AMATH301_Homework7_writeup

March 6, 2023

1 Homework 7 writeup solutions

1.1 Name: Aqua Karaman

1.2 Problem 1

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import time
import scipy.integrate

s = 77.27
w = 0.161
q = 1

y1_prime = lambda y1, y2, y3: s*(y2 - y1*y2 + y1 - q*y1**2)
y2_prime = lambda y1, y2, y3: s**(-1) * (-y2 - y1 * y2 + y3)
y3_prime = lambda y1, y2, y3: w * (y1 - y3)

y0 = np.array([1, 2, 3])

odefun = lambda t, y: np.array([y1_prime(y[0], y[1], y[2]),␣
↪y2_prime(y[0],y[1],y[2]), y3_prime(y[0], y[1], y[2])])
A3 = odefun(1, [2, 3, 4])
print('2a:   ', A3)
```

```
2a:    [-3.86350000e+02 -6.47081662e-02 -3.22000000e-01]
```

1.2.1 Part a - Timing RK45 and BDF

[ ]:
```
# Define the 10 logarithmically spaced points
A4 = np.zeros([3, 10])

q_vals = np.logspace(0, -5, 10)

# Loop over them
startrk = np.zeros(10)
stoprk = np.zeros(10)
```

```
for k in range(len(q_vals)-1):
    startrk[k] = time.time()
    q = q_vals[k]
    sol = scipy.integrate.solve_ivp(odefun, [0, 30], y0)
    A4[:,k] = sol.y[:,-1]
    stoprk[k] = time.time()
startrk[9] = time.time()
q = 1e-5
sol = scipy.integrate.solve_ivp(odefun, [0, 30], y0)
A4[:,9] = sol.y[:,-1]
stoprk[9] = time.time()

print('2b:   ', A4)

A5 = np.zeros([3, 10])
t_bdf = np.zeros(10)

startbdf = np.zeros(10)
stopbdf = np.zeros(10)

for k in range(len(q_vals)):
    startbdf[k] = time.time()
    q = q_vals[k]
    sol = scipy.integrate.solve_ivp(odefun, [0, 30], y0, method='BDF')
    A5[:,k] = sol.y[:,-1]
    stopbdf[k] = time.time()

