

LECTURE NOTES

CS 371 / AMATH 242: INTRODUCTION TO COMPUTATIONAL MATHEMATICS

Prepared by: CALVIN KENT  
[www.student.math.uwaterloo.ca/~c2kent/](http://www.student.math.uwaterloo.ca/~c2kent/)

Instructor: AD TAYAL  
Term: WINTER 2018

Last revised: August 11, 2018

# Table of Contents

	<b>Page</b>
<b>PREFACE AND ACKNOWLEDGMENTS</b>	<b>i</b>
<b>CHAPTER 1 - ERRORS AND PROPAGATION</b>	<b>1</b>
Stability and Conditioning . . . . .	1
Unstable Algorithm Example . . . . .	1
Floating Point Arithmetic . . . . .	3
Binary Example . . . . .	4
Standard Floating Systems . . . . .	5
Machine Precision and Machine Epsilon . . . . .	6
Relative Error . . . . .	8
Floating Point Operations . . . . .	8
Condition Number . . . . .	10
Stability . . . . .	12
<b>Summary of Conditioning</b> . . . . .	<b>14</b>
Asymptotic Behavior of Polynomials . . . . .	15
Big-Oh . . . . .	16
<b>CHAPTER 2 - ROOT FINDING</b>	<b>16</b>
Intermediate Value Theorem . . . . .	20
Bisection Algorithm . . . . .	20
Fixed Point Iteration . . . . .	22
Roots vs. Fixed Points . . . . .	23
Newton's Iteration . . . . .	23
Newton's Iteration Algorithm . . . . .	24
Forward Difference . . . . .	25
Secant Method . . . . .	25
Stopping Criteria . . . . .	25
Convergence . . . . .	29
Convergence in Bisection . . . . .	29
Convergence in Fixed Point Iteration . . . . .	30
Contraction . . . . .	30
Contraction Mapping . . . . .	31

Convergence in Newton's Iterations . . . . .	34
Convergence in Secant Method . . . . .	37
<b>Summary of Convergence . . . . .</b>	<b>37</b>
<b>CHAPTER 3 - NUMERICAL LINEAR ALGEBRA . . . . .</b>	<b>38</b>
Google Page Rank . . . . .	38
Problems a Surfer May Encounter . . . . .	40
Matrix Representation of Dead End and Cycling Pages . . . . .	41
Google Matrix . . . . .	44
Markov Transition Matrices . . . . .	45
Page Rank Algorithm . . . . .	46
Roots vs. Fixed Points . . . . .	49
Gaussian Elimination . . . . .	49
LU Factorization . . . . .	51
<b>Summary of LU Decomposition . . . . .</b>	<b>52</b>
Work Estimates . . . . .	52
LU Factoring Flop Count . . . . .	53
Computing Inverse of a Matrix . . . . .	54
Stability of Factorization . . . . .	56
Row Pivoting . . . . .	57
Diagonally and Column Diagonally Dominant Matrices . . . . .	59
Condition Numbers and Norms . . . . .	59
Matrix Norms . . . . .	61
Stability and Conditioning . . . . .	62
Observations about Condition Number . . . . .	66
Residual . . . . .	67
Iterative Methods to Solve Systems . . . . .	69
Jacobi Iteration . . . . .	69
Gauss-Siedel Method . . . . .	71
General Iterative Methods . . . . .	71
Power Method . . . . .	74
Power Algorithm . . . . .	75
Newton Iteration for System of Equations . . . . .	76
<b>CHAPTER 4 - INTERPOLATION . . . . .</b>	<b>79</b>
Polynomial Interpolation . . . . .	80
Lagrange Basis Functions . . . . .	82
Piecewise Polynomial Interpolation . . . . .	85
Piecewise Hermite Interpolation . . . . .	90
Knots and Nodes . . . . .	90
Spline Interpolation . . . . .	91
Spline Interpolant . . . . .	91
Cubic Spline . . . . .	91
Number of Unknowns and Equations in Cubic Spline . . . . .	91
Free, Clamped and Periodic Boundary Conditions in Cubic Spline . . . . .	92

Not-a-knot Spline . . . . .	93
Cubic Spline Example . . . . .	94
Application: Zero Coupon Yield Interpolation . . . . .	96
Polynomial Interpolation Error . . . . .	100
<b>CHAPTER 5 - INTEGRATION</b>	<b>101</b>
Numerical Integration . . . . .	101
Trapezoidal Rule . . . . .	101
Quadratic Rule . . . . .	102
Error for Trapezoidal Method . . . . .	103
Simpson's Rule . . . . .	107
Degree of Precision . . . . .	108
<b>Summary of Properties for Quadratic Rules</b> . . . . .	<b>110</b>
Gaussian Quadrature . . . . .	111
<b>CHAPTER 6 - DISCRETE FOURIER METHODS</b>	<b>114</b>
Fourier Approximation . . . . .	116
Fourier Series . . . . .	118
Fourier Series Coefficients . . . . .	119
Comparison of Fourier Series to Taylor Series . . . . .	120
Complex Form of Fourier Series . . . . .	122
Discrete Fourier Transform . . . . .	125
Roots of Unity . . . . .	126
Computing Fourier Coefficients . . . . .	127

## Preface and Acknowledgments

This PDF document includes lecture notes for CS 371 / AMATH 242 - Introduction to Computational Mathematics taught by Ad Tayal in Winter 2018.

For any questions e-mail me at `c2kent(at)uwaterloo(dot)ca`.

---

Calvin KENT



Jan 3, 2018

## AMATH 242 / CS 371

- $\mathbb{R}$ , Real numbers are used in real world problems.
- Real numbers are approximated as floating point numbers.
- Algebraic operations are rarely exact.

1) Stability - cumulative effect of errors.

2) Conditioning - effect of small perturbations in the input data.

Ex 1: Unstable algorithm.

Compute  $I_n = \int_0^1 \frac{x^n}{(x+\alpha)} dx$ ,  $\alpha > 0$  constant

$n = 0, 1, 2, \dots$  iteratively

$$\text{If } n=0 \Rightarrow I_0 = \log(x+\alpha) \Big|_0^1$$

$$\begin{aligned} n > 0: I_n &= \int_0^1 \frac{x^{n+1}}{x+\alpha} dx = \int_0^1 \frac{x^{n+1}(x+\alpha - \alpha)}{x+\alpha} dx \\ &= \int_0^1 x^{n+1} dx - \alpha \int_0^1 \frac{x^{n+1}}{x+\alpha} dx \\ &= \frac{1}{n+1} - \alpha I_{n+1} \end{aligned}$$

Algo:  $I_0 = \log\left(\frac{1+\alpha}{\alpha}\right)$  for  $i=1, \dots, n$

$$I_i = \frac{1}{i} - \alpha I_{i-1}$$

For  $\alpha = 0.5 \Rightarrow I_{100} \approx 6.64 \cdot 10^{-3}$

$\alpha = 2 \Rightarrow I_{100} \approx 2.1 \cdot 10^{22}$  } wrong because  $I_{100} < \frac{1}{101}$

Ex:

Approximate  $e^{-5.5}$  to 5th decimal point.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

$$\text{and } e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots}$$

$$\Rightarrow e^{-5.5} = 1 - 5.5 + 15.125 - 27.730 + \dots$$

25 terms until convergence in desired decimal point.

$$\Rightarrow e^{-5.5} = \frac{1}{e^{5.5}} = \frac{1}{1 + 5.5 + 15.125 + \dots}$$

25 terms

$$\Rightarrow e^{-5.5} \approx 0.0040865 \rightarrow \text{correct}$$

$$e^{5.5} \approx 0.0026363$$

# Floating Point Arithmetic

Normalized digital exponential of a real number

$$\pi = 3.14159\dots$$

$$\pi = \underset{\substack{\swarrow \\ \text{sign}}}{+} \underbrace{0.314159}_{\text{mantissa}} \cdot \underset{\substack{\downarrow \\ \text{base}}}{10}^{\leftarrow \text{exponent}}$$

$$0.10000 < \text{mantissa} < 0.99999$$

Let  $b$  be the base,  $b$  is a positive number

$b=10$  decimal

$b=2$  binary

$b=16$  hexadecimal

$$\pm .d_1 d_2 \dots \cdot b^p$$

$$d_i \in \{0, \dots, b-1\}$$

Normalized:  $d_1 \neq 0$

$p$  is an integer

Floating point systems use a finite set of bits

- mantissa
- exponent

$$F(b, m, e)$$

base

# bits use  
mantissa

# bits exponent

Ex  $F(b=10, m=5, e=3)$

Input:  $0.00012345689$

Normalize:  $.12345689 \cdot 10^{-3}$

Round:  $.12346 \cdot 10^{-003}$

$x = 0.00012345689 \Rightarrow f(x) = .12346 \cdot 10^{-3}$

Binary Example:

$x = (1101.101)_b$

$x = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$   
 $= 8 + 4 + 1 + \frac{1}{2} + \frac{1}{8} = \underline{\underline{13.625}}$

$F(b=2, m=4, e=3)$

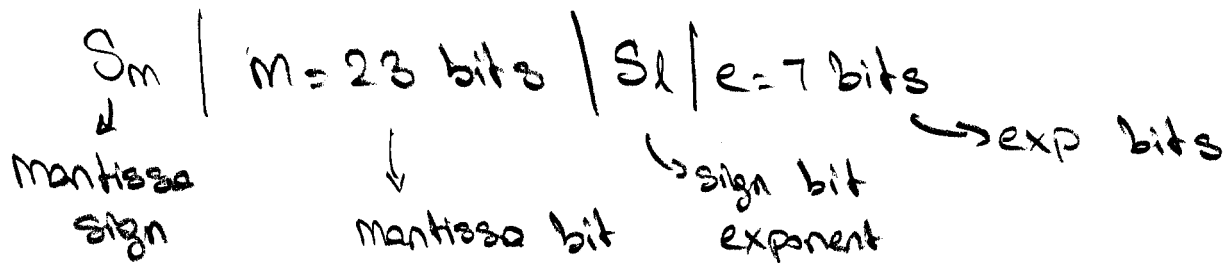
input:  $(1101.101)_b$

normalize:  $(.1101101)_b \cdot 2^4$   
 $= (.1101101)_b \cdot 2^{(100)_b}$

round:  $(.1110)_b \cdot 2^{(100)_b} = \underline{\underline{14}}$

# Standard floating point systems

Single precision: 32 bits, normalized



$$0100 \dots 101000 \dots 1 \Rightarrow (1.1)_b \cdot (2)^{11}_b = \frac{1}{2} \cdot 2^1 = 1.0$$

$$F(2, 23, 7)$$

$$\hat{x}_{\max} = \underbrace{0111 \dots 1}_{23 \text{ terms}} \mid \underbrace{0111 \dots 1}_{7 \text{ terms}}$$

$$= \left( \frac{1}{2} + \frac{1}{2^2} + \dots + \frac{1}{2^{23}} \right) \cdot 2^{127} = \left( 1 - \frac{1}{2^{23}} \right) \cdot 2^{127} \approx 1.7 \cdot 10^{38}$$

$$\hat{x}_{\min} = 0100 \dots 1111 \dots 1$$

$$= \frac{1}{2} \cdot 2^{-127} = 2^{-128} \approx 2.9 \cdot 10^{-39}$$

Single precision: 32 bits

$$F(2, 23, 7)$$

Double precision: 64 bits

$$F(b=2, m=52, e=10)$$

# Machine Precision

$$x \rightarrow fl(x)$$

Relative error,  $\delta_x$

$$\delta_x = \frac{x - fl(x)}{x}, \quad x \neq 0$$

Machine Epsilon is the smallest number  $\epsilon > 0$  such that  $fl(1+\epsilon) > 1$ , strictly speaking  $fl(1+\epsilon) \neq 1$ .

- Chopping:  $\epsilon_{\text{machine}} = \epsilon_{\text{mach}} = b^{1-m}$

- Rounding:  $\epsilon_{\text{mach}} = \frac{1}{2} b^{1-m}$

$m=5$ . Normalize  $1.0 = 0.10000 \cdot b^1$

$$\epsilon_{\text{mach}} = b^{-4} = 0.0001$$

(i)  
(ii)

$$b^{1-5} = b^{-4}$$

(i) + (ii)

$$\begin{array}{r} 0.10000 \cdot b^1 \\ + 0.00001 \cdot b^1 \\ \hline 0.10001 \cdot b^1 > 0.10000 \cdot b^1 \end{array}$$

- Theorem: For any floating point system  $R$  under chopping we have

$$|\delta_x| = \left| \frac{x - fl(x)}{x} \right| \leq \epsilon_{\text{mach}} \quad x \neq 0$$

Corollary:  $|x - fl(x)| = |\eta x|$  where  $|\eta| \leq \epsilon_{mach}$

$fl(x) = x(1 + \eta)$

Single precision:  $m = 23$

$$\epsilon_{mach} = 2^{-22} \approx 0.24 \cdot 10^{-6}$$

6-7 precision digits

Double precision:  $m = 52$

$$\epsilon_{mach} = 2^{-51} \approx 0.44 \cdot 10^{-15}$$

15-16 digits

Note: Machine epsilon depends on

- Chopping vs rounding
  - round up
  - randomized rounding

Jan 08, 2018

Assignment 1, last question, financial toolbox.

Given a floating point system  $F(b, m, e)$

$x \in \mathbb{R}$  is stored in  $F$ , denote as  $fl(x)$  (chopping/rounding)

- Special constant "machine epsilon" for f.p system  $F$

$$\epsilon_{\text{mach}} \equiv \min \{ \epsilon > 0 \mid fl(1 + \epsilon) > 1 \} \text{ smallest } \epsilon > 0$$

such that  $fl(1 + \epsilon) > 1$

Chopping:  $\epsilon_{\text{mach}} = b^{1-m}$

Rounding:  $\epsilon_{\text{mach}} = \frac{1}{2} b^{1-m}$

Theorem: Relative error.

$$|d_x| = \left| \frac{x - fl(x)}{x} \right| \leq \epsilon_{\text{mach}}$$

Also implies  $fl(x) = x(1 + \eta)$ ,  $|\eta| \leq \epsilon_{\text{mach}}$

## Floating Point Operations

The symbol  $\oplus$  is used to denote floating pt. addition

Suppose  $x, y \in \mathbb{R}$ , then  $x \oplus y \stackrel{\text{def}}{=} fl(fl(x) + fl(y))$



- We represent  $x$  as  $fl(x)$
- " "  $y$  as  $fl(y)$
- and carry out the sum  $fl(x) + fl(y)$
- chop/round result  $fl(fl(x) + fl(y))$

Recall:  $fl(z) = z(1 + \eta)$  ,  $|\eta| \leq \epsilon_{mach}$

$$\begin{aligned}
 \text{So } x \oplus y &= fl(fl(x) + fl(y)) \\
 &= [fl(x) + fl(y)](1 + \eta_1) \\
 &= [x(1 + \eta_2) + y(1 + \eta_3)](1 + \eta_1) \quad \text{where } |\eta_i| \leq \epsilon_{mach}
 \end{aligned}$$

Now, let's consider the error in  $x \oplus y$

$$\begin{aligned}
 |x \oplus y - (x + y)| &= |[x(1 + \eta_2) + y(1 + \eta_3)](1 + \eta_1) - (x + y)| \\
 &= |x\eta_1 + x\eta_2 + y\eta_3 + x\eta_1\eta_2 + y\eta_1\eta_3|
 \end{aligned}$$

Since  $|\eta_i| \leq \epsilon_{mach}$  then

$$|x \oplus y - (x + y)| \leq |x\epsilon_{mach} + x\epsilon_{mach} + y\epsilon_{mach} + x\epsilon_{mach}^2 + y\epsilon_{mach}^2|$$

$$\leq 2\epsilon_{mach} [ |x| + |y| ] + \epsilon_{mach}^2 [ |x| + |y| ]$$

$$= (|x| + |y|) (2\epsilon_{mach} + \epsilon_{mach}^2)$$

→ Worst case scenario

This error is corrected in next lecture.

Therefore the relative errors in computing  $x \oplus y$  is

$$\frac{|x \oplus y - (x+y)|}{|x+y|} \leq \frac{|x|+|y|}{|x+y|} (2\epsilon_{\text{mach}} + \epsilon_{\text{mach}}^2)$$

Since  $\epsilon_{\text{mach}} \ll 1$  is small, we can write the

above inequality as

$$\frac{|x \oplus y - (x+y)|}{|x+y|} \leq 2\epsilon_{\text{mach}} \left( \frac{|x|+|y|}{|x+y|} \right) \quad \text{because } \epsilon_{\text{mach}}^2 \rightarrow 0$$

↓  
approx.

### Condition Number

The above inequality can be written as

$$\frac{|x \oplus y - (x+y)|}{|x+y|} \leq K_R \cdot \epsilon_{\text{mach}}$$

$$\text{where } K_R = 2 \frac{|x|+|y|}{|x+y|}$$

The multiplier  $K_R$  is called the condition number for this elementary operation given data  $x, y$ .

- the floating point representation of data  $(x, y)$  has an error of size  $\epsilon_{\text{mach}}$

- perform  $x \oplus y$  magnifies this error by factor  $K_R$ .

- If  $K_R \approx 1$ , the operation is "well-conditioned".

- If  $K_R$  is large (relatively large depending on the problem) then the problem is "ill-conditioned".

$$e^{-5.5} = 1 - 5.5 + 15.125 - 27 = \text{bad answer}$$

$$e^{-5.5} = \frac{1}{1 + 5.5 + 15.125} = \text{good answer}$$

We have the condition number for f.p. addition

$$K_R = \frac{|x| + |y|}{|x + y|}$$

- If  $x$  and  $y$  are of opposite signs and approximately the same size then  $K_R$  is large

-- This phenomena is called "catastrophic cancellation" or "cancellation error"

$$\text{Now } e^{-9.5} = 1 - 5.5 + 15.125 - 27.73 + 38.128 - \dots$$

- here we are adding and subtracting large numbers compared to the final result of  $e^{-9.5} \approx 0.004$

- rapid accumulation of round-off errors

$$\text{Compare with } e^{-9.5} = \frac{1}{1 + 5.5 + 15.125}$$

- in this case  $K_R \approx 1$ , so round-off error does not accumulate quickly.

# Stability

$$I_n = \int_0^1 \frac{x^n}{x+\alpha} dx \quad \text{Can be solved using recurrence relationship. } I_n = \frac{1}{n} - \alpha I_{n-1}, \quad n > 1$$

Suppose that the f.p representation of  $I_0$  introduces an error.

$$\begin{aligned} \hat{I}_0 &\stackrel{\text{def}}{=} f(I_0) \\ &= I_0(1+\eta) \\ &= I_0 + I_0\eta \\ &= \underline{\underline{I_0 + E_0}} \end{aligned}$$

Let  $\hat{I}_n \equiv$  approximate value of  $I_n$  including the effect of  $E_0$ .

$$\begin{aligned} \text{Let } E_n &= \hat{I}_n - I_n \\ &= \text{error at the } n^{\text{th}} \text{ stage due to initial error } E_0. \end{aligned}$$

Exact solution satisfies  $I_n = \frac{1}{n} - \alpha I_{n-1}$

Approximate solution also satisfies  $\hat{I}_n = \frac{1}{n} - \alpha \hat{I}_{n-1}$

$$\text{Then } \hat{I}_n - I_n = -\alpha (\hat{I}_{n-1} - I_{n-1})$$

$$E_n = -\alpha E_{n-1}$$

$$\text{Then } E_n = -\alpha E_{n-1} =$$

$$= -\alpha (-\alpha E_{n-2})$$

$$= (-\alpha)^n E_0$$

then if

$|\alpha| < 1$ , recursion is stable

$|\alpha| > 1$ , recursion is unstable

- Matlab:
- Campus computers
  - VPN
  - Torrent
  - Octave

### Correction/Clarification

In deriving the error for  $x \oplus y$

$$\begin{aligned}
 |x \oplus y - (x+y)| &= |x\eta_1 + x\eta_2 + y\eta_3 + y\eta_1 + x\eta_1\eta_2 + y\eta_1\eta_3| \quad \text{where } |\eta_i| \leq \epsilon_{\text{mach}} \\
 &\leq |x\eta_1| + |x\eta_2| + |y\eta_1| + |y\eta_3| + |x\eta_1\eta_2| + |y\eta_1\eta_3| \\
 &\leq |x|\epsilon_{\text{mach}} + |x|\epsilon_{\text{mach}} + |y|\epsilon_{\text{mach}} + |y|\epsilon_{\text{mach}} + |x|\epsilon_{\text{mach}}^2 + |y|\epsilon_{\text{mach}}^2 \\
 &= 2\epsilon_{\text{mach}}(|x|+|y|) + \epsilon_{\text{mach}}^2(|x|+|y|)
 \end{aligned}$$

$$\therefore |x \oplus y - (x+y)| \leq (|x|+|y|)(2\epsilon_{\text{mach}} + \epsilon_{\text{mach}}^2)$$

Note: Subtraction is same as addition with negative entries.

Intuition: Recall:  $f(x) = \overbrace{\hspace{2cm}}^{\text{mantissa}} \underbrace{\hspace{1cm}}_{\text{exponent}}$

$$\begin{aligned}
 x &= m_1 \cdot 2^{p_1} \\
 y &= m_2 \cdot 2^{p_2}
 \end{aligned}
 \quad \text{then} \quad
 \begin{aligned}
 x \cdot y &= (m_1 m_2) \cdot 2^{p_1 + p_2} \\
 x \div y &= (m_1 \div m_2) \cdot 2^{p_1 - p_2}
 \end{aligned}$$

Notice:  $p_1 + p_2$  and  $p_1 - p_2$  are integer operations (not  $(\cdot)$ )

→ No precision loss.

$m_1 - m_2, m_1 \div m_2$  can be carried out full precision  
(i.e. use all digits)

$$\implies K_R \approx 1$$

Contrast this with addition and subtraction we need to align  $P_1, P_2$ . In the "catastrophic cancellation" case what is happening after alignment

$$x = \boxed{\quad m_1 \quad} \boxed{P}$$

$$y = \boxed{\quad m_2 \quad} \boxed{P}$$

If  $m_1 \approx m_2$  only differ in few least significant digits

Conclusion: For basic arithmetic f.p operations only "catastrophic cancellation" we should look out for.

## Summary

### - Conditioning

Consider a problem  $P$  with input  $I$  and output  $Y$ .

• If we change the input to  $I + \Delta I$ , the output changes to  $Y + \Delta Y$ .

• If  $\Delta Y$  is "comparable" or smaller than  $\Delta I$  then the problem  $P$  is "well-conditioned".

• Otherwise  $P$  is "ill-conditioned".

• Precisely measured using "condition numbers".

Note: This definition is independent of any algorithm, it is a statement about a mathematical problem.

### Stability:

An algorithm should give us reasonably correct answers even if small errors are introduced.

- An algorithm is "stable" if small errors do not become exponentially amplified (context: iterative algorithm)
- Stability is a property of the algorithm.

### Asymptotic Behavior of Polynomials

- Given a polynomial  $P(x)$  we say that

$P(x) = O(x^n)$ ,  $x \rightarrow \infty$  if  $\exists c > 0, x_0 > 0$  such that

$$|P(x)| < c|x|^n \quad \forall x, |x| > |x_0| \quad \longrightarrow \text{highest power}$$

i.e.  $P(x) = 2x^3 + 5x^2 + 7 = O(x^3)$  as  $x \rightarrow \infty$

-  $P(x) = O(x^n)$  as  $x \rightarrow 0$  if  $\exists c > 0, x_0 > 0$  such that

$$|P(x)| < c|x|^n \quad \forall x, |x| < |x_0| \quad \longrightarrow \text{lowest power}$$

i.e.  $P(x) = 5x^3 + 2x = O(x)$  as  $x \rightarrow 0$

## Manipulation of terms involving "Big-Oh"

$$f(x) = x + O(x^2) \quad x \rightarrow 0$$

$$g(x) = 2x + O(x^3) \quad x \rightarrow 0$$

$$\begin{aligned} \text{then } f(x) + g(x) &= 3x + O(x^2) + O(x^3) \\ &= 3x + O(x^2) \quad \text{as } x \rightarrow 0 \end{aligned}$$

Taylor series

$$\begin{aligned} f(x_0+h) &= f(x_0) + \frac{f'(x_0)}{1!} h + \frac{f''(x_0)}{2!} h^2 + \frac{f'''(x_0)}{3!} h^3 + \dots \\ &= f(x_0) + f'(x_0)h + O(h^2) \end{aligned}$$

## Root Finding

Motivation: Financial problem

A stock option is a contract which gives the buyer the right, but not the obligation, to buy (call option) or sell (put option) an underlying stock at a specified price (strike price) on or before a specified date (Maturity date).

Say 3M call option for GOOG with strike \$1100  
(with today value \$1106)

The no-arbitrage price of an option is given by Black-Scholes model.



$$P = f_{BS}(S_0, T, K, r, \sigma, \text{Type})$$

$P$  = Value of option

$S_0$  = current stock (underlying value)

$K$  = strike price

$T$  = maturity date (e.g. 0.25 = 3 months)

$r$  = risk free rate

Type: Call or put

$\sigma$  = Volatility of the stock

And volatility is the measure of riskiness or randomness of price movements.

Jan 15, 2018

A-1 revised PDF.

Open book exams.

## Root Finding

### Motivation: Financial Options

The no-arbitrage price of an option is

$$P = f_{BS}(S_0, T, K, r, \sigma, \text{Type})$$

$\sigma$   $\equiv$  volatility of stock

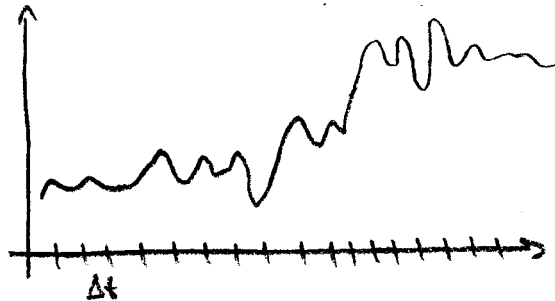
Volatility - measure of riskiness/randomness of price movements

More precisely,  $r_{\Delta t} = \frac{S_{t+\Delta t} - S_t}{S_t}$

$$\sigma^2 = \lim_{\Delta t \rightarrow 0} E \left( r_{\Delta t}^2 \right)$$

$\searrow$  exp operator

- $\sigma$  could be estimated using historical prices.
- - this is backward looking.
- options are actively traded  
→ prices are set by market forces.



- What is actually done is we look at the traded price,  $P$ , and given  $(P, S_0, T, K, r, \text{Type})$  determine or "back-out"  $\sigma$ .
- - This is called "implied volatility"

i.e.  $P = P^*$ ,  $S_0 = S_0^*$ ,  $T = T^*$ ,  $K = K^*$ ,  $r = r^*$ ,  $\text{type} = \text{type}^*$  are known.

Solve  $P = f_{BS}(S_0, T, K, r, \sigma, \text{Type})$  for  $\sigma$ .

$$\text{So, } P^* - f_{BS}(S_0^*, T^*, K^*, r^*, \sigma, \text{Type}^*) = 0$$

- In general, there may be no closed form expression for  $f_{BS}$

e.g. an American option.

-  $f_{BS}(\cdot)$  may be an algorithm, e.g. a lattice model.

- This is an example of a more general problem

\* Given a function  $f(x)$ , find the root  $x^*$  such that  $f(x^*) = 0$

Since nothing in f.p. arithmetic is exact, we rephrase the question in \* as:

\* Given a function  $f(x)$ , find  $x^*$  such that  $|f(x^*)| \leq \text{tol}$   
(tolerance)

[tol does not need to equal  $\epsilon_{\text{mach}}$ ]

~~P2x~~

$$f(x) = x^2 + x - 6 = (x-2)(x+3) \Rightarrow 2 \text{ roots: } x=2, x=-3$$

~~P2x~~

$$f(x) = x^2 - 2x + 1 = (x-1)^2 \Rightarrow x=1 \text{ is a double root}$$
$$f(1) = 0, f'(1) = 0$$

# General Approach

- We will use an iterative method to find roots

Consider a sequence of iterates

$$x_1, x_2, \dots, x_k$$

We want to develop algorithms which generate the seq  $\{x_k\}$  such that  $\lim_{k \rightarrow \infty} x_k = x^*$  where  $f(x^*) = 0$ .

## Intermediate Value Theorem

If  $g(x)$  is cts on  $[a, b]$  and  $c \in [g(a), g(b)]$  then there exists  $x^* \in [a, b]$  such that  $g(x^*) = c$

Then, it follows that,

If we find  $a, b$  such that  $f(a) \cdot f(b) < 0$  and  $f(x)$  is cts on  $[a, b]$  then  $\exists x^* \in [a, b]$  such that  $f(x^*) = 0$ . → opposite sign

## Bisection Algorithm

- Guaranteed to converge.

- Consistently works if we have  $[a, b]$  such that  $f(a)f(b) < 0$  for continuous  $f$ .

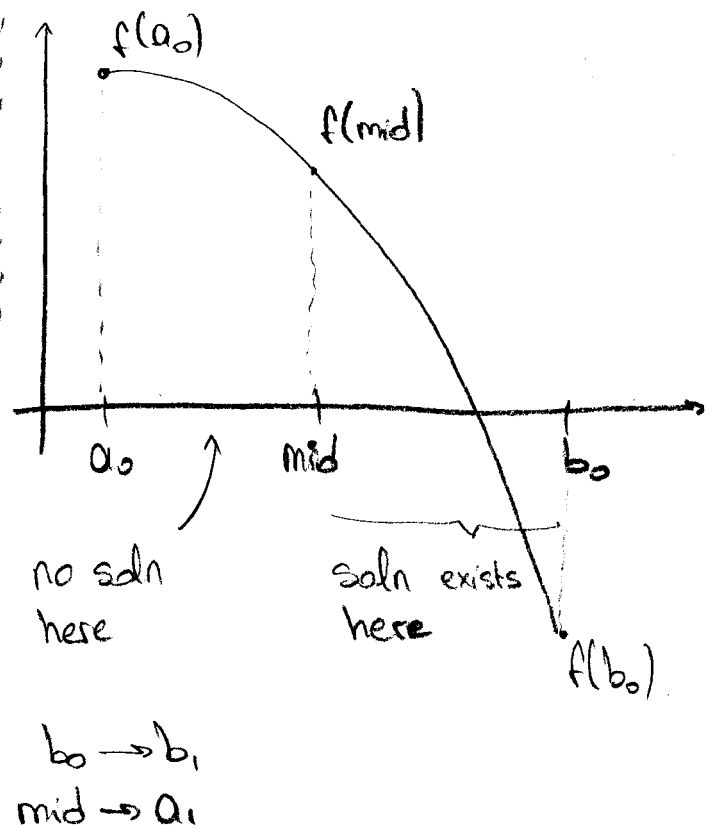
- It is slow

- Used as a starting point for faster method such as Newton's iterations.

