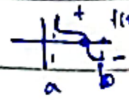
Date:

Ch#05

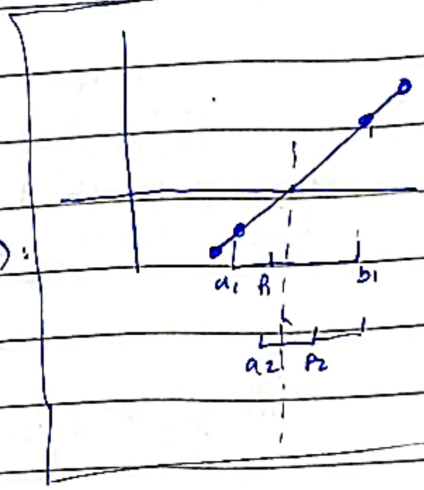
★ Bisection Method.



- If sign changes in interval f(a), f(b) then Keep halfing
  - if both f(a) * f(b) = positive then ~~x=a~~ a = x

so def bisection (f, a, b, tol, ~~er~~)

```
    if( f(a) * f(b) < 0.0)
        while ( (b-a) > tol ):
            x = a+b/2
            if ( f(a) * f(x) < 0)
                b = x
            elseif ( f(x) * f(b) < 0.0):
                a = x
            else break
    return x
```



★ ~~error~~ Convergence rate

$$\rightarrow \mu = \frac{\log\left|\frac{\overset{\text{error}}{e_2}}{e_3}\right|}{\log\left|\frac{e_1}{e_2}\right|}$$

★ Fixed point

- write eq in terms of x
- put x_{n+1} in subject
- pick a value, plug in, Keep plugging

e.g

$x^2 - x - 1 = 0$

$x = 1 + 1/x$

$x_{n+1} = 1 + 1/x_n$

plug $x = 2$ → $x_1 = 1 + \frac{1}{2} = 1.5$

$x_2 = 1 + \frac{1}{1.5} = 1.66$

$x_3 = 1.62$

Rule of conv → if $g'(\text{root}) < 1$ it conv

e.g $g(x) = 1 + \frac{1}{x}$

$g'(x) = + \frac{-1}{x^2}$

$g'(-0.381) = -$ so converges

f is original func, g is the func where x is subject

```
def fixedpoint ( g, x0 , tol , maxit)
    error = 1
    xk = x0
    while (error > tol)
        error = xk
        xk = g(xk)
        error = np.abs ( error - xk)
    return
def f(x):
    y = x**2 - x - 1
    return y
def g(x)
    y = np.sqrt(x+1)
    return y
```

x subject

Date:

Newton Raphson

└ Pick x, find $f'(x)$, plug into $\boxed{x_{n+1} = x_n - \frac{f(x)}{f'(x)}}$, repeat plugging

order of conv = 2.

Code

```
def Newton( f, df, x0, tol):
    same as fixed just
        xk = xk - f(xk) / df(xk)
    return
def f(x):
def df(x):
```

Secant Method

└ same as Newton, substitute slope formula in $f'(x)$ so $x_{n+1} = x_n - f(x_k) \div \underbrace{\left(\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}\right)}_{\text{slope formula}}$

Scipy

```
import scipy.optimize as spo
    x = spo.bisect( f, a, b, (), tol)
    x = spo.fixed_point( g, x0, (), tol, maxit)
    x = spo.newton ( same as fixed )
    x = spo.fsolve
```

CH#06 - Interpolation: find unknown values within set of known data points

Lagrange

$$y = \frac{(x-x_1)(x-x_2)\cdots(x-x_n)}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_n)} \cdot y_0 + \frac{(x-x_0)(x-x_2)(\cdots}{(x_1-x_0)(x_1-x_2)} \cdot y_1 +$$

doesn't contain x_0

Q1,

x	y
0	5
2	7
3	8
5	10
6	12

find ~~y~~ $f(4)$

for $x=4$ $y=?$ $\rightarrow$ $\frac{(4-2)(4-3)\cdots\cdots(4-6)}{(0-2)(0-3)\cdots\cdots} \cdot 5 + (4-0)\cdots\cdots$

~~Newton Method~~

Date:

Newton Method

$(x_0, y_0) = (1, 2)$, $(x_1, y_1) = (7, 10)$, $(x_2, y_2) = (8, 12$

x	y	Δy	Δy^2
1	2		
7	10	4/3	
8	10	2	2/21

a

$$P(x) = a_0 + a_1 (x - x_0) + a_2 (x - x_0)(x - x_1) \dots$$

$$P(x) = 2 + 4/3 (x - 1) - 2/21 (x - 1)(x - 7)$$

Linear Spline

$$f(x) = y_0 + \frac{y_1 - y_0}{x_1 - x_0} (x - x_0)$$

Q1,

x	22	52	82	100
y	4187	4186	4199	4217

b1

find $f(b1)$

b1 b/w $\underbrace{52}_{x_0}$, $\underbrace{82}_{x_1}$ so

$$f(x) = \underbrace{52}_{\text{first point}} + \underbrace{\frac{4199 - 4184.0}{82 - 52}}_{\text{gradient at first}} \cdot (\underbrace{\hat{x}}_{\text{input}} - \underbrace{82}_{\text{second point}})$$

b1

Least Square approx

$A^{-1} b = c$

x	y
2	9
3	29
4	62
5	126
6	215

$$\begin{bmatrix} \Sigma x^0 & \Sigma x^1 & \Sigma x^2 \\ \Sigma x^1 & \Sigma x^2 & \Sigma x^3 \\ \Sigma x^2 & \Sigma x^3 & \Sigma x^4 \end{bmatrix} = \underbrace{\begin{bmatrix} (2+3+4+5+6)^0 & (\quad)^1 \\ & \end{bmatrix}}_{A} = \begin{bmatrix} 5 & 20 & 90 \\ 20 & 90 & 440 \\ 90 & 440 & 2274 \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} \Sigma x^0 y \\ \Sigma x^1 y \\ \Sigma x^2 y \end{bmatrix}}_{c} = \begin{bmatrix} 1279 \\ 2275 \\ 441 \\ 50 \end{bmatrix}$$

Hermite

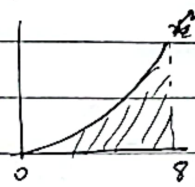
Hermatic

Date:

CH# 07 - integration Approx

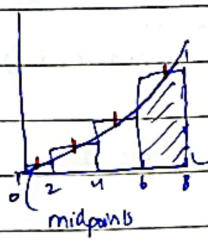
☆ Midpoint

Q1)



find area of curve till $\int_0^8 x^2$

$\underline{n} = 4$
no of steps interval

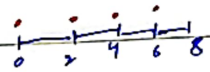


$\frac{8-0}{4} = 2$ } stepsize

sum area of rectangle for approx $= 2 \times \text{midpoint1} + (4-2) \times \text{midpoint2} \ldots$

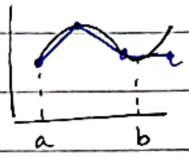
$A = \text{stepsize} * \sum_{i=1}^{n} f(x_i)$

