MATH 151B - Applied Numerical Methods - Homework 3

Darren Tsang, 405433124, Discussion 1A

Produced on Friday, Aug. 28 2020 @ 04:15:05 PM

Question 1

Part A

Note: This part is the exact same as Question 3A on the Midterm, so I will do the exact same derivation as I did on the Midterm.

We are given the following IVP:

$$\frac{dy}{dx} = x - x^2 = f(x, y), \qquad 0 \leq x \leq 1, \qquad y(0) = 0$$

The following is Euler's Method:

$$w_0 = \alpha,$$
$$w_{i+1} = w_i + hf(x_i, w_i)$$

Applied to the IVP we are given, Euler's Method is:

$$w_0 = 0,$$
$$w_i = w_{i-1} + h(x_{i-1} - (x_{i-1})^2)$$

Since $x_{i-1} = a + (i-1)h = 0 + (i-1)h = (i-1)h$:

$$w_0 = 0,$$
$$w_i = w_{i-1} + h((i-1)h - (i-1)^2h^2) = w_{i-1} + (i-1)h(h - (i-1)h^2)$$

We look at the first couple approximations:

$$w_0 = 0,$$
$$w_1 = 0 + (1-1)h(h - (1-1)h^2) = 0 + 0,$$
$$w_2 = 0 + (2-1)h(h - (2-1)h^2) = 0 + h(h - h^2) = h(h - h^2),$$
$$w_3 = h(h - h^2) + (3-1)h(h - (3-1)h^2) = h(h - h^2) + 2h(h - 2h^2)$$
$$w_4 = h(h - h^2) + 2h(h - 2h^2) + (4-1)h(h - (4-1)h^2) = h(h - h^2) + 2h(h - 2h^2) + 3h(h - 3h^2)$$

We notice a pattern for w_i:

$$w_i = h(h - h^2) + 2h(h - 2h^2) + ... + (i-1)h(h - (i-1)h^2) = h^2 \sum_{n=1}^{i} \left[(n-1) - (n-1)^2 h\right],$$

which is an explicit formula for w_i in terms of i and h.

Part B

We are given the following solution to the IVP:

$$y(x) = \frac{x^2}{2} - \frac{x^3}{3}$$

Thus, $y(x_i)$ can be written as the following:

$$y(x_i) = y(a + ih) = y(ih) = \frac{(ih)^2}{2} - \frac{(ih)^3}{3}$$

We take the limit:

$$\lim_{h \to 0} |w_i - y(x)| = \lim_{h \to 0} \left| \left(h^2 \sum_{n=1}^{i} [(n-1) - (n-1)^2 h] \right) - \frac{(ih)^2}{2} + \frac{(ih)^3}{3} \right| = 0,$$

which means Euler's method is convergent, as desired.