# Algorithm

Input:  $f(x)$ ,  $a_0, b_0$ , such that  $f(a_0)f(b_0) < 0$

```
for  $k=0,1,\dots$  [infinite loop or max iterations]
  if  $|b_k - a_k| \leq \text{tol}$  then
     $x^* = (a_k + b_k) / 2$ 
    quit
  end if
   $\text{mid} = (a_k + b_k) / 2$ 
  if  $f(\text{mid})f(a_k) < 0$  then
     $a_{k+1} = a_k$ 
     $b_{k+1} = \text{mid}$ 
  else
     $a_{k+1} = \text{mid}$ 
     $b_{k+1} = b_k$ 
  end if
end for
```



This works by repeatedly halving the interval where a solution is guaranteed to lie.

At convergence we know that  $|b_k - a_k| \leq \text{tol}$

So in  $k$  steps the interval is

$$|b_k - a_k| = \frac{|b_0 - a_0|}{2^k} \leq \text{tol}$$

$$\text{then } \log_2 \frac{|b_0 - a_0|}{\text{tol}} \leq k$$

Take the equality ( $k = \log_2 \frac{|b_0 - a_0|}{\text{tol}}$ ) to find minimum number of

iterations required.

# Fixed Point Iteration

Often we can write  $f(x) = 0$  as  $x - g(x) = 0$

~~But~~ Given  $f(x)$ , define  $g(x) = x - f(x)$ . Then  $x - g(x) = 0$   
 $\Rightarrow x - x + f(x) = 0$   
 $\Rightarrow f(x) = 0$

Note: We can always do the transformation on  $g(x)$  but often the original problem may be given in form  $x - g(x) = 0$

if  $x^*$  is a root of  $f(x)$ , which means  $f(x^*) = 0$ , then  $x^* = g(x^*)$ .

Def<sup>n</sup>:  $x^*$  is a fixed point of  $g(x)$  if  $g(x^*) = x^*$ .

The repeated application of  $g$  converges to this fixed point,  $x^*$ .

$$(g \circ g \circ \dots \circ g)(x^*) = x^*$$

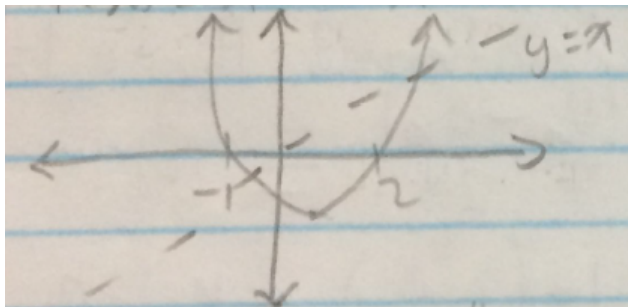
## Algorithm

Input:  $g(x), x_0$

```
For  $k = 0, 1, \dots$  (infinite or max iterations)
   $x_{k+1} = g(x_k)$ 
  if  $|x_{k+1} - x_k| < \text{tol}$  then
    quit
  end if
end for
```

**Roots vs. Fixed Points:**

**Example 5.4:**  $f(x) = x^2 - x - 2 = (x - 2)(x + 1)$ , roots  $x = -1, 2$ . To find fixed points, we need to solve  $f(x) = x - 2 - x = x \implies x^2 - 2x - 2 = 0$ . Graphically, it's usually where  $f(x)$  intersects with  $y = x$ .



**Remark 5.5:** We want roots of  $f(x)$  to be the fixed points of another function  $g(x)$ . So  $x = x^2 - 2$ , let  $g(x) = x^2 - 2$ , get  $f(x) = 0 \implies x^2 - x - 2 = 0$ . Add  $x$  to both sides.

**Newton's Iteration 5.6:** Suppose  $f(x)$  is continuous with finite derivatives  $f'(x), f''(x)$ . Then

$f(x^*) = f(x_0) + f'(x_0)(x^* - x_0) + O((x^* - x_0)^2)$  (obtained by Taylor expansion approximation for  $x^*$  near  $x_0$ ).

This is the linear approximation where  $(x^* - x_0) \rightarrow 0$ . We can say  $f(x^*) \approx f(x_0) + f'(x_0)(x^* - x_0)$ . We want to find  $x^*$  such that  $f(x^*) = 0$ . That is,

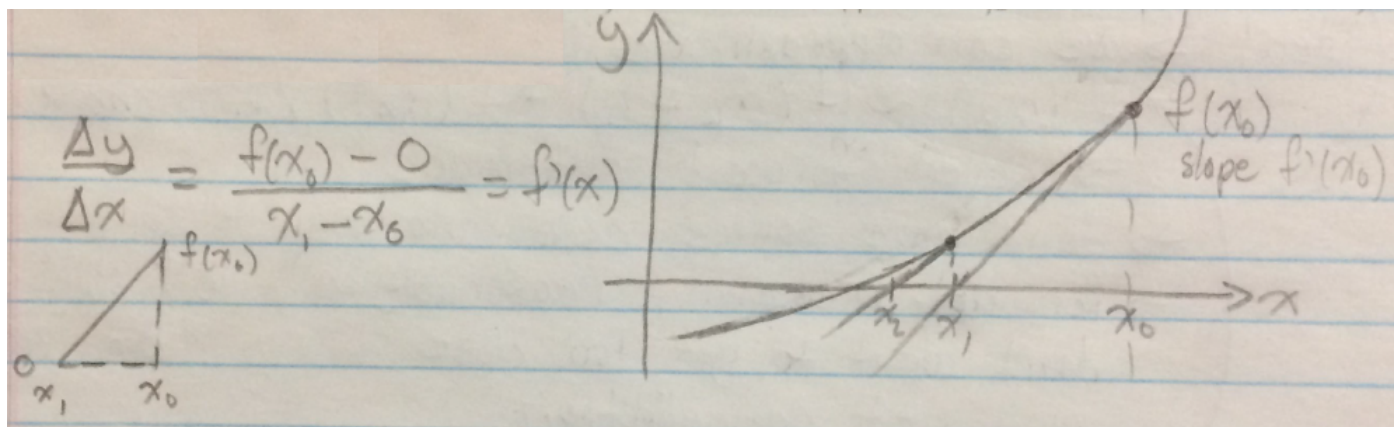
$$f(x_0) + f'(x_0)(x^* - x_0) \approx 0 \implies x^* \approx \frac{-f(x_0)}{f'(x_0)} + x_0. \quad (\star)$$

**Remark 5.7:** We ignored higher order terms  $O((x^* - x_0)^2)$ . Note that  $(\star)$  will not give us  $x^*$  in the first step. We get an estimation  $x_1$  of the root from  $(\star)$ . That is,

$$x_1 = \frac{-f(x_0)}{f'(x_0)} + x_0.$$

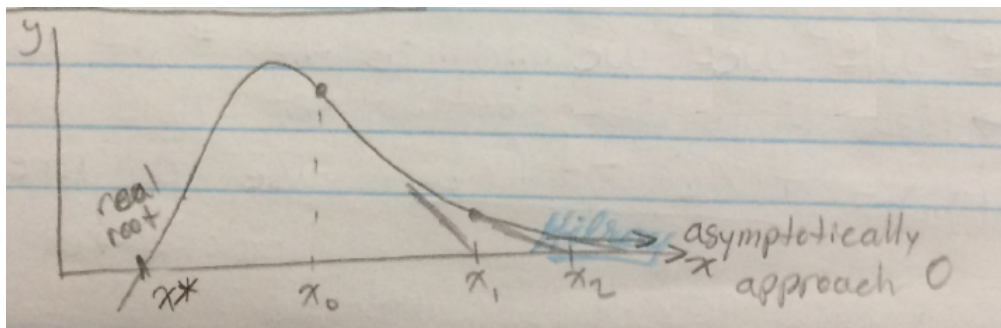
Repeated application gives  $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ .

**Remark 5.8:** (Graph description)

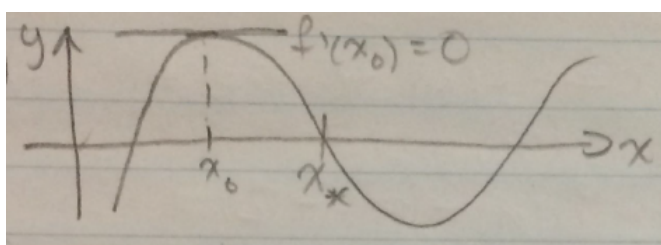


**Remark 5.9:** Newton's method doesn't always converge. We have 3 problematic cases.

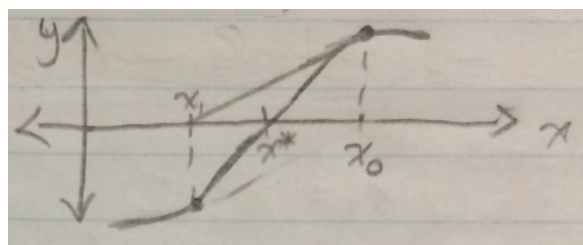
Runaway:



Flat spot:



Cycle:



**Newton's Iteration Algorithm 5.10:**

**Input:**  $f(x), f'(x), x_0, \varepsilon, \text{tol}$ .

**Output:**  $x_k$ .

```

for  $k = 0, 1, 2, \dots$  do                                     // until tolerance (tol) or max iteration
    if  $|f'(x_k)| \leq \varepsilon$  then
        | throw exception
    end
     $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ 
    if  $|x_k - x_{k+1}| < \text{tol}$  then
        | quit
    end
end

```



**Approximate Derivatives 5.11:** In many cases we may not be able to compute  $f'(x)$  exactly. eg.  $f(x)$  may be a complex function or we may not know how  $f(x)$  is computed (we may not have the source code). However, we can approximate  $f(x)$  numerically.

$$f'(x_k) \approx \frac{f(x_k + h) - f(x_k)}{h}, \text{ choose small } h.$$

This is called the forward difference which requires twice as many function evaluations/iterations. For double precision, reasonable  $h$  is in the order of  $10^{-6}, 10^{-9} \sim h$ . We do not want to get too close to  $10^{-15}$  or  $10^{-16}$  to avoid cancellation. Hence we can usually be safe when choosing  $h \sim 10^{-6}, 10^{-9}$ . If we want to minimize the number of function evaluations we need to get a better estimation.

$$f'(x_k) = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \text{ use a previously calculated value.}$$

This method is called the *secant method*.

**Secant Method 5.12:** The iterative algorithm is

$$x_{k+1} = x_k - f(x_k) \left[ \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \right].$$

To compute  $x_{k+1}$ , we need  $x_k$  and  $x_{k-1}$ . We start off (1st iteration) using forward difference. Secant method is similar to Newton's method but it approximate to  $f'(x)$ .

**Remark 5.13:** Secant method will converge slightly slower than Newton's method since it approximates to  $f'(x)$ .

**Stopping Criteria 5.14:** To any iterative algorithm we need to specify a **stopping criteria** that has the following properties.

1. Maximum number of steps: Stop iterating at step  $k = k_{\text{MAX}}$  (this is just a safeguard to avoid infinite loop).
2. Tolerance on size of correction. In other words,  $|x_{k+1} - x_k| < \text{tol}$ . In other words  $x$  converges with bisection. This guarantees that  $|x^* - x_{k+1}| \leq \text{tol}$  eventually. This is not true in Newton's approximation.
3. Tolerance on the function value itself: Stop if  $|f(x_{k+1})| \leq \text{tol}$ . This may not work as well as (2) if  $f(x^*)$  is very small.

**Remark 5.15:** Good stopping criteria depends on the given problem.

Jan 22, 2018

## Convergence

We looked at some root-finding methods which result in a sequence of iterates,  $x_0, x_1, \dots, x_k \rightarrow x^*$  as  $k \rightarrow \infty$

Let  $e_k = x_k - x^*$ , the error at the  $k^{\text{th}}$  iterate.

Defn: The sequence  $(x_k)$  converges to  $x^*$  with order  $q$

if and only if  $\lim_{k \rightarrow \infty} x_k = x^*$ ;

$$\lim_{k \rightarrow \infty} \frac{|e_{k+1}|}{|e_k|^q} = A, \quad A \in (0, \infty)$$

More informally:  $|e_{k+1}| = C_k |e_k|^q$ ,  $\lim_{k \rightarrow \infty} C_k = A$

We would like  $q$  to be as large as possible

$q = 1 \rightarrow$  linear convergence

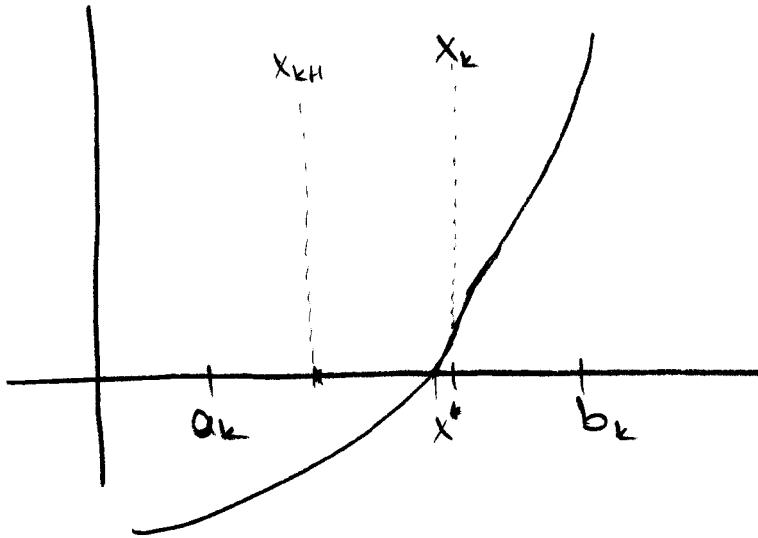
$q = 2 \rightarrow$  quadratic convergence

## Bisection

From our previous discussion of bisection, we know that

$$L_{k+1} \stackrel{\text{def}}{=} |b_{k+1} - a_{k+1}| = \frac{1}{2} L_k$$

- So the sequence  $\{L_k\}$  converges to 0 linearly
- however, we cannot directly use our definition of order of convergence, since bisection can do strange things



$x_k$  is closer to root than  $x_{k+1}$

- error may actually increase on some iterations.
- But  $|e_k| \leq L_k$ , we can say that the bound on the error is linearly convergent. Recall  $L := \text{interval length}$  (i.e.:  $b_k - a_k$ )

### FIXED POINT ITERATION

#### Defn: Contraction

Given a continuous bounded function  $g(x)$  defined on  $x \in [a, b]$ , then  $g(x)$  is a contraction on  $[a, b]$  if  $\exists L \in (0, 1)$  such that  $|g(x) - g(y)| \leq L|x - y| \quad \forall x, y \in [a, b]$ .

Theorem: If  $g(x)$  is differentiable on  $[a, b]$ , then

$$\forall x, y \in [a, b], \quad |g(x) - g(y)| \leq k|x - y|$$

$$\text{where } k = \max_{x \in [a, b]} |g'(x)|$$

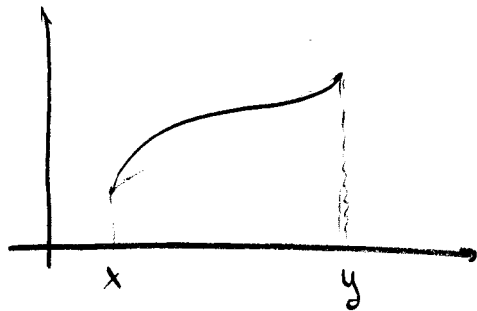
Proof: From mean value theorem,  $\exists \theta \in [a, b]$  such that

$$\forall x, y \in [a, b], \quad g'(\theta) = \frac{g(x) - g(y)}{x - y}$$

$$g(x) - g(y) = g'(\theta)(x - y)$$

$$|g(x) - g(y)| = |g'(\theta)| |x - y|$$

$$\leq \max_{\theta \in [a, b]} |g'(\theta)| |x - y|$$



Corollary: If  $g(x)$  is continuous and differentiable on  $[a, b]$  then  $g(x)$  is a contraction if  $\max_{x \in [a, b]} |g'(x)| \leq k$ ;  $k < 1$ .

Theorem: Contraction mapping

Suppose ①  $g(x)$  is continuous on  $[a, b]$

②  $g(x) \in [a, b]$  for  $x \in [a, b]$

③  $g(x)$  is a contraction on  $[a, b]$

Then ④  $g(x)$  has a unique fixed point  $x^*$ , i.e.  $x^* = g(x^*)$  on  $[a, b]$ .

2 The sequence  $x_{k+1} = g(x_k)$  converges to  $x^*$  for any starting value  $x_0 \in [a, b]$ .

Proof: Existence

Let  $u(x) = x - g(x)$ , then since  $a \leq g(x) \leq b$   $x \in [a, b]$

$$u(a) = a - g(a) \leq 0$$

$$u(b) = b - g(b) \geq 0$$

By the intermediate value theorem,  $\exists x^* \in [a, b]$  such that  $u(x^*) = 0$  or  $x^* = g(x^*)$

Uniqueness

Suppose  $\exists$  two fixed points  $x_1, x_2 \in [a, b]$  such that

$$x_1 = g(x_1) \quad \text{and} \quad x_2 = g(x_2)$$

Then since  $g(x)$  is a contraction,

$$|g(x_1) - g(x_2)| \leq L|x_1 - x_2|$$

$$\text{then, } |x_1 - x_2| \leq L|x_1 - x_2| \Rightarrow L \geq 1$$

Since  $g(x)$  is a contraction,  $L \in (0, 1)$ . #Contradiction.

Therefore, fixed point is unique.

□

## Convergence

Let  $x_0 \in [a, b]$  then  $x_{k+1} = g(x_k)$  where  $x_k \in [a, b]$  then  $x_{k+1} \in [a, b]$ . By contraction property for  $x_{k-1}, x^*$  we get,

$$|g(x_{k-1}) - g(x^*)| \leq L |x_{k-1} - x^*|$$

Since  $x_k = g(x_{k-1})$  and  $x^* = g(x^*)$

$$|x_k - x^*| \leq L |x_{k-1} - x^*| \rightarrow \text{order of convergence}$$

Apply above recursively and get,

$$|x_k - x^*| \leq L^k |x_0 - x^*|$$

$$\lim_{k \rightarrow \infty} |x_k - x^*| \leq |x_0 - x^*| \lim_{k \rightarrow \infty} L^k$$

But  $L \in (0, 1)$ , hence  $\lim_{k \rightarrow \infty} L^k = 0$ . Therefore  $\lim_{k \rightarrow \infty} |x_k - x^*| = 0$ .  $\square$

## Corollary of Contraction Mapping Theorem

Let  $x^* = g(x^*)$  be a fixed point of  $g(x)$ . Assume  $\exists \delta$  such that  $g'(x)$  is continuous in  $[x - \delta, x + \delta]$  then

① If  $|g'(x^*)| < 1$  then  $\exists \delta'$  such that sequence converges to  $x^*$  for any starting value  $x_0 \in [x^* - \delta', x^* + \delta']$ . Furthermore, convergence is linear with

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = |g'(x^*)|$$

② If  $|g'(x^*)| > 1$  then the sequence  $x_{k+1} = g(x_k)$  diverges for any starting value of  $x_0$ .

### Theorem: Convergence of Newton's Iterations

If  $f(x^*) = 0$ ,  $f'(x^*) \neq 0$  and  $f, f', f''$  are all continuous in interval  $[x^* - \delta, x^* + \delta]$ ,  $\delta > 0$  and  $x_0$  is sufficiently close to  $x^*$ , then the sequence  $x_0, x_1, \dots$  converges quadratically to  $x^*$  with

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} = \left| \frac{f''(x^*)}{2f'(x^*)} \right|$$

In other words,

$$|x_{k+1} - x^*| = C_k |x_k - x^*|^2$$

$$\text{with } \lim_{k \rightarrow \infty} C_k = \left| \frac{f''(x^*)}{2f'(x^*)} \right|$$

This convergence result can be used to answer

A1 Q4b.

Proof: Recall the Newton's iteration is defined as,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad f'(x_k) \neq 0$$

Note, if we define  $g(x) = x - \frac{f(x)}{f'(x)}$

Then if  $x^*$  is a root of  $f(x)$ , i.e.  $f(x^*) = 0$ . Then  $x^*$  is a fixed point of  $g(x)$  since  $\underline{g(x^*) = x^* - 0 = x^*}$ .

$$\text{Now } g'(x) = 1 - \frac{f'(x)}{f'(x)} - \frac{f(x) \cdot f''(x)}{(f'(x))^2} = \frac{f(x) - f''(x)}{(f'(x))^2}$$

then  $g'(x^*) = 0$  because  $f(x^*) = 0$ . Therefore, by corollary of contraction mapping the seq  $\{x_k\}$  is convergent.



Don 24, 2018

## Rate of Convergence - Newton's Method

By Taylor's Theorem (in Lagrange form)

$$f(x^*) = f(x_k) + f'(x_k)(x^* - x_k) + f''(\xi_k) \frac{(x^* - x_k)^2}{2}$$

with  $\xi_k$  between  $x_k$  and  $x^*$ . Since  $f(x^*) = 0$ , then

$$0 = f(x_k) + f'(x_k)(x^* - x_k) + f''(\xi)$$

$$0 = \frac{f(x_k)}{f'(x_k)} + x^* - x_k + \frac{f''(\xi)}{2f'(x_k)} (x^* - x_k)^2$$

$$= x_k - x_{k+1} + x^* - x_k + \frac{f''(\xi)}{2f'(x_k)} (x^* - x_k)^2$$

$$\text{Therefore } x_{k+1} - x^* = \frac{f''(\xi_k)}{2f'(x_k)} (x^* - x_k)^2$$

Take limit as  $k \rightarrow \infty$

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} = \left| \frac{f''(x^*)}{2f'(x^*)} \right|$$

Note\*\*: If  $f'(x^*) = 0$  rate of converge degrades to linear. i.e.  $x^2 = f(x)$

## Secant Method Convergence

Similar to Newton's Method but is sub-quadratic, i.e.

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^q} = \left| \frac{f''(x^*)}{2f'(x^*)} \right|^{q-1} \quad \text{where} \quad q = \frac{1}{2}(1 + \sqrt{5}) \approx 1.6$$

## Some practical considerations

- In practice it's often impossible to verify that all conditions needed to ensure convergence are satisfied.

- Note: Conditions derived are mostly sufficient conditions, convergence may still occur if conditions are violated.

## Summary

### Contraction on $[a, b]$

1) By definition  $|g(x) - g(y)| \leq L|x - y|$ ,  $L \in (0, 1) \forall x, y \in [a, b]$

2) Derivate  $|g'(x)| < 1$ ,  $x \in [a, b]$

### Fixed Point Convergence

1)  $x \in [a, b]$ ,  $g(x) \in [a, b]$   $g(x)$  is a contraction.

- By defn: on  $[a, b]$

-  $|g'(x)| < 1$  on  $[a, b]$

Then  $x^* = g(x^*)$  fixed point exists, unique and convergent  $x_0 \in [a, b]$ .

$$2) x^* = g(x^*)$$

$$a) |g'(x^*)| < 1 \Rightarrow \text{convergent } x_0 \in [x^* - \delta, x^* + \delta]$$

$$b) |g'(x^*)| > 1 \Rightarrow \text{diverges } \forall x_0$$

3) If we have a convergent fixed point,

$$\text{Linear convergence: } \lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = |g'(x^*)|$$

### Newton's Convergence

1) If  $f'(x^*) \neq 0 \Rightarrow$  quadratic convergence

$$\lim_{k \rightarrow \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} = \left| \frac{f''(x^*)}{2f'(x^*)} \right|$$

2) If  $f'(x^*) = 0 \Rightarrow$  linear convergence

### Google Page Rank

Search engines perform two tasks:

1) Finding pages containing key words.

2) Ranking pages in order of importance

Search engines differ mostly in 2).

# Google Ideas

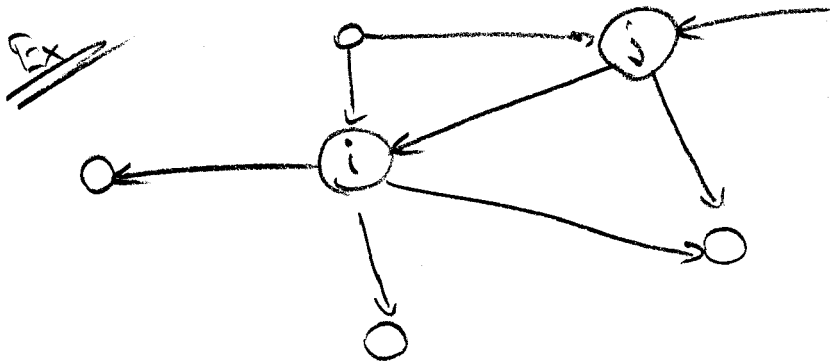
- Model the structure of the web as a directed graph.

- Nodes - web pages

- arcs - A directed arc is drawn from page  $j$  to page  $i$  if  $\exists$  a link from  $j \rightarrow i$

Let  $\text{deg}(j)$  be the out-degree of node  $j$

i.e. The number of arcs leaving  $j$ .



$$\text{deg}(i) = 3$$

$$\text{deg}(j) = 2$$

## - Basic idea

- A link from  $j \rightarrow i$  can be viewed as a vote of importance of  $i$  as seen by  $j$ .

- The "importance" conferred on  $i$  by  $j$  can be defined by

$$\frac{1}{\text{deg}(j)} \text{ if } \exists \text{ a link from } j \rightarrow i. \text{ Otherwise } 0.$$

- This is a local measure of importance.
- For a "global" importance, we need to know the importance of  $y$ .
- Examine the nodes which point to  $j$  and so on.

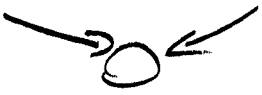
One way to do this:

- Suppose a random web surfer picks a random page to start.
- From the initial page, the surfer then selects an outlink at random  $\rightarrow$  visits the selected page.
- Another outlink is selected from new page at random, and so on.
- Surfer keeps track of the number of times any given page is visited.

$$\Rightarrow \text{Importance} \equiv \frac{\# \text{ of visits to page } i}{\text{total } \# \text{ of pages visited}}$$

- We can view this "importance measure" as the probability that the surfer ends up at page  $i$  after making a large number of hops.

Two major problems a surfer may encounter

① Dead end pages 

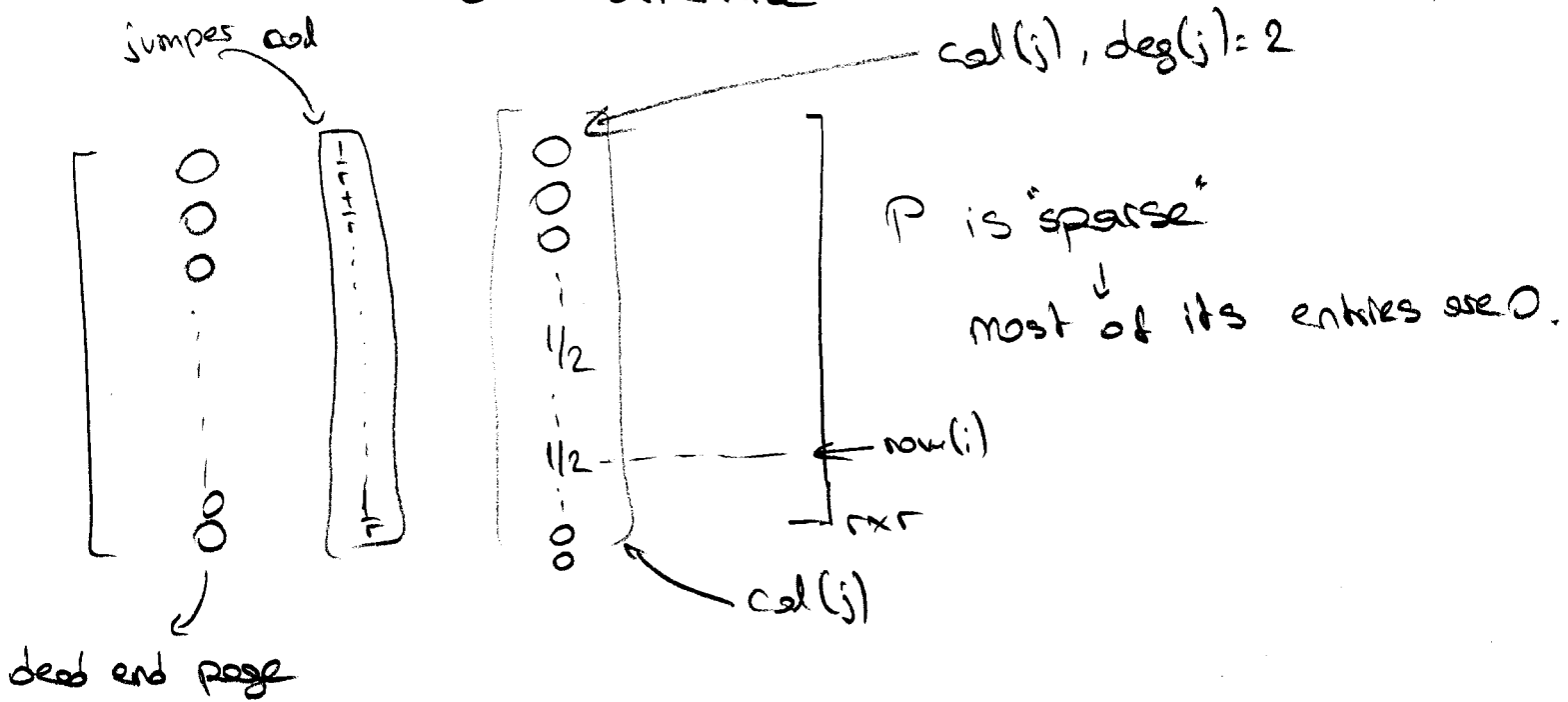
② Cycling pages 

# Matrix Representation

- Suppose say  $R$  pages exist in web.  $P$  is a matrix of size  $R \times R$  where  $R \sim O(10^7, 10^{10})$
- Let  $P_{ij} \equiv$  probability that the random surfer visits node  $i$ , given that she is at node  $j$ .