$n = 4$



use 4 points

☆ Newton cotes



take area under each straight line to approx

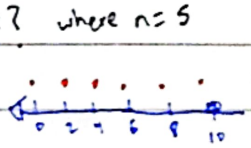
☆ Trapezoid Rule

$\cancel{\int_a^b f(x)} = \cancel{\int_a^b f_1(x)\,dx}$

$\cancel{f_1(x) = f(x_0) + \frac{f(x_1)-f(x_0)}{x_1-x_0}(x-x_0)}$

organic Q1) $\int_0^{10} x^2 = ?$ where $n = 5$

$\Delta x = \frac{b-a}{n} = \frac{10-0}{5} = 2$ so



use all 6

so

$$T_n = \frac{\Delta x}{2}\left[f(x_0) + 2f(x_1) + \ldots 2f(n-1) + f(x_n)\right]$$

first & last have 2

so $T_5 = \frac{2}{2}\left[0 + 2(4) + 2(16) + 2(36) + 2(64) + 100\right]$

$T_5 = 340$

National University Exams

Roll No 23-1613

Name Muhammad Mustafa Kamran

S.no	Code	Name	
1	CS3003	Numerical Computing	BC
2	CS3001	Computer Networks	BCS-
3	CL3001	Computer Networks Lab	BCS-5K
4	CS3004	Software Design and Analysis	BCS-5K
5	CS3005	Theory of Automata	BCS-5J
6	SS2007	Technical and Business Writing	

Manager Academics

Student Signature

FAST

Date:

Composite Simpson

$$S_n = \frac{h}{3}\left[f(x_0) + \boxed{4}(f(x_1) + \boxed{2}(f(x_2)) + \boxed{4} \cdots + f(x_n)\right]$$

· alternating 4, 2 except for 1st and last

CH# 02 Banded Matrix

$$A = \begin{bmatrix} \ddots & & 0 & 0 \\ & \ddots & & 0 \\ 0 & & \ddots & \\ 0 & 0 & & \ddots \end{bmatrix}$$

$L = 1$

$U = 2$

upper bandwidth

convert to banded form

for j in range (n)

$m = \max(1, j+1-U)$

$n = \min$

CH#03

1. Finite Precision

↳ limited memory so Floating point $1.1 + 0.1 = 1.20000002$

2. catastrophic cancel

↳ ~~ex~~ nearly equal numbers $= 0$

3. Machine epsilon

↳ smallest no when added 1, result in value diff from 1

```
x = 1
while (x+1 == 1)
        x = x/2
eps = 2*x
```

4. Underflow vs Over flow

Numbers smaller than 'tiny' are zero

numbers greater than max or less than min = inf, NAN

Day: Date: / /

Python Syntax.

- Write in scientific notation 1.3×10^{-2} cm

★ Python list can have diff data types

e.g ['R', '1', 'Hello']

★ Tuples are constant list

```
x = ('R','a','h','a','t')
print(x[0])
print(x[-1])      // Last index printed.
leng(x)           // length of x
```

x = (3,) comma must

★ Adding to list will append

```
grades = [9,6,7]
grades = grades + [6.2, 8.2, 5.9]
print(grades)     output => [9,6,7,6.2,8.2,5.9]
```

★ ':' is slice operator

```
grades [2:4]      python includes 2 but excludes 4
output:- 7, 6.2
grades [:4]
output :- 9,6,7,6.2
grades [4:]
      :- 8.2, 5.9
```

[i,j) ✓ python
[i,j] ✗
(i,j] ✓
(i,j) ✗

★ '=' causes ~~shallow~~ deep copy

```
list2 = grades    // changing ~~grades~~ list2 will ~~not~~ change list2
list2[0] = 14
print both :-  14,6,7....
               14,6,7....
```

★ '[:]' cause shallow copy

```
list2 = grades[:]
list2[0] = 14
..9,6,7
14,6,7
```

Yousaf

Day: Date: / /

★ Matrices

$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \Rightarrow A = \{[1,2],[3,4]\}$

★ Loops

- for i in [0, 1, 2]:
  print ('Hellowoild')

  i=0
  i=1
  i=2

  output= hello
  helwo
  hello.

- for animal in [cat, dog, bird]:
  print (animal)

  :- cat
  dog
  bird.

- Range: is range [start, stop, step]

  for i in range(5): 0,1,2,3,4
  for i in range(2,10,3): 2,5,8

- Pythonis case, space sensitive

★ Conditions

- a = $\underbrace{1<2}_{\text{true so a=true}}$

- if a != true:
  print (Not true)

- x = int (input ('Give me x = ')) // cin

- type (int(a))

  5! factorial

  !5 not

Yousaf

Day: Date: / /

* Functions

max( )
swap( )

- def double(x):
  y = 2* x
  return y

calling func :- w = double(x) + 1

- If, else

if x >= y :
  return x
else :
  return y

- Ternary operator

y = 2 if x > 3 else y = 3

* Lamda / alternative Func Defination

f = lambda x: f(x)

e.g

f = lambda x: x+1
print(f(1)) output :- 2

* Recursive Function

Yousaf

Day: Date: / /

The Library NumPy

import numpy as np

A = np.array([[1,2],[3,4],[5,6]]) // Matrix 3×2 array, only num allowed.

- Directly add scaler without loop

- slicing

  A[:, 0] //extracts the first column.

Mean

Vectors

print(a.shape) // prints shape of vector

a.reshape(1,3) // restape 3×2 vector to 1×3

Special Matrices

A = np.zeros(3,4) // gives 3×4 matrice $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

A.transpose.

I = np.eye(3) // $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Yousaf

Day: Date: / /

+ Triangular matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

L = np.tril(A) // $\begin{pmatrix} 1 & 0 & 0 \\ 4 & 5 & 0 \\ 7 & 8 & 9 \end{pmatrix}$ upper triangle

U = np.triu(A) // $\begin{pmatrix} 1 & 2 & 3 \\ 0 & 5 & 6 \\ 0 & 0 & 9 \end{pmatrix}$

- Shallow copy

B = A.copy()

- Add matrix, multiply / Divide with scalar.

A = B + C

- Performance

import time

A = np.random.rand(100,100) // random 100x100 matrix generated

B = np.ra . . . . .

t_0 = time.time()

C = A+B

t_1 = time.time()

print ('Python', $t_1 - t_0$, 'Seconds') . . . . . time for C=A+B numpy lowest time

t_0 = time.time()

C = add(A,B) . . . . . //c

t_1 = time.time()

print (; $t_1 - t_0$) time for C = add(A,B)

time for C = addv(A,B) second lowest time

Yousaf

Day: Date: / /

Multiplication

first col, X first row

for i = 1 : n

for j = 1 : p C(i,j) = A(i,:) • B(:,j)

End for

End for

* For Dot product of matrices e.g a • b, Numpy automatically tranposes it so a • b

• If A × B

C = A * B // this will cause elements to be multiplied

C = A @ B // matrix multiplication

E = np.dot (A,B) // Dot product if same dimension

* Extra Functionality

import numpy.linalg as npl

A_squared = npl.matrix_power (A,2) // whole matrix power

npl.power (A,2) // elemental power

npl.det (A)

A_inverse = npl.inv(A) $AA^{-1} = I$

Yousaf

☆ Linear Systems

$x_1 + x_2 + x_3 = 1$
$2x_1 + x_2 + x_3 = 2$
$1x_1 + 1x_2 + 1x_3 = 3$

So $A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ $B = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

```
from numpy.linalg import solve

x = solve (A,B)
```

☆ Eigenvalue / vectors

```
from numpy.linalg import eig

A = [   ]
(w,v) = eig(A)
```

list of eigen values — eigen vectors

```
print('w='); print(w)      :- [-1, 2]
print('v') . . .           :- [[1, 2]
                               [2, 4]]
```

Numerical Computing.

