Jacob Dennis

Physics 411

23 February 2016

Prof. Gull

Homework 5 – Code and Results

**Forward and Backward Euler**

*Code*

import numpy as np

import matplotlib.pyplot as plt

def yprime(y):

A = np.matrix([[-7.0, 3.0], [3.0, -7.0]])

return np.dot(A, y)

y0 = np.matrix([[2.0], [0.0]])

#Forward Euler Method

Y1 = [y0]

A = np.matrix([[-6.0, 3.0], [3.0, -6.0]])

for i in range(5):

Y1.append(np.dot(A, Y1[i]))

Y1 = np.asarray(Y1)

xRange1 = np.rot90(np.rot90(Y1)[1])

yRange1 = np.rot90(np.rot90(Y1)[0])

#Backward Euler Method

Y2 = [y0]

for i in range(5):

B = np.matrix([[8.0/55.0, 3.0/55.0], [3.0/55.0, 8.0/55.0]])

Y2.append(np.dot(B, Y2[i]))

Y2 = np.asarray(Y2)

xRange2 = np.rot90(np.rot90(Y2)[1])

yRange2 = np.rot90(np.rot90(Y2)[0])

print 'Forward Euler Results:\n', Y1

print '\nBackward Euler Results:\n', Y2

plt.clf()

plt.plot(xRange1[0], yRange1[0], label = 'Forwards Euler Method')

plt.plot(xRange2[0], yRange2[0], label = 'Backwards Euler Method')

plt.legend()

plt.title('Forwards and Backwards Euler Approximations')

plt.xlabel('y[0]')

plt.ylabel('y[1]')

plt.savefig('Homework 5 Problem 1.png')

*Results*

Forward Euler Results:

[[[ 2.00000000e+00]

[ 0.00000000e+00]]

[[ -1.20000000e+01]

[ 6.00000000e+00]]

[[ 9.00000000e+01]

[ -7.20000000e+01]]

[[ -7.56000000e+02]

[ 7.02000000e+02]]

[[ 6.64200000e+03]

[ -6.48000000e+03]]

[[ -5.92920000e+04]

[ 5.88060000e+04]]]

Backward Euler Results:

[[[ 2.00000000e+00]

[ 0.00000000e+00]]

[[ 2.90909091e-01]

[ 1.09090909e-01]]

[[ 4.82644628e-02]

[ 3.17355372e-02]]

[[ 8.75131480e-03]

[ 7.24868520e-03]]

[[ 1.66830135e-03]

[ 1.53169865e-03]]

[[ 3.26209213e-04]

[ 3.13790787e-04]]]

**Bulirsch – Stoer**

*Code*

import numpy as np

def yprime(y, t):

return 2.0 \* np.sin(t) + np.tan(t) \* y

y0 = 1.0

def ModifiedMidpoint(y0, t0, f, H, n):

h = H / n

X = [y0]

Y = [0.0, X[0] + h / 2.0 \* f(X[0], t0)]

X.append(X[0] + h \* f(Y[1], t0 + h \* 1.0/2.0))

for m in range(1, n):

Y.append(Y[m] + h \* f(X[m], t0 + m \* h))

X.append(X[m] + h \* f(Y[m + 1], t0 + (m + 0.5) \* h))

return 0.5 \* (X[n] + Y[n] + h / 2.0 \* f(X[n], t0 + H))

def BulirschStoer(y0, t0, f, H, N):

R = []

errors = []

for n in range(N):

Rrow = [ModifiedMidpoint(y0, t0, f, H, n + 1)]

errorRow= []

for m in range(1, n + 1):

Rnew = Rrow[m - 1] + (Rrow[m - 1] - R[n - 1][m - 1]) / (((n + 1.0) / n)\*\*(2.0 \* m) - 1.0)

Rrow.append(Rnew)

errorRow.append((Rrow[m - 1] - R[n - 1][m - 1]) / (((n + 1.0) / n)\*\*(2.0 \* m) - 1.0))

errors.append(np.array(errorRow))

R.append(np.asarray(Rrow))

return np.array([np.asarray(R), np.asarray(errors)])

results = BulirschStoer(y0, 0.0, yprime, 1.0, 6)

print 'Approximations:\n'

for i in results[0]:

print i

print '\nErrors:\n\n',

for i in results[1]:

print i

*Results*

Approximations:

[ 3.14869866]

[ 3.1668799 3.17294032]

[ 3.16481557 3.16316411 3.16075766]

[ 3.16352043 3.16185526 3.16124944 3.16135592]

[ 3.16280437 3.16153136 3.16130666 3.16132699 3.16132115]

[ 3.16238207 3.16142231 3.16132073 3.16132782 3.16132807 3.1613294 ]

Errors:

[]

[ 0.00606042]

[-0.00165146 -0.00240645]

[-0.00166518 -0.00060581 0.00010648]

[ -1.27300475e-03 -2.24705096e-04 2.03276671e-05 -5.83288346e-06]

[ -9.59763862e-04 -1.01578993e-04 7.08510092e-06 2.51023272e-07 1.33139224e-06]