Thus

$$P_{ij} = \begin{cases} \frac{1}{\text{deg}(j)} & \text{if } \exists \text{ link from } j \rightarrow i \\ 0 & \text{otherwise} \end{cases}$$



Dead end pages: To avoid this problem, we assume the surfer jumps to another page at random. Surfer moves the surfer to any other page with equal probability.

More precisely, let  $\vec{d} \in \mathbb{R}^R$  be a column vector

$$[\vec{d}]_j = \begin{cases} 1 & \text{if } \deg(j) = 0 \\ 0 & \text{otherwise} \end{cases}$$

Let  $\vec{e} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}_{R \times 1}$   $R$  dimensional column vector of ones  
 [e.g. ones(10,1)]

$$\frac{1}{R} \vec{e} \vec{d}^T = \begin{bmatrix} 0 & \frac{1}{R} & 0 & \frac{1}{R} & 0 \\ 0 & \frac{1}{R} & 0 & \frac{1}{R} & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \frac{1}{R} & 0 & \frac{1}{R} & 0 \end{bmatrix}$$

$\begin{matrix} \text{7th} & & & & \\ \text{10th} & & & & \end{matrix}$

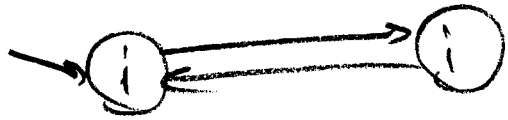
Then we can define a new matrix of transition probability

$$P' = P + \frac{1}{R} \vec{e} \vec{d}^T$$

i.e. we replace any column  $j$  in  $P$  which is identically zero by  $[\frac{1}{R}, \frac{1}{R}, \dots, \frac{1}{R}]^T$  which means the surfer jumps to any page with equal probability

## Cycling pages

Suppose we have a cycle



- We again use "teleportation" <sup>add</sup> to allow the surfer to escape from this cycle.

Let  $\alpha \in (0,1)$  i.e. google

Define a new set of probabilities

$$M = \alpha P' + (1-\alpha) \frac{1}{R} \vec{e} \vec{e}^T$$

Note,  $\vec{e} \vec{e}^T$ :

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix}$$

just a matrix  
of ones

What does  $\textcircled{1}$  do?

- As well as the usual probability of visiting other pages ( $P'$ ), the surfer can teleport to any other page with equal probability.



Jan 29, 2018

## Google Matrix

1)  $P \in \mathbb{R}^{R \times R}$ ;  $R \approx \#$  of web pages

$$\text{where } P_{ij} = \begin{cases} 1/\text{deg}(j) & \text{deg}(j) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

2)  $P' = P + \underbrace{\frac{1}{R} \vec{e} \cdot \vec{e}^T}$  to handle dead end pages by teleportation.

Fills in columns of 0 with  $1/R$

3)  $M = \alpha P' + (1-\alpha) \frac{1}{R} \vec{e} \vec{e}^T$  global teleportation to avoid cycles.

So  $M_{ij}$  is now the probability of visiting  $i$ , given we're at  $j$ , including "teleportation" effects.

Note: Defn of  $M$  we have

$$0 \leq M_{ij} \leq 1$$

$$\sum_i M_{ij} = 1, \text{ i.e. } \text{column}(M) = 1$$

- total probability of visiting some page is 1.

# Markov Transition Matrices

Def<sup>n</sup>: A matrix  $Q$  is a Markov matrix if

$$0 \leq Q_{ij} \leq 1$$

$$\sum_i Q_{ij} = 1 \quad \forall_j \quad (\text{column}(Q) = 1 \text{ for every column})$$

$\Rightarrow$  Google matrix  $M$  is a Markov Matrix.

Def<sup>n</sup>: A vector  $\vec{q}$  is a probability vector if

$$0 \leq [\vec{q}]_i \leq 1 \quad \text{and} \quad \sum_i [\vec{q}]_i = 1$$

Define the initial probability vector of visiting a page as

$$\vec{p}^0 = \begin{bmatrix} 1/R \\ 1/R \\ \vdots \\ 1/R \end{bmatrix} = \frac{1}{R} \vec{e}^n$$

i.e. initial page is selected with equal probability

Then the probability vector after  $l$  hop of the random surfer is

$$\vec{p}^1 = M \vec{p}^0 \quad ; \quad \vec{p}^1 \text{ is also a probability vector.}$$

So given probability vector at hop  $n$  is  $\vec{p}^n$ , the probability vector at hop  $(n+1)$  is

$$\vec{p}^{n+1} = M \vec{p}^n \quad ; \quad \vec{p}^{n+1} \text{ is a probability vector.}$$

# Page Rank Algorithm

Given the assumptions above, we can now state the precise page rank algorithm

The "global importance" or rank score of page  $i$  is given by  $[p^{\infty}]_i$ , where

$$p^{\infty} = \lim_{k \rightarrow \infty} (M)^k p^0$$

the  $k^{\text{th}}$  power of matrix  $M$ .

Each multiplication by  $M$  represents one hop of the random surfer.

## Intermission: Matlab sparse matrices

Sparse matrix only stores non-zero elements in the matrix

- sparse
- speye
- spdiags
- spalloc

} creating sparse matrices in matlab

Matlab avoids unnecessary computations (including 0) when working with sparse matrices

(e.g. matrix multiplication, addition)

This extends to many built-in algorithms in matlab.

e.g. solving a linear system  $A\vec{x} = \vec{b}$   
↳ sparse

Rule of thumb: Operations on sparse matrices return sparse matrices.

Recall: Google Matrix

- Recall:  $P$  is sparse.

↳ Is  $M$  sparse?

↳ Recall  $M = \alpha P' + (1 - \alpha) \frac{1}{R} \vec{e} \vec{e}^T$

↳ sparse

↳ Dense matrix

⇒ result is dense

• Because  $M$  is dense, we would like to avoid storing  $M$ , only  $P$ , which is sparse.

e.g.  $R = 100,000 \Rightarrow M$  is  $100k \times 100k$

$$\text{memory} = 10^5 \times 10^5 \times 8 \text{ bytes} \times \frac{1 \text{ GB}}{2^{30} \text{ Bytes}} \approx 754 \text{ B}$$

•  $M\vec{p}^n$  requires  $O(R^2)$  multiplications but  $P$  is quite sparse

$P\vec{p}^n$  requires  $O(R)$  multiplications.

$$\begin{aligned}
 \text{Recall: } M &= \alpha P' + (1-\alpha) \frac{1}{R} \vec{e} \vec{e}^T \\
 &= \alpha \left( P + \frac{1}{R} \vec{e} \vec{e}^T \right) + (1-\alpha) \frac{1}{R} \vec{e} \vec{e}^T \\
 &= \alpha \left( P + \frac{\alpha}{R} \vec{e} \vec{e}^T \right) + (1-\alpha) \frac{1}{R} \vec{e} \vec{e}^T
 \end{aligned}$$

If  $\vec{p}^n$  is a probability vector

$$\begin{aligned}
 M \vec{p}^n &= \left[ \alpha P + \frac{\alpha}{R} \vec{e} \vec{e}^T + (1-\alpha) \frac{1}{R} \vec{e} \vec{e}^T \right] \vec{p}^n \\
 &= \alpha P \vec{p}^n + \left( \frac{\alpha}{R} \vec{e} \vec{e}^T \right) \vec{p}^n + (1-\alpha) \left( \frac{1}{R} \vec{e} \vec{e}^T \right) \vec{p}^n
 \end{aligned}$$

$$\text{Now } \frac{\vec{e} \vec{e}^T \vec{p}^n}{R} = \frac{\vec{e}}{R} \underbrace{(\vec{e}^T \vec{p}^n)}_1 = \frac{\vec{e}}{R}$$

$$\text{and } \underbrace{(\vec{e} \cdot \vec{e}^T)}_{(R \times R) \times (R \times 1)} \vec{p}^n = \vec{e} \underbrace{(\vec{e}^T \cdot \vec{p}^n)}_{(1 \times R) \times (R \times 1)}$$

Scalar

$$\text{So } M \vec{p}^n = \alpha P \vec{p}^n + \frac{\alpha}{R} \vec{e} \underbrace{(\vec{e}^T \cdot \vec{p}^n)}_{\text{Dot product}} + \frac{(1-\alpha)}{R} \vec{e}$$

↙  
Sparse matrix vector  
multiplication

$\sim O(R)$

Dot product  
 $\sim O(R)$

## Convergence Analysis

Does  $\vec{p}^{\infty} = \lim_{k \rightarrow \infty} M^k \vec{p}^0$  converge? If so, how fast?

• Every Markov matrix has one eigenvalue.

↳ The eigenvalues of  $Q$  and  $Q^T$  are the same. Then

$$\text{rowsum}(Q^T) = 1 \Rightarrow Q^T \vec{e} = \vec{e}$$

$$\Rightarrow \vec{e} \text{ is an eigenvector of } Q^T \text{ with } \lambda = 1$$

$$\Rightarrow \lambda = 1 \text{ is an eigenvalue of } Q.$$

• Every (possibly complex) eigenvalue,  $\lambda$ , of Markov matrix  $Q$  satisfies

$$|\lambda| \leq 1$$

---

Defn: A Markov matrix  $Q$  is a positive Markov matrix if

$$Q_{ij} > 0 \quad \forall i, j$$

---

• If  $Q$  is a positive Markov matrix then  $\exists!$  eigenvector of  $Q$  with eigenvalue  $|\lambda| = 1$ .

### Convergence Proof

• If  $M$  is a positive Markov matrix, then the iteration

$\lim_{k \rightarrow \infty} M^k \vec{p}^0$  converges to a unique vector  $\vec{p}^{\infty}$  for any initial

probability vector  $\vec{p}^0$ .

Proof: Let  $\vec{x}_1$  be an eigenvector of  $M$  with eigenvalue  $\lambda_1$ . We assume  $M$  has  $R$  linearly independent eigenvectors. Then we can find constants,  $c_l$  such that

$$\vec{p}^0 = \sum_l c_l \vec{x}_l$$

Suppose that we order the eigenvectors so that

$$\underbrace{|\lambda_1|}_{=1} > |\lambda_2| \geq |\lambda_3| \geq \dots$$

So that  $\vec{x}_1$  corresponds to unique eigenvectors with  $|\lambda_1|=1$ .

$$\begin{aligned} \text{Then } (M)^k \vec{p}^0 &= M^k \sum_l c_l \vec{x}_l = \sum_l c_l M^k \vec{x}_l = \sum_l c_l (\lambda_l)^k \vec{x}_l \\ &= c_1 \vec{x}_1 + \sum_{l=2}^R c_l (\lambda_l)^k \vec{x}_l \end{aligned}$$

For all  $l \geq 2$  we have  $|\lambda_l| < 1$  then  $\lim_{k \rightarrow \infty} M^k \vec{p}^0 = c_1 \vec{x}_1$

### Uniqueness

Since  $\vec{p}^0$  is a probability vector,  $\vec{p}^\infty$  is also a probability vector. If we start iteration with another probability vector

$$\vec{q}^0 = \sum_l b_l \vec{x}_l$$

$$\text{then } \vec{q}^\infty = \lim_{k \rightarrow \infty} M^k \vec{q}^0 = b_1 \vec{x}_1$$

Since  $\vec{p}^\infty, \vec{q}^\infty$  are probability vectors,  $b_1 = c_1$ .

## Speed of Convergence

Remark: The speed of convergence will be determined by the size of the largest eigenvalue that is less than one,  $\lambda_2$ , of  $M$ . It can be shown that  $|\lambda_2| \sim \alpha$ .

Question 1: Why not choose  $\alpha$  very small?

$$\alpha = 0 \Rightarrow M = \frac{1}{R} \vec{e} \vec{e}^T = \begin{bmatrix} 1 & \dots & 1 \\ 1 & & 1 \\ \vdots & & \vdots \\ 1 & & 1 \end{bmatrix} \Rightarrow \text{no information}$$

$$\text{then } \vec{p}^{\infty} = \frac{1}{R} \vec{e}$$

Generally smaller  $\alpha$  gives faster convergence but less significant ranking information.

$$\alpha = 1 \Rightarrow M = P' \Rightarrow \text{Does not avoid cycles}$$

$$\Rightarrow M \text{ is not a positive Markov Matrix}$$

$$\Rightarrow \text{may not converge}$$



## Summary of Page Rank Algorithm 9.7: 49

```

Input:  $\mathbf{p}^0 = \frac{1}{R}\mathbf{e}$ 

for  $n = 0, 1, 2, \dots, \text{Max\_ITER}$  do
     $\mathbf{p}^{n+1} = M\mathbf{p}^n$ 
    if  $\max : |[\mathbf{p}^{n+1}] : -[\mathbf{p}^n] :| < \text{tol}$  then
        | quit
    end
end

```

**Gaussian Elimination 9.8:** Consider a square non-singular matrix  $A \in \mathbb{R}^{N \times N}$  with unknown solution vector  $x \in \mathbb{R}^N$ . Right hand side vector is  $\mathbf{b} \in \mathbb{R}^N$ ,  $A\mathbf{x} = \mathbf{b}$ . This represents a system of linear equations.

**Example 9.9:** ( $3 \times 3$  Example of Gaussian elimination)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \mathbf{b}$$

and proceed in a methodical way to solve the system of equations.

Step 1: Eliminate  $a_{21}$  and  $a_{31}$ .

$$\begin{array}{l} \text{Row 2} - \frac{a_{21}}{a_{11}} \text{Row 1} \\ \text{Row 3} - \frac{a_{31}}{a_{11}} \text{Row 1} \end{array} \sim \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \end{bmatrix}.$$

Note  $a_{ij}^{(k)}$  indicates that the entry has been modified by  $(k-1)$  steps of elimination  $a_{ij}^{(1)} = a_{ij}$  = original.

Step 2: Eliminate  $a_{32}^{(2)}$ .

$$\text{Row 3} - \frac{a_{32}^{(2)}}{a_{22}^{(2)}} \text{Row 2} \sim \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \end{bmatrix}.$$

We do not need to fully row reduce because we can do a *back solve* to get  $\mathbf{x}$ , as in

$$\begin{aligned} x_3 &= \frac{b_3^{(3)}}{a_{33}^{(3)}} \implies x_2 = \frac{(b_2^{(2)} - a_{23}^{(2)} x_3)}{a_{22}^{(2)}} \\ x_1 &= \frac{(b_1 - a_{12}^{(1)} x_2 - a_{13}^{(1)} x_3)}{a_{11}^{(1)}}. \end{aligned}$$

**Notation 9.10:**  $A^{(k)}$  matrix is the matrix which is produced by  $(k-1)$  steps of elimination. Columns 1 to  $k-1$  have all elements below the diagonal eliminated. We have,

$$A^{(1)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad \text{and} \quad A^{(2)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix}.$$

Where

$$A^{(1)}M_1 = A^{(2)} \quad \text{with} \quad M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{a_{21}}{a_{11}} & 1 & 0 \\ -\frac{a_{31}}{a_{11}} & 0 & 1 \end{bmatrix}.$$

So we have

$$M_2A^{(2)} = A^{(3)} \implies \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{a_{32}^{(2)}}{a_{22}^{(2)}} & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{bmatrix}$$

Note that  $M$ 's are just elementary row matrices from elementary row operators.

**Remark 9.11:** We can generalize this to an  $N \times N$  matrix as follows:

Consider the  $k^{\text{th}}$  step of elimination. Eliminate  $a_{ik}^{(k)}$  using a multiplier  $\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$ . After  $(k-1)$  steps we obtain

$$\begin{bmatrix} \diagdown & \vdots & & \vdots & \diagup \\ \vdots & a_{kk}^{(k)} & \dots & a_{kj}^{(k)} & \vdots \\ 0 & \vdots & & \vdots & \\ \vdots & a_{ik}^{(k)} & \dots & a_{ij}^{(k)} & \vdots \\ 0 & \vdots & & \vdots & \end{bmatrix}.$$

Modify  $a_{ij}^{(k)}$  to get  $a_{ij}^{(k+1)} = a_{ij}^{(k)} - a_{kj}^{(k)} \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$ . To eliminate the entry  $a_{ik}$ , we form a multiplier  $\ell_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)}$ . In matrix form, we start with  $A^{(1)} = A, b^{(1)} = b$  and  $A^{(k)} = M_{(k-1)}A^{(k-1)}$  so we have  $A^{(N)} = (M_{N-1}M_{N-2}\dots M_1)A^{(1)}$  or  $(M_1^{-1}M_2^{-1}\dots M_{N-1}^{-1})A^{(N)} = A^{(1)} = A$ . Where

$$(M_1^{-1}\dots M_{N-1}^{-1}) = \begin{bmatrix} 1 & \dots & \dots & \dots & 0 \\ \ell_{21} & 1 & & & \vdots \\ \ell_{31} & \ell_{32} & 1 & & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \ell_{N1} & \ell_{N2} & \dots & \ell_{N,N-1} & 1 \end{bmatrix}$$

Define  $L = (M_1^{-1}\dots M_{N-1}^{-1})$  where  $L$  is simply the matrix of multipliers.  $L$  is *unit lower triangular*, meaning all diagonal entries are 1 and lower triangular. Now

$$A^{(N)} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ 0 & a_{22}^{(2)} & \dots & a_{2N}^{(2)} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & a_{NN}^{(N)} \end{bmatrix} \quad \begin{array}{l} \text{upper triangular matrix} \\ \text{so all entries below} \\ \text{diagonal are 0.} \end{array}$$

**Example 9.12:** Let  $U \equiv A^{(N)}, (M_1^{-1}\dots M_{N-1}^{-1})A^{(N)} = LU = A$ . We have factored  $A$  into an upper and lower triangular matrix. Suppose we want to solve  $Ax = b$ . Having factored  $A = LU$ , we can find  $x$  by  $LUx = b$ . Let  $z = Ux$ . Then  $Lz = b$  and  $Ux = z$ . Since  $LU$  is a triangular matrix, we can efficiently solve for  $z$  and  $x$  by using forward or backward substitution.

**Remark 9.13:** Once we have factored  $A$  into  $L$  and  $U$  then we can solve for multiple solutions (different  $\mathbf{b}$ 's) by simply doing forward and backward substitution without having to refactor  $A$ . Note that no additional work required to compute  $L$  and  $U$  other than Gaussian elimination.

**Remark 9.14:** If we have  $A\mathbf{x} = \mathbf{b}$ ,  $A\mathbf{y} = \mathbf{c}$ ,  $A\mathbf{z} = \mathbf{d}$ . We factor  $A = LU$  and solve for  $LU\mathbf{x} = \mathbf{b}$ ,  $LU\mathbf{y} = \mathbf{c}$ ,  $LU\mathbf{z} = \mathbf{d}$ . This is efficient due to low computational requirement of forward/backward substitution.

We could also find  $A^{-1}$ , then find  $A^{-1}\mathbf{b}$ ,  $A^{-1}\mathbf{c}$ ,  $A^{-1}\mathbf{d}$ . An efficient way to compute  $A^{-1}$  is done by ***LU factorization***.

Given non-singular  $A$ , we have  $AA^{-1} = I$ . So we want to find  $A^{-1}$  using  $LU$  factorization. This is equivalent to solving  $N$  systems where

$$A\mathbf{x}_1 = \mathbf{e}_1, A\mathbf{x}_2 = \mathbf{e}_2, \dots, A\mathbf{x}_N = \mathbf{e}_N \implies A^{-1} = [\mathbf{x}_1 \dots \mathbf{x}_N]$$

First factor  $A$  into  $L$  and  $U$ . Then do  $N$  forward and backward solves. We find  $\mathbf{x}_1, \dots, \mathbf{x}_N$  as columns as  $A^{-1}$ .

**Remark 9.15:** Often we are given  $A$  and we are interested in computing  $A^{-1}\mathbf{b}$ . We know that it's the same as solving  $A\mathbf{x} = \mathbf{b}$ . So instead of computing  $A^{-1}$  first, we can use forward/backward substitution.

# Recap

~~Jan~~ 5, 2018  
Feb

Given non-singular matrix  $A \in \mathbb{R}^{n \times n}$  we can factor  $A = LU$  using Gaussian elimination, where  $L$  is lower triangular matrix,  $U$  is an upper triangle matrix.

Hence we can solve  $A\vec{x} = \vec{b}$  by

① Factor  $A = LU$

$$\text{So } \underbrace{LU}_{\vec{y}} \vec{x} = \vec{b}$$

$$\text{Let } \vec{y} = U\vec{x}$$

② Forward solve  $L\vec{y} = \vec{b}$

$$\begin{bmatrix} \diagup & & & \\ & \circ & & \\ & & \ddots & \\ & & & \diagdown \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

③ Back solve  $U\vec{x} = \vec{y}$

## Work Estimates

How do we count work (i.e. computational complexity)?

- total number of floating point operations (flops)

i.e. # of multiples + # of adds + # of divides + # of subtracts

Note: Usually multiplication is paired with an addition/subtraction

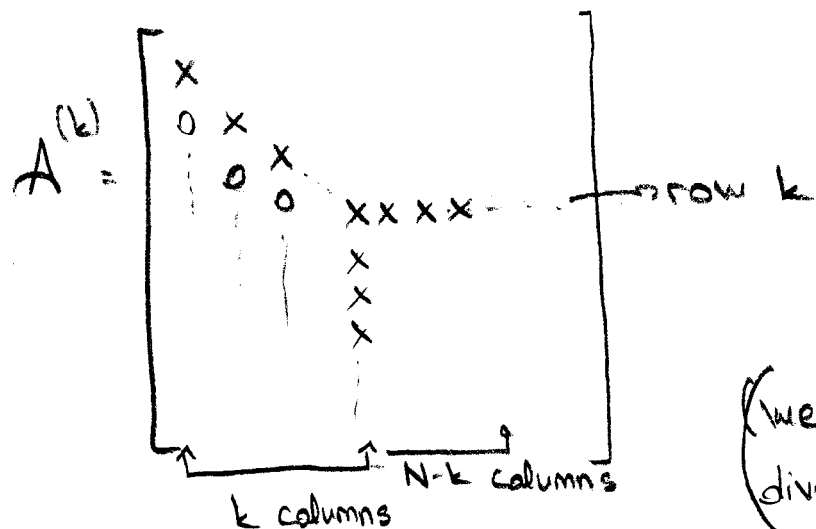
$$z_2 = b_2 - l_{2,1} z_1$$

we can also count multiply-adds

No standard terminology, both approaches are valid flop counts. We adopt the convention of counting total number of flops. i.e multiply-add 2 flops.

## LU factoring - flop count

Assume  $(k-1)$  steps completed.



At this stage we need

$$2(N-k)^2$$

flops to eliminate  $a_{ik}^{(k)}$ ,  $i = k+1, \dots, N$

(we are ignoring the  $(N-k)$  divides to form multipliers)

$$\text{Work} \approx \sum_{k=1}^{N-1} 2(N-k)^2 = 2[1^2 + 2^2 + \dots + (N-1)^2] = \frac{2}{3}N^3 + O(N^2)$$

Recall:

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

Similar analysis of each forward/backward solves

$$\text{Forward solve work} = 2[1 + 2 + \dots + (N-1)]$$

$$= \frac{2N^2}{2} + O(N)$$

$$\text{Back solve work} = N^2 + O(N)$$

$$\text{Total} = 2N^2 + O(N)$$

## Computing $A^{-1}$

Given  $A$ , find  $A^{-1}$  such that  $AA^{-1} = I$ .

Method 1: Use Gauss-Jordan Elimination of augmented matrix

$$\text{i.e. } (A|I) = \left[ \begin{array}{ccc|ccc} a_{11} & \dots & a_{1n} & 1 & \dots & 0 \\ a_{21} & & & 0 & 1 & \\ \vdots & & & \vdots & & \vdots \\ a_{n1} & & a_{nn} & 0 & \dots & 0 \end{array} \right] \rightsquigarrow \left[ \begin{array}{ccc|ccc} 1 & \dots & 0 & b_{11} & b_{12} & \dots & b_{1n} \\ 0 & 1 & & & & & \\ 0 & 0 & & & & & \\ \vdots & & & & & & \\ 0 & 0 & \dots & 0 & 1 & & b_{n1} & \dots & b_{nn} \end{array} \right]$$

How many flops? - reduced form  $\Rightarrow \frac{8}{3}N^3 + O(N^2)$  flops  
- update Augmented matrix

Method 2: Using LU factorization

① Factor  $A = LU$

$\frac{2}{3}N^3 + O(N^2)$  flops

② Forward/Backward solve

$A\vec{x}_1 = \vec{b}_1 \leftarrow$  first column of  $I$

$[2N^2 + O(N)]$  flops  
 $\times N$  times

$$LU\vec{x}_1 = \vec{b}_1$$

$$\begin{array}{c} \vec{x}_2 \\ \vdots \\ \vec{x}_n \end{array}$$

$N$  times

$$\frac{2}{3}N^3 + \frac{2N^3}{1} + O(N^2) = \frac{8N^3}{3} + O(N^2)$$

• Often we need not to compute  $A^{-1}$  directly.

# Example

$$A^{-1}b$$

Method 1:

- ① compute  $A^{-1}$   $\frac{8}{3}N^3 + O(N^2)$
  - ② compute  $A^{-1}b$   $O(N^2)$
- $\frac{8}{3}N^3 + O(N^2)$

Method 2:

- ① Factor  $A = LU$   $\frac{2}{3}N^3 + O(N^2)$
  - ② Forward/Backward solve  $2N^2 + O(N)$
- $\frac{2}{3}N^3 + O(N^2)$

- Method 2 is more efficient

- Also reduces roundoff error accumulation

- Matlab backslash operator (i.e.  $A \setminus b$ ) does this.

i.e.  $A^{-1}b$ ,  $A^{-1}e$ ,  $A^{-1}0$

Inverse approach:  $\frac{8}{3}N^3 + O(N^2)$  vs LU approach:  $\frac{2}{3}N^3 + O(N^2)$

## Ex

Assume  $A = LU$  has already been factored.

a) Solving  $A^T x = b$

$$(LU)^T x = b \Rightarrow U^T L^T x = b$$

$\swarrow$  lower triangular       $\swarrow$  upper triangular

① Let  $y = L^T x$

then  $U^T y = b$

Forward solve:  $n^2 + O(n)$

Back solve:  $L^T x = y$ :  $n^2 + O(n)$

Total  $\Rightarrow 2n^2 + O(n)$

$$b) A^2 \vec{x} = \vec{b}$$

$$(LU)(LU)\vec{x} = \vec{b}$$

$$L(U(L(U\vec{x}))) = \vec{b}$$

$\underbrace{\quad\quad\quad}_{\vec{w}}$   
 $\underbrace{\quad\quad\quad}_{\vec{y}}$   
 $\underbrace{\quad\quad\quad}_{\vec{z}}$

Let  $\vec{w} = U\vec{x}$

$$\vec{y} = L\vec{w} = LU\vec{x}$$

$$\vec{z} = U\vec{y} = ULU\vec{x}$$

- 1) Forward solve  $L\vec{z} = \vec{b}$   $n^2 + O(n)$
- 2) Back solve  $U\vec{y} = \vec{z}$   $n^2 + O(n)$
- 3) Forward solve  $L\vec{w} = \vec{y}$   $n^2 + O(n)$
- 4) Back solve  $U\vec{x} = \vec{w}$   $n^2 + O(n)$

$$\text{Total} = 4n^2 + O(n)$$

## Stability of Factorization ↑ Midterm

Problems can arise when using matrix factorization.

— suppose that  $a_{kk}^{(k)} = 0$

multiplier  $\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \Rightarrow$  Division by zero

Or if  $a_{kk}^{(k)}$  is very small, then the multiplier will be large.





Feb 7, 2018

## Row Pivoting

$$A = \begin{bmatrix} a_{11}^{(1)} & & a_{1N}^{(1)} \\ 0 & a_{22}^{(2)} & a_{2N}^{(2)} \\ \vdots & \vdots & \vdots \\ a_{k1}^{(k)} & \dots & a_{kN}^{(k)} \\ a_{k+11}^{(k+1)} & \dots & a_{k+1N}^{(k+1)} \\ \vdots & \vdots & \vdots \\ a_{n1}^{(n)} & \dots & a_{nN}^{(n)} \end{bmatrix} \quad l_{kj} = \frac{a_{kj}^{(k)}}{a_{kk}^{(k)}}$$

Swapping two rows of  $A^{(k)}$  can be viewed as multiplying  $A^{(k)}$  by a permutation matrix.

Defn: Permutation matrix is a matrix obtained by interchanging row of  $I$ .

Ex  
Consider  $3 \times 3$  matrix  $\rightarrow$  swap 2<sup>nd</sup> and 3<sup>rd</sup> rows.

$$P \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \quad \text{Then, } P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Note:  $P$  is equivalent to swapping rows  $i$  and  $j$  of  $A$  is the matrix obtained by swapping rows  $i$  and  $j$  of  $I$ .

- the product of several permutation matrices is also a permutation.

- Consequently, the process of reordering rows of  $A$  can be written as  $A \rightarrow PA$

- So, factoring and solving  $A$  with row pivoting can be described as

$$Ax = b$$

$$PAx = Pb \quad \leftarrow \text{reorder}$$

$$\text{Factor } PA = LU$$

$$LUX = Pb \quad \leftarrow \text{forward/backward solve}$$

Note: We actually do the pivoting during factorization, but this is the same as factoring  $PA$  mathematically.

Note:  $PA \Rightarrow$  consumes no additional flops.

$$[L, U, P] = \text{lu}(A) \quad \rightsquigarrow \text{matlab code}$$

Note: No pivoting is required if the original matrix is row or column dominant.

Defn: A matrix is row diagonally dominant if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^N |a_{ij}|, \quad i=1, \dots, N$$

Defn: A matrix is column diagonally dominant if

$$|a_{jj}| > \sum_{\substack{i=1 \\ i \neq j}}^N |a_{ij}|, \quad j=1, \dots, N$$

If a matrix is row and column diagonally dominant then it can be shown Gaussian Elimination without pivoting is stable.

### Condition Numbers / Norms

The norm of a vector is a measure of its "size" or "length."