CH# 5

Roots of eq, find x values for eq $f(x)=0$:

1. Fixed point
2. Newton method (Newton Raphson)
3. secant Method
4. scipy.optimize
5. Application in Astrophysics

5.1 ★ Fixed point

- write eq in terms of x
- put it $= x_{n+1}$
- Keep plugging value

Q1) $x^2 - x - 1 = 0$

$x^2 = x + 1$

$x = \frac{x}{x} + \frac{1}{x}$

$x = 1 + \frac{1}{x}$

$x_{n+1} = 1 + \frac{1}{x_n}$

plug $x = 2$] random value

$x_2 = 1 + \frac{1}{2} = 1.5$

$x_3 = 1 + \frac{1}{1.5} = 1.66$

$= 1.6$

$= 1.625$

$= 1.6125$

converges

$x^2 - x = 1$

$x(x-1) = 1$

$x = \frac{1}{x-1}$

$x_{n+1} = \frac{1}{x_n - 1}$

plug $x = 1.6$

$x_2 = \frac{1}{1.6-1} = 1.66$

$= 1.5$

$= 2$

doesn't converge

Date:

Rule of Convergence = converges if $\boxed{|g'(root) < 1|}$

$g(x) = 1 + \frac{1}{x}$

$g'(x) = +\frac{-1}{x^2}$

$= |-0.3819| < 1$

= converges

$g(x) = \frac{1}{x-1}$

$g'(x) = -1/(x-1)^2$

$g(root) = |-2.618| > 1$

= doesn't

Code:

```
def fixedpoint (g, x0, tol=1.e^-6, maxit=100):
    error = 1
    iteration = 0
    x_K = x0
    while ( error > tol and iteration < maxit ):
        iteration++
        error = x_K
        * x_K = g(x_K)
        error = np.abs (error - x_K)
    return x_K

def f(x) :
    y = x**2 - x - 1.0
    return y

def g(x):
    y = np.sqrt(x+1)
    return y

x = fixedpoint ( g, x0, tol, maxit)
```

Date:

5.2 Newton Raphson

- Pick x near root
- find $f'(x)$
- plug into $\boxed{x_{n+1} = x_n - \frac{f(x)}{f'(x)}}$
- repeat until converg

Q1, $f(x) = x^2 - 10$

$f'(x) = 2x$

$x_{n+1} = x_n - \frac{x^2-10}{2x}$

$x_0 = 3$

$x_1 = 3 - \frac{3^2-10}{6} = 2.767$

$x_2 = \quad = 2.67$

Code:

```
def Newton( f, df, x0, tol, maxit)
    error ''
    iteration ''
    xk ''
    while( '' ):
        iteration++
        err = xk
        xk = xk - f(xk) / df (xk)
    '' same
```

Order of convergence = 2

$f(x_n) - f[x_n$

x_{n-1}

Date:

5.3, Secant Method

- same as newton guess no derivative required

just ~~substitute~~ slope formula in $f'(x)$

- $$f'(x) = \underbrace{\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}}_{\text{slope}}$$

- $$\boxed{x_{k+1} = x_k - f(x_k) \cdot \underbrace{\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}}_{\text{reciprocal of slope}}}$$

Q1) ~~Find root, perform 4 intervals of secant,~~ $x^3 + x^2 - 1$

code:

```
secant (f, x1, x2, tol, error)

    while ( err > tol .... )
        xk1 = x1
        xk = x2
        iteration++
        err = xk1
        xk+1 = xk - ((xk - xk1) / f(xk) - f(xk1))*f(xk)
        error = np.abs(err . xk1)
        x1 = 2
        x2 = xk1

def f(x):
def
```

Date:

Q, $f(x) = x^2 - 4$ actual root = 2

initial guess $x_0 = 3$, $f(x_0) = 5$

$x_1 = 2.5$ $f(x_1) = 2.25$

$x_0 = 2$

secant for

$$x_{n+1} = x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

so

$$x_2 = x_1 - f(x_1) \cdot \frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

$$x_2 = 2.5 - 2.25 \left(\frac{2.5 - 3}{2.25 - 5} \right)$$

$$= 2.095$$

$$x_3 = x_2 - f(x_2) \cdot \frac{f(x_2) - f(x_1)}{f(x}$$

Date:

5.4 ★ Scipy Module

→ Bisection code:

```
import scipy.optimize as spo
def f(x):
    '''
x = spo.bisect (f, a, b, (), tol)
```

→ fixedpoint

```
import scipy.optimize as spo
def f(x)
def g(x)
x0 , tol, maxit
x = spo.fixed_point ( g, x0, (), tol, maxit)
```

→ Newton

```
import scipy.optimze as spo
def f(x)
def df(x)
x0
x = spo.newton (f, x0, df, tol , maxit)
```

→ fsolve (Hybrid work on broader spectrum)

```
import ..
def f(x):
def df(x):
x0
x = spo.fsolve (f, x0, fprime = df, xtol=1.e-4)
```

Date:

5.4 ☆ Application

$$\underbrace{\theta}_{\text{eccentric anomaly}} = \overbrace{\psi}^{\text{anomaly of planet}} + \underbrace{e}_{\text{Eccentricity}} \sin\theta$$

- use fixed point

```
def g(theta):
        e = 1.e^-6
        psi = np.pi/6
        return = psi + e* np.sin(theta)
f0 = np.pi/6
f1 = $po.fixed-point (g, theta0)
```

Date:

C#6 Interpolation

way to find unknown value within set of know data points

1. ~~Monomial Basis~~
2. Lagrange polynom
3. Failure of Lagrange
4. Newton method
5. Hermite
6. Spline
7. Least square
8. Polynomial fit
9. Non Poly Fit
10. Scipy

2/ Lagrange Polynom

No of Datapoints

$$= \sum_{j=1}^{n} \left[y_j \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} \right]$$

summation

product Notation

Degree of Poly $= n-1$

(0,5)
(2,1)

Date:

Q

	x	y	
x_0	0	5	y_0
x_1	2	7	y_1
x_2	3	8	y_2
x_3	5	10	y_3
x_4	6	12	y_4

find $f(4)$

Formula ⇒

numerator must not contain x_0

for point 0 ⇒ x_0, y_0

$$y_{A \times 4} = \frac{(x-x_1)(x-x_2)\cdots(x-x_n)}{(x_0-x_1)(x_0-x_2)(x-)\cdots(x_0-x_n)} \cdot y_0 +$$

x_0 - everything.

