Numerical Analysis Homework 4
APPM 4650

Zachary Vogel

November 20, 2015

Problem 1

The first problem asks for an approximation to the integral $\int_0^1 x^2 e^{-x} dx$ using Guassian Quadrature using $n = 3$. It also asks for a comparison of the results to the exact value of the integral.
First, rewriting the integral we get:

$$\alpha = \int_0^1 x^2 e^{-x} dx = \frac{1}{2}\int_{-1}^{1}\left(\frac{1}{2}x + \frac{1}{2}\right)^2 e^{-\left(\frac{1}{2}x+\frac{1}{2}\right)} dx$$

Then, we use our weights and evaluations to estimate with 3 points:

$$\alpha = \frac{1}{2}\left(\frac{8}{9}\frac{1}{4}e^{-\frac{1}{2}} + \frac{5}{9}\left(\left(\frac{1}{2}\sqrt{\frac{3}{5}}+\frac{1}{2}\right)^2 e^{-\left(\frac{1}{2}\sqrt{\frac{3}{5}}+\frac{1}{2}\right)} + \left(-\frac{1}{2}\sqrt{\frac{3}{5}}+\frac{1}{2}\right)^2 e^{\frac{1}{2}\sqrt{\frac{3}{5}}-\frac{1}{2}}\right)\right)$$

Solving, we get that the integral is approximately $\alpha = 0.160595$. The actual value wolfram alpha gave me was 0.16060. Therefore, we are really really close to the exact value from a 3 point approximation.

Problem 2

Here, use Runge-Kutta 4 to approximate the solution to the following first order system of equations for $0 \le t \le 1$, and $h = 0.2$. It also wants a comparison to the actual solution. The system is defined by:

$$u_1' = 3u_1 + 2u_2 - (2t^2+1)e^{2t} \quad u_1(0) = 1$$

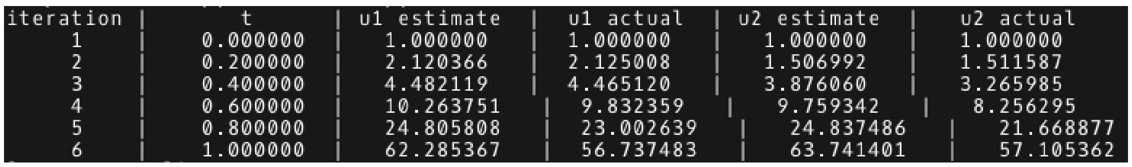
$$u_2' = 4u_1 + u_2 + (t^2 + 2t - 4)e^{2t} \quad u_2(0) = 1$$

The exact solutions for this problem are:

$$u_1(t) = \frac{1}{3}e^{5t} - \frac{1}{3}e^{-t} + e^{2t}$$

$$u_2(t) = \frac{1}{3}e^{5t} + \frac{2}{3}e^{-t} + t^2 e^{2t}$$

I solved this problem with a python script that I wrote. The code for it is in Appendix 1, here is the table of results for the problem. The problem of methods like this is visible. As the iteration number increases, the error gets bigger with every step.



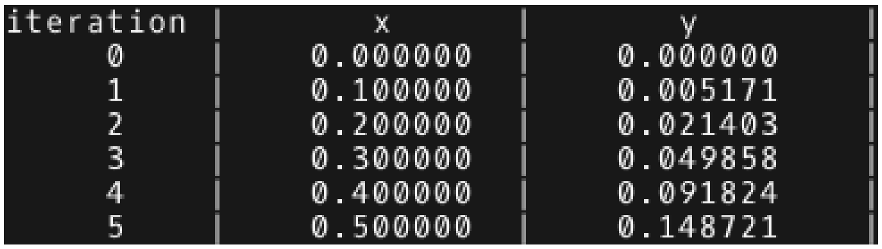
```
iteration |       t       | u1 estimate  |  u1 actual  | u2 estimate  |   u2 actual
    1     |   0.000000    |   1.000000   |  1.000000   |   1.000000   |  1.000000
    2     |   0.200000    |   2.120366   |  2.125008   |   1.506992   |  1.511587
    3     |   0.400000    |   4.482119   |  4.465120   |   3.876060   |  3.265985
    4     |   0.600000    |   10.263751   |  9.832359   |   9.759342   |  8.256295
    5     |   0.800000    |   24.805808   |  23.002639   |   24.837486   |   21.668877
    6     |   1.000000    |   62.285367   |  56.737483   |   63.741401   |   57.105362
```