$\vec{x} = [x_1, x_2, \dots, x_N]^T$ . Absolute norm:  $\|\vec{x}\|_1 = \sum_{i=1}^N |x_i|$

Euclidean norm:  $\|\vec{x}\|_2 = \sqrt{\sum_{i=1}^N x_i^2}$

Infinity norm:  $\|\vec{x}\|_\infty = \max_i |x_i|$

Usually these norms are referred to as  $-p$  norms.

$$\|\vec{x}\|_p, \quad p=1, 2, \infty$$

Note:  $\|\cdot\|_p = \|\cdot\|$  means valid for any  $p$ .

## Properties of Norms

$$\cdot \|\vec{x}\| = 0 \Rightarrow x_i = 0 \quad \forall i$$

$$\cdot \|\alpha \vec{x}\| = |\alpha| \|\vec{x}\|, \quad \alpha \in \mathbb{R}$$

$$\cdot \|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$$

## Matrix Norms

For an  $N \times N$  matrix  $A$  we define the norm of  $A$  as

$$\|A\| = \max_{\|\vec{x}\| \neq 0} \frac{\|A\vec{x}\|}{\|\vec{x}\|} \quad \text{for any } p\text{-norm.}$$

It can be shown that 
$$\|A\|_1 = \max_j \sum_{i=1}^N |a_{ij}|$$

= max absolute column sum.

$$\|A\|_\infty = \max_j \sum_{i=1}^N |a_{ij}|$$

= max absolute row sum.

For the  $\|\cdot\|_2$  norm, we have to consider we have to consider the eigenvalues of  $A^T A$ .

Recall that matrix  $D$  has an eigenvalue  $\lambda$  (scalar) associated with non-zero eigenvector  $\vec{y}$ , if  $D\vec{y} = \lambda\vec{y}$

If  $\lambda_i, i=1, \dots, N$  are the eigenvalues of  $A^T A$ , then

$$\|A\|_2 = \max_i \sqrt{\lambda_i}$$

Note:  $\lambda_i \geq 0$  since  $A^T A$  is positive semi-definite.

### Properties of Matrix Norms

$A \equiv N \times N$  matrix

$B \equiv N \times N$  matrix

$x \equiv N \times 1$  matrix

$\alpha \equiv \text{scalar}$

$$\|A\| = 0 \text{ iff } a_{ij} = 0 \quad \forall i, j$$

$$\|\alpha A\| = |\alpha| \|A\| \quad \text{and} \quad \|A+B\| \leq \|A\| + \|B\|$$

$$\|A x\| \leq \|A\| \|x\| \quad \text{and} \quad \|AB\| \leq \|A\| \|B\| \quad \text{and} \quad \|I\| = 1$$

### Stability and Conditioning

In practice, neither data nor computer arithmetic is exact.

- What is the effect of this on the solution to  $A \vec{x} = \vec{b}$ ?

• Two questions: 1) Is the computed solution  $\vec{x}^c$  the exact solution to a "nearby" problem?

2) If small changes are made in the given problem, and the problem is solved using exact arithmetic, are the changes in the solution to this perturbed problem small compared to the solution of the original problem?

Question 1 is asking if "can we control the growth of round-off error in finite precision arithmetic?"

↳ If yes, then the algorithm is stable

Question 2 is asking if small changes in the data implies small changes in the exact solution, which implies problem is well-conditioned.

Conversely, it is asking if small changes in the data implies large changes in the exact solution, which implies problem is ill-conditioned.

Note: Conditioning is a property of the problem, not the algorithm.

In practice, Gaussian Elimination with (complete) pivoting can be considered to be numerically stable

### Conditioning

Suppose we perturb the right hand side of

$$Ax = b$$

What happens to  $x$ ?

If  $b \rightarrow b + \Delta b$ , then  $x \rightarrow x + \Delta x$ . Then how big is  $\Delta x$ ?

$$A(x + \Delta x) = b + \Delta b \quad (1)$$

Since  $Ax = b$ , (1) becomes  $\Delta x = A^{-1} \Delta b$

$$\text{or } \|\Delta x\| \leq \|A^{-1}\| \|\Delta b\| \quad (2)$$

From  $Ax = b$  we have  $\|b\| \leq \|A\| \|x\|$

$$\Rightarrow \frac{1}{\|x\|} \leq \frac{\|A\|}{\|b\|} \quad (3)$$

(2)  $\times$  (3) gives

$$\frac{\|\Delta x\|}{\|x\|} \leq \|A^{-1}\| \|A\| \frac{\|\Delta b\|}{\|b\|} \quad (4)$$

(4) holds for any p-norm.

Let  $\kappa(A) \equiv \|A^{-1}\| \|A\|$  be the condition number of  $A$ .

Eqn (4) says that

$$\left( \begin{array}{c} \text{relative change} \\ \text{in } x \end{array} \right) \leq \left( \begin{array}{c} \text{condition} \\ \text{number} \end{array} \right) \left( \begin{array}{c} \text{relative change} \\ \text{in } b \end{array} \right)$$

If  $\kappa(A)$  is large, problem may be ill conditioned (bound very sharp)

- may be a gross overestimate

- If  $\kappa(A)$  is near one, problem is okay for sure.



Now suppose we perturb, the matrix elements

$A \rightarrow A + \Delta A$  so that

$$(A + \Delta A)(x + \Delta x) = b \quad (5)$$

Assuming  $Ax = b$ , then (5) gives

$$A \Delta x = -\Delta A(x + \Delta x)$$

$$\Delta x = -A^{-1} \Delta A(x + \Delta x)$$

which gives

$$\|\Delta x\| \leq \|A^{-1}\| \|\Delta A\| \|x + \Delta x\|$$

$$\Rightarrow \frac{\|\Delta x\|}{\|x + \Delta x\|} \leq \underbrace{\|A^{-1}\| \|A\|}_{= \kappa(A)} \cdot \frac{\|\Delta A\|}{\|A\|}$$

$$= \kappa(A) \frac{\|\Delta A\|}{\|A\|} \quad (6)$$

(6) Says  $\left( \begin{array}{c} \text{relative change} \\ \text{in } x \end{array} \right) \leq \left( \begin{array}{c} \text{condition} \\ \text{number} \end{array} \right) \times \left( \begin{array}{c} \text{relative change} \\ \text{in } A \end{array} \right)$

$\Rightarrow$  Condition number,  $\kappa(A)$  is a measure of solution sensitivity to change in  $A$  or  $b$ .

$$Ax = \textcircled{b} \quad b \rightarrow b + \Delta b$$

both  $A, b$

Feb 12, 2018

Recall:

Condition of problem:  $A\vec{x} = \vec{b}$

Condition number ( $\kappa(A)$ ) is a measure of solution sensitivity to change in  $A$  or  $b$  (or both)

$$\kappa(A) = \|A\| \|A^{-1}\| \quad (1)$$

for any  $p$ -norm.

Observations about  $\kappa(A)$ :

$$1) \kappa(A) = \|A\| \|A^{-1}\| \geq \|AA^{-1}\| = \|I\| = 1 \quad (2)$$

$$\text{so } \kappa(A) \geq 1$$

$$2) \kappa(\alpha A) = \|\alpha A\| \|\frac{1}{\alpha} A^{-1}\| = \|A\| \|A^{-1}\| = \kappa(A) \quad (3)$$

3) 2-norm condition number ( $\kappa_2(A)$ ) =  $\|A\|_2 \|A^{-1}\|_2$  can be computed without computing inverse of  $A$ , which is  $A^{-1}$ .

$$\kappa_2(A) = \|A\|_2 \|A^{-1}\|_2 = \sqrt{\frac{\lambda_{\max}(A^T A)}{\lambda_{\min}(A^T A)}} \quad (4)$$

(1)

## Residual:

The accuracy of the computed solution is sometimes measured by the size of the residual, (5)

i.e. Let the computed solution to  $A\vec{x} = \vec{b}$  be  $\vec{x} + \Delta\vec{x}$  (i.e. its not exact)

Let  $\vec{r} = \vec{b} - A(\vec{x} + \Delta\vec{x})$  be the residual (r) (6)

If  $\vec{r} = \vec{0} \Rightarrow \Delta\vec{x} = \vec{0}$  and we have the exact solution.

Similar analysis as in equations (1) - (4) gives

$$\frac{\|\Delta\vec{x}\|}{\|\vec{x}\|} \leq \kappa(A) \frac{\|\vec{r}\|}{\|\vec{b}\|} \quad (7)$$

$\Rightarrow$  Small residue does not necessarily mean small error.

If, for example,  $\frac{\|\vec{r}\|}{\|\vec{b}\|} \sim \epsilon_{\text{mach}}$  then (7) implies  $\frac{\|\Delta\vec{x}\|}{\|\vec{x}\|} \leq \kappa(A) \epsilon_{\text{mach}}$

which may not be small if  $\kappa(A)$  is large.

## Pivoting

It is known that Gaussian elimination with pivoting produces the exact solution to

$$(A + E)\vec{x} = \vec{b} \quad (8)$$

where  $\|E\| = \rho \epsilon_{\text{mach}} \|A\|$  (9)

Practically  $\rho = O(1)$  [but pathological cases can be constructed where  $\rho$  is large, but this never happens in practice]

Assuming  $\rho = 1$  in (9) and using (6) we get

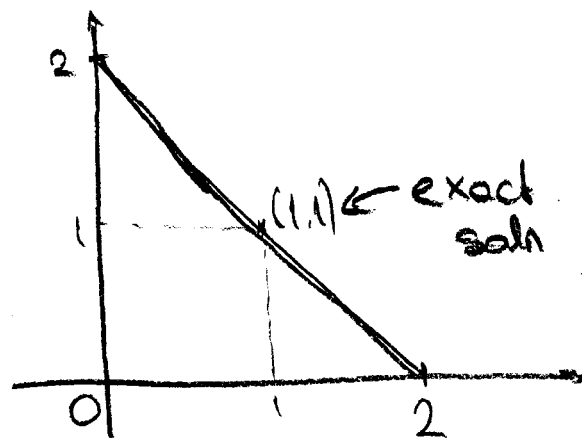
$$\frac{\|x - \tilde{x}\|}{\|\tilde{x}\|} \leq \kappa(A) \epsilon_{\text{mach}} \quad \text{(10)}$$

If matrix is badly conditioned, a stable method may not give useful results, due to property of the problem.

Ex

$$A \vec{x} = \vec{b}$$

$$\begin{bmatrix} 1 & 1 \\ 1.0001 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2.0001 \end{bmatrix}$$



$$\|A\|_{\infty} = \max \text{ abs row sum} = 2.0001$$

$$A^{-1} = \frac{1}{-0.0001} \begin{bmatrix} 1 & -1 \\ -1.0001 & 1 \end{bmatrix} = \begin{bmatrix} -10,000 & 10,000 \\ 10,001 & -10,000 \end{bmatrix}$$

$$\|A^{-1}\|_{\infty} = 20,001. \text{ So } \kappa_{\infty}(A) = \|A\|_{\infty} \|A^{-1}\|_{\infty} = 40,004,0001 \approx 4 \cdot 10^4$$

(10)  $\frac{\|x - \tilde{x}\|}{\|\tilde{x}\|} \leq \kappa(A) \epsilon_{\text{mach}}$

If we solve using pivoting, (10) says

$$\frac{\|\Delta x\|_{\infty}}{\|\tilde{x}\|_{\infty}} \leq \kappa(A) \epsilon_{\text{mach}}$$

Since  $\kappa(A) \approx 4 \cdot 10^4$  and in double precision,  $\epsilon_{\text{mach}} \approx 10^{-16}$ . So we should expect about  $16 - 4 = 12$  digits of accuracy in the solution of  $x$ .

## Iterative methods to solve $A\vec{x} = \vec{b}$

- For sparse matrices which arise in many applications we would like to avoid

- 1) A dense solve, work  $\sim O(N^3)$  ;  $N = \text{size of matrix}$
- 2) A sparse direct (Gaussian elimination) solve, work  $\sim O(N^2)$

## Jacobi Iteration

$$A\vec{x} = \vec{b}$$

Let the  $k^{\text{th}}$  iterate be denoted by  $\vec{x}^{(k)} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_N^{(k)} \end{bmatrix}$

The equation of the  $i^{\text{th}}$  row:

$$\sum_{j=1}^N a_{ij} x_j = b_i, \quad i=1, \dots, N \quad (11)$$

Given  $\vec{x}^{(k)}$  we update each element of  $\vec{x}^{(k+1)}$  as follows:

$$x_j^{(k+1)} = \frac{1}{a_{ii}} \left[ b_i - \sum_{\substack{j=1 \\ j \neq i}}^N a_{ij} x_j^{(k)} \right] \quad (12) \\ i=1, \dots, N$$

i.e. substituting  $\vec{x}^k$  in (11) and solving for  $x_i^{(k+1)}$

Recall a matrix  $A$  is row diagonally dominant if

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^N |a_{ij}|, \quad i=1, \dots, N$$

Theorem: If  $A$  is row diagonally dominant, the Jacobi iteration converges for any starting vector,  $\vec{x}^0$ .

Proof: Let  $\vec{x}$  be the solution to  $A\vec{x} = \vec{b}$ , then

$$x_i = \frac{1}{a_{ii}} \left[ b_i - \sum_{j \neq i} a_{ij} x_j \right] \quad (13)$$

(12) - (13) gives

$$x_i^{(k+1)} - x_i = \frac{1}{a_{ii}} \left[ - \sum_{j \neq i} a_{ij} (x_j^{(k)} - x_j) \right] \quad (14)$$

Define  $e_i^{(k+1)} = x_i^{(k+1)} - x_i$  (the error), then (14) becomes

$$e_i^{(k+1)} = \frac{1}{a_{ii}} \left[ - \sum_{j \neq i} a_{ij} e_j^{(k)} \right]$$

$$|e_i^{(k+1)}| = \frac{1}{|a_{ii}|} \left| \sum_{j \neq i} a_{ij} e_j^{(k)} \right|$$

$$\leq \frac{1}{|a_{ii}|} \sum_{j \neq i} |a_{ij}| \underbrace{|e_j^{(k)}|}_{\leq \|e^k\|_\infty} \leq \|e^k\|_\infty \sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|}$$

But  $\sum_{j \neq i} \frac{|a_{ij}|}{|a_{ii}|} < 1$  since  $A$  is row diagonally dominant

Hence  $|e_i^{(k+1)}| < \|e^{(k)}\|_{\infty} \quad \forall i$

$$\Rightarrow \|e^{(k+1)}\|_{\infty} \leq \|e^{(k)}\|_{\infty}$$

$$\text{or } \|e^{(k+1)}\|_{\infty} = \rho \|e^{(k)}\|_{\infty}; \quad \rho \in [0, 1)$$

$$= \rho^{(k+1)} \|e^{(0)}\|_{\infty}$$

Hence  $\lim_{k \rightarrow \infty} \|e^{(k)}\| = 0 \Rightarrow \vec{x}^k \rightarrow \vec{x}^*$

### Gauss - Siedel Method

- Similar to Jacobi, except most recently updated values of  $x_i^{(k+1)}$  are used in the current update step.

$$\text{i.e. } x_i^{(k+1)} = \frac{1}{a_{ii}} \left[ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^N a_{ij} x_j^{(k)} \right]$$

Like Jacobi, Gauss-Siedel is guaranteed to converge if  $A$  is row diagonally dominant.

### General Iterative Methods

Say we have a guess  $\vec{x}^0$  for the solution of  $A\vec{x} = \vec{b}$ .

The residual is:  $\vec{r}^0 = \vec{b} - A\vec{x}^0$

The error in the solution is:  $e^0 = \vec{x} - \vec{x}^0$

Error is related to the residual as follows:

$$Ae^0 = A(\bar{x} - \bar{x}^0) = \bar{b} - A\bar{x}^0 = \bar{r}^0$$

$$\text{or } e^0 = A^{-1}\bar{r}^0$$

Now by definition  $\bar{x} = \bar{x}^0 + e^0$

$$\Rightarrow \bar{x} = \bar{x}^0$$

Of course we do not want to compute  $A^{-1}$ . Instead we can choose a matrix  $B$ , that is easy to invert,  $B^{-1} \approx A^{-1}$ .

$$\text{So } \bar{x}^0 \approx \bar{x}^0 + B^{-1}\bar{r}^0$$

Repeated application yields iterative formula

$$\bar{x}^{k+1} = \bar{x}^k + B^{-1}\bar{r}^k = \bar{x}^k + B^{-1}(\bar{b} - A\bar{x}^k) \quad (15)$$

### Convergence

Now if  $\bar{x}$  is the exact solution of  $A\bar{x} = \bar{b}$  we have

$$\bar{x} = \bar{x} + \underbrace{B^{-1}(\bar{b} - A\bar{x})}_{=0} \quad (16)$$

$$(15)(16) \quad \bar{x}^{k+1} - \bar{x} = (\bar{x}^k - \bar{x}) - B^{-1}A(\bar{x}^k - \bar{x}) \quad (17)$$

$$e^{k+1} = [I - B^{-1}A] e^k$$

which then implies

$$\|e^{k+1}\| \leq \|I - B^{-1}A\| \|e^k\| \text{ which will converge if } \|I - B^{-1}A\| < 1$$

in some norm.



# Notes:

For Jacobi iteration, we are choosing

$$B = \text{diag}(A) = \begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn} \end{bmatrix}$$

For Gauss-Seidel we are choosing

$$B = \text{lower}(A) = \begin{bmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & \ddots & \\ \vdots & \vdots & \vdots & a_{nn} \end{bmatrix}$$

Feb 14, 2018

Recall:

$$\frac{\|\Delta x\|}{\|x\|} \leq 10^{-12} \Rightarrow \approx 12 \text{ significant digits.}$$

Power Method: [not in course notes]

For the Google problem, we solved  $M \vec{p}^{\infty} = \vec{p}^{\infty}$

Since  $M$  is Markov Matrix, its largest eigenvalue is  $\lambda_1 = 1$   
then  $\vec{p}^{\infty}$  is the eigenvector associated with the largest magnitude eigenvalue.

With a slight modification to the algorithm, we can find the largest eigenvalue and the associated eigenvector.

For a more general matrix  $A \in \mathbb{R}^{n \times n}$

- e.g. dimensionality reduction, recommendation engines  
such as those used by Amazon etc.

Suppose  $A \in \mathbb{R}^{n \times n}$  and its eigenvalues (which we don't know) have the following property.

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \dots \geq |\lambda_n| \quad \left( \text{i.e. the largest eigenvalue in magnitude is unique} \right)$$

As before, if we assume  $n$  linearly independent eigenvectors such that  $\lambda \vec{x}_l = A \vec{x}_l$  for  $l=1, \dots, n$ .

Then for any starting vector  $\vec{y}^0$  we can write

$$\vec{y}^0 = \sum_l c_l \vec{x}_l \quad ; \quad c_l \text{ are constants}$$

$$\begin{aligned} \text{So, } \vec{y}^k &= A^k \vec{y}^0 = \sum_l c_l (\lambda_l)^k \vec{x}_l \\ &= c_1 \lambda_1^k \vec{x}_1 + \sum_{l=2}^n c_l (\lambda_l)^k \vec{x}_l \\ &= \lambda_1^k \left[ c_1 \vec{x}_1 + \sum_{l=2}^n c_l \left( \frac{\lambda_l}{\lambda_1} \right)^k \vec{x}_l \right] \end{aligned}$$

Since for  $l > 1$ ,  $\lambda_l / \lambda_1 < 1$  then as  $k \rightarrow \infty$ ,  $\frac{A^k \vec{y}^0}{\lambda_1^k} = c_1 \vec{x}_1$ .

$$\text{Also if we define } r_k = \frac{\|\vec{y}^{(k+1)}\|_\infty}{\|\vec{y}^{(k)}\|_\infty} = \lambda_1 \frac{\|c_1 \vec{x}_1 + O\left(\frac{\lambda_2}{\lambda_1}\right)^{k+1}\|}{\|c_1 \vec{x}_1 + O\left(\frac{\lambda_2}{\lambda_1}\right)^k\|}$$

Then  $\lim_{k \rightarrow \infty} r_k = \lambda_1$ .

### Power Algorithm

$$\vec{z}^0 = \vec{y}^0$$

For  $k=1, \dots$ , until converged

$$\vec{y} = A \vec{z}$$

$$r_k = \|\vec{y}\|_\infty / \|\vec{z}\|_\infty$$

$$\vec{z}^k = \vec{y} / r_k$$

End for

# Newton Iterations for System of Eqns

Suppose we have three coupled non-linear equations

$$f_1(x_1, x_2, x_3) = 0$$

$$f_2(x_1, x_2, x_3) = 0$$

$$f_3(x_1, x_2, x_3) = 0$$

We want to solve for the root  $(x_1^*, x_2^*, x_3^*)$  such that

$$f_i(x_1^*, x_2^*, x_3^*) = 0 \text{ for } i=1, 2, 3$$

Define vector  $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$  and  $\vec{F}(\vec{x}) = \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \\ f_3(x_1, x_2, x_3) \end{bmatrix}$

Thus we want to solve  $\vec{x}^*$  s.t.  $\vec{F}(\vec{x}^*) = 0$

The analogue of derivative  $f'$  in one-variable case is the Jacobian matrix:

$$J(\vec{x}) = \begin{bmatrix} \frac{\partial f_i}{\partial x_j} \end{bmatrix}_{m \times n} \quad \text{where } \begin{matrix} 1 \leq i \leq m \\ 1 \leq j \leq n \end{matrix}$$

• Taylor Expansion for vector valued functions around  $\vec{x}^0$  is

$$\vec{F}(\vec{x}) = \vec{F}(\vec{x}^0) + J(\vec{x}^0)(\vec{x} - \vec{x}^0) + o(\|\vec{x} - \vec{x}^0\|^2) \quad (1)$$

We want to find  $\vec{x}^*$  such that  $\vec{F}(\vec{x}^*) = 0 \quad (2)$

Setting  $\vec{x} = \vec{x}^*$  in (1) and sub (2)  $\rightarrow$  (1)

$$0 \approx F(\vec{x}^0) + J(\vec{x}^0)(\vec{x}^* - \vec{x}^0)$$

$$\vec{x}^* \approx \vec{x}^0 - [J(\vec{x}^0)]^{-1} F(\vec{x}^0)$$

Repeated application yield iterations

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - [J(\vec{x}^{(k)})]^{-1} F(\vec{x}^{(k)})$$

Note: In practice, instead of computing the inverse of Jacobian matrix,  $[J(\vec{x}^{(k)})]^{-1}$ , at each step, we compute

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - \vec{s}$$

where  $\vec{s}$  is the solution to the system

$$J(\vec{x}^{(k)}) \vec{s} = F(\vec{x}^{(k)})$$

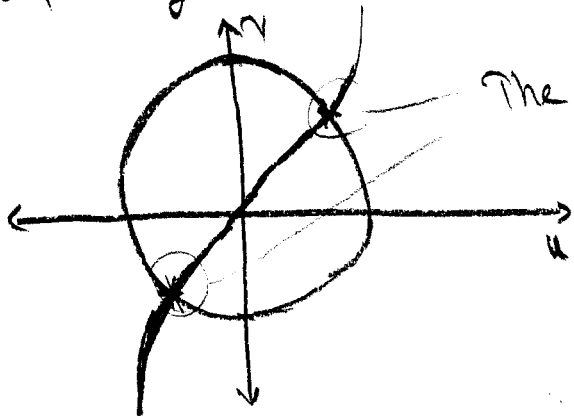
which we can solve using LU factorization.

~~P2x~~ Find soln to the following system, with initial guess

$$\begin{aligned} n - u^3 &= 0 \\ u^2 + n^2 &= 1 \end{aligned}$$

$$\vec{x}^0 = \begin{bmatrix} u \\ n \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Graphically we have:



The intersection points are the solution.

We have

$$J\left(\begin{bmatrix} u \\ v \end{bmatrix}\right) = \begin{bmatrix} -3u^2 & 1 \\ 2u & 2v \end{bmatrix} \cdot \text{Starting point } \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

solve  $\vec{s}$ .

$$J\vec{s} = \vec{r}$$

$$\nearrow F(\vec{x}^0)$$

$$\begin{bmatrix} -3 & 1 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix} \Rightarrow \vec{s} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\text{So, } \vec{x}^1 = \vec{x}^0 - \vec{s} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

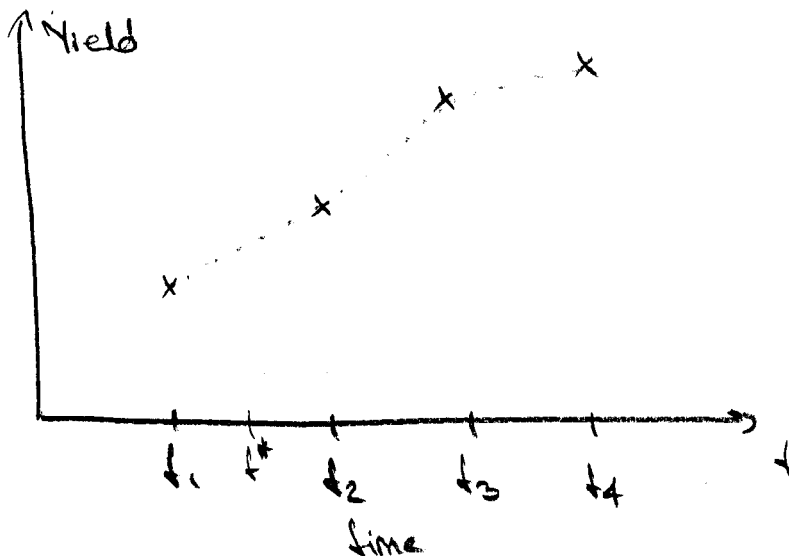
and so on...

Interpolation

Consider the following problem.

- In financial markets there are bonds that trade at various maturities.

- By examining the traded price of a bond and its cash flows, we can determine the yield for a bond maturing at time  $t$ .



- Suppose we want to issue a bond with maturity,  $t^*$ ,

$$t_1 < t^* < t_2$$

i.e. between prices of listed bonds

- How do we set the coupon (i.e. the yield) for this bond, consistent with other traded bonds?

- We need to interpolate the existence data to give estimate of the "missing" information

## Applications:

- Animation, fonts, computer graphics, engineering, automobile and aircraft design.

## Polynomial Interpolation

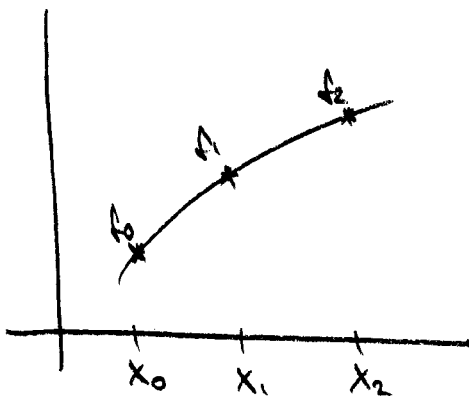
- Many ways to choose interpolating function
- Polynomial is widely used

Note: Taylor series tells us that any smooth function can be approximated by a polynomial

Suppose we have 3 data points:

x	f
$x_0$	$f_0$
$x_1$	$f_1$
$x_2$	$f_2$

We want to draw a smooth curve which passes through the three points.



Let the polynomial interpolant be denoted by

$$P(x) = a_0 + a_1x + a_2x^2$$

How can we determine  $\{a_0, a_1, a_2\}$

- Force  $p(x)$  to agree with data points at  $\{x_0, x_1, x_2\}$

i.e.  $p(x_i) = f_i$  for  $i=0,1,2$



$$a_0 + a_1 x_0 + a_2 x_0^2 = f_0$$

$$a_0 + a_1 x_1 + a_2 x_1^2 = f_1$$

$$a_0 + a_1 x_2 + a_2 x_2^2 = f_2$$

or in  
matrix form

$$\begin{bmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \end{bmatrix}$$

In general, for a polynomial of degree  $n$  and  $n+1$  points,  
the conditions

$$p(x_i) = f_i$$

gives a system of the form

$$V = \begin{bmatrix} 1 & x_0 & \dots & x_0^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1 & \dots & x_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \dots & x_n^n \end{bmatrix} \quad \vec{a} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \quad \vec{f} = \begin{bmatrix} f_0 \\ \vdots \\ f_n \end{bmatrix}$$

$V \vec{a} = \vec{f}$ .  $V$  is known as the Vandermonde matrix.

**Thm** Given  $n+1$  data points  $(x_i, f_i)$ , with distinct  $x_i$ , there is a unique polynomial of degree not exceeding  $n$  which interpolates the data.

Proof:  $\det(V) = \prod_{i < j} (x_i - x_j) \neq 0 \Rightarrow V$  is non-singular

Unique? Suppose there are two polynomials of degree  $n$ .

$p^A(x), p^B(x)$  which interpolate the data.

$$P^A(x) = a_0^A + a_1^A x + \dots + a_n^A x^n$$

$$P^B(x) = a_0^B + a_1^B x + \dots + a_n^B x^n$$

such  
that

$$V \vec{a}^A = \vec{f} \quad (1)$$

$$V \vec{a}^B = \vec{f} \quad (2)$$

$$(1) \cdot (2) \quad V(\vec{a}^A - \vec{a}^B) = 0 \Rightarrow \vec{a}^A = \vec{a}^B \quad [V \text{ is non-singular}]$$

then  $P^A(x) = P^B(x)$

## Lagrange Basis Functions

- Vandermonde is a brute force way to determine the interpolating polynomial.

Original problem:

x	f
$x_0$	$f_0$
$x_1$	$f_1$
$x_2$	$f_2$

We can write the solution we are looking for as

$$P(x_0) = 1 \cdot f_0 + 0 \cdot f_1 + 0 \cdot f_2$$

$$P(x_1) = 0 \cdot f_0 + 1 \cdot f_1 + 0 \cdot f_2$$

$$P(x_2) = 0 \cdot f_0 + 0 \cdot f_1 + 1 \cdot f_2$$

So we would like to find three polynomials of degree 2,

$$L_0(x), L_1(x), L_2(x)$$

such that

$$L_0(x_0) = 1, L_1(x_0) = 0, L_2(x_0) = 0$$

$$L_0(x_1) = 0, L_1(x_1) = 1, L_2(x_1) = 0$$

$$L_0(x_2) = 0, L_1(x_2) = 0, L_2(x_2) = 1$$

Then the degree 2 polynomial which agrees with the data

$$\{x_0, x_1, x_2\}$$

is

$$p(x) = L_0(x)f_0 + L_1(x)f_1 + L_2(x)f_2$$

**Proof:**  $P(x) = \sum$  degree 2 polynomials

$\implies P(x)$  is also 2<sup>nd</sup> degree polynomial.

