Homework 7

November 27, 2020

Note: I had several issues trying to get some of the code working for the scipy.integrate.ode function. The Jacobian Matrix definition and even the order in which the symbols t and Y appear in the Bessel Function gave me errors.

To compensate, I used another function 'solveivp' from Python. By default, this function solves a given ODE with the RK45 Method, ie 5th order Runge Kutta. I have included the documentation page in the References at the end.

1 Problem 1

1.1 Part A

Before applying a method to numerically integrate this differential equation in Python, we will introduce a change of variables to obtain a different system. From the Bessel Differential Equation, we can introduce a change of variables. With the function definition below, we see how this is done

```
>>> def fbessel(t,Y):
...     y = Y[0]
...     z = Y[1]
...     dydt = z
...     dzdt = 1.00/t**2 * (-t*z - (t**2)*y)
...     return np.array([dydt,dzdt])
```

As demonstrated on the Python documentation page, we will next apply the method to numerically integrate this differential equation. Observe that because we want to have 21 equally spaced points on the interval $[1, 30]$, we want to implement a step size of $\frac{29}{21} \approx 1.38095\cdots$. [1] So we have that

```
>>> result = integrate.solve_ivp(fbessel, (1,30) , y0 = np.array([0,0.448]) ,
max_step = 1 , vectorized= False, t_eval = None, atol = 1e-6 , rtol = 1e-6)
```

Which helps us solve the Bessel Differential Equation with the 5 order Runge Kutta Method that we discussed. The short version of the output is

```
>>> result
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 458
     njev: 0
      nlu: 0
      sol: None
   status: 0
  success: True
```

[1] After running the code with a maximum step size of either 1 or $1.38\cdots$, I saw that number of function evaluations and other information given in the output was the same. I did not change this small part of the function call in what I have given for **A**.

So with the RK45 Method that we have by default in Python, we see that we can set the relative tolerance to what we desire. In the case of **A**, we see that the RK45 Method is able to reach the desired tolerance that we set. In the call of our Python function, observe that we made use of the fact that $J_1(1) \approx 0.440050587\cdots$, and that we want our interval of integration to be from $(1, 30)$ in order to determine the value of $J_1(30)$.

With the code that we have given already, we can try another method of integration out, as given on the Python Documentation Page. We will run the same code on the function 'fbessel', with the same initial conditions but just a different method. From the Documentation page for $'$solveivp$'$, one higher order method than RK45 is the 'Radau' Method, which denotes an Implicit Runge-Kutta method of the Radau IIA family of order 5. From Python, we get that

```
>>> result = integrate.solve_ivp(fbessel, (1,30) , y0 = np.array([0,0.448]) ,method =
'LSODA', max_step = 1 , vectorized= False, t_eval = None, atol = 1e-6 , rtol = 1e-6)
>>> result
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 283
     njev: 0
      nlu: 0
      sol: None
   status: 0
  success: True
```

So we observe that with the same tolerance, we obtain the desired error with a smaller number of steps. In terms of the 2 methods, we clearly see that the RK45 Method has a slower convergence to the actual value of $J_1(30)$ that we were solving for from the Bessel Differential Equation. More specifically, this is reflective of the characteristics of this numerical method, in which the higher order method requires a **smaller** number of function evaluations. We can see from **each** output of the results that I have given the number of function evaluations which is denoted by nfev. I have attached the **complete** output from each routine at the end of this HW.

1.2 Part B

We repeat the same code from **A**. For a higher tolerance per step of 10^{-10} we see from the code that

```
>>> result = integrate.solve_ivp(fbessel, (1,30) , y0 = np.array([0,0.448]), max_step =
1.38095 , vectorized= False, t_eval = None, atol = 1e-10 , rtol = 1e-10)
>>> result
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 2816
     njev: 0
      nlu: 0
      sol: None
   status: 0
  success: True
```

For the higher order method we have that

```
>>> result = integrate.solve_ivp(fbessel, (1,30) , y0 = np.array([0,0.448]) ,method =
'LSODA', max_step = 1.38095 , vectorized= False, t_eval = None, atol = 1e-10 , rtol = 1e-10)
>>> result
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 531
     njev: 0
      nlu: 0
      sol: None
```

```
   status: 0
  success: True
```

Directly from the number of function evaluations, we can observe that the higher order method is much better for obtaining a higher tolerance of 10^{-10}. This is somewhat in line with what we saw from **A** and what anyone could have expected for **B**. We conclude that the higher order method, with a much smaller number of function evaluations, allows us to achieve the desired tolerance more efficiently and quickly. This completes **B**. In short, the disparity between the efficiency of either method that I have chosen, RK45 or 'Radau', becomes much larger when we look at a smaller error tolerance of 10^{-10}.