$$\frac{(x-x_0)(x-x_2)(x-x_3)\cdots}{(x_1-x_0)(x_1-x_2)\cdots} y_1$$

so solving for $x=4$ find y

$$\hookrightarrow y = \frac{(4-2)(4-3)(4-5)(4-6)}{(0-2)(0-3)(0-5)(0-6)} \cdot 5 +$$

$$\frac{(4-0)(4-3)(4-5)(4-6)}{(2-0)(2-3)(2-5)(2-6)} \cdot 7 + \cdots$$

$$y = 8.89$$

Code:

```
def lagrangeBasis (z, x):
    n = len(x)
    m = len(z)
    basis = np.ones((n,m))

    for i in range(n) :
    for j in range (n):
        if i != j :
            basis[i, :] *= (z - x[j] / x[i] - x[j])

    return basis
def lagrange_interpol (z, x, y)
    n = len(x)
    m = len(y)
    P = np.zeros(m)
    basis = lagrange_basis(z, x)
    for i in range(m):
        P += basis[i, :] * y[i]
    return P

def f(x):
z = np.linspace (1, 4, 100)
x = np.array([1, 1.5, 4])
y = f(x)
yz = = lagrange_interpolant (z, x, y)
plt.plot (z, yz, '-')
plt.plot ( x, y, 'o')
plt.plot (z, f(z), '-')
plt.legend
plt.xlabel , ylabel
plt.show().
```

no of node

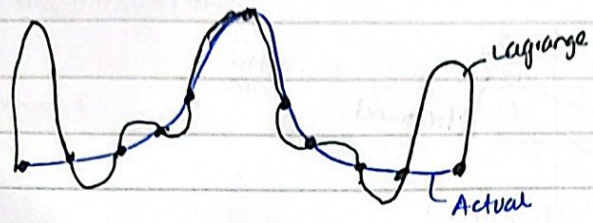
no of points to calculate

unknown points

known points

★ Lagrange Failure

for too many known points



★ Newton Method

Q1,

x	24	28	32	36	40
y	28.06	30.19	32.75	34.94	40.

find ★ $f(33)$

$$y = y_n + \frac{v \cdot 1}{1!} y_{n-1} + \frac{v(v+1) \cdot 1}{2!} y_{n-2} \cdots y_{n-n}$$

$$\boxed{v = \frac{x - x_n}{\cancel{n}}}$$ — gradient

so $v \Rightarrow$ $n = 4$, $x = 33$, $x_n = 40$ so

$$v = \frac{33-40}{4} = -7/4$$

★ Divided Diff

x	y a_0	Δy a_1	Δy^2 a_2	Δy^3
24	28.06	$\frac{30.19-28.06}{28-24}$	"	"
28	30.19	"	"	
32	32.95	"		
36	34.94	"		
40	40			

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$$

$$\cancel{f(x) = a_0 + a_1(x-x_1) + a_2(x-x_1)(x-x_2)\cdots}$$

$\frac{150-180}{5-3}$

-30

$5-3=h$

Q1)

x	3	5	7	9
y	180	150	120	90

find $f(4)$

- x point must be equidistant
- Newton forward — if 4 in beggining of data point (in this case)
- backward — end

	x	y (a_0)	Δy (a_1)	$\Delta^2 y$ (a_2)	$\Delta^3 y$ (a_3)
x_0	3	180	-30	0	0
x_1	5	150	-30	0	
x_2	7	120	-30		
x_3	9	90			

for forward

for backward

$$\boxed{v = \frac{x - x_0}{h}} = \frac{4-3}{2} = 1/2 = \boxed{0.5 = v}$$

h — diff in x values

* Newton's forward form

+1 for backward

$$\boxed{y = y_0 + \frac{v}{1!}\Delta y_0 + \frac{v(v-1)}{2!}\cdot\Delta^2 y_0 \cdots\cdots}$$

$$y = 180 + \frac{0.5}{1}(-30) + \frac{0.5\cdot(-0.5)}{2!}\cdot(0) + \frac{0.5(-0.5)(-1.5)}{3!}\cdot(0)$$

$$y = 180 - 15 = 165$$

$$P(x) = 180 + (-30)(x-3) + 0(x-x_0)(x-x_1)$$

~~or $P(x) = y_0 + \Delta y_0 \cdot (x - x_0$~~

$$P(x) = y_0 + \Delta y_0 \cdot (x - x_0) + \Delta^2 y_0 \cdot (x-x_0)(x-x_1) + \Delta^3 y_0$$

code to interpolat at np.linspace (3, 9, 20)

```
z = np.linspace(3, 9, 20)
x = [3, 5, 7, 9]
a = [180, -30, 0, 0]
p = a[4];
for k in range(1, 4+1)
    p = a[4-k] + (z-x[4+k])*p
```

Date:

Hermatite Interpolation Skipped

Spline Method

1) Linear spline

	$i=1$	2	3
x	0	1	2
y	0	1	8

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

$$S_1(x) = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) = 0 + \frac{(1-0)}{1-0}(x-0)$$

Formula:-)

$$f(x) = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$$

Q1)

			61		
x	22	42	52	82	100
y	4187	4178	4186	4199	4217

find $f(61)$ between $\underbrace{52}_{x_0} - \underbrace{81}_{x_1}$

$y_0 = 4186$ $y_1 = 4199$

$$\text{so } f(x) = \overbrace{52}^{\text{first point}} + \overbrace{\frac{4199-4186}{82-52}}^{\text{gradient of first}} \cdot (\underbrace{x}_{\text{input}} - \overbrace{82}^{\text{second point}})$$

$$f(61) = 52 + 0.4.33 \cdot (61-82)$$

* Cubic Spline Skipped

Least Square Method

Q

x	1	2	3	4	5	6	7
y	1.5	3.8	6.7	9	11.2	13.6	16

x	y	xy	x^2
1	1.5	1.5	1
2	3.8	7.6	4
3	6.7	20.1	9
4	9	36	16
5	11.2	56	25
6	13.6	81.6	36
7	16	112	49
Σ	28	61.8	314.8	140
sum colum	Σx = p	Σy = q	Σxy = r	Σx^2 = s

$A X = B$

$X = BA^{-1}$

$X = \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix}$

$\hookrightarrow P(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1)$

3×3

4×4

A

$$\begin{bmatrix} \Sigma x^0 & \Sigma x^1 & \Sigma x^2 & \Sigma x^3 \\ \Sigma x^1 & \Sigma x^2 & \Sigma x^3 & \\ \Sigma x^2 & \Sigma x^3 & \Sigma x^4 & \\ \Sigma & & & \end{bmatrix}$$

$$A \begin{bmatrix} 7 & & \\ - & & \end{bmatrix}$$

Mat A

$$B \begin{bmatrix} \Sigma x^0 y \\ \Sigma x^1 y \\ \Sigma x^2 y \end{bmatrix}$$

$$\begin{bmatrix} s & p \\ p & \underbrace{N}_{\text{total no points}}+1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ q \end{bmatrix}$$

$A \quad x \quad b$

$Ax = B$

$x = A^{-1}B$

find x so $\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ q \end{bmatrix} \begin{bmatrix} s & p \\ p & N+1 \end{bmatrix}^{-1}$ use calculator

$P(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1)$

$P(4.2)$

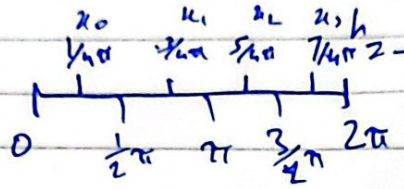
$$\begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix}$$

Date:

Midpoint integration

$$\int_0^{2\pi} \sin(x)\,dx \qquad [\overset{a}{0} \quad \overset{b}{2\pi}]$$

$$h = 4$$



$$\frac{b-a}{4} = \frac{1}{2}\pi$$

$$= f(x_0)*h + f(x_1)*h + f(x_2)*h + f(x_3)*h$$

$$\sin(\tfrac{1}{4}\pi)*\tfrac{1}{2}\pi +$$

$$= \ldots \text{ app}$$

$$\text{Relative} = \frac{|\text{error}|}{\text{exact}}$$

Absolute error

CH#09

Intro + overview

└ Solving linear eq using computational methods

e.g $6x + 2y = 4$
$4x + 5y = 5$ so $A = \begin{bmatrix} 6 & 2 \\ 4 & 1 \end{bmatrix}$ $x = \begin{bmatrix} x \\ y \end{bmatrix}$ $b = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$