By construction  $p(x_i) = f_i$   $i = 0, 1, 2$ .

How can we find these polynomials?

$$L_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} = \begin{cases} 1 & \text{at } x=x_0 \\ 0 & \text{at } x=x_1, x_2 \end{cases}$$

$$L_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} = \begin{cases} 1 & \text{at } x=x_1 \\ 0 & \text{otherwise} \end{cases}$$

$$L_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} = \begin{cases} 1 & \text{at } x=x_2 \\ 0 & \text{otherwise} \end{cases}$$

Question: Is this the same polynomial we get by solving the Vandermonde system?

- Yes -

[Polynomial which interpolates the data is unique.]

## General Result

Given a set of  $n+1$  data points  $(x_i, y_i)$  where  $i = 0, 1, \dots, n$   
the set of  $n+1$  Lagrange basis functions is defined by

$$L_k(x) = \frac{(x-x_0)(x-x_1)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_n)}{(x_k-x_0)(x_k-x_1)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_n)}$$

$$k = 0, 1, \dots, n$$

Note:  $L_k(x_j) = \begin{cases} 1 & \text{if } k=j \\ 0 & \text{otherwise} \end{cases}$

More compact notation

$$L_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^n \frac{(x-x_i)}{(x_k-x_i)}$$

Final result:

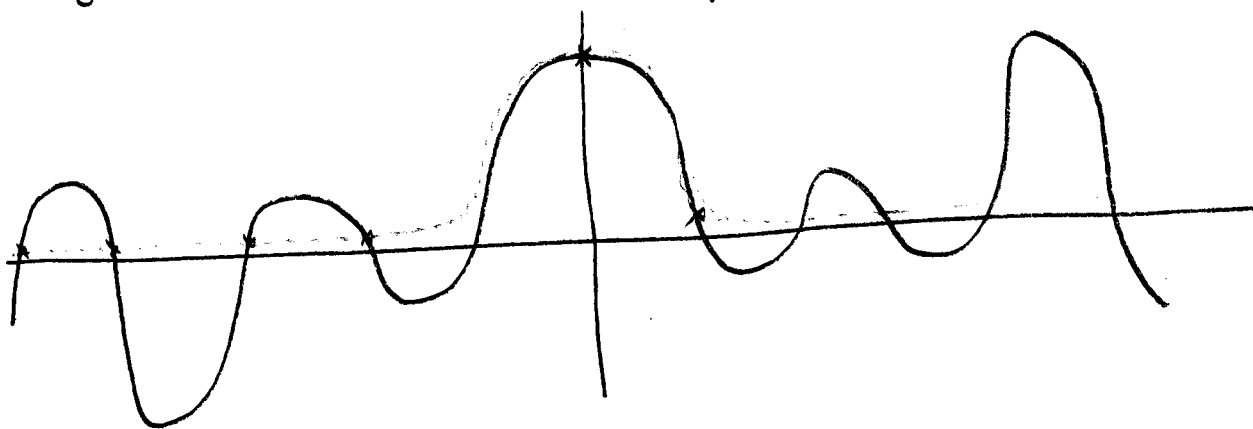
The polynomial of degree  $n$  which interpolates data

$$(x_i, f_i) \quad i = 0, 1, \dots, n \quad \text{is}$$

$$\bar{f}(x) = \sum_{i=0}^n f_i L_i(x)$$

## Problem with Polynomial Interpolation

If the number of data points is large ( $> 4$ ) then a high degree polynomial is required to fit data points.



high degree polynomials are "oscillatory" unstable

## Piecewise Polynomial Interpolation

Another possibility is to use a piecewise interpolant

- Suppose we have  $n+1$  data points  $(x_0, y_0), \dots, (x_n, y_n)$   
with  $x_0 < x_1 < x_2 < \dots < x_n$

Then a piecewise interpolating polynomial for these  $n+1$  points is a function  $p(x)$  satisfying

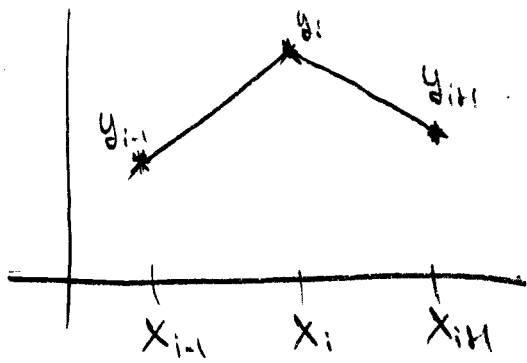
1)  $p(x)$  is a function of  $x$ ,  $x \in [x_0, x_n]$

2)  $p(x)$  interpolates the data

$$\text{i.e. } p(x_i) = y_i \quad i = 0, 1, \dots, n$$

3) Continuous on the interval  $[x_0, x_n]$

Simplest example is piecewise linear interpolation.



Let  $P_i(x)$  be the polynomial defined on  $[x_{i-1}, x_i]$ . Then,  $P_i(x)$  is equal to

$$P_i(x) = y_{i-1} \left( \frac{x - x_i}{x_{i-1} - x_i} \right) + y_i \left( \frac{x - x_{i-1}}{x_i - x_{i-1}} \right)$$

where  $x \in [x_{i-1}, x_i]$

Lagrange basis functions

For  $x \in [x_i, x_{i+1}]$

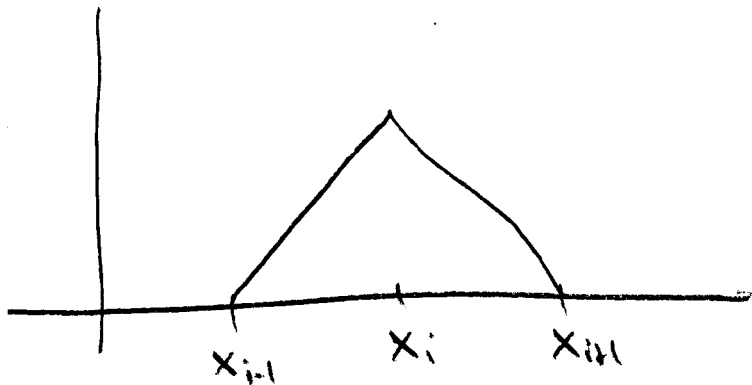
$$P_i(x) = y_i \left( \frac{x - x_{i+1}}{x_i - x_{i+1}} \right) + y_{i+1} \left( \frac{x - x_i}{x_{i+1} - x_i} \right)$$

Point  $x_i$  is associated with one Lagrange basis function

We say the piecewise Lagrange basis function associated with node  $i$  is

$$L_i^p(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & x \in [x_i, x_{i+1}] \\ 0 & \text{otherwise} \end{cases}$$

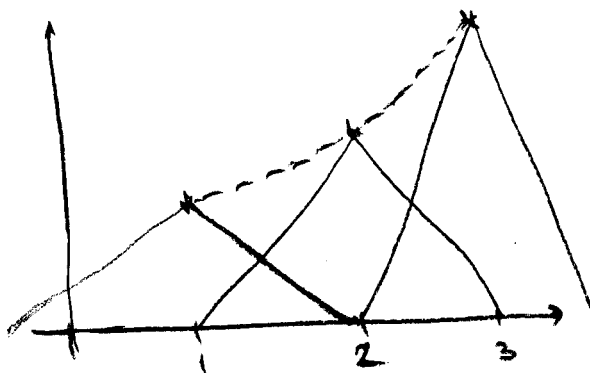
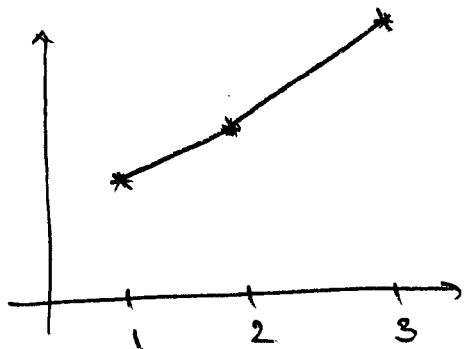
$L_i^P(x)$



"kinks"

We can write the piecewise linear interpolation as

$$p(x) = \sum_{i=0}^n y_i L_i^P(x) \leftarrow \text{piecewise Lagrange basis functions}$$



Linear basis functions are not very smooth "kinks" at each point

Consider a quadratic piecewise

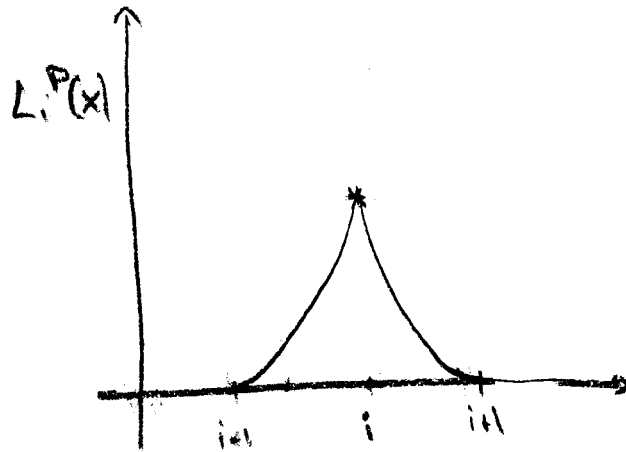
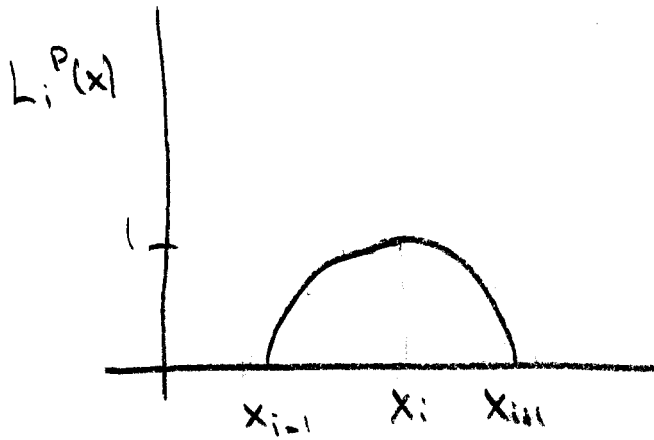
$$p_i(x) = y_{i-1} \frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})}$$

$$x \in [x_{i-1}, x_{i+1}] \quad + \quad y_i = \frac{(x-x_{i-1})(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})}$$

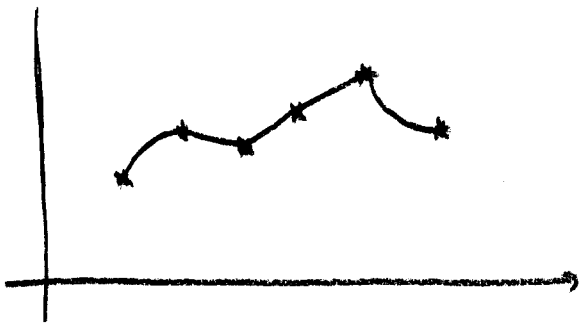
for  $i=1, 3, 5, \dots$

$$+ \quad y_{i+1} = \frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)}$$

In this case the piecewise basis functions for nodes  $i=1, 3, 5, \dots$



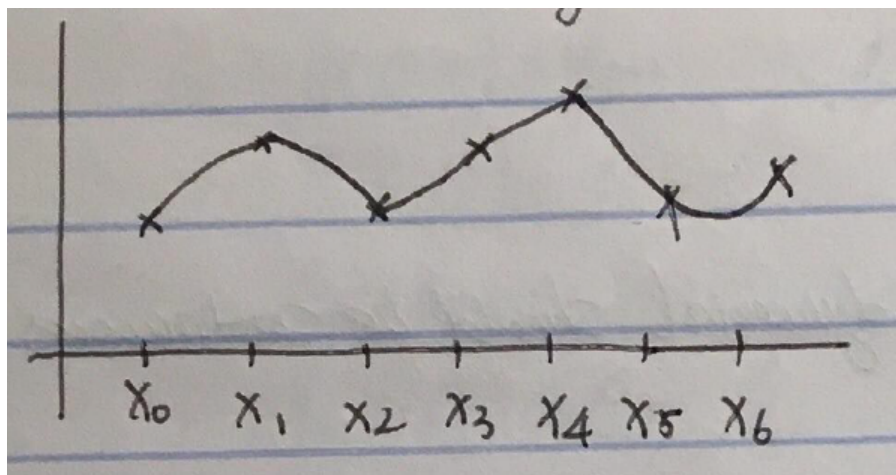
So  $p(x)$  will look like



- Curve is cts on some points but kinks appear where the quadratics join



Piecewise Quadratic Polynomial 15.4:



Use a quadratic polynomial on each subinterval  $[x_{i-1}, x_{i+1}]$  for  $i = 1, 3, 5, \dots$ ,

$$p_i(x) = \begin{matrix} y_{i-1} \frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})} \\ + y_i \dots \\ + y_{i+1} \dots \end{matrix} \quad x \in [x_{i-1}, x_{i+1}] \text{ where } i = 1, 3, 5, \dots,$$

So we have

$$p_{i+2}(x) = y_{i+1} \frac{(x-x_{i+2})(x-x_{i+3})}{(x_{i+1}-x_{i+2})(x_{i+1}-x_{i+3})} + y_{i+2} \dots + y_{i+3} \dots \quad \text{where } i \text{ is odd.}$$

Case 1:  $x_i : i$  is odd, and it is associated with the Lagrange basis function from  $p_i$ .

$$L_i^p = \begin{cases} \frac{x+x_{i-1}}{x-x_{i-2}} \frac{x+x_{i+1}}{x-x_{i+2}} & x \in [x_{i-1}, x_{i+1}] \\ 0 & \text{otherwise} \end{cases} \quad i=1,2,5,\dots$$

Case 2:  $x_i : i$  is even, it is associated with Lagrange basis function from  $p_{i-1}$  and  $p_{i+1}$

$$L_i^p = \begin{cases} \frac{(x+x_{i-2})(x+x_{i-1})}{(x-x_{i-2})(x-x_{i-1})} & x \in [x_{i-2}, x_i] \\ \frac{(x+x_{i+1})(x+x_{i+2})}{(x-x_{i+1})(x-x_{i+2})} & x \in [x_i, x_{i+2}] \end{cases} \quad i=2,4,6,\dots$$

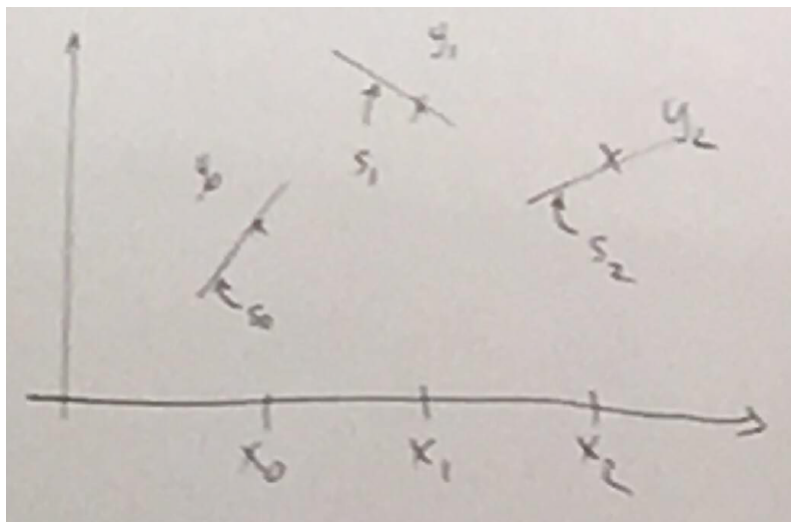
**Remark 15.5:**

$$p(x) = \sum_{i=0}^n y_i L_i^p(x)$$

- has kinks at even  $i$
- can we eliminate the kink problem?
- derivative of interpolating polynomial should be continuous.

**Piecewise Hermite Interpolation 15.6:** Suppose we want to interpolate both the function and the derivatives. In other words, we are given the data

$x$	$y$	slopes
$x_0$	$y_0$	$s_0$
$x_1$	$y_1$	$s_1$
$x_2$	$y_2$	$s_2$
$\vdots$	$\vdots$	$\vdots$



we want to draw a smooth curve through  $x_0, \dots, x_n$  such that the piecewise polynomial agrees with the data and the derivatives  $s_0, \dots, s_n$ . If we do this, then curve will appear smooth, since derivatives are continuous at the data points. Suppose we have  $x_0, \dots, x_n$  which means there are  $n$  intervals so that  $[x_{i-1}, x_i]$   $i = 1, \dots, n$ . Let  $p_i(x)$  be a piecewise cubic polynomial defined on  $[x_{i-1}, x_i]$ . In other words,

$$p_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^2(x - x_{i-1}) \quad \text{for } x \in [x_{i-1}, x_i] \quad (1)$$

Note that we have cleverly picked from  $d_i((x - x_{i-1})^2(x - x_{i-1}))$  for last term in (1) to minimize the algebra.

**Remark 15.7:** If  $p_i(x)$  interpolates the data then we have

$$\left. \begin{aligned} p_i(x_{i-1}) &= y_{i-1} \\ p_i(x_i) &= y_i \end{aligned} \right\} \text{data}$$

$$\left. \begin{aligned} p'_i(x_{i-1}) &= s_{i-1} \\ p'_i(x_i) &= s_i \end{aligned} \right\} \text{slopes}$$

So we have four condition and four unknown parameters  $\{a_i, b_i, c_i, d_i\}$  on each subinterval. Our solution is

$$a_i = y_{i-1} \quad b_i = s_{i-1} \quad c_i = \frac{m_{i-1} - s_{i-1}}{\delta x_{i-1}} \quad \text{where} \quad \begin{aligned} \delta_{i-1} &= x_i - x_{i-1} \\ m_{i-1} &= \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \end{aligned}$$

It follows that at each of the  $(n - 1)$  interior points  $\{x_1, \dots, x_{n-1}\}$  we have

$$\begin{aligned} p_i(x_i) &= p_{i+1}(x_i) \\ p'_i(x_i) &= p'_{i+1}(x_i) \quad \text{where } i = 1, 2, \dots, n - 1 \end{aligned}$$

The interpolant and its first derivative are continuous at the *knots*.

**Definition 15.8:** We call the points where the interpolant changes behavior from one polynomial to another as **knots**.

**Definition 15.9:** We call the points where the data is specified as **nodes**.

**Remark 15.10:** Note that in Remark 15.7 nodes and knots are the same. However we are generally not so lucky as to have both data values and slopes specified at the knots. Usually we only have the data  $(x_i, y_i)$ . So how can we produce smooth interpolant using only data points  $(x_i, y_i)$ ?

**Spline Interpolation 15.11:**

**Definition 15.12:** A **spline interpolant** is a piecewise polynomial of degree  $n$  such that

1. The splines agree with the interpolated function at the knots.
2. The first  $(n - 1)$  derivatives are also continuous at the knots.

**Remark 15.13:** In practice, we usually have *cubic splines*. This allows continuity of the first and second derivatives at the knots which makes them appear smooth to the eye.

**Definition 15.14:** A **cubic spline**  $S(x)$  is a piecewise cubic polynomial such that  $S(x), S'(x), S''(x)$  are all continuous at the knots. Suppose we have the data

$x$	$y$
$x_0$	$y_0$
$x_1$	$y_1$
$\vdots$	$\vdots$

Denote the cubic spline on the interval  $[x_{i-1}, x_i]$  by  $S_i(x)$ . For  $S_i(x)$  to be a cubic spline, the following conditions must hold:

1. Splines interpolates the data:

$$\begin{aligned} S_i(x_{i-1}) &= y_{i-1} \\ S_i(x_i) &= y_i \quad \text{where } i = 1, 2, \dots, n \end{aligned} \tag{2}$$

2. First and second derivatives are continuous at the knots (interior points):

$$\begin{aligned} S'_i(x_i) &= S'_{i+1}(x_i) \\ S''_i(x_i) &= S''_{i+1}(x_i) \quad \text{where } i = 1, 2, \dots, n \end{aligned} \tag{3}$$

**Remark 15.15:** We check to see if we have the right number of equations and unknowns.

Number of unknowns:

- Each of  $S_i(x)$  has four parameters. That is,

$$S_i(x) = a_i + b_i x + c_i x^2 + d_i x^3, \quad \text{for } x \in [x_{i-1}, x_i], i = 1, 2, \dots, n$$

- Number of  $S_i$ 's is the same of number of intervals which is  $n$ .
- So we get the total number of unknowns as  $4n$ .

Number of equations:

- From the continuity conditions (2), we get 2 equations for each  $S_i$  where  $i = 1, 2, \dots, n$ . So we get  $2n$  equations.
- From the derivative (smoothness) conditions (3), we get 2 equations for each  $(n - 1)$  interior points, so get  $2n - 2$  equations.

- So we get the total number of equations as  $4n - 4$ .

**Boundary Conditions in Cubic Spline 15.16:** We can have three of the following boundary conditions in a cubic spline:

---

**Free boundary:** A free boundary has  $S_1''(x_0) = 0, S_n''(x_n) = 0$ .

It minimizes average square curvature on  $[x_0, x_n]$ .

A cubic spline with free boundary conditions is known as a **natural cubic spline**.

**Clamped boundary:** A clamped boundary has  $S_1'(x_0) = \text{specified}$  and  $S_n'(x_n) = \text{specified}$ .

This is useful for controlling behavior at boundaries directly. A cubic spline with clamped boundary conditions is known as a **complete cubic spline**.

**Periodic boundary:** A periodic boundary has  $S_1'(x_0) = S_n'(x_n)$  and  $S_1''(x_0) = S_n''(x_n)$ .

A cubic spline with periodic boundary conditions is known as a **periodic cubic spline**.

March 05  
2018

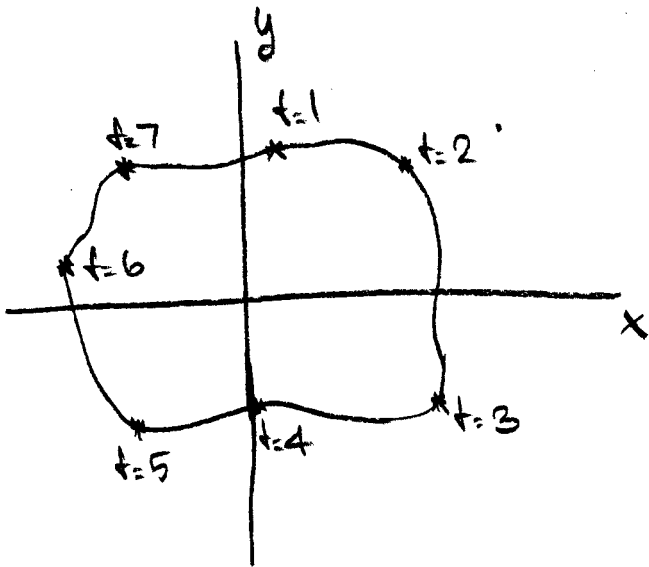
## Periodic Spline

A periodic spline has

$$S''(x_0) = S''(x_n)$$

$$S'(x_0) = S'(x_n)$$

Useful for parametric 2-d closed loops



Two independent splines  $x(t), y(t)$   
as a function of parameter  $t$ .

Recall the conditions for cubic  
spline:

$$S_1(x_i) = S_2(x_i)$$

$$S'_1(x_i) = S'_2(x_i)$$

$$S''_1(x_i) = S''_2(x_i)$$

## Not-a-knot spline

$$S_1'''(x_1) = S_2'''(x_1)$$

$$S_{n-1}'''(x_{n-1}) = S_n'''(x_{n-1})$$

$$\Rightarrow S_1 \equiv S_2$$

$$S_{n-1} \equiv S_n$$

$\Rightarrow$  dropping knot pts  $x_1$  and  $x_{n-1}$

use same polynomial  $[x_0, x_2]$  and

$[x_{n-2}, x_n]$ .

In general then, once we impose two additional constraints, we will have

$4n$  eqns,  $4n$  unknowns

- however, some (tedious) algebra reduces this set of equations to a  $(n+1)$  tridiagonal matrix for the slopes

$$S_i, i=0, 1, \dots, n, S_i = \frac{dS_i(x)}{dx}$$

$$\begin{bmatrix} \text{---} & & & & \\ & \text{---} & & & \\ & & \text{---} & & \\ & & & \text{---} & \\ & & & & \text{---} \end{bmatrix} \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_n \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

$(n+1) \times (n+1)$

A tridiagonal can be solved in  $O(n)$  operations  $\rightarrow$  very fast

- Once we know the  $S_i$ , then the coefficients of the spline on each subinterval are easily determined.

### Cubic Spline Example

Derive the natural cubic interpolating spline for the data

x	y
-1	1
0	2
1	-1

$$\text{Let } S(x) = \begin{cases} S_1(x) = a_1 + b_1x + c_1x^2 + d_1x^3, & x \in [-1, 0] \\ S_2(x) = a_2 + b_2x + c_2x^2 + d_2x^3, & x \in [0, 1] \end{cases}$$

From the continuity conditions we have,

$$S_1(-1) = 1 \Rightarrow a_1 - b_1 + c_1 - d_1 = 1 \quad (5)$$

$$S_1(0) = 2 \Rightarrow a_1 = 2 \quad (6)$$

$$S_2(0) = 2 \Rightarrow a_2 = 2 \quad (7)$$

$$S_2(1) = -1 \Rightarrow a_2 + b_2 + c_2 + d_2 = -1 \quad (8)$$

1<sup>st</sup> and 2<sup>nd</sup> derivatives:

$$S'(x) = \begin{cases} S_1'(x) = b_1 + 2c_1x + 3d_1x^2 & , x \in [-1, 0] \\ S_2'(x) = b_2 + 2c_2x + 3d_2x^2 & , x \in [0, 1] \end{cases}$$

$$S''(x) = \begin{cases} S_1''(x) = 2c_1 + 6d_1x & , x \in [-1, 0] \\ S_2''(x) = 2c_2 + 6d_2x & , x \in [0, 1] \end{cases}$$

From smoothness conditions (4) we have:

$$S_1'(0) = S_2'(0) \Rightarrow b_1 = b_2 \quad (9)$$

$$S_1''(0) = S_2''(0) \Rightarrow c_1 = c_2 \quad (10)$$

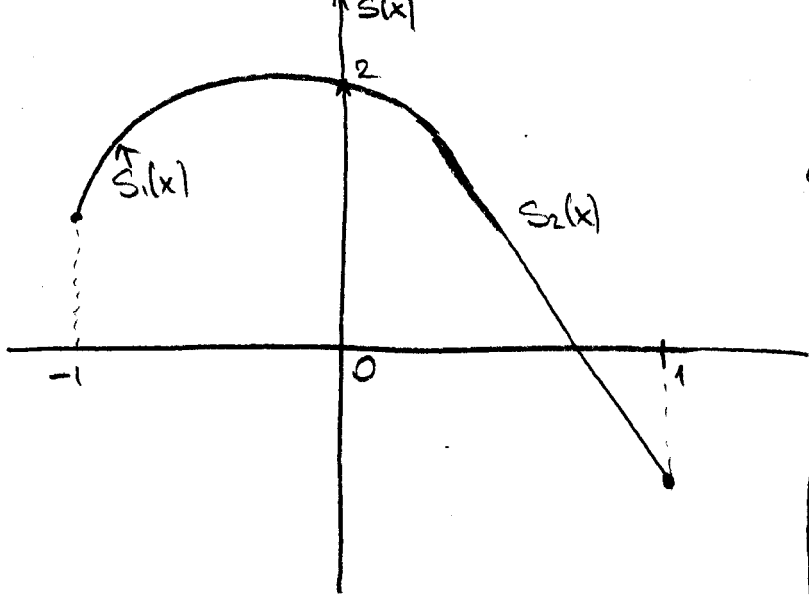
For  $S$  to be a natural cubic spline:

$$S_1''(-1) = 0 \Rightarrow c_1 = 3d_1 \quad (11)$$

$$S_2''(1) = 0 \Rightarrow c_2 = -3d_2 \quad (12)$$

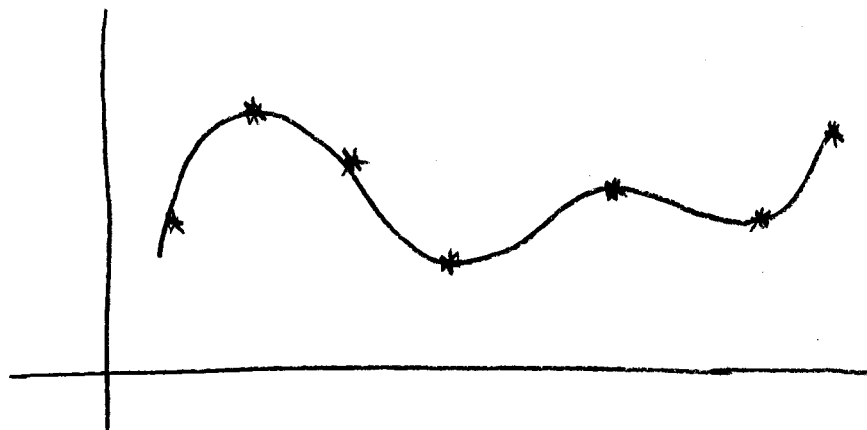
From (5) - (12) we obtain

$$S(x) = \begin{cases} 2 - x - 3x^2 - x^3 & , x \in [-1, 0] \\ 2 - x - 3x^2 + x^3 & , x \in [0, 1] \end{cases}$$



In general, spline interpolant gives us a smooth curve through our data

- while reducing unnecessary curvature.