Question 2

```
import math
import pandas as pd

def rkf12(func, a, b, alpha, tol, hmax, hmin):
  h = [hmax]
  t = [a]
  w = [alpha]
  all_R = [0]

  i = 0
  flag = True
  while(flag):
    k1 = h[i]*func(t[i], w[i])
    k2 = h[i]*func(t[i] + h[i], w[i] + k1)
    R = abs(.5*(k1 - k2)/h[i])
    all_R.append(R)

    if(R <= tol):
      t.append(t[i] + h[i])
      w.append(w[i] + k1)
    else:
      t.append(t[-1])
      w.append(w[-1])


    delta = .84 * ((tol/R)**.25)
    if(delta <= .1):
      temp_h = .1*h[i]
    elif(delta >= 4):
      temp_h = 4*h[i]
    else:
      temp_h = delta*h[i]

    if (temp_h > hmax):
      temp_h = hmax

    if (t[i+1] >= b):
      print("Stopped because t too big!")
      temp_h = 0
      flag = False
    elif (temp_h > b-t[i+1]):
      temp_h = b-t[i+1]
    elif (temp_h < hmin):
      print("Stopped because h too small!")
      temp_h = 0
      flag = False

    h.append(temp_h)
    i += 1

  return(t, h, w, all_R)
```

```
def rkf45(func, a, b, alpha, tol, hmax, hmin):
  h = [hmax]
  t = [a]
  w = [alpha]
  all_R = [0]

  i = 0
  flag = True
  while(flag):
    k1 = h[i]*func(t[i], w[i])
    k2 = h[i]*func(t[i] + h[i]/4, w[i] + k1/4)
    k3 = h[i]*func(t[i] + 3*h[i]/8, w[i] + 3*k1/32 + 9*k2/32)
    k4 = h[i]*func(t[i] + 12*h[i]/13, w[i] + 1932*k1/2197 - 7200*k2/2197 + 7296*k3/2197)
    k5 = h[i]*func(t[i] + h[i], w[i] + 439*k1/216 - 8*k2 + 3680*k3/513 - 845*k4/4104)
    k6 = h[i]*func(t[i] + .5*h[i], w[i] - 8*k1/27 + 2*k2 - 3544*k3/2565 + 1859*k4/4104 - 11*k5/40)

    R = abs((k1/360 - 128*k3/4275 - 2197*k4/75240 + k5/50 + 2*k6/55)/h[i])
    all_R.append(R)

    if(R <= tol):
      t.append(t[i] + h[i])
      w.append(w[i] + 25*k1/216 + 1408*k3/2565 + 2197*k4/4104 - k5/5)
    else:
      t.append(t[-1])
      w.append(w[-1])


    delta = .84 * ((tol/R)**.25)
    if(delta <= .1):
      temp_h = .1*h[i]
    elif(delta >= 4):
      temp_h = 4*h[i]
    else:
      temp_h = delta*h[i]

    if (temp_h > hmax):
      temp_h = hmax

    if (t[i+1] >= b):
      print("Stopped because t too big!")
      temp_h = 0
      flag = False
    elif (temp_h > b-t[i+1]):
      temp_h = b-t[i+1]
    elif (temp_h < hmin):
      print("Stopped because h too small!")
      temp_h = 0
      flag = False

    h.append(temp_h)
    i += 1

  return(t, h, w, all_R)
```

```
def f(t, y):
  return(y - t*t + 1)

def y_act(t):
  return(t*t + 2*t + 1 - .5*math.exp(t))
```

My code for RKF12 and RKF45 is shown above. For RKF45, the code is pretty much the implementation of the pseudocode presented on the course textbook. For RKF12, the only thing that is different from RKF45 are the k constants. In RKF12, we only use k_1 and k_2 as defined below.

Euler's Method can be written as the following:

$$w_{i+1} = w_i + k_1$$

and Modifed Euler's Method can be written as the following:

$$w_{i+1} = w_i + \frac{k_1 + k_2}{2},$$

where $k_1 = hf(t_i, w_i)$ and $k_2 = hf(t_i + h, w_i + k_1)$.

Also, I will be using the following IVP to test my RKF12() and RKF45() functions:

$$\frac{dy}{dt} = y - t^2 - 1, \qquad y(0) = .5, \qquad 0 \leq t \leq 2$$

Note that the actual solution is for the IVP is:

$$y(t) = (t+1)^2 - .5e^t = t^2 + 2t + 1 - .5e^t$$

```
a = 0
b = 2
alpha = .5
tol = .075
hmax = .1
hmin = .01

t_rkf12, h_rkf12, w_rkf12, R_rkf12 = rkf12(f, a, b, alpha, tol, hmax, hmin)
```

```
## Stopped because t too big!
```

```
t_rkf45, h_rkf45, w_rkf45, R_rkf45 = rkf45(f, a, b, alpha, tol, hmax, hmin)
```

```
## Stopped because t too big!
```

```
actual_val12 = [y_act(i) for i in t_rkf12]
actual_val45 = [y_act(i) for i in t_rkf45]

diff_12 = []
for i in range(0, len(actual_val12)):
  diff_12.append(abs(actual_val12[i] - w_rkf12[i]))

diff_45 = []
for i in range(0, len(actual_val45)):
  diff_45.append(abs(actual_val45[i] - w_rkf45[i]))
```

```
first = pd.DataFrame(data={'i': range(0, len(t_rkf12)),
                    't_i':t_rkf12,
                    'h_i':h_rkf12,
                    'R_i':R_rkf12,
                    'w_i':w_rkf12,
                    'y_act(t_i)':actual_val12,
                    '|w_i - y_act(t_i)|':diff_12}).to_string(index=False)

second = pd.DataFrame(data={'i': range(0, len(t_rkf45)),
                    't_i':t_rkf45,
                    'h_i':h_rkf45,
                    'R_i':R_rkf45,
                    'w_i':w_rkf45,
                    'y_act(t_i)':actual_val45,
                    '|w_i - y_act(t_i)|':diff_45}).to_string(index=False)
```

```
print(first)
```

```
##   i       t_i       h_i       R_i       w_i  y_act(t_i)  |w_i - y_act(t_i)|
##   0  0.000000  0.100000  0.000000  0.500000    0.500000           0.000000
##   1  0.100000  0.085461  0.070000  0.650000    0.657415           0.007415
##   2  0.185461  0.076592  0.057880  0.790157    0.803432           0.013275
##   3  0.262053  0.071165  0.050100  0.924633    0.942980           0.018347
##   4  0.333218  0.067975  0.044858  1.056713    1.079745           0.023032
##   5  0.401193  0.066337  0.041168  1.188970    1.216539           0.027569
```

```
##   6  0.467530  0.065852  0.038452  1.323502    1.355621            0.032118
##   7  0.533382  0.066296  0.036351  1.462116    1.498916            0.036801
##   8  0.599678  0.067559  0.034625  1.606483    1.648203            0.041720
##   9  0.667237  0.069625  0.033102  1.758279    1.805256            0.046977
##  10  0.736862  0.072567  0.031644  1.919326    1.972004            0.052678
##  11  0.809428  0.076573  0.030118  2.091771    2.150719            0.058948
##  12  0.886001  0.082012  0.028377  2.278348    2.344295            0.065947
##  13  0.968014  0.089594  0.026217  2.482834    2.556723            0.073889
##  14  1.057608  0.100000  0.023302  2.710921    2.794013            0.083091
##  15  1.157608  0.100000  0.018859  2.970160    3.064116            0.093956
##  16  1.257608  0.100000  0.010744  3.233170    3.338294            0.105124
##  17  1.357608  0.100000  0.001819  3.498330    3.614873            0.116543
##  18  1.457608  0.100000  0.007999  3.763853    3.892001            0.128148
##  19  1.557608  0.100000  0.018799  4.027776    4.167632            0.139856
##  20  1.657608  0.100000  0.030679  4.287939    4.439507            0.151568
##  21  1.757608  0.096119  0.043747  4.541967    4.705126            0.163159
##  22  1.853726  0.086982  0.055679  4.777725    4.951973            0.174248
##  23  1.940708  0.059292  0.063193  4.981386    5.165925            0.184538
##  24  2.000000  0.000000  0.051159  5.112720    5.305472            0.192752
```

```
print(second)
```

```
##   i  t_i  h_i           R_i       w_i  y_act(t_i)  |w_i - y_act(t_i)|
##   0  0.0  0.1  0.000000e+00  0.500000    0.500000        0.000000e+00
##   1  0.1  0.1  1.691907e-07  0.657415    0.657415        1.592885e-08
##   2  0.2  0.1  1.627018e-07  0.829299    0.829299        3.293287e-08
##   3  0.3  0.1  1.555304e-07  1.015071    1.015071        5.106203e-08
##   4  0.4  0.1  1.476049e-07  1.214088    1.214088        7.036491e-08
##   5  0.5  0.1  1.388458e-07  1.425639    1.425639        9.088786e-08
##   6  0.6  0.1  1.291654e-07  1.648941    1.648941        1.126740e-07
##   7  0.7  0.1  1.184670e-07  1.883124    1.883124        1.357621e-07
##   8  0.8  0.1  1.066435e-07  2.127230    2.127230        1.601849e-07
##   9  0.9  0.1  9.357642e-08  2.380199    2.380198        1.859679e-07
##  10  1.0  0.1  7.913509e-08  2.640859    2.640859        2.131270e-07
##  11  1.1  0.1  6.317495e-08  2.907917    2.907917        2.416665e-07
##  12  1.2  0.1  4.553626e-08  3.179942    3.179942        2.715764e-07
##  13  1.3  0.1  2.604250e-08  3.455352    3.455352        3.028291e-07
##  14  1.4  0.1  4.498560e-09  3.732400    3.732400        3.353764e-07
##  15  1.5  0.1  1.931117e-08  4.009156    4.009155        3.691449e-07
##  16  1.6  0.1  4.562500e-08  4.283484    4.283484        4.040313e-07
##  17  1.7  0.1  7.470628e-08  4.553027    4.553026        4.398974e-07
##  18  1.8  0.1  1.068461e-07  4.815177    4.815176        4.765634e-07
##  19  1.9  0.1  1.423660e-07  5.067053    5.067053        5.138008e-07
##  20  2.0  0.0  1.816216e-07  5.305473    5.305472        5.513241e-07
```