$\underbrace{A}_{\text{coefficients}} \overset{\text{unknowns}}{\underset{\text{solve for } x}{x}} = \underbrace{b}_{\text{constants}}$

- Direct Methods
  - pivoting
  1. Gaussian Elimination : transform A to upper triangle and solve x
  2. LU factorization : Decompose A into L, U (lower upper triangl
  3. Cholesky Factorization : if symmetric and positive decompose A into LL^T
- Iterative methods
  1. Jacobi Method : update all variables simultaneously based on prev iteration
  2. Guass Seidal Method : update each variable immediatly as it's computed leading to faster convergence..
- Sparse Matrices (scipy tools for effizency)
- Practical Implementation : Kirchoff's law (circuit')

Naive Gaussian Elimination.

Steps,

1. Foward Elimination : (Eliminate elements below main diagnol to make upper Triangle)
2. Back substitution : starting from last row, calculate each unKnown.

e.g
$$\left[\begin{array}{ccc|c} 1 & 1 & -1 & -2 \\ 2 & -1 & 1 & 5 \\ -1 & 2 & 2 & 1 \end{array}\right] \xrightarrow{\text{forward Eli}} \begin{array}{c} \begin{array}{ccc} x & y & z \end{array} \\ \left[\begin{array}{ccc|c} 1 & 1 & -1 & 2 \\ 0 & 1 & -1 & -3 \\ 0 & 0 & 1 & 2 \end{array}\right] \end{array}$$

Back Sub ⇒ $\boxed{1z = 2}$ so

$y + z = -3$ so $y + 2 = -3$ $\boxed{y = -5}$

$x + y - z = 2$ so $x - 5 + 2 = 2$, $\boxed{x = 5}$

Date:

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

→ eliminating b ⇒ $mik = \frac{b}{a}$

then $b - a(mik) = 1$

Algorithm:

```
import numpy as np
def Gaus_elim (A, b):
    n= len(b)
    for K in range (n-1):
        for i in range (K+1, n):
            if A[K,K] != 0.0:
                mik = A[i,K] / A[K,K]
                A[i, K+1:n] = A[i, K+1:n] - mik * A[K, K+1:n]
                b[i] = b[i] - mik * b[K]

    # backward sub
    for K in range (n-1, -1, -1):
        b[K] = (b[K] - np.dot (A[K, K+1:n], b[K+1:n])) / A[K,K]

    return b
```

$$\left[\begin{array}{ccc|c} 4 & -2 & 1 & 11 \\ -2 & 4 & -2 & -16 \\ 1 & -2 & 4 & 17 \end{array}\right]$$

1- 4 must not be 0

~~2. 4/-2 = row1/row2 = mik =~~

2. 4/-2 = mik = -2

3. Row2 = Row2 − (0.5) * row2

do for b too

so −2 = −2 − (−2 * $\frac{-0.5}{2}$) = 0

e.g2) $\begin{pmatrix} 2 & 3 \\ 4 & -1 \end{pmatrix} \rightarrow \begin{pmatrix} 2 & 3 \\ 4-(2*2) & -1-(2*3) \end{pmatrix} \rightarrow \begin{pmatrix} 2 & 3 \\ 0 & -7 \end{pmatrix}$

Date:

LU Factorisation

$L = \begin{bmatrix} 1 & 0 & 0 \\ * & 1 & 0 \\ * & * & 1 \end{bmatrix}$ $U = \begin{bmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix}$

Decompose A to $\underbrace{L}_{\text{lower}}\overset{\text{upper}}{U}$ triangle such that $LU = A$.
so

Steps)

1. ~~start L as identity mat, U as copy of A~~
2. ~~Elimination:~~

By Hand method:

1. Use guassian for upper triangle 'U'
2. collect multiplyers of row operation for 'L'

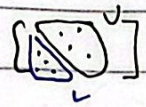
a) $\begin{bmatrix} 1 & -2 & 0 \\ -3 & -2 & 1 \\ 0 & -2 & 8 \end{bmatrix} \xrightarrow{R_2 \leftarrow R_2 + 3R_1} \begin{bmatrix} 1 & -2 & 0 \\ 0 & -8 & 1 \\ 0 & -2 & 8 \end{bmatrix} \xrightarrow{R_3 \leftarrow R_3 - R_2/4}$

$\begin{bmatrix} 1 & -2 & 0 \\ 0 & -8 & 1 \\ 0 & 0 & 31/4 \end{bmatrix} = U$

$L = \begin{bmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ 0 & +1/4 & 1 \end{bmatrix}$

Algorithim

modify A / Use Gaussian for 'U', and 'L'

so # A after this = 

```
def LU(A)
    n = len(A)
    for k in range(n-1):
        for i in range(k+1, n):
            mik = A[i,k] / A[k,k]
            A[i, k+1:n] = A[i, k+1:n] - mik * A[k, k+1:n]
            A[i,k] = mik   # L assigned here
return A
```

Date:

```
main();
    B = LU(A)
    L = np.tril(B)
    U = np.triu(B)
```

☆ Solving LU

$Ax = b$

$(LU)x = b$

$Ly = b$

$Ux = y$

e.g $\begin{bmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{bmatrix}$ $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$, $c = \begin{bmatrix} 1 \\ 6 \\ 4 \end{bmatrix}$

• Dont make change to c in LU, not needed

$\Rightarrow \left[\begin{array}{ccc|c} 1 & 1 & 1 & \\ 4 & 3 & -1 & \\ 3 & 5 & 3 & \end{array}\right] \xrightarrow[R_3 - (+3R_1)]{R_2 - (+4)R_1} \left[\begin{array}{ccc|c} 1 & 1 & 1 & \\ 0 & -1 & -5 & \\ 0 & 2 & 0 & \end{array}\right] \xrightarrow{R_3 - (-2)R_2}$

$\left[\begin{array}{ccc|c} 1 & 1 & 1 & \\ 0 & -1 & -5 & \\ 0 & 0 & -10 & \end{array}\right]$ so $U = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{bmatrix}$

$L = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & -2 & 1 \end{bmatrix}$ $L = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & -2 & 1 \end{bmatrix}$

Assume $y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$ so $\boxed{Ly = b}$

so $\begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & -2 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ original & $c = \begin{bmatrix} 1 \\ 6 \\ 4 \end{bmatrix}$

$y_1 = \boxed{1}$

$4(1) + y_2 = 6$ so $y_2 = \boxed{-2}$

$y_3 = \boxed{5}$

so

$\boxed{Ux = y}$

$\hookrightarrow \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix}$

solve

Date:

☆ Pivoting

error if guassian normal tails as

so $\frac{1}{0}$ ↑

$$A_1 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \qquad A_2 = \begin{bmatrix} \overbrace{10^{-17}}^{\text{error}} & 1 \\ 1 & 1 \end{bmatrix}$$

so solution is pivoting change order of rows

☆ Gaussian Partial Pivot

- before ~~redu~~ elimination of below elements to zero
  - check for Highest No, swap its row.

```
def pGauss_ ( A, b, tol = 1.0e-15):
    n = Len(b)
    for k in range(n-1):
        p = np.argmax(np.abs(A[k:n, k])) + k   find pivot row (row with largest value in current column)

        if np.abs(A[p,k]) < tol:   // if pivot too small
            raise ValueError (" Matrix is singular or nearly singular")

        if p != k:                 // swap rows
            A[[k,p]] = A[p,k]      // swap row in A matrix
            b[[k,p]] = b[p,k]      // swap row in B

        for i in range (k+1 : n):  // elimination same as guass
            same Mik...

        if np.abs(A[n-1, n-1]) < tol:   // if last pivot element too small.
            raise error ("Matrix singular")

        Back substitution same
        rest same
```

$-2/6 = -1/3$ $x = -6$ $-2 - \frac{1}{3}$

Date:

LU Pivoting

★ LU Pivoting:

$$\underbrace{P}_{\text{permutation identity matrix}}A = LU = b$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \ldots$$

Q1, $A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & 7 & 9 \end{bmatrix} \xrightarrow{\text{swap } R_1 \leftrightarrow R_2} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 2 & 7 & 9 \end{bmatrix} \xrightarrow[R_3 + 2R_1 \to R_3]{R_3 - 2R_1 \to R_3} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 7 \end{bmatrix}$

$\xrightarrow[R_3 + 3R_2 \to R_3]{R_3 - 3R_2 \to R_3} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix}$

- P initially $= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ after swap $R_1 \leftrightarrow R_2$ $= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = P$

- $L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix}$

$R_3 - 2R_1$ — 2 put in R_3 unlike prev

instep 2 this element removed by $-3R_2$ so put here

so $PA = LU$

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 1 \\ 2 & 7 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix}$$

To solve

$Ax \times PAB$

$\boxed{LUx = \boxed{P}b}$ as ~~PAx=b~~

~~PPA = LU~~ so 1. $Ly = Pb$

$Ux = y$

Dont multipy instead result = $b[P[i]]$ points to elements before swap

Date:

Code:

```
def rowswap (v, i, j):
def LUpivot (A, tol = 1.0e-15):
    n = len(A)
    perm = np.eye(n)                    ] Pa identity matrix
    for k in range (0, n-1):
        maxrow = np.argmax (np.abs (A[k:n, k])) + k   ] find row with max value in current column
        if np.abs (A[p, k]) < tol :
            " Singular Matrix"
        if maxrow != k :
            rowswap (A, k, maxrow)
            rowswap( perm, k, maxrow)   } swap rows, A in Perm too
    for i in range(k+1, n):
        if A[i, k] != 0.0
            mik = A[i, k] / A[k, k]     } Lower = b/a
            . . . .                     Upper = normal elimination

    return A, prem
def solve (A, b, prem):
    n = len(A)
    x = b.copy
    for i in range (n)
        x[i] = b[prem[i]]
    for k in range (1, n)
        x[k] = x[k] - np.dot(A[k, 0:k], x[0:k])
    x[n-1] = x[n-1] / A[n-1, n-1]

    for i in range (n-2, -1, -1);
        x[k] = (x[k] - np.dot(A (k, k+1:n), A(k+1:n))) / A[k,k
```

To solve an given Array
↳ L^x, L^T
↳ $\begin{bmatrix} a^2 & ab & ad \\ ab & b^2+c^2 & \\ & & d^2+e^2+f^2 \end{bmatrix}$
↳ find values of L

Cholesky's Decomp

condition
1- A is symmetric ie $A^t = A$
2- A is positive definate |first element|, |top left matrix

Steps
1. $AX = B$
2. $A = L^x L^T$
3. $LY = B$
4. $L^T X = Y$

By Hand:

Q1) $\begin{bmatrix} 4 & 10 & 8 \\ 10 & 26 & 26 \\ 8 & 26 & 61 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 144 \\ 128 \\ 214 \end{bmatrix}$

A X B

condition 1. ✓
2. Positive definate ✓ determinant $|4| > 0$ ✓, $\begin{vmatrix} 4 & 10 \\ 10 & 26 \end{vmatrix} > 0$ ✓, $|4|$

step 2) $A = L \cdot L^T$

$\begin{bmatrix} 4 & 10 & 8 \\ 10 & 26 & 26 \\ 8 & 26 & 61 \end{bmatrix} = \begin{bmatrix} a & 0 & 0 \\ b & c & 0 \\ d & e & f \end{bmatrix} \times \begin{bmatrix} a & b & d \\ 0 & c & e \\ 0 & 0 & f \end{bmatrix}$ transpose L for this

$\begin{bmatrix} 4 & 10 & 8 \\ 10 & 26 & 26 \\ 8 & 26 & 61 \end{bmatrix} =$ cross product $\begin{bmatrix} a^2 & ab & ad \\ ab & b^2+c^2 & bd+ce \\ da & bd+ce & d^2+e^2+f^2 \end{bmatrix}$

use calculator for direct

so $4 = a^2$ $\quad ab = 10$

$\boxed{2 = a}$ $\quad 2b = 10$ $\quad \boxed{d = 4}$

$\boxed{b = 5}$

$\boxed{c = 1}$ $\quad \boxed{e = 6}$ $\quad \boxed{f = 3}$

put into L, L^T

so $L, L^T \Rightarrow$ $L = \begin{bmatrix} 2 & 0 & 0 \\ 5 & 1 & 0 \\ 4 & 6 & 3 \end{bmatrix}$ $L^T = \begin{bmatrix} 2 & 5 & 4 \\ 0 & 1 & 6 \\ 0 & 0 & 3 \end{bmatrix}$

Step3)

$LY = B$ let $y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$

so

solve for Y

Step 4) $L^T X = Y$

solve for X.

Algorithm:

a^2

$\begin{bmatrix} a^2 & ab & ad \\ ab & b^2+c^2 & bd+ce \\ da & bd+ce & d^2+e^2+f^2 \end{bmatrix}$

* Diagnol elements

square root of different b/w from all prev elements in row e.g $L = \begin{bmatrix} \vdots & \vdots & \vdots \\ & & f \end{bmatrix}$ to calculate $\Rightarrow$ $d^2+e^2+f^2 = 61$

$f = \sqrt{61-(e^2+d^2)}$

* Non Diagnol

For $i = 1 : n$

$L(i,i) = \sqrt{A(i,i) - \sum_{k=1}^{i-1} L^2(i,k)}$] Main Diagnol

for $j = i+1 : n$

$L(j,i) = \left(A(j,i) - \sum_{k=1}^{i-1} L(j,k) L(i,k) \right) / L(i,i)$

Date:

```
import scipy.linalg as spl
(P, L, U) = spl.LU(A)
or
(LU, P) = spl.lu_factor(A)
x = spl.lu_solve((LU, P), b)
```

LU

cholesky

```
H = spl.cholesky(A)
y = spl.solve_triangular(H, b, lower=True)
x = spl.solve_triangular(H, y, trans=1, lower=True)
```

Jacobi Method

Condition for convergence:

$Ax=b$, A should be diagonally dominant ie

$$|A_{ii}| \geq \sum_{j=1, j\neq i}^{n} |A_{ij}|$$

Q1,

$$\begin{array}{c} \quad A \qquad\qquad B \\ \left[\begin{array}{ccc|c} 27 & 6 & 1 & 85 \\ 6 & 15 & 2 & 72 \\ 1 & 1 & 54 & 110 \end{array}\right] \end{array}$$

must be dominant or coefficient must be biggest in row after even summing of other elements

