**Numerical Method UTS**

**Kodingan Python**

1. Python : Jawaban : [1, 2, -2, -1, 3]

from numpy import array

from scipy.linalg import solve

import numpy as np

def gaussElimination(A,b):

#A = array([[4,-2,1],[-2,4,-2],[1,-2,4]])

#b = array([11,-16,17])

x = solve(A, b)

print("X1,X2,X3 =",x)

check = np.dot(A, x) - b

print("A\*x-b harus [0 0 0]")

print("A\*x-b = ", check)

return x,check

A = array([[1, -2, 1, -3, 1], [2, 1, -3, 1, 2], [-1, -1, -1, -1, -1], [1, 1, 1, -1, -1], [2, -2, 1, -1, 1]])

b = array([1, 15, -3, -1, 0])

x, check = gaussElimination(A, b)

#x adalah jawabannya

1. Python: Jawaban : -5.02

import numpy as np

from pypoly import Polynomial

def chunkIt(seq, num):

avg = len(seq) / float(num)

out = []

last = 0.0

while last < len(seq):

out.append(seq[int(last):int(last + avg)])

last += avg

return out

def product(list):

p = 1

for i in list:

p \*= i

return p

def Lagrange(x,X):

T = np.zeros((2,len(X)))

list = []

for i in range(len(X)):

for j in range(len(X)):

if i != j:

list.append((x-X[j][0])/(X[i][0]-X[j][0]))

p = []

for i in chunkIt(list,len(X)):

p.append(product(i))

for i in range(len(X)):

T[0][i] = p[i]

T[1][i] = X[i][1]

list2 = []

for i in range(len(X)):

list2.append(T[0][i]\*T[1][i])

return sum(list2)

# Input

# x kecil buat mau dicari nilai brpa ( misal soalnya tau ln1 sama ln2 , disuru cari ln3 berarti x kecil = 3)

# X gede [x,y]

#Change HERE

x, X = 0.5, [[-1.2, -5.5],[0.3, -5.39],[1.1, -3.4],[1.4, -2.3]]

order = len(X)

equations = np.array([[point[0] \*\* i for i in range(order)] for point in X])

values = np.array([point[1] for point in X])

coefficients = np.linalg.solve(equations, values)

print ('coefficients', list(coefficients))

p = Polynomial(\*coefficients)

print(p)

# p hasilnya , p(x) fungsi hasil interpolasi nya

print(p(x))

#p(x) adalah nilai y pada saat t = 0.5

1. Python (Newton Rapshon) Jawaban: 3.2190

# Python3 code for implementation of Newton

# Raphson Method for solving equations

# An example function whose solution

# is determined using Bisection Method.

# The function is x \*\*2 - 7\*x -7

def func( x ):

#CHANGE HERE

return x\*\*5 - 4\*x\*\*4 + 2\*x\*\*3 + x\*\*2 - x + 10

# Derivative of the above function

# which is 3\*x^x - 2\*x

def derivFunc( x ):

#MANNUAL DERIVATIVE THEN TYPE HERE

return 5\*x\*\*4 - 16\*x\*\*3 + 6\*x\*\*2 + 2\*x - 1

# Function to find the root

def newtonRaphson( x ):

h = func(x) / derivFunc(x)

while abs(h) >= 0.0001:

h = func(x)/derivFunc(x)

# x(i+1) = x(i) - f(x) / f'(x)

x = x - h

print("The value of the root is : ",

"%.4f"% x)

# Driver program to test above

x0 = 1 # Initial values assumed

newtonRaphson(x0)

1. Python (brents) Jawaban: h = 0.519786

def brents(f, x0, x1, max\_iter=50, tolerance=1e-5):

fx0 = f(x0)

fx1 = f(x1)

assert (fx0 \* fx1) <= 0, "Root not bracketed"

if abs(fx0) < abs(fx1):

x0, x1 = x1, x0

fx0, fx1 = fx1, fx0

x2, fx2 = x0, fx0

mflag = True

steps\_taken = 0

while steps\_taken < max\_iter and abs(x1-x0) > tolerance:

fx0 = f(x0)

fx1 = f(x1)

fx2 = f(x2)

if fx0 != fx2 and fx1 != fx2:

L0 = (x0 \* fx1 \* fx2) / ((fx0 - fx1) \* (fx0 - fx2))

L1 = (x1 \* fx0 \* fx2) / ((fx1 - fx0) \* (fx1 - fx2))

L2 = (x2 \* fx1 \* fx0) / ((fx2 - fx0) \* (fx2 - fx1))

new = L0 + L1 + L2

else:

new = x1 - ( (fx1 \* (x1 - x0)) / (fx1 - fx0) )

if ((new < ((3 \* x0 + x1) / 4) or new > x1) or

(mflag == True and (abs(new - x1)) >= (abs(x1 - x2) / 2)) or

(mflag == False and (abs(new - x1)) >= (abs(x2 - d) / 2)) or

(mflag == True and (abs(x1 - x2)) < tolerance) or

(mflag == False and (abs(x2 - d)) < tolerance)):

new = (x0 + x1) / 2

mflag = True

else:

mflag = False

fnew = f(new)

d, x2 = x2, x1

if (fx0 \* fnew) < 0:

x1 = new

else:

x0 = new

if abs(fx0) < abs(fx1):

x0, x1 = x1, x0

steps\_taken += 1

return x1, steps\_taken

# CHANGE HERE

f = lambda x: 63.5688\*x\*\*3 - 34.37463\*x\*\*2 + 0.36 #ganti soal

root, steps = brents(f, -20 , 20, tolerance= 1e-7)

print ("root is:", root)

print ("steps taken:", steps)

1. Python (differensial method) Jawaban: Di paling bawah

#real function

def f(x):

y = x\*\*4 + x\*\*3 + 2\*x\*\*2 - 3\*x + 10 #insert eq here

return y

#Differential eq

def actual\_diff\_f(x):

y = 4\*x\*\*3 + 3\*x\*\*2 + 4\*x - 3 #insert differential equation here

return y

#Forward

def f\_Forward(x,h):

fh = f(x+h)

fx = f(x)

eq = (fh-fx)/h

return eq

#Backward

def f\_Backward(x,h):

fh = f(x-h)

fx = f(x)

eq = (fx-fh)/h

return eq

#Middle

def f\_Middle(x,h):

fh = f(x+h)

fx = f(x-h)

eq = (fh-fx)/(2\*h)

return eq

#Data

x= 1.7

h= 0.02

Forward = f\_Forward(x,h)

print("using forward method, predicted f'(x):",Forward)

Backward = f\_Backward(x,h)

print("using backward method, predicted f'(x):",Backward)

Middle = f\_Middle(x,h)

print("using middle method, predicted f'(x):",Middle)

Actual = actual\_diff\_f(x)

print("actual f'(x):",Actual)