**NC ASSIGNMENT 03**

**Name:** Bilal Ahmed Khan **Sec:** B **Roll No:** 20k-0183

1. **Euler Method of solving Differential Equations:**
2. **First Order Equations:**

def odeEuler(f,y0,t):

'''Approximate the solution of y'=f(y,t) by Euler's method.

Parameters

----------

f : function

Right-hand side of the differential equation y'=f(t,y), y(t\_0)=y\_0

y0 : number

Initial value y(t0)=y0 wher t0 is the entry at index 0 in the array t

t : array

1D NumPy array of t values where we approximate y values. Time step

at each iteration is given by t[n+1] - t[n].

Returns

-------

y : 1D NumPy array

Approximation y[n] of the solution y(t\_n) computed by Euler's method.

'''

y = np.zeros(len(t))

y[0] = y0

for n in range(0,len(t)-1):

y[n+1] = y[n] + f(y[n],t[n])\*(t[n+1] - t[n])

return y

1. **Exponential Equations:**

t = np.linspace(0,2,21)

y0 = 1

f = lambda y,t: y

y = odeEuler(f,y0,t)

y\_true = np.exp(t)

plt.plot(t,y,'b.-',t,y\_true,'r-')

plt.legend(['Euler','True'])

plt.axis([0,2,0,9])

plt.grid(True)

plt.title("Solution of $y'=y , y(0)=1$")

plt.show()

1. **4 RK Method:**

# RK-4 method python program

# function to be solved

def f(x,y):

return x+y

# or

# f = lambda x: x+y

# RK-4 method

def rk4(x0,y0,xn,n):

# Calculating step size

h = (xn-x0)/n

print('\n--------SOLUTION--------')

print('-------------------------')

print('x0\ty0\tyn')

print('-------------------------')

for i in range(n):

k1 = h \* (f(x0, y0))

k2 = h \* (f((x0+h/2), (y0+k1/2)))

k3 = h \* (f((x0+h/2), (y0+k2/2)))

k4 = h \* (f((x0+h), (y0+k3)))

k = (k1+2\*k2+2\*k3+k4)/6

yn = y0 + k

print('%.4f\t%.4f\t%.4f'% (x0,y0,yn) )

print('-------------------------')

y0 = yn

x0 = x0+h

print('\nAt x=%.4f, y=%.4f' %(xn,yn))

# Inputs

print('Enter initial conditions:')

x0 = float(input('x0 = '))

y0 = float(input('y0 = '))

print('Enter calculation point: ')

xn = float(input('xn = '))

print('Enter number of steps:')

step = int(input('Number of steps = '))

# RK4 method call

rk4(x0,y0,xn,step)

1. **LU Decomposition Method:**

import scipy.linalg

A = scipy.array([[1, 2, 3],

[4, 5, 6],

[10, 11, 9]])

P, L, U = scipy.linalg.lu(A)

print(P)

print(L)

print(U)

A = scipy.array([[1, 2, 3],

[4, 5, 6],

[10, 11, 9]])

P, L, U = scipy.linalg.lu(A)

mult = P.dot((L.dot(U)))

print(mult)

1. **LDLt Factorization:**

import math

MAX = 100;

def Cholesky\_ Factorisation (matrix, n):

lower = [[0 for x in range(n + 1)]

for y in range(n + 1)];

# Factorizing a matrix

# into Lower Triangular

for i in range(n):

for j in range(i + 1):

sum1 = 0;

# summation for diagonals

if (j == i):

for k in range(j):

sum1 += pow(lower[j][k], 2);

lower[j][j] = int(math.sqrt(matrix[j][j] - sum1));

else:

# Evaluating L(i, j)

# using L(j, j)

for k in range(j):

sum1 += (lower[i][k] \*lower[j][k]);

if(lower[j][j] > 0):

lower[i][j] = int((matrix[i][j] - sum1) /

lower[j][j]);

# Displaying Lower Triangular

# and its Transpose

print("Lower Triangular\t\tTranspose");

for i in range(n):

# Lower Triangular

for j in range(n):

print(lower[i][j], end = "\t");

print("", end = "\t");

# Transpose of

# Lower Triangular

for j in range(n):

print(lower[j][i], end = "\t");

print("");

# Driver Code

n = 3;

matrix = [[4, 12, -16],

[12, 37, -43],

[-16, -43, 98]];

Cholesky\_Factorisation (matrix, n);

1. **Gauss-Siedel Method:**

# Gauss Seidel Iteration

# Defining equations to be solved

# in diagonally dominant form

f1 = lambda x,y,z: (17-y+2\*z)/20

f2 = lambda x,y,z: (-18-3\*x+z)/20

f3 = lambda x,y,z: (25-2\*x+3\*y)/20

# Initial setup

x0 = 0

y0 = 0

z0 = 0

count = 1

# Reading tolerable error

e = float(input('Enter tolerable error: '))

# Implementation of Gauss Seidel Iteration

print('\nCount\tx\ty\tz\n')

condition = True

while condition:

x1 = f1(x0,y0,z0)

y1 = f2(x1,y0,z0)

z1 = f3(x1,y1,z0)

print('%d\t%0.4f\t%0.4f\t%0.4f\n' %(count, x1,y1,z1))

e1 = abs(x0-x1);

e2 = abs(y0-y1);

e3 = abs(z0-z1);

count += 1

x0 = x1

y0 = y1

z0 = z1

condition = e1>e and e2>e and e3>e

print('\nSolution: x=%0.3f, y=%0.3f and z = %0.3f\n'% (x1,y1,z1))

1. **Jacobi’s Method:**

from pprint import pprint

from numpy import array, zeros, diag, diagflat, dot

def jacobi(A,b,N=25,x=None):

"""Solves the equation Ax=b via the Jacobi iterative method."""

# Create an initial guess if needed

if x is None:

x = zeros(len(A[0]))

# Create a vector of the diagonal elements of A

# and subtract them from A

D = diag(A)

R = A - diagflat(D)

# Iterate for N times

for i in range(N):

x = (b - dot(R,x)) / D

return x

A = array([[2.0,1.0],[5.0,7.0]])

b = array([11.0,13.0])

guess = array([1.0,1.0])

sol = jacobi(A,b,N=25,x=guess)

print "A:"

pprint(A)

print "b:"

pprint(b)

print "x:"

pprint(sol)