Problem 3

Here, we are again given a differential equation, $y' = x + y$ with $y(0) = 0$. Find $y(x)$ for $0 \leq x \leq 0.5$ with a step size of $h = 0.1$. It wants the solution using several different methods, while noting that any seed values can be approximated by $y \approx \frac{x^2}{2}$ when $x << 1$.

(a)

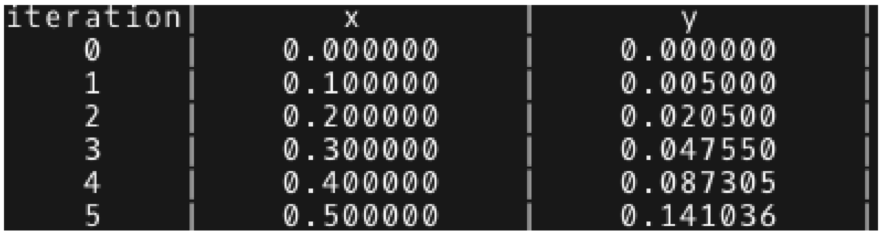
First, the solution with Runge-Kutta 4.
Once again, I used a python script I wrote to solve this problem. The code is in Appendix 2, here are the results.



```
iteration |       x       |       y       |
    0     |   0.000000    |   0.000000    |
    1     |   0.100000    |   0.005171    |
    2     |   0.200000    |   0.021403    |
    3     |   0.300000    |   0.049858    |
    4     |   0.400000    |   0.091824    |
    5     |   0.500000    |   0.148721    |
```

(b)

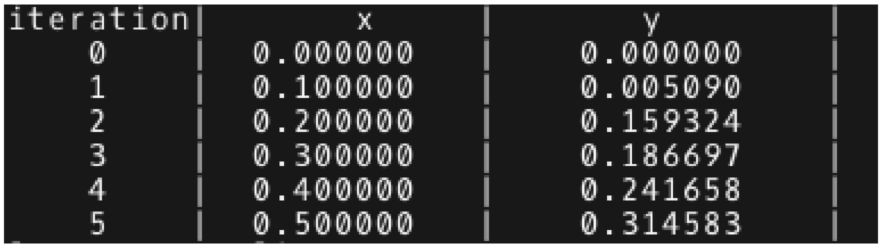
Next with the improved Euler method.
Yay, python scripts. Here are the results, code is in Appendix 3:



```
iteration|       x        |       y        |
    0    |   0.000000     |   0.000000     |
    1    |   0.100000     |   0.005000     |
    2    |   0.200000     |   0.020500     |
    3    |   0.300000     |   0.047550     |
    4    |   0.400000     |   0.087305     |
    5    |   0.500000     |   0.141036     |
```

(c)

Finally, solve with the Predictor-corrector using the Adams-bashforth 4-step for the predictor and the Adams-Moulton 3-step as the corrector. Again I implemented in python. I didn't want to do boring things with the seed values, so my code is relatively unusual (various step sizes until I have enough seed values). Here is the table of results, the code is in Appendix 4. This doesn't seem correct, probably because of my attempt at a good way to iterate.



```
iteration|        x         |         y         |
    0    |   0.000000       |    0.000000       |
    1    |   0.100000       |    0.005090       |
    2    |   0.200000       |    0.159324       |
    3    |   0.300000       |    0.186697       |
    4    |   0.400000       |    0.241658       |
    5    |   0.500000       |    0.314583       |
```