2 Problem 2

We can integrate Equation (1) in Python by directly applying the 'quad' method to integrate the Bessel Function which solves the given differential equation in (1). With the code below, we call on a special library of functions and apply the method to get a solution within the desired tolerance of 10^{-6}. Directly again from the code, we will change the initial conditions slightly, observing that we want to integrate from $x = 0$ to $x = 1$, as well as making use of the initial condition, in which $J_1(0) = 0$. To avoid inputting 0, thereby giving an error, I set the boundary of integration to be a small number > 0.

In the code below, observe that the different method that I choose was the 'BDF' one, which stands for 'Implicit multi-step variable-order (1 to 5) method based on a backward differentiation formula for the derivative approximation'. The output is

```
>>> result = integrate.solve_ivp(fbessel, (0.000001,1) ,
y0 = np.array([0,1]),method='BDF',  max_step = 0.01 , vectorized= False, t_eval = None, atol = 1e-6,
>>> result
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 695
     njev: 6
      nlu: 46
      sol: None
   status: 0
  success: True
```

But as we saw in **2B**, the number of function evaluations for the 'BDF' method from Python increases dramatically when we want a smaller desired error tolerance of 10^{-10}. We see that

```
>>> result = integrate.solve_ivp(fbessel, (0.000001,1) ,
y0 = np.array([0,1]),method='BDF',  max_step = 0.01 ,
vectorized= False, t_eval = None, atol = 1e-10, rtol = 1e-10)
>>> result
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 2489
     njev: 6
      nlu: 139
      sol: None
   status: 0
  success: True
```

$\Rightarrow$ Even from this method and the ones that we chose for **1**, we can see that as we try to apply some integration method for the Bessel Differential Equation, the differential equation itself is singular for $x = 1$. Therefore, we would expect a much higher number of function evaluations because the terms that are impacted when $x \approx 1$ make the differential equation harder to approximate and study, which completes Problem **2**.

3 References

Python Documentation Page
https : //docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solveivp.html.