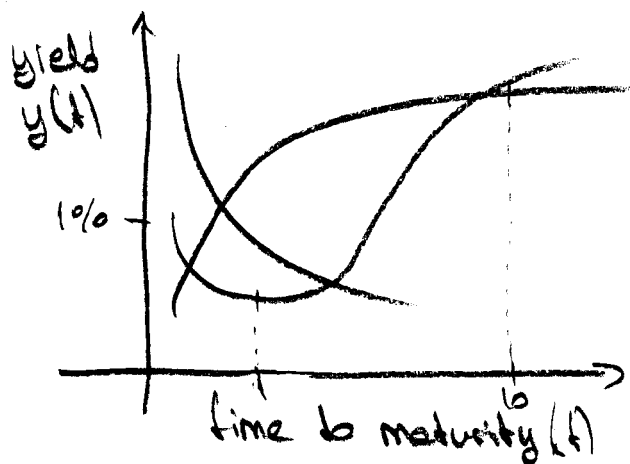
### Application: Zero Coupon Yield Interpolation (Q6, A3)

The yield curve is the relationship between yields (interest rates) and different terms of maturities.

- In finance, we require an accurate picture of the yield curve

→ to price bonds, derivatives, any cash flow analysis

→ macro economic analysis, forecasting and arbitrage.



- the clearest picture of the yield curve is obtained by looking at the yield of zero-coupon bonds of different maturities.



Def<sup>n</sup>: Zero-coupon bonds are bonds which have no intermediate (coupon) payments - all principle and interest is paid at maturity.

e.g. Say we have a set of listed zero-coupon bonds which pay 100 at maturity

maturity (t)	1	2	3	...
current price	95	93	90	

Let  $y(t)$  = the zero-yield curve function

Under continuous compounding:  $V_0 = V_t e^{-rt}$  annualized yield  
 Value at  $t=0$       Value at time  $t$

So we can compute the zero yields for the given maturities:

$$95 = 100 e^{-y(1)} \Rightarrow y(1) = \dots$$

$$93 = 100 e^{-2y(2)} \Rightarrow y(2) = \dots$$

$$90 = 100 e^{-3y(3)} \Rightarrow y(3) = \dots$$

Problem: Most traded bonds are coupon bonds, not zero-coupon bonds (e.g. US treasury bonds)

Can we determine the zero-yield from a coupon paying bond?  
 Yes, but require a set of bonds.

1) bootstrapping  $\rightarrow$  requires a sufficiently complete set of bonds which have matching coupon dates.

$\rightarrow$  provides point estimates only

2) "fit" to a rigid parametric curve

- useful for macroanalysis, but can lose detail.

We'll use a cubic spline to interpolate/extract the entire zero-curve from prices of coupon bonds.

- min restriction of the set of bonds required.

- obtain accurate (flexible) smooth yield curve

Defn: Discount curve is a function  $f(t)$  that represents the present value of receiving one unit of currency,  $t$  years in the future

$$f(t) = e^{-ty(t)}$$

$$y(t) = \frac{-\log f(t)}{t}$$

It is easier to interpolate the discount curve than the yield curve directly.

Divide the maturity axis into subintervals defined by  $(n+1)$  knot pts.

$$0 = t_1 < t_2 < t_3 < \dots < t_n < t_{n+1} = \text{max maturity}$$

NB: using 1-indexing to be consistent with matlab.

We can define the interpolating cubic spline for the discount curve.

$$f(t) = \sum_{i=1}^n P_i(t) I_i(t)$$

(1)

where  $P_i(t) = a_i + b_i(t - t_i) + c_i(t - t_i)^2 + d_i(t - t_i)^3$

and  $I_i(t) = \begin{cases} 1 & \text{if } t > t_i \\ 0 & \text{otherwise} \end{cases}$

In other words

$$f(t) = \begin{cases} P_1(t) & , t \in [t_1, t_2] \\ P_1(t) + P_2(t) & , t \in [t_2, t_3] \\ P_1(t) + P_2(t) + P_3(t) & , t \in [t_3, t_4] \end{cases}$$

Define the interpolating cubic spline for the discount curve as

$$\begin{aligned}
 f(t) &= \sum_{i=1}^n p_i(t) I_i(t) & (1) \\
 p_i(t) &= a_i + b_i(t - t_i) + c_i(t - t_i)^2 + d_i(t - t_i)^3 \\
 I_i(t) &= \begin{cases} 1 & \text{if } t > t_i \\ 0 & \text{otherwise} \end{cases}
 \end{aligned}$$

For  $(n + 1)$  knot points  $0 = t_1 < t_2 < \dots < t_{n+1} = \text{max maturity}$ . If we impose continuity and smoothness constraints:

$$\begin{aligned}
 f(t_i^-) &= f(t_i^+) \\
 f'(t_i^-) &= f'(t_i^+) \\
 f''(t_i^-) &= f''(t_i^+).
 \end{aligned}$$

Then (1) is reduced to

$$f(t) = a_1 + b_1(t - t_1) + c_1(t - t_1)^2 + \sum_{i=1}^n d_i(t - t_1)^3 I_i(t) \quad (2)$$

which consists of  $n + 3$  unknowns. In Question 6b in the assignment we were also asked to impose a left end clamped boundary  $f'(t_1) = 0$  which further reduces (2) to  $b_1 = 0$  which gives us

$$f(t) = a_1 + c_1(t - t_1)^2 + \sum_{i=1}^n d_i(t - t_1)^3 I_i(t) \implies n + 2 \text{ unknowns} \quad (3)$$

Now we say we have a set of  $J$  coupon-paying bonds. Let  $T_1, T_2, \dots, T_k$  denote the set of all payment times all bonds in the data set. Let  $C_{j,k}$  be the payment bond  $J$  at time  $T_k$ . From a no-arbitrage argument the price of bond  $J$  is equal to the present value of discounted cash flows.

$$p_j = \sum_{k=1}^K C_{j,k} \quad \text{where } p_j \text{ is the current price for bond } j \quad (4)$$

When we sub (3)  $\rightarrow$  (4) we get an equation for each bond  $j$

$$\begin{aligned}
 p_j &= \sum_{k=1}^K C_{j,k} \left[ a_1 + c_1(T_k - t_1)^2 + \underbrace{\sum_{i=1}^n d_i(T_k - t_i)^3 I_i(T_k)}_{f(T_k)} \right] \\
 \implies p_j &= a_1 \sum_{k=1}^K C_{j,k} + c_1 \sum_{k=1}^K C_{j,k} (T_k - t_1)^2 + \sum_{i=1}^n d_i \left( \sum_{k=1}^K C_{j,k} (T_k - t_i)^3 I_i(T_k) \right) & (5)
 \end{aligned}$$

For  $(n + 1)$  knots, equation (5) has  $(n + 2)$  unknowns.

**Definition 17.20:** Assume we start with a function  $y = f(x)$  and take data points from it to build an interpolating polynomial  $p(x)$ . We define the **interpolation error** at  $x$  as  $f(x) - p(x)$ .

**Theorem 17.21:** Assume  $p(x)$  is the degree  $n$  or less interpolating polynomial fitting the  $(n + 1)$  points

$$(x_0, y_0), \dots, (x_n, y_n) \quad x_0 < x_1 < \dots < x_n.$$

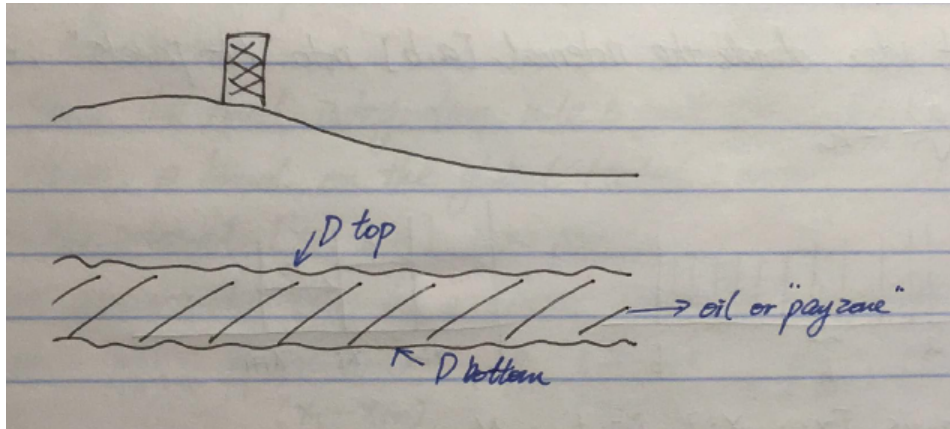
Then the interpolation error is

$$f(x) - p(x) = \frac{f^{(n+1)}(\alpha)}{(n+1)!} (x - x_0)(x - x_1) \cdots (x - x_n) \quad \text{where } \alpha \in [x_0, x_n] \text{ a random number.}$$

At the points  $x_0, \dots, x_n$  the error is zero.

**Numerical Integration 17.22:**

**Motivation:** Suppose we are given the following seismic data. For the cross-section of an oil reservoir:

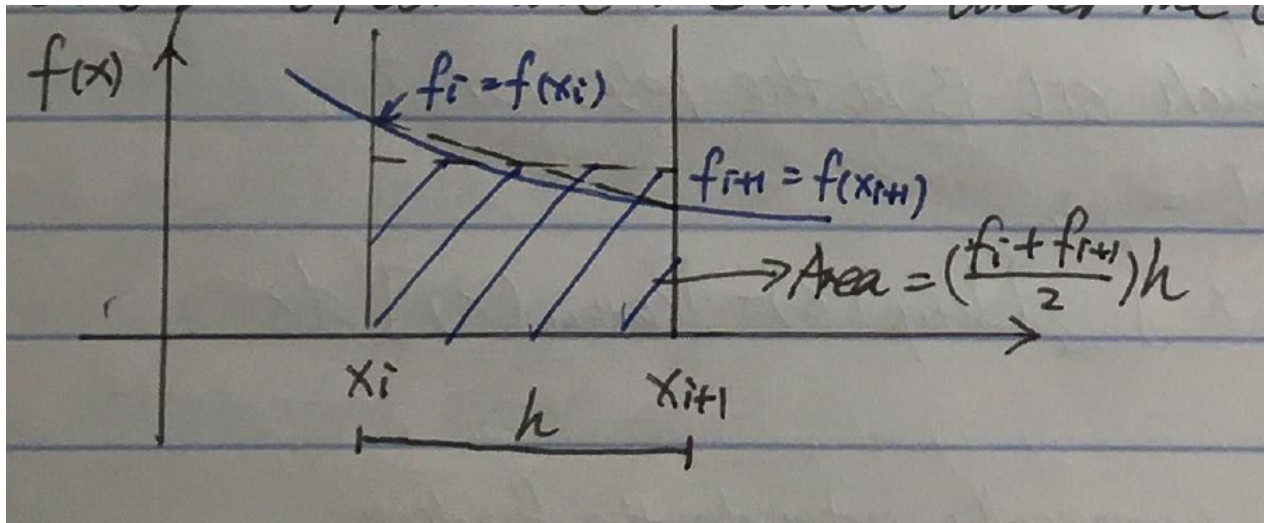


In other words, given the depth to the top and bottom of the reservoir, at discrete points, how much oil is in the reservoir? We have the volume of oil as

$$\text{width} \times \int_{x=a}^{x=b} [D_{\text{top}}(x) - D_{\text{bottom}}(x)] dx$$

i.e, we have to numerically integrate the function  $f(x) = D_{\text{top}}(x) - D_{\text{bottom}}(x)$ . i.e,  $\int_a^b f(x) dx$ .

**Definition 17.23:** There is no closed form expression for  $f(x)$ , since it is experimented data. In general, even if  $f(x)$  can be expressed analytically, a “closed form” solution for the integration is unlikely to exist. Recall that the integral of a function  $f(x)$  is just the area under the curve. Simplest idea is to divide the interval  $[a, b]$  into  $N$ -panels, each with length  $\frac{(b-a)}{N} = h$ . For each panel, estimate the area under the curve  $f(x)$  for this panel as follows:



So we have

$$\begin{aligned} \int_a^b f(x) dx &= \sum_{i=0}^{h-1} \frac{h}{2} [f_i + f_{i+1}] \\ &= \frac{h}{2} [f_0 + 2f_1 + 2f_2 + \dots + 2f_{N-1} + f_N]. \end{aligned} \quad (\text{i})$$

This method is called the **Trapezoidal rule**.

**Definition 17.24:** Note that (i) can be written as

$$\int_a^b f(x) = \frac{h}{2} [f_0 + 2f_1 + 2f_2 + \dots + 2f_{N-1} + f_N] = \sum_{i=0}^N \omega_i f_i \quad \text{where } \omega_i \text{ are the integration rates.} \quad (\text{ii})$$

An expansion of the form (ii) is called the **quadratic rule**.

March 12, 2018

## Error for Trapezoidal Method

On interval  $[x_i, x_{i+1}]$  (one panel) approximate  $f(x)$  by Linear Lagrange interpolating polynomial

$$f(x) = \left. \begin{aligned} & f_i \left( \frac{x - x_{i+1}}{x_i - x_{i+1}} \right) + f_{i+1} \left( \frac{x - x_i}{x_{i+1} - x_i} \right) \\ & + f''(\xi_x) \frac{(x - x_i)(x - x_{i+1})}{2} \end{aligned} \right\} \textcircled{3}$$

$$x \in [x_i, x_{i+1}], \xi_x \in [x_i, x_{i+1}]$$

The unknown point  $\xi_x$  depends continuously on  $x$ .

Let  $f(x) = P_L(x) + E_L(x)$  where  $P_L(x) = f_i \frac{(x - x_{i+1})}{(x_i - x_{i+1})} + f_{i+1} \frac{(x - x_i)}{(x_{i+1} - x_i)}$

$$E_L(x) = f''(\xi_x) \frac{(x - x_i)(x - x_{i+1})}{2}$$

So

$$\int_{x_i}^{x_{i+1}} f(x) dx = \int_{x_i}^{x_{i+1}} P_L(x) dx + \int_{x_i}^{x_{i+1}} E_L(x) dx \quad \textcircled{4}$$

We have

$$\int_{x_i}^{x_{i+1}} P_L(x) dx = \int_{x_i}^{x_{i+1}} \left[ f_i \frac{(x - x_{i+1})}{(x_i - x_{i+1})} + f_{i+1} \frac{(x - x_i)}{(x_{i+1} - x_i)} \right] dx = \frac{h}{2} [f_i + f_{i+1}]$$

$\textcircled{5}$

$\textcircled{1}$

$$\int_{x_i}^{x_{i+1}} E_L(x) dx = \frac{1}{2} \int_{x_i}^{x_{i+1}} f''(\xi(x)) (x-x_i)(x-x_{i+1}) dx$$

$$= \frac{f''(\beta)}{2} \int_{x_i}^{x_{i+1}} (x-x_i)(x-x_{i+1}) dx$$

$$= \frac{f''(\beta)}{2} \int_0^h (u)(u-h) du$$

$$= -\frac{f''(\beta)}{12} h^3 \quad \text{--- (6)}$$

By mean value  
thm for integrals,  
since integrand does  
not change sign.  
 $\beta \in [x_i, x_{i+1}]$

$$\Rightarrow \left| \int_{x_i}^{x_{i+1}} E_L(x) dx \right| \leq M_i \frac{h^3}{12} ; M_i = \max_{x \in [x_i, x_{i+1}]} |f''(x)|$$

So (4), (5), (6) gives us the single panel form for the trapezoidal rule.

$$\int_{x_i}^{x_{i+1}} f(x) dx = \frac{h}{2} [f_i + f_{i+1}] + E_i$$

$$\text{where } |E_i| \leq M_i \frac{h^3}{12} ; M_i = \max_{x \in [x_i, x_{i+1}]} |f''(x)| \quad \text{--- (7)}$$

(7) Gives us the local error of the trapezoidal rule  
i.e. the error over one panel.

What is the ~~get~~ global error?



$$\int_a^b f(x) dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x) dx$$

$$= \sum_{i=0}^{N-1} \left( \frac{f_i + f_{i+1}}{2} \right) h + \sum_{i=0}^{N-1} B_i$$

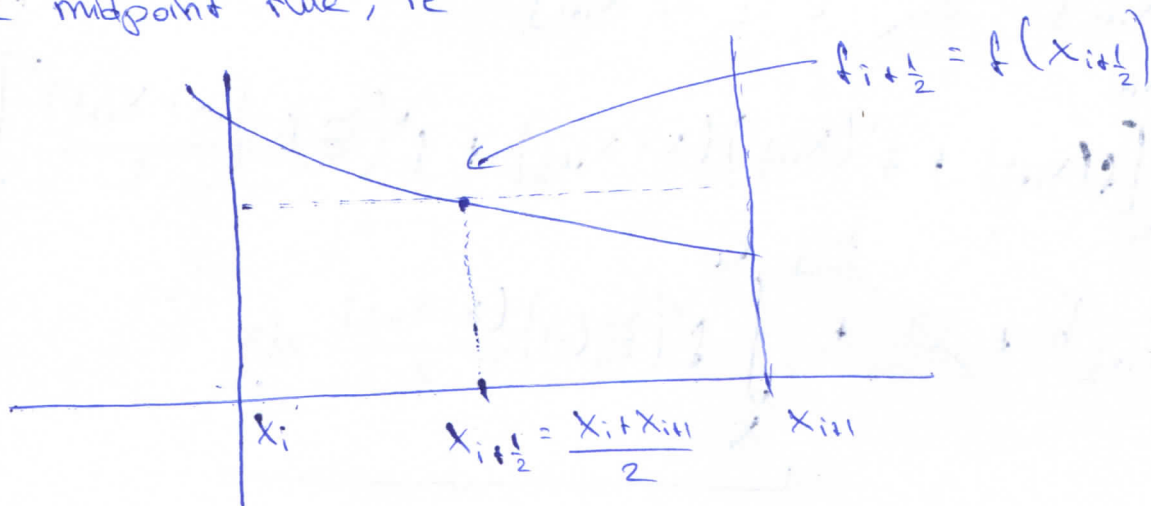
$$\text{So } |\text{Error Global}| = \sum_{i=0}^{N-1} B_i \leq \sum_{i=0}^{N-1} M_i \frac{h^3}{12} \leq \frac{Mh^2}{12} \sum_{i=0}^{N-1} h$$

$$\text{where } M = \max_i M_i = \max_{x \in [a,b]} |f''(x)|$$

$$\Rightarrow |\text{Error Global}| \leq \frac{Mh^2}{12} (b-a) = O(h^2)$$

coefficient depends on 2<sup>nd</sup> derivative ( $f''(x)$ ).

Now suppose we use an even simpler function (quadrature rule) the midpoint rule, ie



$$\int_{x_i}^{x_{i+1}} f(x) dx \sim h f_{i+1/2}$$

This is a degree zero interpolating polynomial.

So what is a legitimate guess ~~for~~ for the local error global error of this quadrature rule?

- zero-order Lagrange interpolation

$$\Rightarrow \text{interpolation error} = O(x - x_{i+\frac{1}{2}}) = O(h)$$

- Integrate this term  $\sim O(h^2)$

$$\text{Global error} = \sum \text{local errors}$$

$$= \sum O(h^2) = O(h)$$

↳ this would be inaccurate

- In order to get the more accurate estimate here, we need to use Taylor Series.

$$f(x) = f(x_{i+\frac{1}{2}}) + f'(x_{i+\frac{1}{2}})(x - x_{i+\frac{1}{2}}) + f''(\xi_x) \frac{(x - x_{i+\frac{1}{2}})^2}{2}$$

$$x \in [x_i, x_{i+1}], \quad \xi_x \in [x_i, x_{i+1}]$$

$$\begin{aligned} \int_{x_i}^{x_{i+1}} f(x) dx &\approx \int_{x_i}^{x_{i+1}} \left[ f(x_{i+\frac{1}{2}}) + f'(x_{i+\frac{1}{2}})(x - x_{i+\frac{1}{2}}) + f''(\xi(x)) \frac{(x - x_{i+\frac{1}{2}})^2}{2} \right] dx \\ &= f_{i+\frac{1}{2}} h + \underbrace{\int_{x_i}^{x_{i+1}} f''(\xi(x)) \frac{(x - x_{i+\frac{1}{2}})^2}{2} dx}_{E_i} \end{aligned}$$

*lucky break* →

$$\text{then } |E_i| = \left| \int_{x_i}^{x_{i+1}} f''(\xi(x)) \frac{(x - x_{i+\frac{1}{2}})^2}{2} dx \right| \leq M_i \int_{x_i}^{x_{i+1}} \frac{(x - x_{i+\frac{1}{2}})^2}{2} dx = M_i \int_{-h/2}^{h/2} \frac{u^2}{2} du$$

$$\text{where } M_i = \max_{x \in [x_i, x_{i+1}]} |f''(x)|$$

$$= M_i \frac{h^3}{24}$$

Usual argument for the global error:

$$|\text{Global Error}| = \sum \text{local errors}$$

$$\leq \sum O(h^3)$$

$$= O(h^2)$$

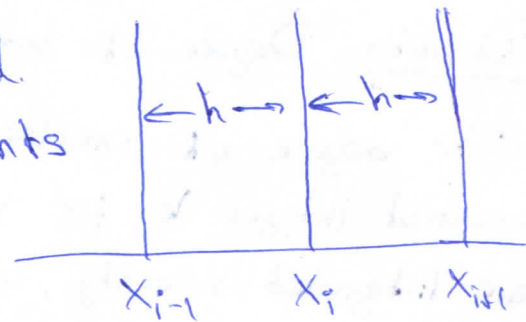
Coefficient depends on  $f''(x)$

$\Rightarrow$  Trapezoidal rule, mid point rule both have global error  $= O(h^2)$ ,

Coefficient depends on  $\max_{x \in [a,b]} |f''(x)|$ .

### Simpson's Rule

- Use a quadratic interpolating polynomial.
- two panels, need three interpolating points



$$P_2(x) = f_{i-1} \frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})}$$

$$+ f_i \frac{(x-x_{i-1})(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})}$$

$$+ f_{i+1} \frac{(x-x_{i-1})(x-x_i)}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)}$$

$$\int_{x_i}^{x_{i+1}} P_2(x) dx = \frac{h}{3} [f_{i-1} + 4f_i + f_{i+1}]$$

$$\int_a^b f(x) dx = \sum \int_{x_{i-1}}^{x_i} f(x) dx \approx \frac{h}{3} [f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \dots + 2f_{N-2} + 4f_{N-1} + f_N]$$

What is the error of this ~~quadratic~~ quadrature rule?

2<sup>nd</sup> order interpolating polynomial

→ error of interpolating polynomial  $O(x - x_i)^3$

→ integrate  $O(h^4)$

→ global error =  $O(h^3)$  ← wrong

Again, requires tedious work with Taylor Series. Global error of Simpson's Rule

$$|\text{Error}_{\text{Global}}| \leq \frac{Mh^4(b-a)}{180} = O(h^4); \quad M = \max_{x \in [a,b]} |f^{(4)}(x)|$$

Definition: Degree of precision.

The degree of precision of a numerical integration method is the greatest integer  $k$  for which all degree  $k$  or less polynomials are integrated exactly by the method.

E.g. error of the trapezoidal rule  $\leq \frac{Mh^2}{12}(b-a)$ ,  $M = \max_{x \in [a,b]} |f''(x)|$

⇒ if integrating function,  $f(x)$ , is of degree 1 or less, the error will be zero

⇒ degree of precision = 1 for trapezoidal method

• What is the degree of precision for Simpson's Rule?

Since  $f^{(4)}(x) = 0$  for degree 3 or less polynomials, the

degree of precision = 3.

For next lecture:

$$\int_{x_i}^{x_{i+3}} f_i(x) \approx \frac{3h}{8} [f_i + 3f_{i+1} + 3f_{i+2} + f_{i+3}]$$



Degree of Precision

Quadratic interpolating polynomial  $\rightarrow$  Simpson's rule  
 $\rightarrow$  degree of precision = 3

If we use a cubic interpolating polynomial, we obtain the following ~~quadratic~~ quadrature rule (called Simpson's 3/8 Rule)

$$\int_{x_i}^{x_{i+3}} f(x) \approx \frac{3h}{8} [f_i + 3f_{i+1} + 3f_{i+2} + f_{i+3}]$$

How can we determine the degree of precision for this rule?

Since  $\{1, (x-x_i), (x-x_i)^2, \dots\}$  forms a basis for polynomials, it suffices to test these in succession

E.g.  $ax^2 + bx + c = a'(x-x_i)^2 + b'(x-x_i) + c'$

$$\Rightarrow \int (ax^2 + bx + c) dx = \underbrace{a' \int (x-x_i)^2 dx + b' \int (x-x_i) dx + c' \int 1 dx}$$

If these are exact,

$\int (ax^2 + bx + c) dx$  is exact.

For Simpson's 3/8 rule, if we test  $f(x) = (x - x_i)^3$ ;  $\int_{x_i}^{x_{i+3}} f(x) dx$

$$\text{Rule} = \frac{3h}{8} \left[ 0 + 3h^3 + 3 \cdot (2h)^3 + (3h)^3 \right] = \frac{81}{4} h^4$$

Using sub  $v = x - x_i$

$$\text{Exact} = \int_0^{3h} u^3 dv = \frac{u^4}{4} \Big|_0^{3h} = \frac{81}{4} h^4$$

Similarly we can verify equality for  $1, (x - x_i), (x - x_i)^2$

But for  $f(x) = (x - x_i)^4$  we find

$$\text{Rule} = \frac{3h}{8} \left[ 0 + 3 \cdot h^4 + 3(2h)^4 + (3h)^4 \right] = \frac{99}{2} h^5$$

$$\text{Exact} = \int_0^{3h} u^4 dv = \frac{v^5}{5} \Big|_0^{3h} = \frac{243}{5} h^5 \neq \frac{99}{2} h^5$$

$\therefore$  Degree of precision for Simpson's 3/8 rule is 3.

### Summary of Results

Quadrature Rule	Order of Interpolating Polynomial	Global Error	Degree of Precision
Midpoint	0	$O(h^2)$	1
Trapezoid	1	$O(h^2)$	1
Simpson's	2	$O(h^4)$	3
Simpson's 3/8	3	$O(h^4)$	3
	4	$O(h^6)$	5
	5	$O(h^6)$	5
	⋮	⋮	⋮
	⋮	⋮	⋮

## Gaussian Quadrature

Previously, we have specified that the function  $f(x)$  was given at equally spaced pts,  $x_i, x_{i+1}, \dots$  with  $(x_{i+1} - x_i) = \text{const} = h$

For ex. Trapezoid rule involves 2 terms

$$\int_{x_i}^{x_{i+1}} f(x) dx \approx \frac{h}{2} f_i + \frac{h}{2} f_{i+1} \quad (1)$$

evaluated at  $x_i, x_{i+1}$

$$= w_i f_i + w_{i+1} f_{i+1}$$

(1) is exact for degree 1 and lower polynomials

Can we do better (with same number of evaluations)?

Suppose we remove the requirement that the function be evaluated at predetermined  $x$ -values.

→ a two term quadratic rule would contain four parameters

$$\{w_i, w_{i+1}, x_i, x_{i+1}\}$$

→ should be able to determine a quadrature rule which is exact for polynomials up to degree 3.

This idea is called Gaussian Quadrature.

Derivation: Two term case. Suppose we have formula

$$\int_{-1}^{+1} f(x) dx \approx a f(x_1) + b f(x_2) \quad (2)$$

4 parameters  $\{a, b, x_1, x_2\}$ . We want to make (2) correct for any

polynomial of the form  $P(x) = \alpha + \beta x + \gamma x^2 + \delta x^3$  (3)

If we determine  $\{a, b, x_1, x_2\}$  such that (2) is exact for each polynomial

$$f = x^3 \quad f = x^2 \quad f = x \quad f = 1 \quad (4)$$

Independently, then since (3) is a linear combination of type (4) then it'll be exact for any polynomial of type (3).

Let  $f(x) = x^3$  in (2)

$$\int_{-1}^1 x^3 dx = 0 = ax_1^3 + bx_2^3 \quad \text{--- (i)}$$

Let  $f(x) = x^2$  in (2) :  $\int_{-1}^1 x^2 dx = \frac{2}{3} = ax_1^2 + bx_2^2 \quad \text{--- (ii)}$

Let  $f(x) = x$  in (2) :  $\int_{-1}^1 x dx = 0 = ax_1 + bx_2 \quad \text{--- (iii)}$

Let  $f(x) = 1$  in (2) :  $\int_{-1}^1 1 dx = 2 = a + b \quad \text{--- (iv)}$

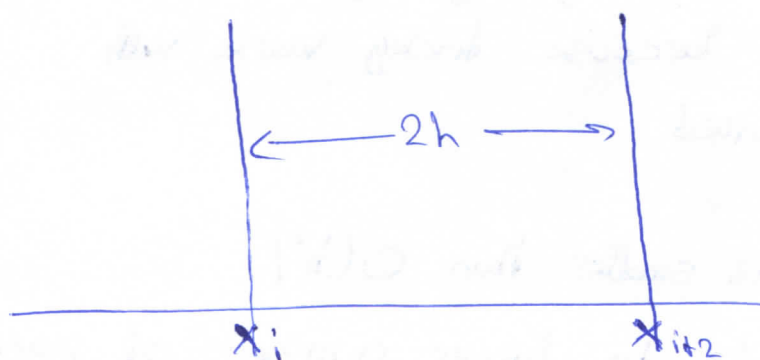
From (i) - (iv) we have  $a = 1 = b$  and  $x_{1,2} = \pm \sqrt{\frac{1}{3}}$ .

Therefore  $\int_{-1}^1 f(x) dx \approx f\left(-\sqrt{\frac{1}{3}} = -0.5773\right) + f\left(\frac{1}{\sqrt{3}} = 0.5773\right) \quad (5)$

(5) is exact for any polynomial of degree 3 or lower



We can transform the limits of integration to arbitrary panel



Note: Using  $2h$  width so that the number of function evaluations  $\sim$  midpoint/trapezoidal rules for given  $h$

Let  $x = x_i + h(1+t)$ . So when  $t = -1$ :  $x = x_i$

$t = 1$ :  $x = x_i + 2h$

We have  $dx = h dt$ . Thus

$$\int_{x_i}^{x_i+2h} f(x) dx = \int_{-1}^1 f(x_i + h(1+t)) dt$$

$$= h \int_{-1}^1 g(t) dt; \quad g(t) = f(x_i + h(1+t))$$

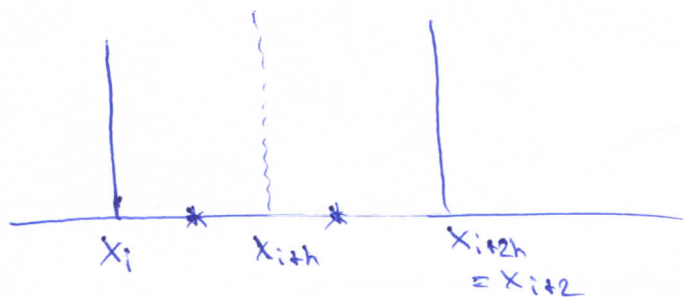
Since  $\int_{-1}^1 g(t) dt \approx g\left(-\frac{1}{\sqrt{3}}\right) + g\left(\frac{1}{\sqrt{3}}\right)$

(6)

Sub in (6) and get

$$\int_{x_i}^{x_i+2h} f(x) dx \approx h \left[ f\left(x_i + h\left(1 - \frac{1}{\sqrt{3}}\right)\right) + f\left(x_i + h\left(1 + \frac{1}{\sqrt{3}}\right)\right) \right]$$

$\approx 0.4$                        $\approx 1.6$



This ~~exam~~ course does not cover getting higher order Gaussian Quadrature rules because tricky work with orthogonal polynomials is required.