A Appendix 1

```
import numpy as np

u1diff="3*u1+2*u2-(2*t**2+1)*np.exp(2*t)"

u2diff="4*u1+u2+(t**2+2*t-4)*np.exp(2*t)"

u1act="1/3*np.exp(5*t)-1/3*np.exp(-t)+np.exp(2*t)"
u2act="1/3*np.exp(5*t)+2/3*np.exp(-t)+t**2*np.exp(2*t)"

u10=1.0
u20=1.0
t0=0.0

tmax=1.0
h=0.2

u1a=np.array([u10, 0, 0, 0, 0, 0])
u2a=np.array([u20, 0, 0, 0, 0, 0])
ta=np.array([t0, t0+h, t0+2*h, 3*h, 4*h, 5*h])


u1e=np.array([0.0,0,0,0,0,0])
u2e=np.array([0.0,0,0,0,0,0])


def my_range(start,end,step):
    while start<=end:
        yield start
        start+=step

for t in my_range(t0,tmax-h,h):
    n=t*5
    u1e[n]=eval(u1act)
    u2e[n]=eval(u2act)
    u1=u1a[n]
    u2=u2a[n]
    t=ta[n]
    k1u1=eval(u1diff)
    k1u2=eval(u2diff)
    u1=u1a[n]+h/2*k1u1
    u2=u2a[n]+h/2*k1u2
    t=ta[n]+h/2
    k2u1=eval(u1diff)
    k2u2=eval(u2diff)
    u1=u1a[n]+h/2*k2u1
    u2=u2a[n]+h/2*k2u2
    t=ta[n]+h/2
    k3u1=eval(u1diff)
    k3u2=eval(u2diff)
    u1=u1a[n]+h*k3u1
    u2=u1a[n]+h*k3u2
    t=ta[n]+h
    k4u1=eval(u1diff)
    k4u2=eval(u2diff)
    u1a[n+1]=u1a[n]+h/6*(k1u1+2*k2u1+2*k3u1+k4u1)
    u2a[n+1]=u1a[n]+h/6*(k1u2+2*k2u2+2*k3u2+k4u2)
```

```
t=1
u1e[5]=eval(u1act)
u2e[5]=eval(u2act)

r="iteration |       t       | u1 estimate   |  u1 actual   | u2 estimate   |   u2 actual"
print(r)
for i in my_range(0,5,1):
    s="     %d     |     %f   |    %f    |   %f    |    %f    |    %f    " %(i+1,ta[i],u1a[i], u1e[i],u2a[i
    print(s)
```

B Appendix 2

```
import numpy as np

ydiff="x+y"

y0=0.0
x0=0.0

xmax=0.5
h=0.1

ya=np.array([y0, 0, 0, 0, 0, 0])
xa=np.array([x0,h,2*h,3*h,4*h,5*h])


def my_range(start,end,step):
    while start<=end:
        yield start
        start+=step

r="iteration |       x       |       y         |"
print(r)


for x in my_range(x0,xmax-h,h):
    n=x*10
    y=ya[n]
    x=xa[n]
    k1=eval(ydiff)
    y=ya[n]+h/2*k1
    x=xa[n]+h/2
    k2=eval(ydiff)
    y=ya[n]+h/2*k2
    x=xa[n]+h/2
    k3=eval(ydiff)
    y=ya[n]+h*k3
    x=xa[n]+h
    k4=eval(ydiff)
    ya[n+1]=ya[n]+h/6*(k1+2*k2+2*k3+k4)
    s="     %d     |     %f   |     %f     |" %(n,xa[n],ya[n])
    print(s)

s="     %d     |     %f   |     %f     |" %(n+1,xa[n+1],ya[n+1])
print(s)
```