As you can see on this page and the previous page, when we use $a = 0, b = 2, \alpha = .5, tol = .075, hmax = .1, hmin = .01$, the absolute error for the approximation of $y(2)$ is .192752 and $5.513241 \cdot 10^{-7}$ for RKF12 and RKF45 respectively. We can see that the error for RKF12 is much larger. Furthermore, RKF12 took 24 iterations to run, while RKF45 only took 20 iterations to run.

In this case, RKF12 took $24 * 2 = 48$ calls to the function f, while RKF45 took $20 * 6 = 120$ calls to the function f. Since RKF12 took less calls, we can say that RKF12 was computationally less expensive in this case. We should note that there is a tradeoff though; RKF12 was less computationally heavy, but it resulted in a much less accurate approximation.

```
a = 0
b = 2
alpha = .5
tol = .025
hmax = .1
hmin = .01

t_rkf12, h_rkf12, w_rkf12, R_rkf12 = rkf12(f, a, b, alpha, tol, hmax, hmin)
```

```
## Stopped because t too big!
```

```
t_rkf45, h_rkf45, w_rkf45, R_rkf45 = rkf45(f, a, b, alpha, tol, hmax, hmin)
```

```
## Stopped because t too big!
```

```
actual_val12 = [y_act(i) for i in t_rkf12]
actual_val45 = [y_act(i) for i in t_rkf45]

diff_12 = []
for i in range(0, len(actual_val12)):
  diff_12.append(abs(actual_val12[i] - w_rkf12[i]))

diff_45 = []
for i in range(0, len(actual_val45)):
  diff_45.append(abs(actual_val45[i] - w_rkf45[i]))
```

```
first = pd.DataFrame(data={'i': range(0, len(t_rkf12)),
                    't_i':t_rkf12,
                    'h_i':h_rkf12,
                    'R_i':R_rkf12,
                    'w_i':w_rkf12,
                    'y_act(t_i)':actual_val12,
                    '|w_i - y_act(t_i)|':diff_12}).to_string(index=False)

second = pd.DataFrame(data={'i': range(0, len(t_rkf45)),
                    't_i':t_rkf45,
                    'h_i':h_rkf45,
                    'R_i':R_rkf45,
                    'w_i':w_rkf45,
                    'y_act(t_i)':actual_val45,
                    '|w_i - y_act(t_i)|':diff_45}).to_string(index=False)
```

```
print(first)
```

```
##   i       t_i       h_i       R_i       w_i  y_act(t_i)  |w_i - y_act(t_i)|
##   0  0.000000  0.100000  0.000000  0.500000    0.500000           0.000000
##   1  0.000000  0.064937  0.070000  0.500000    0.500000           0.000000
##   2  0.000000  0.046684  0.046594  0.500000    0.500000           0.000000
##   3  0.000000  0.036334  0.033924  0.500000    0.500000           0.000000
##   4  0.000000  0.030053  0.026590  0.500000    0.500000           0.000000
##   5  0.030053  0.026039  0.022088  0.545080    0.545755           0.000675
```

```
##   6  0.056092  0.023431  0.018983  0.585288    0.586483           0.001195
##   7  0.079523  0.021691  0.016947  0.622360    0.623985           0.001625
##   8  0.101214  0.020512  0.015567  0.657413    0.659416           0.002003
##   9  0.121726  0.019707  0.014607  0.691200    0.693547           0.002347
##  10  0.141433  0.019162  0.013925  0.724236    0.726908           0.002672
##  11  0.160595  0.018800  0.013434  0.756893    0.759876           0.002984
##  12  0.179395  0.018569  0.013076  0.789437    0.792725           0.003289
##  13  0.197964  0.018435  0.012812  0.822068    0.825658           0.003591
##  14  0.216399  0.018374  0.012615  0.854936    0.858828           0.003893
##  15  0.234773  0.018367  0.012466  0.888158    0.892354           0.004196
##  16  0.253140  0.018402  0.012353  0.921825    0.926328           0.004503
##  17  0.271542  0.018469  0.012265  0.956010    0.960825           0.004815
##  18  0.290011  0.018563  0.012196  0.990774    0.995907           0.005133
##  19  0.308574  0.018679  0.012141  