# Find the time
rk_times = np.abs(startrk - stoprk)
bdf_times = np.abs(startbdf - stopbdf)

print('RK method times:   ', rk_times)
print('BDF times are:   ', bdf_times)
```

```
2b:    [[1.00000000e+00 1.76920129e+00 3.42826976e+00 1.35411976e+01
  3.00035207e+00 1.14100460e+00 1.03726556e+00 1.01038834e+00
  1.00297028e+00 1.00232480e+00]
 [1.27659290e+00 1.16661679e+00 1.03617714e+00 7.62697465e-01
  1.48784476e+00 8.11697415e+00 2.83575682e+01 9.95437951e+01
  3.54099018e+02 1.26740846e+03]
 [1.01618010e+00 1.70229211e+00 2.95199924e+00 7.65951450e+00
  4.84077454e+01 8.06768171e+01 2.18206553e+02 7.05237307e+02
  2.44180348e+03 8.67980449e+03]]
RK method times:    [0.         0.09920239 0.05058002 0.03357005 0.02960968
0.06557369
 0.21628547 0.78039145 2.73382401 9.95656824]
BDF times are:    [0.00450349 0.00551128 0.00601125 0.00750899 0.01552272
0.02653289
```

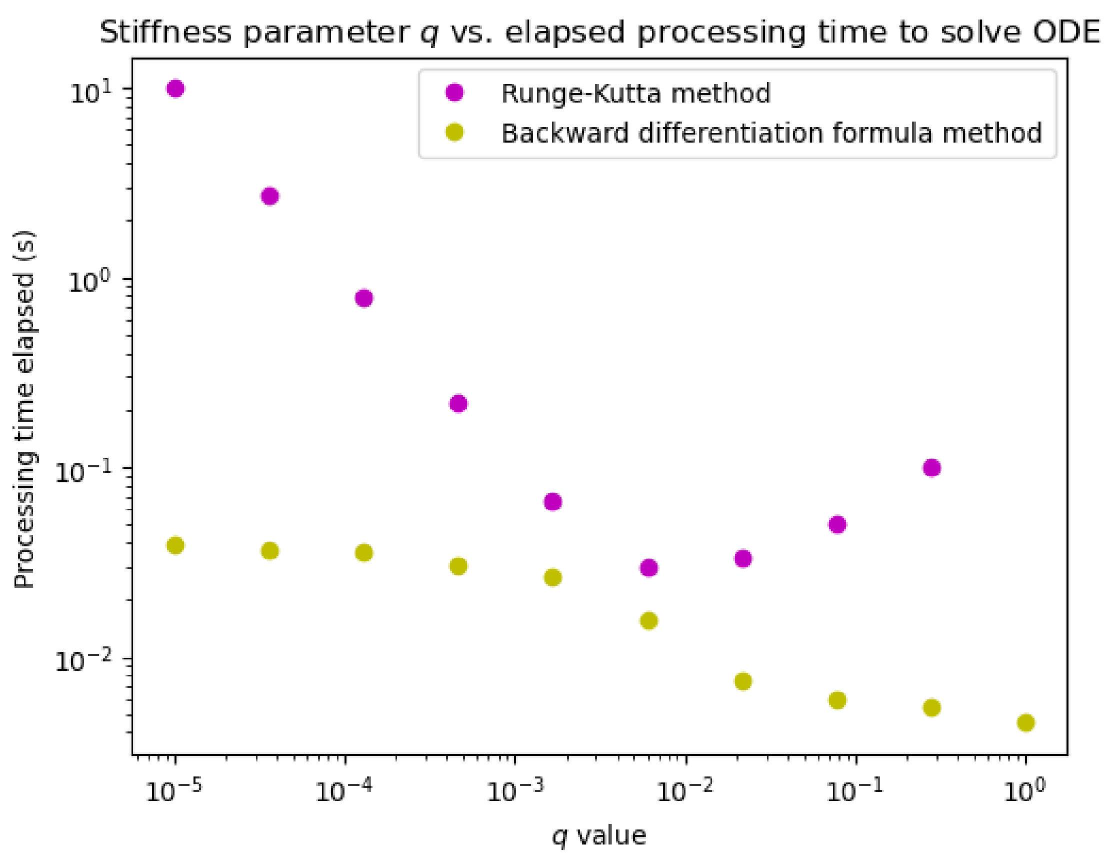
```
0.03053522 0.03554773 0.03610969 0.03955007]
```

1.2.2 Part b - Create a loglog plot

Make sure to use plot *markers* not lines for the data, and label the axes!

```
[ ]: plt.figure()

     plt.loglog(q_vals, rk_times, '.m', label='Runge-Kutta method', markersize=12)
     plt.loglog(q_vals, bdf_times, '.y', label='Backward differentiation formula␣
      ↪method', markersize=12)

     plt.xlabel('$q$ value')
     plt.ylabel('Processing time elapsed (s)')
     plt.title('Stiffness parameter $q$ vs. elapsed processing time to solve ODE')
     plt.legend()
```