4 Code Output

4.1 1A

4.1.1 First Method Output

```
>>> result = integrate.solve_ivp(fbessel, (1,30) , y0 = np.array([0,0.448]) , max_step = 1 , vectoriz
>>> result
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 458
     njev: 0
      nlu: 0
      sol: None
   status: 0
  success: True
        t: array([ 1.        ,  1.03040263,  1.31353675,  1.66971346,  2.03406534,
        2.39841721,  2.76276909,  3.12712097,  3.49147285,  3.85582472,
        4.2201766 ,  4.58452848,  4.94888036,  5.31323224,  5.67758411,
        6.04193599,  6.40628787,  6.77595551,  7.1476483 ,  7.51934109,
        7.89103388,  8.26272668,  8.63563049,  9.0085343 ,  9.38143811,
        9.76092723, 10.14472648, 10.52852573, 10.91232497, 11.29612422,
       11.68125139, 12.06637856, 12.45150572, 12.83989728, 13.23282517,
       13.6259946 , 14.01916403, 14.41233346, 14.8068116 , 15.20128973,
       15.59576786, 15.99246802, 16.39325775, 16.79422382, 17.1951899 ,
       17.59615597, 17.99842518, 18.40069439, 18.80296359, 19.20737915,
       19.61513306, 20.02288696, 20.43064087, 20.83839478, 21.24731599,
       21.6562372 , 22.06515842, 22.47671145, 22.89047988, 23.30424832,
       23.71801675, 24.13182822, 24.54630884, 24.96078946, 25.37527009,
       25.79325571, 26.21186038, 26.63046504, 27.04906971, 27.46853872,
       27.88800773, 28.30747674, 28.72852954, 29.15188047, 29.57523141,
       29.99858235, 30.        ])
 t_events: None
        y: array([[ 0.        ,  0.01341537,  0.12018556,  0.21280848,  0.26392837,
         0.27472763,  0.25041811,  0.19824144,  0.12720254,  0.04733625,
        -0.03125174, -0.09939222, -0.14976231, -0.17758031, -0.18100262,
        -0.16117951, -0.12196976, -0.06852341, -0.00857396,  0.04947711,
         0.09813114,  0.13144935,  0.1457545 ,  0.13984083,  0.11533266,
         0.07544545,  0.02596782, -0.02538085, -0.07126345, -0.10536896,
        -0.12328866, -0.12285671, -0.10472124, -0.07170346, -0.02861873,
         0.01753141,  0.05973115,  0.09177394,  0.1091671 ,  0.10955367,
         0.09331691,  0.06313068,  0.02356953, -0.01874483, -0.05712208,
        -0.08565725, -0.10013518, -0.09848245, -0.08131211, -0.0514772 ,
        -0.01372124,  0.02548463,  0.05976319,  0.08366124,  0.09351659,
         0.08790969,  0.06804657,  0.0372088 ,  0.00057272, -0.03550273,
        -0.06500374, -0.08311914, -0.08700824, -0.07622046, -0.05280848,
        -0.02061416,  0.01463483,  0.04681797,  0.07047711,  0.0817043 ,
         0.07870169,  0.06219944,  0.03511837,  0.00215726, -0.03070668,
```

```
       -0.05772012, -0.05779508],
       [ 0.448     ,  0.43457856,  0.32282301,  0.19923072,  0.08302371,
        -0.02138274, -0.10872892, -0.17352438, -0.21178654, -0.22194555,
        -0.20520297, -0.16545195, -0.10883424, -0.04301906,  0.02369315,
         0.08332672,  0.12912891,  0.15654736,  0.16233015,  0.14662599,
         0.11247691,  0.06512945,  0.01111468, -0.04209058, -0.08753639,
        -0.11996166, -0.13463311, -0.1297162 , -0.10660268, -0.06917075,
        -0.02300321,  0.02495756,  0.06783608,  0.09989219,  0.11651212,
         0.1152382 ,  0.09675623,  0.064297  ,  0.02293253, -0.02080474,
        -0.06027962, -0.08978914, -0.10492391, -0.1033192 , -0.08561197,
        -0.05493304, -0.01622506,  0.02418106,  0.0599053 ,  0.08553761,
         0.09704509,  0.09260653,  0.07326207,  0.04243345,  0.00521788,
        -0.03213845, -0.06355662, -0.08412258, -0.09041388, -0.08149748,
        -0.05912866, -0.02726709,  0.00866683,  0.0425458 ,  0.06873205,
         0.0830274 ,  0.08292154,  0.06863146,  0.04282112,  0.01000822,
        -0.0240272 , -0.05342455, -0.07325704, -0.08011107, -0.0728394 ,
        -0.05291574, -0.05283136]])
```