1.026169    1.031625           0.005457
##  20  0.327253  0.018813  0.012096  1.062237    1.068025           0.005788
##  21  0.346066  0.018963  0.012058  1.099019    1.105147           0.006127
##  22  0.365030  0.019128  0.012024  1.136552    1.143027           0.006475
##  23  0.384157  0.019306  0.011994  1.174871    1.181703           0.006832
##  24  0.403463  0.019497  0.011966  1.214009    1.221208           0.007199
##  25  0.422959  0.019700  0.011940  1.254001    1.261577           0.007576
##  26  0.442660  0.019917  0.011914  1.294881    1.302846           0.007965
##  27  0.462577  0.020147  0.011887  1.336686    1.345050           0.008365
##  28  0.482724  0.020392  0.011861  1.379452    1.388229           0.008777
##  29  0.503116  0.020651  0.011833  1.423222    1.432424           0.009202
##  30  0.523767  0.020926  0.011804  1.468037    1.477678           0.009641
##  31  0.544693  0.021219  0.011774  1.513943    1.524038           0.010095
##  32  0.565913  0.021531  0.011741  1.560992    1.571555           0.010564
##  33  0.587444  0.021864  0.011706  1.609238    1.620287           0.011049
##  34  0.609308  0.022220  0.011669  1.658741    1.670293           0.011552
##  35  0.631527  0.022601  0.011628  1.709568    1.721642           0.012074
##  36  0.654128  0.023010  0.011584  1.761792    1.774408           0.012615
##  37  0.677139  0.023452  0.011536  1.815497    1.828675           0.013179
##  38  0.700591  0.023929  0.011483  1.870772    1.884537           0.013765
##  39  0.724520  0.024448  0.011424  1.927723    1.942099           0.014376
##  40  0.748968  0.025013  0.011360  1.986466    2.001481           0.015014
##  41  0.773981  0.025631  0.011288  2.047135    2.062817           0.015682
##  42  0.799612  0.026313  0.011207  2.109883    2.126265           0.016382
##  43  0.825925  0.027067  0.011117  2.174888    2.192005           0.017117
##  44  0.852991  0.027907  0.011014  2.242358    2.260249           0.017890
##  45  0.880898  0.028851  0.010896  2.312538    2.331245           0.018707
##  46  0.909749  0.029921  0.010760  2.385720    2.405293           0.019572
##  47  0.939670  0.031145  0.010602  2.462260    2.482752           0.020492
##  48  0.970815  0.032564  0.010415  2.542592    2.564065           0.021473
##  49  1.003380  0.034233  0.010191  2.627263    2.649788           0.022525
##  50  1.037613  0.036232  0.009919  2.716972    2.740631           0.023660
##  51  1.073846  0.038682  0.009581  2.812637    2.837529           0.024892
##  52  1.112527  0.041774  0.009151  2.915511    2.941753           0.026242
##  53  1.154302  0.045841  0.008584  3.027374    3.055111           0.027738
##  54  1.200142  0.051514  0.007805  3.150913    3.180332           0.029419
##  55  1.251656  0.060219  0.006665  3.290545    3.321892           0.031347
##  56  1.311876  0.076303  0.004829  3.454577    3.488204           0.033627
##  57  1.388179  0.100000  0.001278  3.663157    3.699627           0.036470
##  58  1.488179  0.100000  0.007012  3.936769    3.976524           0.039755
##  59  1.588179  0.091556  0.017713  4.208978    4.251258           0.042280
```

```