```
[ ]: <matplotlib.legend.Legend at 0x291418d8a90>
```



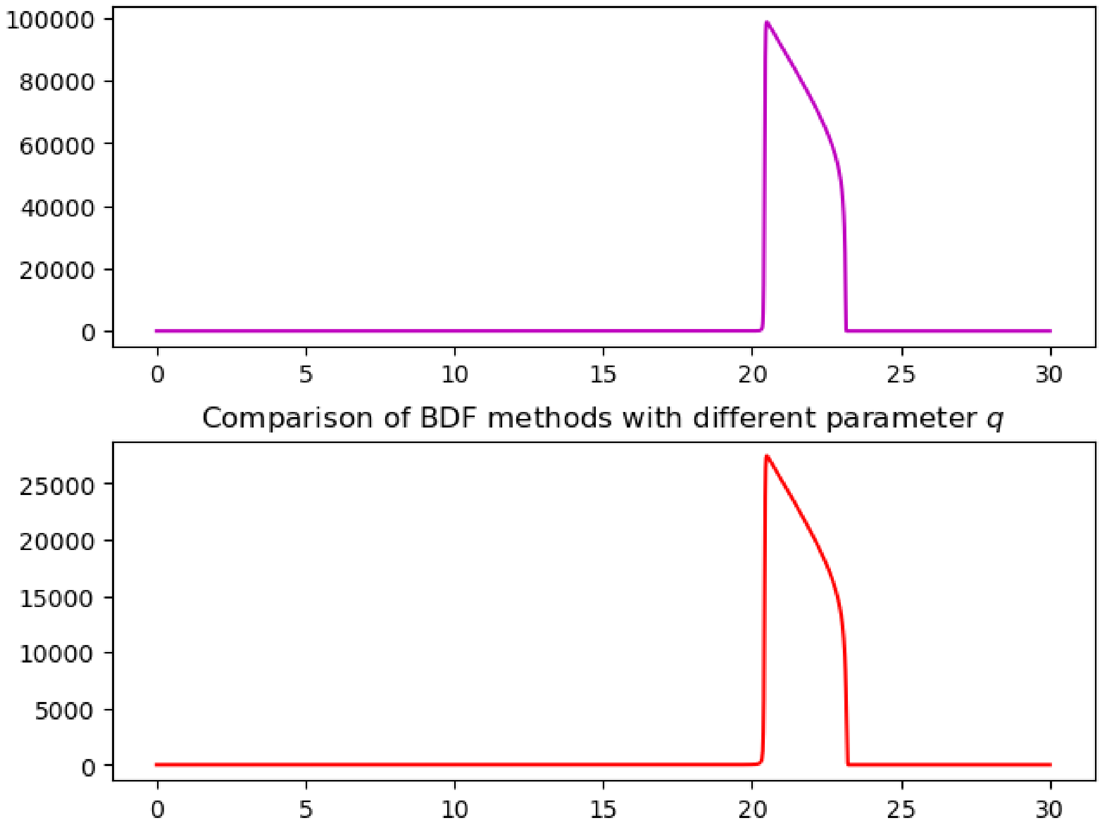
1.2.3 Part c - Create a 2 panel figure.

We have not yet done this in class, but it's about time we teach you because it's an important skill!

Below I'll show how to do that plotting $y = x^2$ in the upper figure and $y = x^3$ in the lower figure.

[ ]:
```python
plt.figure()
q = q_vals[-1]
sol0 = scipy.integrate.solve_ivp(odefun, [0, 30], y0)
y1q0 = sol0.y[0]
t_vals0 = sol0.t

q = q_vals[-2]
sol1 = scipy.integrate.solve_ivp(odefun, [0, 30], y0)
y1q1 = sol1.y[0]
t_vals1 = sol1.t

fig, ax = plt.subplots(2, 1, constrained_layout=True)

ax[0].plot(t_vals0, y1q0, 'm')
ax[1].plot(t_vals1, y1q1, 'r')
# ax[0].set_ylim(0, 10)
# ax[1].set_ylim(0, 10)

plt.title('Comparison of BDF methods with different parameter $q$')

# You can delete this once you've figured it out,
# I just want to give the example.
# x = np.linspace(-1, 1, 100)
# fig, ax = plt.subplots(2, 1, constrained_layout=True)
# the (2, 1) corresponds to "2 rows, 1 column"
# ax[0].plot(x, x**2, 'b')
# ax[0] means the 0th panel.
# ax[1].plot(x, x**3, 'r') # ax[1] means the 1st panel
```

[ ]:
```
Text(0.5, 1.0, 'Comparison of BDF methods with different parameter $q$')