$27 > |6|+|1|$ ✓

$15 > |6|+|2|$ ✓

$54 > 1+1$ ✓

so applicable

Divide A into

$$A = \underbrace{D}_{\text{matrix with diagonal}} + \underbrace{N}_{\text{matrix A with diagonal zero}}$$

so

$Ax=b$

$(D+N)x=b$

$Dx = -Nx+b$

$$\boxed{Dx^{(k+1)} = -Nx^{(k)}+b}$$

so $x^{k+1} = \overbrace{D^{-1}}^{\text{reciprocal of diagonal matrix}}(-Nx^{(k)}+b)$

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

$$\begin{array}{ccc} x & y & z \end{array} \qquad A \qquad\qquad b$$

$$\begin{bmatrix} 27 & 6 & 1 \\ 6 & 15 & 2 \\ 1 & 1 & 54 \end{bmatrix} \begin{bmatrix} 85 \\ 72 \\ 110 \end{bmatrix}$$

=> condition (Diagnol Dominant ✓)

now.

$$x = 1/27\,(85 - 6y + 1z)$$
$$y = 1/15\,(72 - 6x + 2z)$$
$$z = 1/54\,(110 - 1y - 1x)$$

$x_0 = 0$, $y_0 = 0$, $z_0 = 0$

so

First Iteration =

$$x' = \frac{85}{27}$$
$$y = 72/15$$
$$z = 110/54$$

second Iteration

$$x = \frac{1}{27}(85 - 6(\ ) + 1(\ ))$$

code:

Date:

uses newly found values immediately

Guass seidal

- Same condition as Jacobi
  - dominant diagnol or symmetric positive definit
    - condition for convergence
      - if not met still can be applied but no gurantee

★ Dont confuse with guassian elimination

computing steps

$Ax=b$

- $A = LU$
- $x^{k+1} = L^{-1}(b-(U \cdot x^k))$

Q1,
$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \overset{b}{\begin{bmatrix} 7 \\ 1 \\ 1 \end{bmatrix}}$$

make the diagonal

uses k+1 unlike jacobi

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1} a_{ij} x_j^{(k)} \right)$$

$$x = \frac{1}{2}(7+y)$$
$$y = \frac{1}{2}(1+1x+1z)$$
$$z = \frac{1}{2}(1+1y)$$

Initial: $x_0 = 0 \quad y_0 = 0 \quad z_0 = 0$

1st Iteration:

$$x_1 = \frac{1}{2}(7+0) = 3.5$$
$$y_1 = \frac{1}{2}(1+3.5+0) = 2.25$$
$$z = \frac{1}{2}(1+2.25) = 1.625$$

2nd It:

$$x_2 = \frac{1}{2}(7+2.25) = 4.625$$
$$y_2 = \frac{1}{2}(1+4.625+1.625)$$
$$z_2 =$$

Date:

Code:

```
def gauss_seidal ( A,b,x ) tol=1e^-5, maxit):
n=len(b)
err = 1.0
iter = 0

xnew = np.zero_like(x) } initialize initial guess x0
M = np.tri((A) } extract lower triangle from A
U = A - M } extract upperT
while( error > tol and iter > maxit)
        iter++;
x^{k+1} = L^{-1}.(b-(U.x^k)) { xnew = np.dot(npl.inv(M), b - np.dot(U,x))
        err = npl.norm(xnew-x)
return x
```

memoize

SkippedRest.

$L^{-1}.(b-(U.x)$

Date:

CH#10
Best Approximation

Least Square Problem
L Obj: Minimize sum of squares of residuals

$Ax = b$ has no solution so use Least square – approximate sol

$$A^T A x = A^T b$$

1. compute $A^T A$, $A^T b$
2. write in matric eq $A^T A x = A^T b$ in RREF
3. Solution is least square.

code:

```
import numpy.linalg as npl

(x, residual, rank, s) = npl.lstsq(A, b, rcond=None)
```

Moore Penrose Inverse

$$x = ((A^T A x)^{-1} (A^T) b)$$

```
tmp = npl.inv(np.dot(np.transpose(A), A))
Amp = np.dot(tmp, np.transpose
```

Date:

Gram Schmidt

└ Generates orthonormal vectors

original basis $v_1, v_2, v_3 \longrightarrow u_1, u_2, u_3$

1) $u_1 = v_1$

2) $u_2 = v_2 - \dfrac{v_2 \overset{\text{dotprod}}{\cdot} u_1}{\underbrace{|u_1|^2}_{|u_1 \cdot u_1|}} u_1$ or $u_2 =$ [illegible]

3) $u_3 = v_3 - \dfrac{v_3 \cdot u_1}{|u_1|^2} \cdot u_1 - \dfrac{v_3 \cdot u_2}{|u_2|^2} \cdot u_2$

Q1) $x_1 = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$ $x_2 = \begin{bmatrix} 8 \\ 1 \\ -6 \end{bmatrix}$ $x_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

1) Apply Gram to make orthogonal vector u_1, u_2, u_3

2) use $u_1, u_2 \ldots$ compute orthonomal v_1, v_2, v_3

1)

$u_1 = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$

$u_2 = \cancel{\begin{bmatrix} 8 \\ 1 \\ 6 \end{bmatrix}} - \dfrac{\begin{bmatrix} 8 \\ 1 \\ 6 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}}{\left| \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} \right|} \cdot \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \dfrac{8+2+0}{1+4+0} \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \cancel{\dfrac{10}{5}} 2 \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$

$= \begin{pmatrix} 8 \\ 1 \\ 6 \end{pmatrix} - 2 \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix} = \begin{pmatrix} 6 \\ -3 \\ 6 \end{pmatrix}$

$\cancel{u_3 = \dfrac{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}}{5} \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} - \dfrac{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 6 \\ -3 \\ 6 \end{bmatrix}}{\begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix}} \begin{bmatrix} 2 \\ 4 \\ 0 \end{bmatrix} = 0 - 0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}$

$\cancel{u_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}}$

$u_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - \dfrac{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}}{5} \cdot \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} - \dfrac{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 6 \\ -3 \\ 6 \end{bmatrix}}{\begin{pmatrix} 6 \\ -3 \\ 6 \end{pmatrix} \begin{pmatrix} 6 \\ -3 \\ 6 \end{pmatrix}} \begin{bmatrix} 6 \\ -3 \\ 6 \end{bmatrix}$

$= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - \dfrac{6}{36-9+36} \begin{bmatrix} 6 \\ -3 \\ 6 \end{bmatrix}$

$= \cancel{\dfrac{4}{9}} = \begin{bmatrix} 4/9 \\ -2/9 \\ 5/9 \end{bmatrix}$

$|\ | =$ magnitude

so if $u_1 = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ $|u_1| = \sqrt{a^2+b^2+c^2}$

Date:

ii) Normalizat

$$u_1 = \frac{u_1}{|u_1|} = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$$

$$u_2 = \frac{u_2}{|u_2|} = \frac{1}{3}$$

Code:

QR facto

$$A = \underbrace{Q}_{\text{orthonormal matrix}} \underbrace{R}_{\text{upper triangle}}$$