Notes As  $h \rightarrow 0$ ,  $O(h^4)$  is smaller than  $O(h^2)$

Also as  $h \rightarrow 0$  corresponds to larger number of panels.

End of integration.

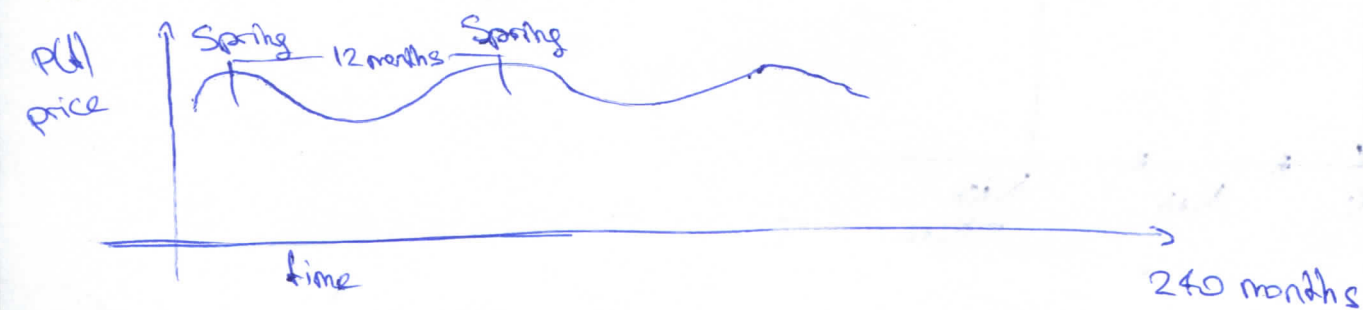
## Discrete Fourier Transform

- method for transforming information from one "form" to another (eg time to freq.)

- applications: Image compression JPEG  
Sound compression MPEG  
medical imaging  
digital communications etc

Motivation: Suppose we have data which represents the price of orange juice (monthly avg spot prices) for last 240 months (20 years)

- We would expect the price of OJ to be low in fall (after harvest) and high in spring



A simple model for this  
 $P(t) = A \sin \left[ \frac{2\pi t}{T} \right]$  → in months

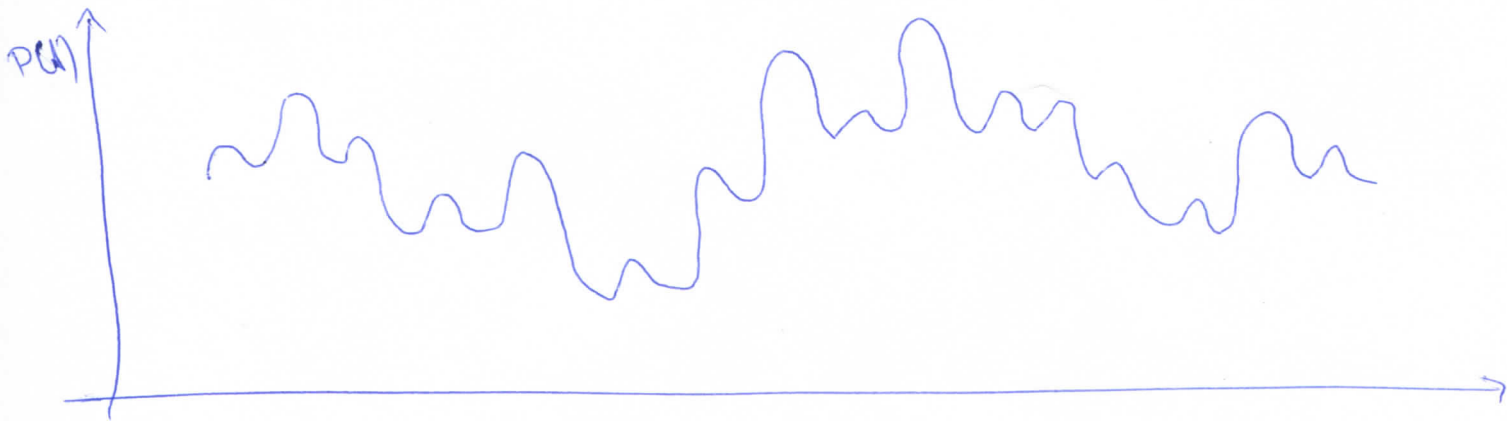
Note:  $\sin \left[ \frac{2\pi(t+T)}{T} \right] = \sin \left[ \frac{2\pi t}{T} + 2\pi \right]$   
 $= \sin \left[ \frac{2\pi t}{T} \right]$   
→  $T = \text{period} = 12$

$P(t)$  is a periodic function, with period  $T=12$

ie.  $P(t+T) = P(t)$

But there may be other effects El nino, sunspot activity, imports, etc.

Actual time series for observed  $P(t)$



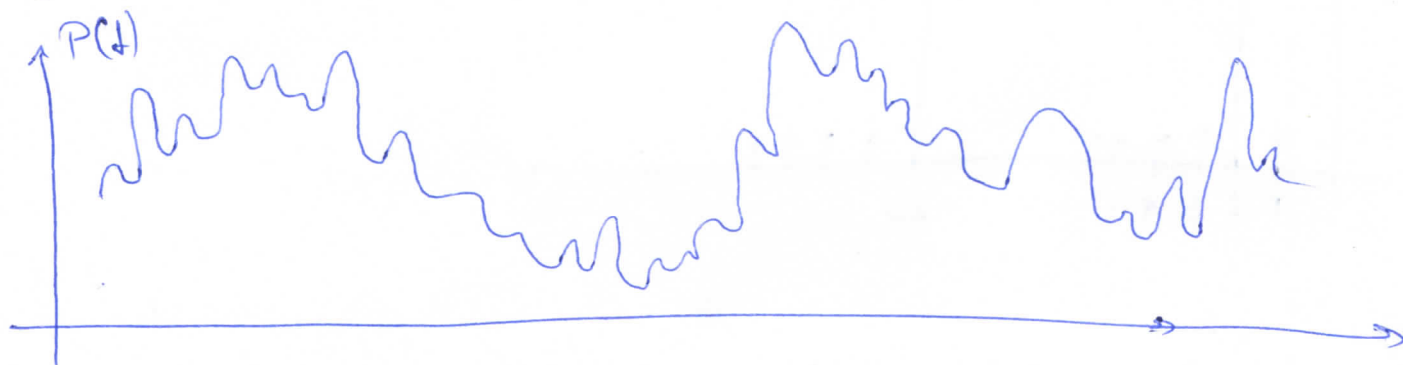
We would like to analyze this data, and identify the cycles or periods embedded in this data.

March 19, 2018

## Motivation: Fourier Approximation

Price of orange juice,  $P(t)$ , may contain several "cyclic" effects, e.g. annual harvest, El Niño, sunspots etc.

Time Series for observed  $P(t)$  might look like



We would like to analyze this data, and identify the cycles or periods in this data.

Represent the function, data, by Fourier Series expansion (data for  $T = 240$  months)

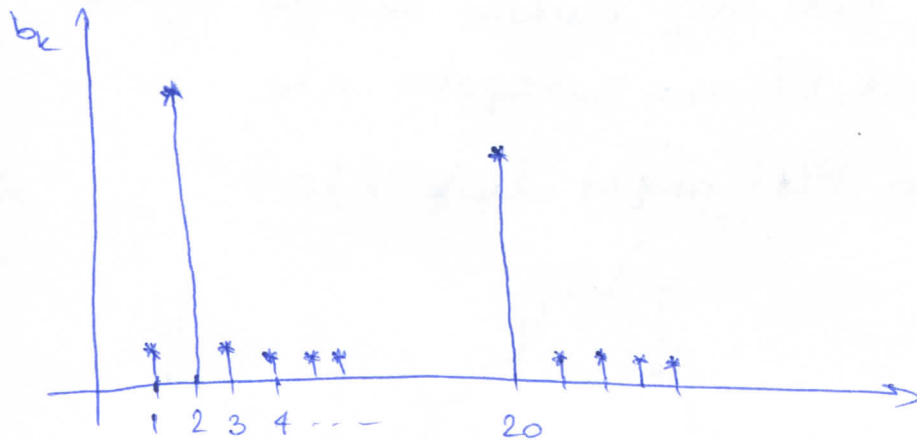
$$P(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi k t}{T}\right) + \sum_{k=1}^{\infty} b_k \sin\left(\frac{2\pi k t}{T}\right)$$

$T = 240$  months

Suppose we find  $a_k, b_k$  that best fits the data. (Describe how to do this later)

$$a_k = 0$$

Plot of  $b_k$  gives



This plot shows that

$$\begin{aligned} P(t) &\sim b_2 \sin\left(\frac{2\pi \cdot 2t}{240}\right) + b_{20} \sin\left(\frac{2\pi \cdot 20t}{240}\right) \\ &= b_2 \sin\left(\frac{2\pi t}{120}\right) + b_{20} \sin\left(\frac{2\pi t}{12}\right) \end{aligned}$$

Suggest that there are two main cycles in the data. The  $b_2$  term represents a pattern which repeats every 120 months (10 years).  
→ sunspots?

The  $b_{20}$  term → pattern which ~~repeats~~ repeats every 12 months  
→ usual yearly harvest



# Fourier Series

Given some time series  $P(t)$ ,  $0 \leq t \leq T$  (assume periodic with period  $T$ ). We can represent this data by Fourier series.

$$P(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi k t}{T}\right) + \sum_{k=1}^{\infty} b_k \sin\left(\frac{2\pi k t}{T}\right)$$

This gives us information about the patterns in the data by looking at the size of the coefficients  $a_k, b_k$ . e.g.  $b_k$  represents "size" or "amplitude" of the pattern with period  $\left(\frac{T}{k}\right)$  or frequency  $\left(\frac{k}{T}\right)$

## Review: Inner Product

- Generalization of dot product for functions.

e.g.  $\langle \vec{x}, \vec{y} \rangle = \sum_i x_i y_i$  ← dot product of two vectors

$\langle f(t), g(t) \rangle = \int_a^b f(t) g(t) dt$  ← inner product of two functions over domain  $[a, b]$

If we have an orthogonal basis

e.g.  $\vec{x} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ ,  $\vec{y} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}$

then any  $\vec{v}$  in the corresponding space ( $\mathbb{R}^2$ ) can be written as

$$\vec{v} = \alpha \vec{x} + \beta \vec{y}$$

Because  $\vec{x}, \vec{y}$  are orthogonal, we can readily determine  $\alpha$  and  $\beta$  by projection

$$\text{e.g. } \vec{v} = \begin{bmatrix} 6 \\ 20 \end{bmatrix}$$

$$\alpha = \frac{\langle \vec{v}, \vec{x} \rangle}{\langle \vec{x}, \vec{x} \rangle} = \frac{6 \cdot 2}{2 \cdot 2} = 3 \quad \text{and} \quad \beta = \frac{\langle \vec{v}, \vec{y} \rangle}{\langle \vec{y}, \vec{y} \rangle} = \frac{20 \cdot 5}{5 \cdot 5} = 4$$

### Fourier Series Coefficients

Fourier Series representation of  $f(t)$

$$f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi k t}{T}\right) + \sum_{k=1}^{\infty} b_k \sin\left(\frac{2\pi k t}{T}\right)$$

$\left\{ 1, \cos\left(\frac{2\pi k t}{T}\right), \sin\left(\frac{2\pi k t}{T}\right) \right\}; k=1, \dots, \infty$  forms an orthogonal

basis.

$$\text{i.e. } \left\langle 1, \cos\left(\frac{2\pi k t}{T}\right) \right\rangle = \int_0^T 1 \cdot \cos\left(\frac{2\pi k t}{T}\right) dt = 0; k \geq 1$$

$$\left\langle 1, \sin\left(\frac{2\pi k t}{T}\right) \right\rangle = \int_0^T 1 \cdot \sin\left(\frac{2\pi k t}{T}\right) dt = 0; k \geq 1$$

$$\left\langle \cos\left(\frac{2\pi k t}{T}\right), \sin\left(\frac{2\pi l t}{T}\right) \right\rangle = \int_0^T \cos\left(\frac{2\pi k t}{T}\right) \sin\left(\frac{2\pi l t}{T}\right) dt = 0; k, l \geq 1$$

$$\left\langle \cos\left(\frac{2\pi k t}{T}\right), \cos\left(\frac{2\pi l t}{T}\right) \right\rangle = 0; k \neq l$$

$$\left\langle \sin\left(\frac{2\pi k t}{T}\right), \sin\left(\frac{2\pi l t}{T}\right) \right\rangle = 0; k \neq l$$

Thus  $a_k, b_k$  can be determined by projection

$$a_k = \frac{\langle f(t), \cos\left(\frac{2\pi kt}{T}\right) \rangle}{\langle \cos\left(\frac{2\pi kt}{T}\right), \cos\left(\frac{2\pi kt}{T}\right) \rangle} = \frac{2}{T} \int_0^T f(t) \cos\left(\frac{2\pi kt}{T}\right) dt$$

$$b_k = \frac{\langle f(t), \sin\left(\frac{2\pi kt}{T}\right) \rangle}{\langle \sin\left(\frac{2\pi kt}{T}\right), \sin\left(\frac{2\pi kt}{T}\right) \rangle} = \frac{2}{T} \int_0^T f(t) \sin\left(\frac{2\pi kt}{T}\right) dt$$

$$\frac{a_0}{2} = \frac{\langle f(t), 1 \rangle}{\langle 1, 1 \rangle} = \frac{1}{T} \int_0^T f(t) dt$$

The space of functions that spans the basis

$$\left\{ 1, \cos\left(\frac{2\pi kt}{T}\right), \sin\left(\frac{2\pi kt}{T}\right) \right\}; k \geq 1$$

is  $V = \left\{ f(x); \langle f(x), f(x) \rangle < \infty \right\}$ , i.e.  $\int_a^b [f(x)]^2 dx < \infty$

Such functions are also referred as square integrable functions.

### Comparison with Taylor Series

Recall Taylor series expresses  $f(t)$  as an infinite sum of polynomials  $\rightarrow$  local approximation

$$f(t) = \sum_{k=0}^{\infty} w_k (x-a)^k; w_k = \frac{f^{(k)}(a)}{k!}$$

Similarly, Fourier series is just expressing  $f(t)$  as an infinite sum of cosines/sines.  $\rightarrow$  global



# Differences

- Taylor Series requires ~~local~~ knowledge of  $f$  around arbitrary small neighborhood,  $t = a$
- Fourier  $\rightarrow$  knowing ~~the~~  $f$  on its whole domain, e.g.  $[0, \pi]$   
i.e. Taylor series is "local"  
Fourier ~~series~~ series is "global".
- Taylor  $\rightarrow$  requires infinitely many derivatives at one point.
- Fourier  $\rightarrow$  requires (square) integrable functions.
- In practice we approximate the series using a finite number of terms  $\Rightarrow$

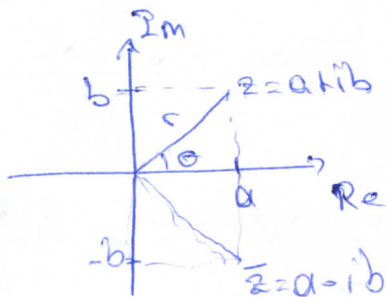
Taylor  $\rightarrow$  the error is small near  $t = a$  but may be large at a distant pt

Fourier  $\rightarrow$  the error is distributed along the domain of  $f$  in a "least-squares" sense.

## Digression: Complex Numbers Review

- trig arithmetic can be simplified by using complex numbers.

Let  $i = \sqrt{-1}$ , then the complex numbers has the form  $z = a + ib$



$$\text{magnitude: } |z| = \sqrt{a^2 + b^2} = r$$

$$\text{conjugate: } \bar{z} = a - ib$$

Polar representation of  $z$  is

$$z = r e^{i\theta} \Rightarrow \bar{z} = r e^{-i\theta}$$

Euler's Identity:  $e^{i\theta} = \cos\theta + i\sin\theta = \text{cis } \theta$ .

### Complex Form of Fourier Series

From Euler's Identity we have  $e^{i\theta} = \cos\theta + i\sin\theta$

$$e^{-i\theta} = \cos\theta - i\sin\theta$$

$$\frac{e^{i\theta} + e^{-i\theta}}{2} = \cos\theta$$

$$\frac{e^{i\theta} - e^{-i\theta}}{2i} = \sin\theta = -\frac{i}{2} (e^{i\theta} - e^{-i\theta})$$

$$\text{Then } f(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left[ a_k \cos\left(\frac{2\pi k t}{T}\right) + b_k \sin\left(\frac{2\pi k t}{T}\right) \right]$$

$$= \frac{a_0}{2} + \sum_{k=1}^{\infty} \left[ \frac{a_k}{2} \left( e^{i\frac{2\pi k t}{T}} + e^{-i\frac{2\pi k t}{T}} \right) - i b_k \left( e^{i\frac{2\pi k t}{T}} - e^{-i\frac{2\pi k t}{T}} \right) \right]$$

$$= \frac{a_0}{2} + \sum_{k=1}^{\infty} \left[ \left( \frac{a_k - i b_k}{2} \right) e^{i\frac{2\pi k t}{T}} + \left( \frac{a_k + i b_k}{2} \right) e^{-i\frac{2\pi k t}{T}} \right]$$

$$\stackrel{\text{LWS}}{=} c_0 + \sum_{k=1}^{\infty} \underbrace{c_k}_{c_k} e^{i2\pi k t / T} + \sum_{k=1}^{\infty} \underbrace{c_{-k}}_{-i2\pi k t / T} e^{-i2\pi k t / T}$$

$$= \sum_{k=-\infty}^{\infty} c_k e^{i2\pi k t} \quad \text{where } c_k = \begin{cases} \frac{a_k + i b_k}{2} & \text{when } k < 0 \\ \frac{a_0}{2} & \text{when } k = 0 \\ \frac{a_k - i b_k}{2} & \text{when } k > 0 \end{cases}$$

## Complex Form of Fourier Series

$$f(t) = \sum_{k=-\infty}^{\infty} C_k e^{\frac{i2\pi kt}{T}} \quad \text{where } C_0 = a_0/2$$

$$C_k = \begin{cases} \frac{a_k - ib_k}{2}, & k > 0 \\ \frac{a_k + ib_k}{2}, & k < 0 \end{cases}$$

$a_k$  and  $b_k$  are the coefficients of real Fourier Series.

### Observations:

1)  $C_k \in \mathbb{C}$ .

2) If  $f$  is a real function then  $a_k, b_k \in \mathbb{R}$ .

Then  $C_{-k} = \overline{C_k}$        $\overline{a+ib} = \overline{C_k} = C_{-k} = a - ib$ .

3)  $\left\{ e^{\frac{i2\pi kt}{T}} \right\}, -\infty < k < \infty$  forms an alternative basis for  $V$ , square integrable functions.

Inner product for complex functions is given by

$$\langle f(t), g(t) \rangle = \int_a^b f(t) \overline{g(t)} dt$$

↑  
complex conjugate of  $g(t)$ .

We can verify that basis functions are ~~ortho~~ orthogonal, which are  $\left\{ e^{\frac{i2\pi kt}{T}} \right\}$

because

$$\left\langle e^{\frac{i2\pi kt}{T}}, e^{\frac{i2\pi lt}{T}} \right\rangle = \int_0^T e^{\frac{i2\pi(k-l)t}{T}} dt = \begin{cases} 0 & k \neq l \\ T & k = l \end{cases}$$

Thus we can directly get an expression for  $C_k$  by projection

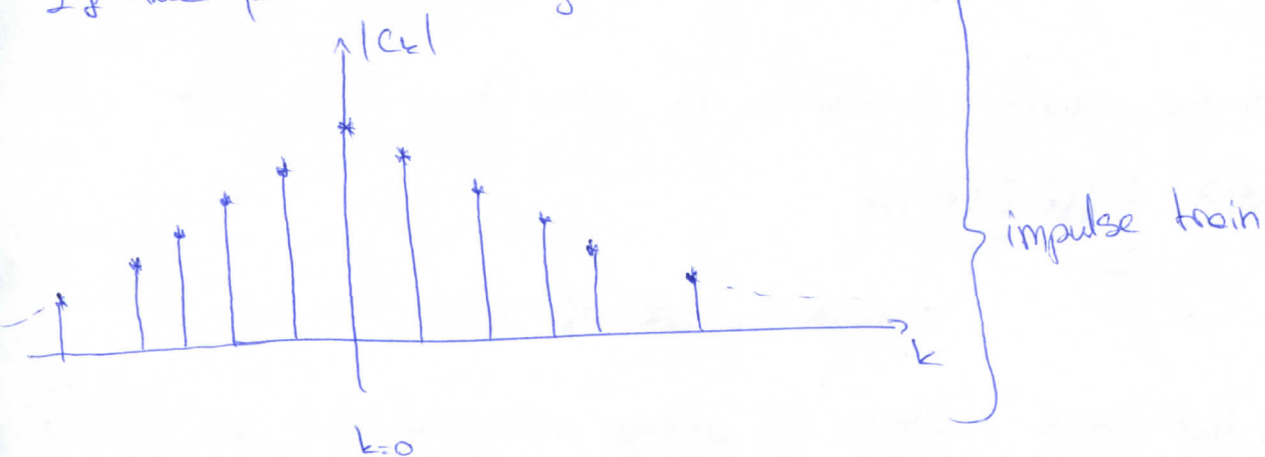
$$C_k = \frac{\langle f(t), e^{\frac{i2\pi kt}{T}} \rangle}{\langle e^{\frac{i2\pi kt}{T}}, e^{\frac{i2\pi kt}{T}} \rangle} = \frac{1}{T} \int_0^T f(x) e^{-\frac{i2\pi kt}{T}} dt$$

$$f(t) = \sum_{k=-\infty}^{\infty} C_k e^{\frac{i2\pi kt}{T}}$$

Magnitude of  $C_k$ , also called the modulus, denoted by  $|C_k|$ , measures the "size" or "amplitude" of the cycle with period  $\left(\frac{T}{k}\right)$  or frequency  $\left(\frac{k}{T}\right)$ . If  $f$  is a real function, then

$$|C_k| = |\bar{C}_k| = |C_{-k}| \quad \text{i.e. } |C_k| \text{ is symmetric about } k=0.$$

If we plot the magnitude of  $C_k$  for typical data



i.e. most of the information is concentrated in the low frequencies

⇒ by ignoring higher frequencies we can compress the signal with minimum loss



# Discrete Fourier Transform

In practice, we do not have an analytic expression for  $f(t)$  the "signal"

— Suppose we have  $N$  samples of the signal spaced  $\Delta t = \frac{T}{N}$  apart (e.g. digital sound recording)

Let the corresponding discrete sample time

$$t_n = n\Delta t = n\frac{T}{N}; \quad n=0, \dots, N-1$$

Denote the input signal  $f(t_n) = f(n\Delta t) = f_n$  so that the input is  $\{f_0, f_1, \dots, f_{N-1}\}$

Note: We assume  $f(t)$  is periodic with period  $T$ . ~~is~~ i.e.  $f(t+T) = f(t)$   
 $\Rightarrow f_{n+N} = f_n$

Since  $f_0 = f_N$ , the value of  $f_N$  is redundant.

Since we have  $N$  points, we can interpolate this data exactly using a truncated Fourier Series, with  $N$  terms.

i.e. 
$$f(t) = \sum_{k=0}^{N-1} c_k e^{\frac{i2\pi kt}{T}}$$

For  $t_n = n\Delta t = n\frac{T}{N}$  we get

$$f_n = f(t_n) = \sum_{k=0}^{N-1} c_k e^{\frac{i2\pi kn}{N}}; \quad n=0, \dots, N-1$$

(1)

Let  $C_k = F_k$ ,  $W_N = e^{\frac{i2\pi}{N}}$  then (1) becomes

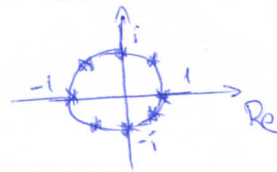
$$f_n = \sum_{k=0}^{N-1} F_k W_N^{kn}$$

~ Digression: Roots of Unity ~

$W_N$  is a special complex number.

Defn: Integer powers of  $W_N$  are called the  $N^{\text{th}}$  roots of unity.

B.g:  $N=8$ ,  $W_8^k = e^{\frac{i2\pi k}{8}} \Rightarrow$



because  $(W_N^k)^N = 1$ .

Then, the distinct  $N^{\text{th}}$  roots of unity are

$$W_N^k = e^{\frac{i2\pi k}{N}} ; 0 \leq k < N.$$

Useful Property

$$\sum_{n=0}^{N-1} W_N^{n(k-l)} = \begin{cases} 0 & k \neq l \\ N & k = l \end{cases} \quad \text{for } k, l = 0, 1, 2, \dots, N-1 \quad (2)$$

Proof of (2):  $\sum_{n=0}^{N-1} r^n = \frac{1-r^N}{1-r}$ ,  $r \neq 1$  (sum of geometric series) (3)

if  $k \neq l$ , then  $r = W_N^{k-l} \neq 1$  (3) becomes

$$\sum_{n=0}^{N-1} W_N^{n(k-l)} = \frac{1 - W_N^{(k-l)N}}{1 - W_N^{(k-l)}} = \frac{1 - (W_N^N)^{(k-l)}}{1 - W_N^{(k-l)}} = 0 \quad (4)$$

If  $k=l$ , then  $W^{k-l} = 1$  and we get  $\sum_{n=0}^{N-1} W^{n(k-l)} = \sum_{n=0}^{N-1} 1$

$$\sum_{n=0}^{N-1} 1 = \underbrace{1+1+\dots+1}_{N \text{ times}} = N \quad (5)$$

(4) and (5) together gives us (2) as required.

We can also write (2) as  $\sum_{n=0}^{N-1} W_N^{n(k-l)} = N \delta_{k,l}$ ,  $\delta_{k,l} = \begin{cases} 1 & \text{if } k=l \\ 0 & \text{if } k \neq l \end{cases}$   
↑  
 Kronecker delta

### Computing Fourier Coefficients

~~$$f_n = \sum_{k=0}^{N-1} f_k W_N^{nk}$$~~

Recall that  $f_n = \sum_{k=0}^{N-1} P_k W_N^{nk}$  (6)

Multiply (6) by  $W^{-nk}$  and sum over  $n$

$$\sum_{n=0}^{N-1} f_n W_N^{-nk} = \sum_{n=0}^{N-1} \sum_{l=0}^{N-1} P_l W_N^{nl} W_N^{-nk}$$

$$= \sum_{l=0}^{N-1} P_l \sum_{n=0}^{N-1} W_N^{n(l-k)} = \sum_{l=0}^{N-1} P_l \sum_{n=0}^{N-1} W_N^{n(l-k)}$$

$$= \sum_{l=0}^{N-1} P_l \sum_{n=0}^{N-1} \delta_{k,l} = \sum_{l=0}^{N-1} P_l N \delta_{k,l} = P_k N \implies P_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W_N^{-nk}$$

We can also arrive at this by noting that (2) implies

$\{W_N^{kn}\}$ ,  $k=0, \dots, N-1$  are orthogonal, i.e.

$$\langle W_N^{kn}, W_N^{ln} \rangle = \sum_{n=0}^{N-1} W_N^{n(k-l)} = \begin{cases} 0 & k \neq l \\ N & k = l \end{cases}$$

So by projection we get

$$F_k = \frac{\langle f_n, W_N^{kn} \rangle}{\langle W_N^{kn}, W_N^{kn} \rangle} = \frac{1}{N} \sum_{n=0}^{N-1} f_n W_N^{-kn}$$



## Discrete Fourier Transform

Given data  $\{f_0, f_1, \dots, f_{N-1}\}$  then

$$F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk} ; W = e^{i \frac{2\pi}{N}} \quad (1)$$

is called forward transform.

$\{F_k\}$  are called Fourier Coefficients or the frequency signal.

Given  $\{F_0, F_1, \dots, F_{N-1}\}$  the operation

$$f_n = \sum_{k=0}^{N-1} F_k W^{nk} \quad (2)$$

is called the reverse or inverse transform.

(1) and (2) is called the DFT pair.

### ~ Periodicity of $F_k$ .

Recall when motivating the DFT, we said that given  $N$  samples of  $f(t)$  we can interpolate data exactly using  $N$  terms of the Fourier Series.

$$\text{ie. } f_n = \sum_{k=0}^{N-1} F_k e^{\frac{i2\pi kn}{N}}$$

Why did we choose terms  $k=0, \dots, N-1$ ? Why not  $k=5, \dots, N+4$

$$k = \lfloor -N/2 \rfloor + 1, \dots, \lfloor N/2 \rfloor$$

As it turns out, for a "discrete" signal  $\{f_n\}$  the Fourier coefficients are periodic with period  $N$ . So our choice doesn't really matter.

•  $k=0, \dots, N-1$  more naturally fits array index conventions, though  $\lfloor -N/2 \rfloor + 1, \dots, \lfloor N/2 \rfloor$  would theoretically be a better choice.

Proposition: Fourier coefficients  $\{P_k\}$  are periodic with period  $N$ , i.e.

$$P_k = P_{k+SN} \quad \text{for integer } S$$

Proof: 
$$P_{k+SN} = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{n(k+SN)} = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{nk} (W^{-Sn})^N = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{nk} = P_k$$

Recall:  $W^N = 1$ .

