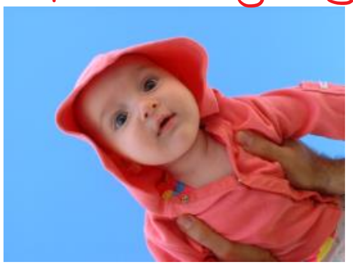
# Principal component analysis (PCA)

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



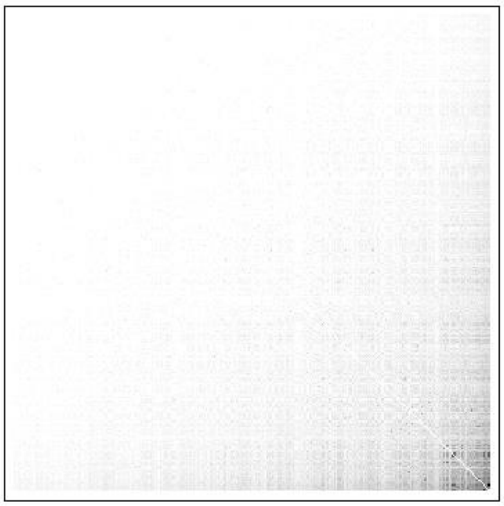

• Even purely combinatorial applications

# Spectral image segmentation



# Spectral clustering

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



```
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 |

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

| | | | |
|---|---|---|---|
| Cold Mountain | A Fish Called Wanda | A Knight's Tale | Adaptation |
| Collateral | Alien: Collector's Edit | Ice Age | A Few Good Men |
| Crash | Back to the Future | Jurassic Park | Air Force One |
| Fahrenheit 9/11 | Back to the Future Part | Lara Croft: Tomb Raider | Armageddon |
| Finding Neverland | Batman | Minority Report | Clear and Present Dange |
| Hotel Rwanda | Die Hard 2: Die Harder | Pirates of the Caribbea | Crimson Tide |
| Man on Fire | Die Hard With a Vengean | Rush Hour | Enemy of the State |
| Master and Commander: T | Goldfinger | Rush Hour 2 | Entrapment |
| Million Dollar Baby | Groundhog Day | Sleeping Beauty: Specia | High Crimes |
| Ocean's Twelve , | Indiana Jones and the T , | Spider-Man 2 | In the Line of Fire |
| Ray | Men in Black | Star Wars: Episode II: , | Lethal Weapon |
| Road to Perdition | Mission: Impossible | Star Wars: Episode I: T | Lethal Weapon 2 , |
| Runaway Jury | Predator: Collector's E | Terminator 3: Rise of t | Lethal Weapon 3 |
| Seabiscuit | Rocky | The Fifth Element | Patriot Games |
| The Manchurian Candidat | Speed | The Incredibles | Rules of Engagement |
| The Notebook | Star Trek II: The Wrath | The Matrix | Swordfish |
| The Phantom of the Oper | Terminator 2: Extreme E | The Matrix: Reloaded | The Bone Collector |
| The Pianist | The Hunt for Red Octobe | The Matrix: Revolutions | The Client |
| | The Terminator | The Mummy | The Fugitive |
| | True Lies | The Mummy Returns | The Negotiator |
| | | X2: X-Men United | The Pelican Brief |
| | | X-Men | The Rock |
| | | | The Sum of All Fears |

| | | | |
|---|---|---|---|
| 12 Monkeys | Ace Ventura: Pet Detect | 50 First Dates | Apollo 13 |
| Almost Famous | A League of Their Own | Anger Management | As Good as It Gets |
| American History X | A River Runs Through It | Bad Boys II | Black Hawk Down |
| Anchorman: The Legend o | Basic Instinct | Behind Enemy Lines | Boys Don't Cry |
| Donnie Darko | Cheaper by the Dozen | Bruce Almighty | Cast Away |
| Garden State | Daddy Day Care | Dodgeball: A True Under | Chocolat |
| GoodFellas: Special Edi | Erin Brockovich | Harold and Kumar Go to | Dances With Wolves: Spe |
| Grosse Pointe Blank | Face/Off | Hero | Dead Man Walking |
| Heat: Special Edition | Father of the Bride | Hidalgo | Driving Miss Daisy |
| Kill Bill: Vol. 1 | Kindergarten Cop | Hitch | Enemy at the Gates |
| Kill Bill: Vol. 2 | Legally Blonde | Hostage | E.T. the Extra-Terrestr |
| Memento | Mrs. Doubtfire | I Robot | Field of Dreams |
| Napoleon Dynamite , | Notting Hill , | Meet the Fockers | Forrest Gump |
| Office Space | Pay It Forward | National Treasure , | Fried Green Tomatoes |
| Pulp Fiction | Phenomenon | Ocean's Eleven | Gladiator |
| Requiem for a Dream | Serendipity | Sahara | Glory |
| Reservoir Dogs | Shall We Dance? | Shrek 2 | Good Will Hunting |
| Seven | Sleepless in Seattle | The Bourne Identity | Jerry Maguire |
| Sin City | Steel Magnolias | The Bourne Supremacy | Moonstruck } |
| | | The Count of Monte Cris | 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] (USC): $\tilde{O}(m n^{1/3}/\varepsilon^{11/3})$
      [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