##  60  1.588179  0.075718  0.026609  4.208978    4.251258           0.042280
##  61  1.663897  0.066119  0.021406  4.412406    4.456424           0.044018
##  62  1.730016  0.055654  0.024797  4.587216    4.632616           0.045401
##  63  1.730016  0.046454  0.025640  4.587216    4.632616           0.045401
##  64  1.776471  0.040669  0.021189  4.707731    4.754307           0.046577
##  65  1.817140  0.035607  0.021183  4.811514    4.859162           0.047648
##  66  1.852747  0.031371  0.020659  4.900869    4.949508           0.048639
##  67  1.884117  0.027898  0.019899  4.978298    5.027861           0.049563
##  68  1.912016  0.025073  0.019079  5.046046    5.096478           0.050432
##  69  1.937089  0.022773  0.018289  5.105976    5.157229           0.051252
##  70  1.959862  0.020892  0.017573  5.159577    5.211608           0.052032
##  71  1.980753  0.019247  0.016944  5.208015    5.260790           0.052775
##  72  2.000000  0.000000  0.016322  5.251986    5.305472           0.053486
```

```
print(second)
```

```
##   i  t_i  h_i           R_i       w_i  y_act(t_i)  |w_i - y_act(t_i)|
##   0  0.0  0.1  0.000000e+00  0.500000    0.500000        0.000000e+00
##   1  0.1  0.1  1.691907e-07  0.657415    0.657415        1.592885e-08
##   2  0.2  0.1  1.627018e-07  0.829299    0.829299        3.293287e-08
##   3  0.3  0.1  1.555304e-07  1.015071    1.015071        5.106203e-08
##   4  0.4  0.1  1.476049e-07  1.214088    1.214088        7.036491e-08
##   5  0.5  0.1  1.388458e-07  1.425639    1.425639        9.088786e-08
##   6  0.6  0.1  1.291654e-07  1.648941    1.648941        1.126740e-07
##   7  0.7  0.1  1.184670e-07  1.883124    1.883124        1.357621e-07
##   8  0.8  0.1  1.066435e-07  2.127230    2.127230        1.601849e-07
##   9  0.9  0.1  9.357642e-08  2.380199    2.380198        1.859679e-07
##  10  1.0  0.1  7.913509e-08  2.640859    2.640859        2.131270e-07
##  11  1.1  0.1  6.317495e-08  2.907917    2.907917        2.416665e-07
##  12  1.2  0.1  4.553626e-08  3.179942    3.179942        2.715764e-07
##  13  1.3  0.1  2.604250e-08  3.455352    3.455352        3.028291e-07
##  14  1.4  0.1  4.498560e-09  3.732400    3.732400        3.353764e-07
##  15  1.5  0.1  1.931117e-08  4.009156    4.009155        3.691449e-07
##  16  1.6  0.1  4.562500e-08  4.283484    4.283484        4.040313e-07
##  17  1.7  0.1  7.470628e-08  4.553027    4.553026        4.398974e-07
##  18  1.8  0.1  1.068461e-07  4.815177    4.815176        4.765634e-07
##  19  1.9  0.1  1.423660e-07  5.067053    5.067053        5.138008e-07
##  20  2.0  0.0  1.816216e-07  5.305473    5.305472        5.513241e-07
```

We ran the same code, but this time we adjust the tolerance to be .025. As we can see from the two tables directly above, the absolute error for the approximation of $y(2)$ is .053486 and $5.513241 \cdot 10^{-7}$ for RKF12 and RKF45 respectively. Just like last time, RKF12 took more iterations and had a less accurate approximation when compared to RKF45.

The thing that is different is the difference between the number of iterations; RKF12 took 72 while RKF45 still only took 20. This is a huge difference that was not seen when tol was larger. RKF12 made $72 * 2 = 144$ calls to f, while RKF45 only made $20 * 6 = 120$ calls to f. Since RKF12, made more calls, it is more computationally heavy.

From this, we can say that RKF12 is good when the tolerance is not super low and when your approximation does not need to be very close. When you have a low tolerance and want an extremely accurate approximation, you should use RKF45.

Note: I continued to play with much smaller tolerance levels, and RKF45 still only took 20 iterations for every tolerance level I tried. On the other hand RKF12's number of iterations grew very fast, some times

even up to 25,000. This makes sense because RKF12 is not as accurate as RKF45, which means it would need more iteration to meet the same tolerance level. Even with so many iterations, RKF12 is still not as accurate as RKF45, which shows that RKF45 is good when you want an extremely accurate approximation.