<Figure size 640x480 with 0 Axes>
```



1.2.4 Part d - Comment on what we see.

Part (i) - Compare the two methods Overall, it seems BDF is the best method. For every size value of q, it seems to consistently take less time to find a solution, according to the plots generated by my code.

Part (ii) - Time as q increases The time RK45 takes seems to have a quadratic relation with q as it decreases on the log-log plot. It seems the vertex of this quadratic relation is around (10^{-2}, $10^{-1.5}$).

Part (iii) - What makes calculation slower for RK45? I'd assume from the plots that the giant jump the solution takes a little after $t = 20$ is what causes the processing time disparity. In this case specifically, the smallest q bounces the solution up to 98695, whereas the solution for the second smallest q reaches only up to 27455.

Part (iv) - Is this equation stiff? How do we know? I would say this equation is stiff since it does have a wild jump that gets bigger as q decreases. In other words, since the solution changes wildly for a smaller q, it is stiff.

1.3 Problem 2

[ ]:
```
mu = 200

dxdt = lambda y: y
dydt = lambda t, xy:  (1 - (xy[0])**2) * (xy[1]) * mu - xy[0]

init_cond = np.array([2, 3])
A6 = dxdt(init_cond[1])
A7 = dydt(0, init_cond)

x0 = 2
y0 = 0
xy0 = np.array([2, 0])
# y0 = ?
solrk = scipy.integrate.solve_ivp(dydt, [0, 400], [2, 0])
A8 = solrk.y[0]

## Part c
# we're gonna see if this works - NOTE: it doesn't
solbdf = scipy.integrate.solve_ivp(dydt, [0, 400], [2,0], method='BDF')
A9 = solbdf.y[0]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_3744\413698620.py in <module>
     17 ## Part c
     18 # we're gonna see if this works - NOTE: it doesn't
---> 19 solbdf = scipy.integrate.solve_ivp(dydt, [0, 400], [2,0], method='BDF')
     20 A9 = solbdf.y[0]

c:\Users\johns\anaconda3\lib\site-packages\scipy\integrate\_ivp\ivp.py in␣
 ↪solve_ivp(fun, t_span, y0, method, t_eval, dense_output, events, vectorized,␣
 ↪args, **options)
    553         method = METHODS[method]
    554
--> 555     solver = method(fun, t0, y0, tf, vectorized=vectorized, **options)
    556
    557     if t_eval is None:

c:\Users\johns\anaconda3\lib\site-packages\scipy\integrate\_ivp\bdf.py in␣
 ↪__init__(self, fun, t0, y0, t_bound, max_step, rtol, atol, jac, jac_sparsity,␣
 ↪vectorized, first_step, **extraneous)
    207
    208         self.jac_factor = None
--> 209         self.jac, self.J = self._validate_jac(jac, jac_sparsity)
    210         if issparse(self.J):
    211             def lu(A):
```

```
c:\Users\johns\anaconda3\lib\site-packages\scipy\integrate\_ivp\bdf.py in
 ↪_validate_jac(self, jac, sparsity)
    263                                                    sparsity)
    264                     return J
--> 265                 J = jac_wrapped(t0, y0)
    266             elif callable(jac):
    267                 J = jac(t0, y0)

c:\Users\johns\anaconda3\lib\site-packages\scipy\integrate\_ivp\bdf.py in
 ↪jac_wrapped(t, y)
    259                     self.njev += 1
    260                     f = self.fun_single(t, y)
--> 261                     J, self.jac_factor = num_jac(self.fun_vectorized, t, y,
 ↪f,
    262                                                    self.atol, self.jac_factor
    263                                                    sparsity)

c:\Users\johns\anaconda3\lib\site-packages\scipy\integrate\_ivp\common.py in
 ↪num_jac(fun, t, y, f, threshold, factor, sparsity)
    316
    317         if sparsity is None:
--> 318             return _dense_num_jac(fun, t, y, f, h, factor, y_scale)
    319         else:
    320             structure, groups = sparsity

c:\Users\johns\anaconda3\lib\site-packages\scipy\integrate\_ivp\common.py in
 ↪_dense_num_jac(fun, t, y, f, h, factor, y_scale)
    327         h_vecs = np.diag(h)
    328         f_new = fun(t, y[:, None] + h_vecs)
--> 329         diff = f_new - f[:, None]
    330         max_ind = np.argmax(np.abs(diff), axis=0)
    331         r = np.arange(n)

