Zhang_Jeffrey_Code6

April 28, 2022

Code for 5.6.14

[58]:
```
import math
import numpy as np
from sympy import *
from scipy.integrate import quad
import scipy.special as special
from sympy.plotting import plot

def f(t,y):
    return 0.0439*math.log(12000/y)*y

def Adam_Fourth_Order_Predictor_Corrector(h, a, alpha, target):
    t = [a]
    w = [alpha]
    K = [0,0,0,0]
    for i in range(1,4):
        K[0] = h*f(t[i - 1], w[i - 1])
        K[1] = h*f(t[i - 1] +h/2, w[i - 1] + K[0]/2)
        K[2] = h*f(t[i - 1] +h/2, w[i - 1] + K[1]/2)
        K[3] = h*f(t[i - 1] +h, w[i - 1] + K[2])
        w.append(w[i-1] + (K[0] + 2*K[1] + 2*K[2] + K[3])/6)
        t.append(a + i*h)
        print(K)

    print(w)
    i = 4
    while (w[3] < target):
        time = a + i*h
        t3w3 = f(t[3],w[3])
        t2w2 = f(t[2],w[2])
        t1w1 = f(t[1],w[1])
        temp = w[3] + (h/24)*(55*t3w3 - 59*t2w2 + 37*t1w1 - 9*f(t[0],w[0]))
        temp = w[3] + (h/24)*(9*f(time,temp) + 19*t3w3 - 5*t2w2 + t1w1)
        for j in range(3):
            t[j] = t[j + 1]
            w[j] = w[j + 1]
        t[3] = time
```

```
        w[3] = temp
        i += 1
    return t[3]
```

[59]:
```
Adam_Fourth_Order_Predictor_Corrector(0.5, 0, 4000, 11000)
```

```
[96.45815894506005, 96.5561961171459, 96.55628931778065, 96.64178194720265]
[96.64177843982728, 96.71484170142273, 96.71489224292509, 96.7756222620943]
[96.77561991326102, 96.82413601689669, 96.82415728901361, 96.8605660013819]
[4000, 4096.554151960353, 4193.266963392122, 4290.0890921465325]
```

[59]: 58.0

1 AMSC 460 HW 6 Part 2

Consider the following IVP:

$$y'(t) = 2ty(t)^2, \quad y(0) = 1$$

The exact solution of this problem is $y(t) = 1/\left(1 - t^2\right)$ and, clearly, it explodes at $t = 1$ even though $f(t, y) = 2ty^2$ is continuous in both variables and satisfies the Lipschitz condition for y in any bounded interval (c, d). This example shows that typically one expects only local-in-time existence of IVP solutions.

- Set $a = 0.9999$ and solve the IVP above on the domain $[0, a]$ by any first order method with step sizes $a/10^k, k = 3, 4, 5$. Report the absolute error at the time $t = a$ for different k.

We can use Euler's method

[89]:
```
def f(t,y):
    return 2*t*y**2

def y(t):
    return 1/(1 - t**2)

def Euler_Method(a,k):
    n = 10**k
    h = a/n
    t = 0
    w = 1
    for i in range(1, n + 1):
        w = w + h*f(t,w)
        t = i*h
    return w
```

[90]:
```
y(0.9999) - Euler_Method(0.9999,3)
```

[90]: 4891.492319076559

[91]:
```python
y(0.9999) - Euler_Method(0.9999,4)
```

[91]: 4319.866942896124

[92]:
```python
y(0.9999) - Euler_Method(0.9999,5)
```

[92]: 2191.1098275180993

The error is quite high but shrinks drastically as k increases

- Repeat the previous part using any second order method.

We can use the Modified Euler's Method

[85]:
```python
def Modified_Euler_Method(a,k):
    n = 10**k
    h = a/n
    t = 0
    w = 1
    for i in range(n):
        temp = f(t,w)
        w = w + (h/2)*(temp + f(t+h, w + h*temp))
        t = i*h
    return w
```

[86]:
```python
y(0.9999) - Modified_Euler_Method(0.9999,3)
```

[86]: 4630.69429888907

[87]:
```python
y(0.9999) - Modified_Euler_Method(0.9999,4)
```

[87]: 2707.958450564152

[88]:
```python
y(0.9999) - Modified_Euler_Method(0.9999,5)
```

[88]: 472.26252222436415

- Do you observe the expected order of convergence of errors for each method?

Yes, firstly, it is quite obvious that the 2nd order method converges faster than the first order method.

$$|f(t, y_1) - f(t, y_2)| = \left|2ty_1^2 - 2ty_2^2\right|$$