$(A \xrightarrow[\text{smidth}]{\text{gram}} \text{orthonormal})$

To calculate R

$A = QR$

$Q^T A = \overbrace{Q^T Q}^{\text{identity matrix}} R$

$\boxed{Q^T A} = R$

code

```
from numpy.linalg import qr

Q, R = qr(A)
```

SVD:

$$A = U \Sigma V^T$$

- U: orthogonal matrix $[\,|\,|\,]$
- V^T: eigen vectors of A^*A^T
- Σ: rectangular diagonal matrix containing $\sqrt{\text{eigenvalues}}$ $\begin{bmatrix} \diagdown \end{bmatrix}$
- V^T: orthonormal matrix contain eigenvectors of A^TA $[\,|\,|\,|\,]$

Steps

1- A^TA

2- use characteristic eq to find λ $\Rightarrow$ $|A^TA - I\lambda| = 0$

so $\begin{bmatrix} a-\lambda & b-\lambda & c-\lambda \end{bmatrix} = 0$

3- find singular value $\sigma_1 = \sqrt{\lambda_1}$, $\sigma_2 = \sqrt{\lambda_2}$

4- for eigen vector

$\hookrightarrow$ $[A^TA - I\lambda_1]\underbrace{[\;]}_{v_1} = 0$ — normalize it divide by magnitude

$[A^TA - I\lambda_2]\underbrace{[\;]}_{v_2} = 0$

so

$$V = \left[\underbrace{[\;]}_{v_1}\underbrace{[\;]}_{v_2}\right]$$

Take Transpose for V^T.

Code:

```
import numpy.linalg as npL
A = np.array([ . . . . . ])
U, S, V = npl.svd(A)
```

Date:

Image Processing

```
import imagio , numpy , numpy.linalg   } import neccessary Libraries
photo = imagio.imread ("Newton.jpg")   #} read image
Red [:,:,0] = photo[:,:,0]
G   ''   ''  ''  ''   1]
B   ''   ''  ''  ''  ''  2]   } extract RGB channels to seperate matrics

U_r, S_r, V_r = svd(Red)
U_g, S_g, V_g = .   (Green)
''   ''   ''  ''  ''  ''  (Blue)   } get USR from each channel using SVD

K = 100   } Rank=100, set of eigenvalue/vectors, higher K = more detail

U_r_c = U_r[ : 0 : K] ; V_r_c = V_r[0 : K; :] ; S_r_c = np.diag (S_r)
U_b_  ''  ''  ''
U_g_  -   -   -   -   -   } perform compression by extracting top K

comp_image_r = np.dot( U_r_c, np.dot( S_r_c, V_r_c))
''  ''  ''  ''  -g  ''  ''  ''  ''  ''  ''  ''
''  ''  ''  ''  -b  ''  ''  ''   } compute each channel from compressed components

comp_image = np.zeros(row, col, 3)   ] initialize result matrix
comp_image [ :, :, 0] = comp_image_r
                        1            - g
                        2            - b   } Add RGB channel to result.

comp_image ( comp_image < 0] = 0 ; comp image [ comp image > 1] = 1   } clip values

plt. show (comp. img)   } show image
plt. show
```

Date:

CH#09 lecture

★ SparseMatrix

* from scipy import sparse

sparse_mat = np.array { { 2, -1, 0, 0
-10 0, 0 ; 0
1, 0 0, 1 .

A = sparse.coo_matrix (s) } coo = non zero stored in an array, and it's store in diff arrays

B = sparse.csr_matrix (A)

coo output			csr output	
(0,0)	2	print(A)	[2, -1, -10, 1, 1]	⇒ print (s.data[:])
(0,1)	-1		[0, 1, 4, 7 8]	⇒ s.indptr[:]
(1,0)	-10		[0 1 1 3 2 4 5]	⇒ s.indices[:]
indice	value			

★ Spar Guass seiden $x_{new} = L^{-1}(b - U \cdot x^{k})$

store A in csr
code is same just for operation on A access through sparse.linalg.inverse etc

```
def sparseGuas( A, b, x, tol, err)
    n = len(b)
    xnew = np.zeros.like(x)
    L = sparse.triL ( A, -1, format=csc)
    U = sparse.triU (A, +1,      =csc )
```

for loop

csp

A = np.array { 1, 0, 1 0, 0 0 } same call

b =

s = sparse.csr_matrix (A)

X = linalg.spsolve (s, b) } solves

Date:

CH#08

Differentiation:

★ Content

1- First order Finite difference approx

- Forward : $f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h}$
- Backward $f'(x_0) = \frac{f(x_0) - f(x_0-h)}{h}$
- Central (second order) $f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h}$

add eq

2- Second order 2nd derivative

- Approximation = $f''(x_0) = \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2} + \underbrace{O(h^2)}_{\text{error Bigo}}$

3- Richards Extrapolation

concept : ~~$D(h) = D(h) + ch^p$~~

$$\underbrace{D}_{\text{derivative}} = \underbrace{d(h)}_{\text{approx eq}} + \underbrace{E(h)}_{\text{error}}$$

- Take two approximation to reduce error (subtract equation)

$D = d(h_1) + ch_1^p$

$D = d(h_2) + ch_2^p$

- final equation

$$\hookrightarrow D = \frac{2^p d(h_1/2) - d(h_1)}{2^p - 1}$$

- In forward diff code

$d_1 = (f(x\theta+h) - f(x\theta)/h)$

- Take two derivative
- $D2 = h/2$
- final = $D2 \times 2 - D1$

$d_2 = (f(x\theta+h/2) - f(x\theta/h/2)$

derivative = $2 \cdot d_2 - d_1$

★ Euler method / Runge Kutta 1st

↳ $y_{n+1} = y_n + h\,\underbrace{f(x_n, y_n)}_{\text{fnc of } \frac{dy}{dx}}$

ex) solve $\frac{dy}{dx} = x + y$ with boundary value condition $y = 1$, $x = 0$.
find approx value of y for $x = 0.1$

$$\boxed{h_{step} = \frac{x^{\text{wanted}} - x^{\text{given}}}{\text{Steps}}} = \frac{0.1 - 0}{5 \text{ assume}} = \boxed{0.02 = h}$$

- need to calculate

$x_0 = 0$	$y_0 = 1$
$x_1 = 0.02$	$y_1 = ? = 1.02$
$x_2 = 0.04$	$y_2 = ?$
$x_3 = 0.06$	$y_3 =$
$x_4 = 0.08$	$y_4 =$
$x_5 = 0.06$	$y_5 =$

- Calculate y_1

$$y_1 = y_0 + h\,\overbrace{f(x_0, y_0)}^{\frac{dy}{dx} = x+y}$$
$$y_1 = y_0 + h(x_0 + y_0)$$
$$y_1 = 1 + 0.02(0 + 1) = 1.02$$

- y_2

$$y_2 = y_1 + h(x_1 + y_1)$$
$$y_2 = 1.02 + h(0.02 + 1.02) = 1.0408$$

- y_3
$= 1.0624$

- $y_4 = 1.0848$

$\boxed{y_5 = 1.1081}$ ✓

Date:

* Improved Eulers / Rungekutta Second

- $y_{n+1} = y_n + h f(x_n, y_n)$

put the answer into this eq

- $y_{n+1} = y_n + \frac{h}{2}\left[f(x_n, y_n) + f(x_{n+1}, y_{n+1}) \right]$

avg out