IndexError: too many indices for array: array is 0-dimensional, but 1 were
 ↪indexed
```

1.3.1 Part a - Ratio of points, RK45 to BDF.

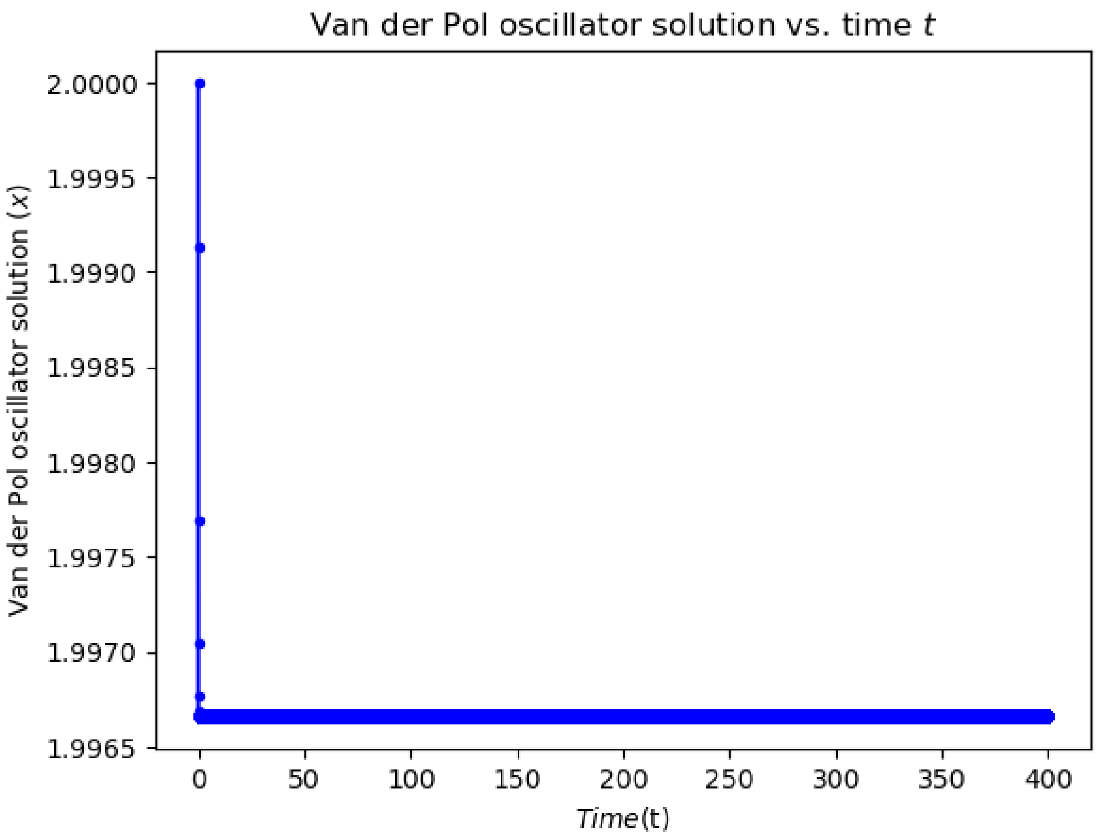
Could not calculate, N/A

Part b - Plot solution, x(t)

[ ]:
```
plt.figure

plt.plot(solrk.t, solrk.y[0], '.-b', label='Runge-Kutta method')
plt.xlabel('$Time ($t$)$')
```

```
plt.ylabel('Van der Pol oscillator solution ($x$)')
plt.title('Van der Pol oscillator solution vs. time $t$')
```

[ ]: Text(0.5, 1.0, 'Van der Pol oscillator solution vs. time t')



Part c - Plot x(t) vs. y(t) (y(t) on vertical axis)

[ ]:
```
# No functioning code for BDF method. Please see above error message.
```

Part d - Discussion No functioning code for BDF method. Please see above error message.