C Appendix 3

```
import numpy as np

ydiff="y+x"

y0=0.0
x0=0.0

xmax=0.5
h=0.1

yest=np.array([y0,0,0,0,0,0])

#improved euler method is:
#y_{j+1}=y_j+\frac{h}{2}(f(x_j,y_j)+f(x_{j+1},y_j+hf(x_j,y_j)))


def my_range(start,end,step):
    while start<=end:
        yield start
        start+=step

y=yest[0]


r="iteration|       x         |       y         |"
print(r)

for x in my_range(x0,xmax-h,h):
    it=x*10
    m1=eval(ydiff)
    x=x+h
    y=y+m1*h
    m2=eval(ydiff)
    yest[it+1]=yest[it]+h*(m1+m2)/2
    s="    %d    |    %f    |    %f    |" %(it,x-h,yest[it])
    print(s)

s="    %d    |    %f    |    %f    |" %(it+1,x,yest[it+1])
print(s)
```

D Appendix 4

```
import numpy as np

#predictor:
#y_{i+1}=y_i+\frac{h}{24}(55f_i-59*f_{i-1}+37f_{i-2}-9f_{i-3})

#corrector:
#y_{i+1}=y_i+\frac{h}{24}(9f_{i+1}+19f_i-5f_{i-1}+f_{i-2})

ydiff="x+y"

seed="x**2/2"

x0=0.0
y0=0.0
```

```
xmax=0.5
h=0.1

def my_range(start,end,step):
    while start<=end:
        yield start
        start+=step




#start with step of 0.025
x=x0
y=y0
f0=eval(ydiff)

x=0.025
y=eval(seed)
f1=eval(ydiff)

x=0.05
y=eval(seed)
f2=eval(ydiff)

x=0.075
y=eval(seed)
f3=eval(ydiff)

y1=y+0.025/24*(55*f3-59*f2+37*f1-9*f0)

#now we have the first predictor value, let us correct with h=0.033
x=1.0/30.0
y=eval(seed)
f1=eval(ydiff)

x=2.0/30.0
y=eval(seed)
ye=y
f2=eval(ydiff)

x=0.1
y=y1
f3=eval(ydiff)

y1=ye+0.033/24*(9*f3+19*f2-5*f1+f0)

#now we do the second predictor with h=0.05
#thus, we need f values at x=0,.05,0.1,0.15


x=0.05
y=eval(seed)
f2=eval(ydiff)

x=0.1
y=y1
f4=eval(ydiff)
```

```
x=0.15
y=eval(seed)
f5=eval(ydiff)

y2=y+h/24*(55*f5-59*f4+37*f2-9*f0)

#still using h=0.05
x=0.2
y=y2
f6=eval(ydiff)

y2=0.15+0.05/25*(9*f6+19*f5-5*f4+f2)

y=y2
f6=eval(ydiff)

x=0.1
y=y1
f8=eval(ydiff)

x=0.25
y=eval(seed)
f7=eval(ydiff)

y3=y+.05/24*(55*f7-59*f6+37*f5-9*f4)

y3=y2+0.1/24*(9*y3+19*f6-5*f8+0)
x=0.3
y=y3
f9=eval(ydiff)

y4=y3+0.1/24*(55*f9-59*f6+37*f8-0)
x=0.4
y=y4
f10=eval(ydiff)

y4=y3+0.1/24*(9*f10+19*f9-5*f6+f8)
y=y4
f10=eval(ydiff)

y5=y4+0.1/24*(55*f10-59*f9+37*f6-9*f8)
x=0.5
y=y5
f11=eval(ydiff)

y5=y4+0.1/24*(9*f11+19*f10-5*f9+f6)

x=np.array([0.0, 0.1, 0.2, 0.3, 0.4, 0.5])
y=np.array([0, y1, y2, y3, y4, y5])

print("iteration|       x      |       y        |")

for n in my_range(0,5,1):
    s="    %d    |  %f   |    %f    |" %(n,x[n],y[n])
    print(s)
```