Note: Fourier coefficients are not necessarily periodic when transforming continuous periodic function  $f(t)$  to its Fourier Series.

i.e. 
$$f(t) = \sum_{k=-\infty}^{\infty} C_k e^{\frac{i2\pi kt}{T}}$$
  
 not periodic

Symmetry Properties:

Note:  $W^{-k} = W^k$  and  $W^N = 1$  and  $W^{N-k} = W^{-k}$ .

$$\overline{P_{N-k}} = \frac{1}{N} \sum_{n=0}^{N-1} \overline{f_n W^{n(N-k)}} = \frac{1}{N} \sum_{n=0}^{N-1} \overline{f_n} \cdot W^{n(N-k)} = \frac{1}{N} \sum_{n=0}^{N-1} f_n \cdot W^{nk} = P_k$$

because  $f_n \in \mathbb{R}$  and  $W^N = 1$ .

Then, if  $f_n$  is real, we have  $P_k = \overline{P_{N-k}} = \overline{P_{SN-k}}$  ;  $S \in \mathbb{R}$

and since  $P_{k+N} = P_k$ , we have  $P_k = \overline{P_{-k}}$ .

## Interpretation of $R_k$

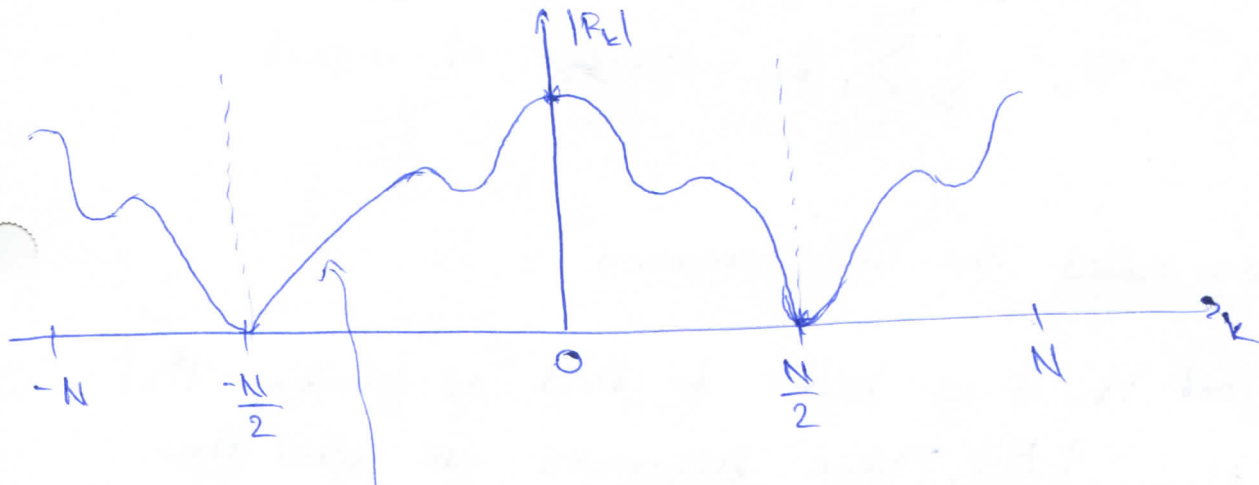
- For algorithmic convenience, we compute  $P_0, \dots, P_{N-1}$  but  $F_k$  is periodic,  $P_{k+N} = P_k$ . For real input (usual case)

$$|F_{-k}| = |F_k|$$

and

$$|P_{N-k}| = |P_k|$$

So if we plot the magnitude of  $P_k$ ,  $|P_k|$ , for real  $f_n$ , we would see something like



~~is~~ supposed to look like symmetric

Note: For convenience, show as cts, though  $P_k$  is a discrete function. i.e. impulse train

$$\text{Recall that } f_n = \sum_{k=0}^{N-1} P_k W^{nk} = \sum_{k=-N/2}^{N/2} P_k W^{nk} \quad (i)$$

In (i) there are two terms,  $k=p$ ,  $k=-p$ , i.e.

$$f_n = \dots + P_{-p} W^{np} + \dots + P_p W^{np} + \dots$$

If  $f_n$  is real then  $R_p = a + ib$  and  $R_{-p} = a - ib$ .

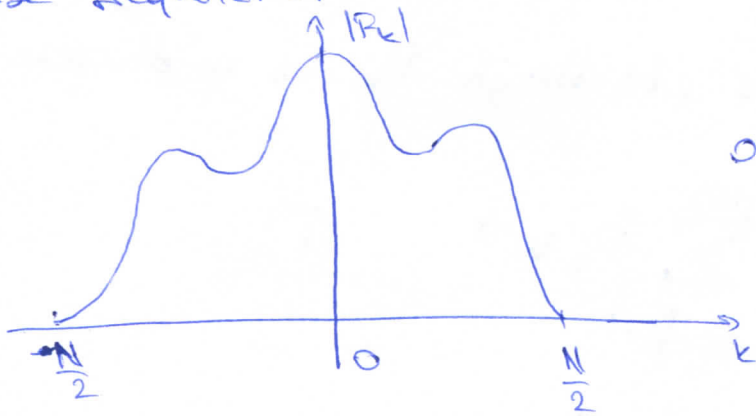
$$\begin{aligned} \text{Then } R_{-p} W^{-np} + R_p W^{np} &= (a - ib) \left[ \cos\left(\frac{-2\pi np}{N}\right) + i \sin\left(\frac{-2\pi np}{N}\right) \right] + \\ &\quad (a + ib) \left[ \cos\left(\frac{2\pi np}{N}\right) + i \sin\left(\frac{2\pi np}{N}\right) \right] \\ &= 2a \cos\left(\frac{2\pi np}{N}\right) - 2b \sin\left(\frac{2\pi np}{N}\right) \end{aligned}$$

$\Rightarrow$  Combination of positive and negative complex coefficients for freq  $\pm p$  gives real sin, cos with frequency  $p$

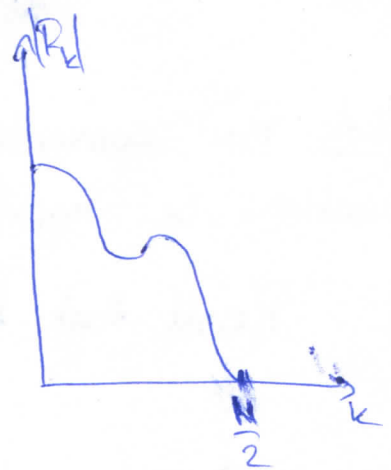
Note:  $R_0 = \frac{1}{N} \sum_{n=0}^{N-1} f_n = \text{average of input}$

$\hookrightarrow$  Also called the "DC component".

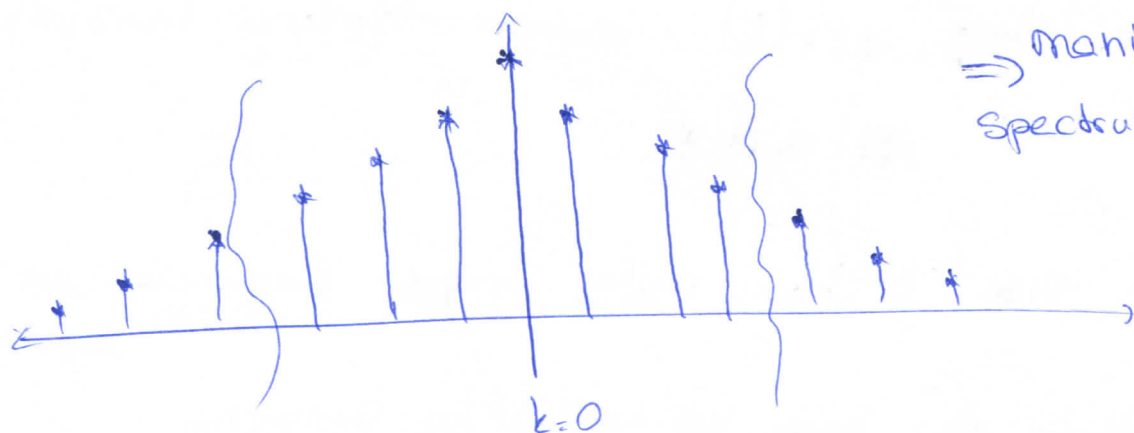
So to interpret  $R_k$ , it is better to think of plotting  $|R_k|$  from  $\lfloor \frac{-N}{2} \rfloor + 1, \dots, \lfloor \frac{N}{2} \rfloor$ . These frequencies are called the "base frequencies".



or even just



If we filter the data, it is necessary to apply the filter to  $R_k$  as plotted (above) to make sense of the filtering operation before reconstructing the signal [e.g. bird train whistle]



### Standardization Problem / Software

Unfortunately, there is a lack of a standard definition of the DFT pair.

- This does not affect basic results.

We can write a general DFT pair as

$$R_k = \frac{S}{N} \sum_{n=0}^{N-1} f_n(\bar{u})^{nk}$$

$$f_n = \frac{1}{S} \sum_{k=0}^{N-1} R_k u^{nk}$$

Where  $S$  = scaling factor,  $u = W$  or  $\bar{W}$

In this class, we use  $S=1$  and  $u = W = e^{\frac{j2\pi}{N}}$

Matlab uses:  $S=N$  and  $u = W = e^{\frac{j2\pi}{N}}$ .

Keep in mind, matlab arrays start at 1, not 0



$$f = \begin{bmatrix} f_0 \\ \vdots \\ f_{N-1} \end{bmatrix}$$

$f \leftarrow$  vector of signal

$$P = \text{fft}(f) \leftarrow \text{forward transform}$$

$$P_2 \begin{bmatrix} P_0 \\ \vdots \\ P_{N-1} \end{bmatrix} \xrightarrow{\text{fftshift}(P)} \begin{bmatrix} P_{-N/2} \\ \vdots \\ P_0 \\ \vdots \\ P_{N/2} \end{bmatrix} \xrightarrow{\text{ifft}(P)} f$$

$\text{fftshift}(P)$        $\text{ifft}(P)$        $\text{ifftshift}(f)$

$\leftarrow$  inverse transform

For sample size  $N$ , how many unique frequencies are represented?

$N/2$ . Everything else can be built by symmetry.

$N$  samples of a signal on  $[0, T] \Rightarrow$  sampling freq.  $= \frac{N}{T}$

What's the largest frequency that is uniquely be presented?

$$\frac{N/2}{T} = \frac{1}{2} \cdot \text{sampling freq.}$$

$\Rightarrow$  Sampling frequency  $> 2 \cdot$  max freq in signal in

order to capture all frequency information in a signal.

- Otherwise we get "aliasing" effects

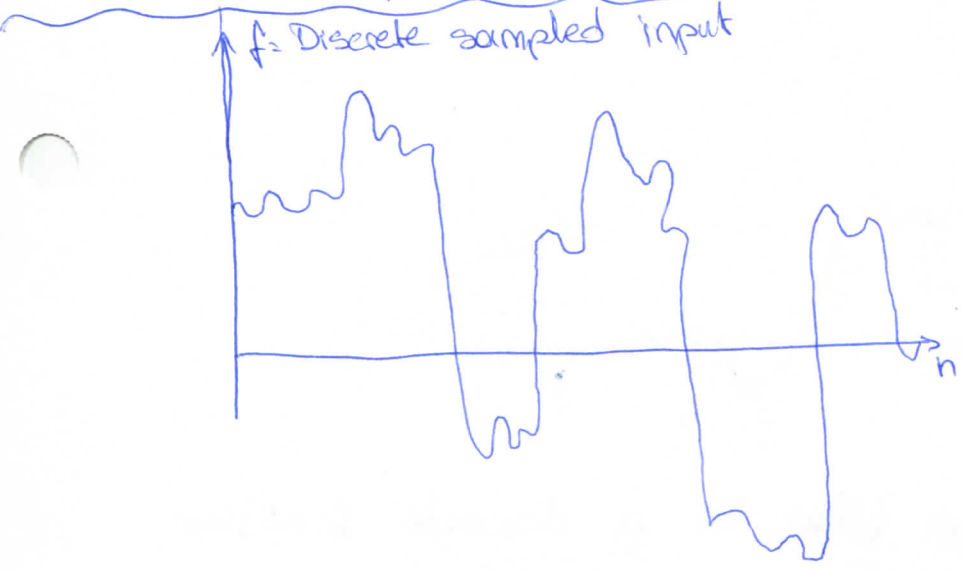
i.e. higher frequency components get "mixed-up" (aliased) with lower frequency components modulo  $N$ .

# Filtering

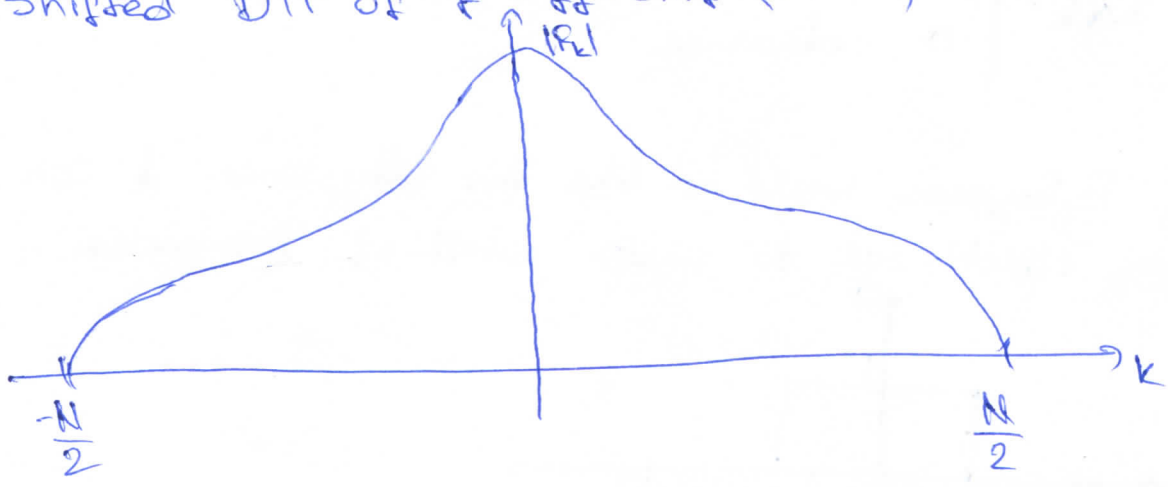
Suppose we want to design a "low-pass" filter  
i.e. we want to kill all frequencies in the input data  
that are larger than some threshold,  $f_{\text{thresh}}$

Want to remove freq  $|\frac{k}{T}| > f_{\text{thresh}}$  or  $|k| > f_{\text{thresh}} \cdot T \equiv P$

## Discrete Sampled Input

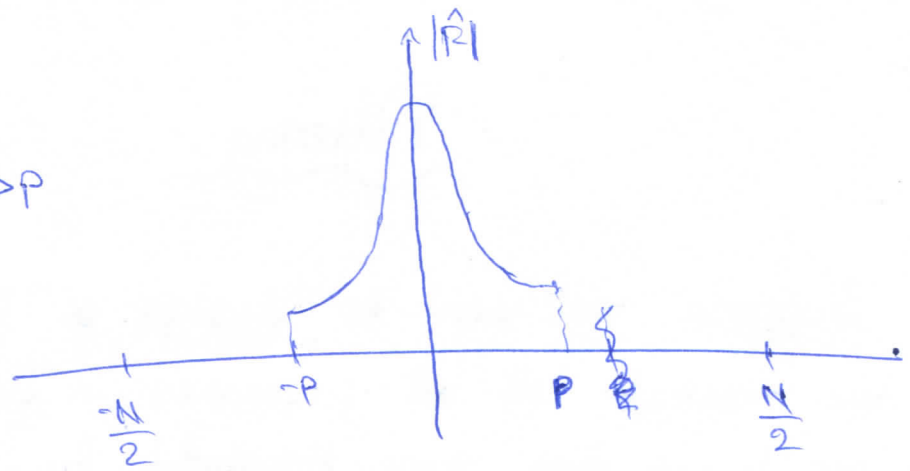


$F = \text{shifted DTR of } f = \text{fft} + \text{shift}(\text{fft}(f))$



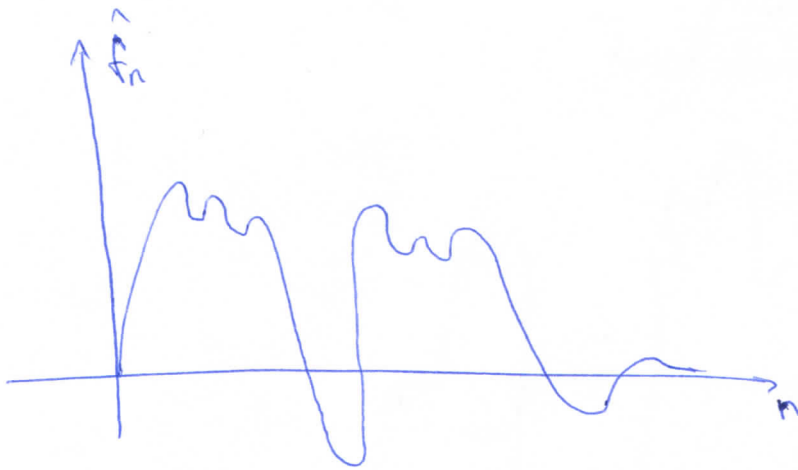
$\hat{P}$  = filtered  $P$

i.e zero out  $P_k$  for  $|k| > P$



Filtered signal =  $\hat{f}$  = inverse DFT of "u shifted"  $\hat{P}$

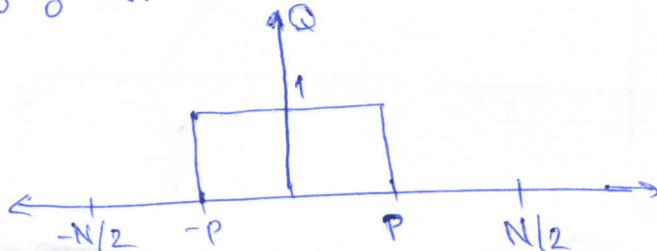
$$= \text{ifft}(\text{ifftshift}(\hat{P}))$$



More formally a low pass filter is a discrete function

$$Q_k = \begin{cases} 1 & |k| \leq P \\ 0 & \text{otherwise} \end{cases}$$

— Actually B. Engineers would not like this filter, since it can produce ringing effects, due to sudden cutoff of frequencies





- This has produced entire field "Digital Filter Design"  
 ↳ which is not in the scope of this course.

### Filtering Algorithm

$$\text{Input} = f = \{f_0, \dots, f_{N-1}\} \in \mathbb{R}$$

$$F = \text{fftshift}(\text{fft}(f))$$

$$\left\{ F_{\lfloor \frac{N}{2} \rfloor + 1}, \dots, F_{\lfloor \frac{N}{2} \rfloor} \right\}$$

$$\hat{P}_k = F_k \times Q_k, \quad k = \left\{ \lfloor \frac{N}{2} \rfloor + 1, \dots, \lfloor \frac{N}{2} \rfloor \right\}$$

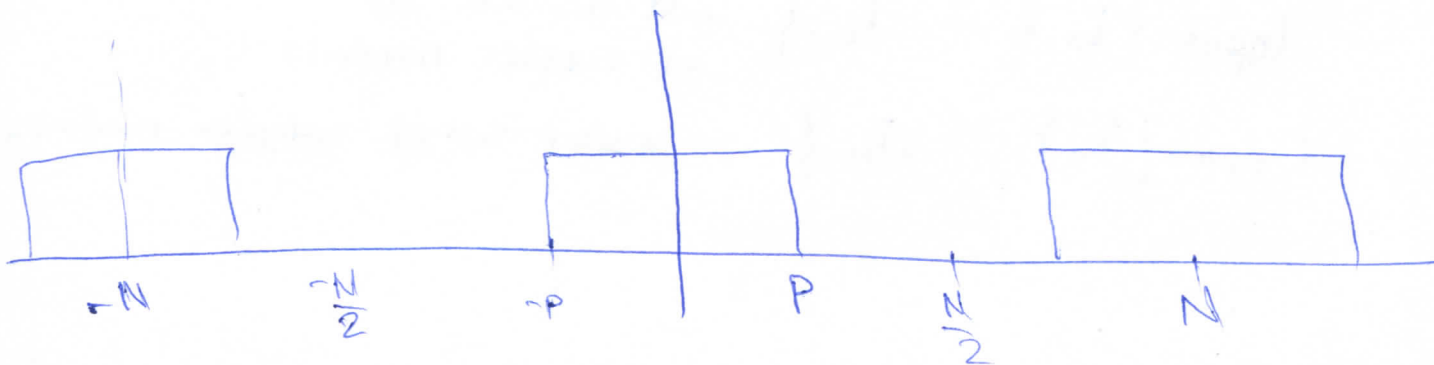
filtered signal  $\rightarrow \hat{f} = \text{real}(\text{ifft}(\text{ifftshift}(\hat{P})))$

needed ~~to~~ due to round/numerical errors, e.g.  $5 \pm 0.0000002$

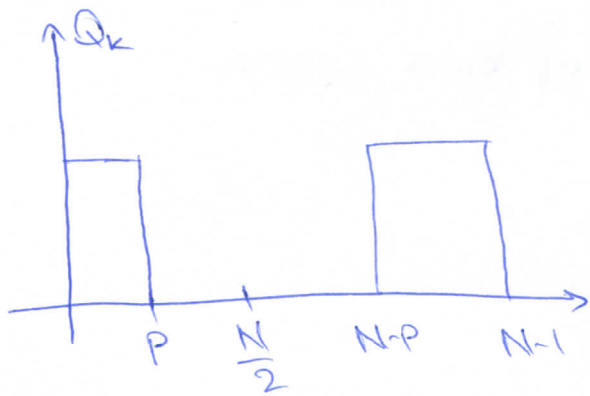
~ Recall we prefer to carry out operations on

$$Q_k, \quad k \in \{0, \dots, N-1\}$$

- We can define our filter so that it acts on data in the range  $\{0, \dots, N-1\}$  by defining the periodic extension of  $Q_k$ , i.e.  $Q_{k \pm N} = Q_k$



We will need to filter in the range  $k \in \{0, \dots, N-1\}$



$$Q_k = \begin{cases} 0 & k \in \{p+1, \dots, N-p-1\} \\ 1 & \text{otherwise} \end{cases}$$

This filter can be directly applied to

$$R_k, k \in \{0, \dots, N-1\}$$

### Filtering Algorithm

$$F = \text{fft}(f)$$

$$\hat{F} = F \times Q$$

$$f = \text{real}(\text{ifft}(\hat{F}))$$

### Another View of DFT

Although we have motivated the derivation of the DFT pair with ref Fourier Series, we can view the whole as a discrete transformation

Input =  $\{f_0, f_1, \dots, f_{N-1}\}$  - a discrete set of complex numbers

Output =  $\{F_0, F_1, \dots, F_{N-1}\}$  - another set of complex numbers

The transformation is easily reversible, i.e. given

$$\{P_0, P_1, \dots, P_{N-1}\}$$

we can generate the original set

$$\{f_0, f_1, \dots, f_{N-1}\}$$

### Matrix form of DFT

Let  $\vec{P} = \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_{N-1} \end{bmatrix}$

$\vec{f} = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{bmatrix}$

then, from  $P_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{nk}$

we can write  $\vec{P} = \frac{1}{N} M \vec{f}$

$$M = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W^1 & W^2 & \dots & W^{(N-1)} \\ 1 & W^{-2} & W^{-4} & \dots & W^{-2(N-1)} \\ 1 & W^{-3} & W^{-6} & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W^{-(N-1)} & W^{-2(N-1)} & \dots & W^{-(N-1)^2} \end{bmatrix}$$

Proof  $\sum_{n=0}^{N-1} W_N^{n(k-l)} = N \delta_{kl}$  we can see that

$$\overline{M^T} M = N \cdot I \implies M^{-1} = \frac{1}{N} \overline{M^T}$$

So the DFT pair in matrix form is:

$$F = \frac{1}{N} M f$$

$$f = \overline{M^T} F$$

### Complexity of DFT

Since  $M$  is a dense matrix of size  $N \times N$  then

$$F = \frac{1}{N} M f$$

requires  $O(N^2)$  complex flops

- But it turns out we can compute the  $N$  values (DFT)

$\{F_0, \dots, F_{N-1}\}$  in  $O(N \log_2 N)$  flops using a divide

and conquer approach.

### Fast Fourier Transform

$$\text{The DFT is } F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W_N^{nk} \quad (1)$$

Assume that  $N=2^m$  for some integer  $m$ . We can split the sum (1) in to two parts:

$$F_k = \frac{1}{N} \sum_{n=0}^{\frac{N}{2}-1} a_n W_N^{-nk} + \sum_{n=\frac{N}{2}}^{N-1} a_n W_N^{-nk} \quad (2)$$

Let  $q = n - \frac{N}{2}$  or  $n = q + \frac{N}{2}$  in 2<sup>nd</sup> term of (2)

$$F_k = \frac{1}{N} \sum_{n=0}^{\frac{N}{2}-1} a_n W_N^{-nk} + \frac{1}{N} \sum_{q=0}^{\frac{N}{2}-1} a_{q+\frac{N}{2}} W_N^{-k(q+\frac{N}{2})}$$

$$= \frac{1}{N} \sum_{n=0}^{\frac{N}{2}-1} a_n W_N^{-nk} + \frac{1}{N} \sum_{n=0}^{\frac{N}{2}-1} a_{n+\frac{N}{2}} W_N^{-kn} W^{-k\frac{N}{2}}$$

$$= \frac{1}{N} \sum_{n=0}^{\frac{N}{2}-1} W_N^{-nk} \left[ a_n + a_{n+\frac{N}{2}} W^{-k\frac{N}{2}} \right] \quad (3)$$

Note that  $W_N^{-k\frac{N}{2}} = e^{-ik\pi} = (e^{-i\pi})^k = (-1)^k$ . Then the even coefficients are

$$F_{2k} = \frac{1}{N} \sum_{n=0}^{\frac{N}{2}-1} (a_n + a_{n+\frac{N}{2}}) W_N^{-2nk} ; k=0, \dots, \frac{N}{2}-1 \quad (4)$$

while the odd coefficients are

$$F_{2k+1} = \frac{1}{N} \sum_{n=0}^{\frac{N}{2}-1} \left[ (a_n - a_{n+\frac{N}{2}}) W_N^{-n} \right] W_N^{-2kn} ; k=0, \dots, \frac{N}{2}-1$$

(5)



$$\text{Let } g_n = f_n + f_{n+\frac{N}{2}} \quad ; \quad n=0, \dots, \frac{N}{2}-1$$

$$h_n = (f_n - f_{n+\frac{N}{2}}) W_N^{-n} \quad ; \quad n=0, \dots, \frac{N}{2}-1$$

(6)

Then (4), (5), (6)  $\Rightarrow$

$$F_{2k} = \frac{1}{2} \cdot \frac{1}{\frac{N}{2}} \sum_{n=0}^{\frac{N}{2}-1} g_n W_N^{-2nk}$$

(7)

$$F_{2k+1} = \frac{1}{2} \cdot \frac{1}{\frac{N}{2}} \sum_{n=0}^{\frac{N}{2}-1} h_n W_N^{-2nk} \quad ; \quad k=0, \dots, \frac{N}{2}-1$$

(8)

$$\text{Now } W_N^{-2kn} = e^{\frac{i2\pi k \cdot 2n}{N}} = \underbrace{\left[ e^{\frac{i2\pi}{(N/2)kn}} \right]^{-kn}}_{W_{N/2}}$$

So that  $W_N^2$  is the "twiddle factor" for input data of size  $\frac{N}{2}$ .

$$\text{i.e. } W_N^2 = W_{N/2}$$

Thus (7) and (8) can be seen as

$$F_{\text{even}} = \frac{1}{2} \text{DFT}(g_n) \quad \text{and} \quad F_{\text{odd}} = \frac{1}{2} \text{DFT}(h_n) \quad ; \quad \frac{N}{2} \text{ points}$$

Therefore we have reduced the problem of computing a single DFT of  $N$  points to two DFTs each of  $\frac{N}{2}$  points.

- We can further reduce this problem to 4 DFTs of  $N/4$  points etc.

- There will be  $\log_2 N$  of these stages.

(8)

- Each stage requires  $O(N)$  complex flops

Total complexity FFT =  $O(N \log_2 N)$ .

We can use essentially the same algorithm in both directions - either forward or inverse transforms by replacing  $W$  with  $\overline{W}$  and adjusting multiplication factor

Office hours will be posted on Piazza.

April 02, 2018

## Fast Fourier Transform

Showed that we can compute the DFT efficiently by splitting the problem into halves recursively. For this we assumed  $N = 2^m$  for an integer  $m$ . How do we handle the case when  $N$  is not a power of 2.

Solution 1:

Pad our data  $\rightarrow$  zeros  
 $\rightarrow$  beginning part of data

E.g.  $f = \{f_0, f_1, f_2, f_3, f_4\}$

then we use

$$f' = \{f_0, f_1, f_2, f_3, f_4, 0, 0, 0\}$$

or

$$\{f_0, \dots, f_4, f_0, f_1, f_2\}$$

This works in the sense that

$$\text{IFFT}(\text{FFT}(f'))$$

will recover our original data.

but if  $P$  is DFT of ~~our~~  $f$  and  
 $P'$  is DFT of  $f'$ ,

then in general  $P'_k \neq P_k$ .



## Solution 2:

Can generalize FFT to an arbitrary split  $N = N_1 N_2$

⇒ Cooley-Tukey ~~algorithm~~ algorithm.

A split factor of  $r$  is called a radix- $r$  FFT.

Eg  $N = 2^7 \cdot 3^4 \cdot 11^2$  can be computed by a combination of radix-2, 3, 11 FFT's. We won't get into details, but point out in practice the algorithm can be done in-place, without recursion  
⇒ butterfly algorithm.

This is the algorithm Matlab's fft/fft uses.

END of Lecture.

## Final Exam Information:

- 10 questions
- Cumulative, heavy on past midterm
- 110 points

- Floating point errors and stability • 1 question

- Non-linear equations

• 1 question

↳ Fixed point, Newton etc.

- Google and Markov iteration

• 1 question

- 3 questions: Numerical Linear Algebra

↳ LU Decomposition

↳ Iterative methods

- 2 questions Interpolation and Integration

• 2 questions

- 2 questions Fourier

• 2 questions

• No FFT / No Matlab

Review: Assignments (+) Integration (+) Lecture Notes