4.2 Second Method Output

```
>>> result
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 283
     njev: 0
      nlu: 0
      sol: None
   status: 0
  success: True
        t: array([ 1.        ,  1.00176269,  1.00352538,  1.00705076,  1.01057614,
        1.01410152,  1.04703726,  1.07997299,  1.11290872,  1.14584445,
        1.19808619,  1.25032792,  1.30256966,  1.35481139,  1.40705313,
        1.48411494,  1.56117676,  1.63823857,  1.71530038,  1.7923622 ,
        1.86942401,  1.99165191,  2.11387982,  2.23610773,  2.35833563,
        2.48056354,  2.60279144,  2.77242976,  2.94206808,  3.11170639,
        3.28134471,  3.45098303,  3.62062134,  3.85397895,  4.08733655,
        4.32069415,  4.55405176,  4.78740936,  5.02076696,  5.25412457,
        5.48748217,  5.72083978,  5.95419738,  6.23283533,  6.51147329,
        6.79011125,  7.0687492 ,  7.34738716,  7.62602511,  7.90466307,
        8.13183153,  8.359     ,  8.58616847,  8.81333694,  9.04050541,
        9.26767387,  9.49484234,  9.75380751, 10.01277269, 10.27173786,
       10.53070303, 10.78966821, 11.04863338, 11.30759855, 11.54795038,
       11.78830221, 12.02865404, 12.26900587, 12.5093577 , 12.74970953,
       12.99006136, 13.23703005, 13.48399874, 13.73096742, 13.97793611,
       14.2249048 , 14.47187348, 14.71884217, 14.96404986, 15.20925754,
       15.45446522, 15.69967291, 15.94488059, 16.19008828, 16.43529596,
       16.68078771, 16.92627946, 17.1717712 , 17.41726295, 17.6627547 ,
       17.90824644, 18.15373819, 18.40600773, 18.65827728, 18.91054682,
       19.16281637, 19.41508591, 19.66735546, 19.919625  , 20.16313974,
       20.40665449, 20.65016923, 20.89368398, 21.13719872, 21.38071347,
       21.62422821, 21.87376372, 22.12329922, 22.37283473, 22.62237023,
       22.87190574, 23.12144124, 23.37097675, 23.61300869, 23.85504064,
       24.09707258, 24.33910453, 24.58113647, 24.82316842, 25.06520036,
```

```
       25.31601967, 25.56683898, 25.8176583 , 26.06847761, 26.31929692,
       26.57011623, 26.82093554, 27.0620769 , 27.30321825, 27.5443596 ,
       27.78550096, 28.02664231, 28.26778367, 28.50892502, 28.75888509,
       29.00884516, 29.25880523, 29.5087653 , 29.75872537, 29.99910961,
       30.        ])
 t_events: None
        y: array([[ 0.        ,  0.0007883 ,  0.0015752 ,  0.00314625,  0.00471177,
         0.00627178,  0.02058307,  0.03442916,  0.04782251,  0.06077411,
         0.08043508,  0.09904195,  0.11662224,  0.13319912,  0.14879211,
         0.17003206,  0.18921254,  0.20637255,  0.22154662,  0.23476742,
         0.24606723,  0.26013214,  0.26959211,  0.2746096 ,  0.27537061,
         0.27208728,  0.26499894,  0.24935904,  0.22770015,  0.20090045,
         0.16990796,  0.1357222 ,  0.09937366,  0.04774724, -0.00335599,
        -0.05144429, -0.09428646, -0.13000282, -0.15713802, -0.17471403,
        -0.18225936, -0.17981467, -0.16791373, -0.14271827, -0.10785506,
        -0.06630643, -0.02141647,  0.023358  ,  0.06469719,  0.09965116,
         0.12174857,  0.13706694,  0.14502558,  0.14542127,  0.13842931,
         0.12458574,  0.10475297,  0.07630309,  0.04357662,  0.00881678,
        -0.0256582 , -0.05760517, -0.08499817, -0.10615351, -0.11911559,
        -0.12503284, -0.1237147 , -0.11538028, -0.10063838, -0.08044612,
        -0.05604912, -0.02813551,  0.0009537 ,  0.02945756,  0.05568157,
         0.07809679,  0.09542765,  0.10672318,  0.11139878,  0.10938789,
         0.10091715,  0.08658973,  0.06734412,  0.044394  ,  0.0191529 ,
        -0.0068817 , -0.03212628, -0.05509475, -0.07445688, -0.08911446,
        -0.09826287, -0.10143454, -0.09835687, -0.08913714, -0.07443644,
....
......
```

4.3 1B

4.3.1 First Method Output

```
>>> result = integrate.solve_ivp(fbessel, (1,30) , y0 = np.array([0,0.448]), max_step = 1.38095 , vec
>>> result
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 2816
     njev: 0
      nlu: 0
      sol: None
   status: 0
  success: True
        t: array([ 1.        ,  1.00481849,  1.04885145,  1.09456302,  1.14421157,
        1.19881927,  1.25962329,  1.32407671,  1.38853013,  1.44517598,
        1.50183811,  1.55850024,  1.61516237,  1.67182449,  1.72848662,
        1.78514875,  1.84181087,  1.898473  ,  1.95513513,  2.01179725,
        2.06845938,  2.12512151,  2.18178364,  2.23844576,  2.29510789,
        2.35177002,  2.40843214,  2.46509427,  2.5217564 ,  2.57841853,
        2.63508065,  2.69174278,  2.74840491,  2.80506703,  2.86172916,
        2.91839129,  2.97505342,  3.03171554,  3.08837767,  3.1450398 ,
        3.20170192,  3.25836405,  3.31502618,  3.37168831,  3.42835043,
        3.48501256,  3.54167469,  3.59833681,  3.65499894,  3.71166107,
        3.76832319,  3.82498532,  3.88164745,  3.93830958,  3.9949717 ,
```

```
       4.05163383,  4.10829596,  4.16495808,  4.22162021,  4.27828234,
       4.33494447,  4.39160659,  4.44826872,  4.50493085,  4.56159297,
       4.6182551 ,  4.6749
       ......
       ...
```

4.3.2 Second Method Output

```
>>> result = integrate.solve_ivp(fbessel, (1,30) , y0 = np.array([0,0.448]) ,method = 'LSODA', max_st
>>> result
  message: 'The solver successfully reached the end of the integration interval.'
     nfev: 531
     njev: 0
      nlu: 0
      sol: None
   status: 0
  success: True
        t: array([ 1.        ,  1.00001761,  1.00003522,  1.00007043,  1.00010565,
       1.00014086,  1.00049301,  1.00084516,  1.00119731,  1.00154947,
       1.00507098,  1.00859249,  1.012114  ,  1.01563551,  1.01915702,
       1.03218417,  1.04521131,  1.05823845,  1.0712656 ,  1.08429274,
       1.09731988,  1.11608424,  1.1348486 ,  1.15361296,  1.17237732,
       1.19114168,  1.20990604,  1.2286704 ,  1.25517818,  1.28168597,
       1.30819375,  1.33470153,  1.36120932,  1.3877171 ,  1.41422488,
       1.44073266,  1.47671085,  1.51268905,  1.54866724,  1.58464543,
       1.62062362,  1.65660181,  1.69258   ,  1.72855819,  1.77187464,
       1.81519109,  1.85850753,  1.90182398,  1.94514042,  1.98845687,
       2.03177331,  2.07508976,  2.1276994 ,  2.18030903,  2.23291867,
       2.28552831,  2.33813795,  2.39074758,  2.44335722,  2.49596686,
       2.56044449,  2.62492212,  2.68939976,  2.75387739,  2.81835502,
       2.88283266,  2.94731029,  3.01178792,  3.09332316,  3.17485839,
       3.25639362,  3.33792886,  3.41946409,  3.50099932,  3.58253456,
       3.66406979,  3.77695222,  3.88983465,  4.00271707,  4.1155995 ,
       4.22848193,  4.34136435,  4.45424678,  4.56712921,  4.69185258,
       4.81657596,  4.94129933,  5.06602271,  5.19074609,  5.31546946,
       5.44019284,  5.56491621,  5.68963959,  5.81755109,  5.9454626 ,
       6.0733741 ,  6.2012856 ,  6.32919711,  6.45710861,  6.58502012,

      ....
      ...
```

4.4 Problem 2 Output

```
nfev: 2489
     njev: 6
      nlu: 139
      sol: None
   status: 0
  success: True
        t: array([1.00000000e-06, 1.00000120e-06, 1.00000239e-06, 1.00001434e-06,
      1.00002629e-06, 1.00006195e-06, 1.00009761e-06, 1.00013327e-06,
      1.00022131e-06, 1.00030936e-06, 1.00039740e-06, 1.00092973e-06,
```

```
1.00146206e-06, 1.00199439e-06, 1.00291563e-06, 1.00383688e-06,
1.00475813e-06, 1.00567937e-06, 1.00766734e-06, 1.00965530e-06,
1.01164327e-06, 1.01363123e-06, 1.01715876e-06, 1.02068630e-06,
1.02421383e-06, 1.02774137e-06, 1.03126890e-06, 1.03754993e-06,
1.04383095e-06, 1.05011198e-06, 1.05639301e-06, 1.06267403e-06,
1.06895506e-06, 1.07861995e-06, 1.08828483e-06, 1.09794972e-06,
1.10761461e-06, 1.11727950e-06, 1.12694439e-06, 1.13737976e-06,
1.14781513e-06, 1.15825050e-06, 1.16868586e-06, 1.17912123e-06,
1.18955660e-06, 1.20054517e-06, 1.21153373e-06, 1.22252230e-06,
1.23351086e-06, 1.24449943e-06, 1.25548799e-06, 1.26711908e-06,
1.27875017e-06, 1.29038125e-06, 1.30201234e-06, 1.31364343e-06,
1.32527452e-06, 1.33761841e-06, 1.34996230e-06, 1.36230619e-06,
1.37465008e-06, 1.38699397e-06, 1.39933786e-06, 1.41244043e-06,
1.42554300e-06, 1.43864557e-06, 1.45174813e-06, 1.46485070e-06,
1.47795327e-06, 1.49186434e-06, 1.50577542e-06, 1.51968650e-06,
1.53359757e-06, 1.547508
....
...
```