CEE 6736: HW #4

Hanfeng Zhai
Mechanical Engineering, Cornell University
hz253@cornell.edu

October 19, 2022

[2]:
```
# !sudo apt install cm-super dvipng texlive-latex-extra texlive-latex-recommended
```

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy.integrate import odeint
```

Problem

Please show all work (i.e. include source code, and include thoughtful, analytical discussions):

(1) Please revisit our example from class, where we applied the *Forward Euler* scheme to the numerical solution of $\frac{dy}{dt} = ry$.

Solution:

Revisiting our lecture notes, using the Forward Euler method the update of each step takes the form:

$$\begin{aligned} y_{n+1} &= y_n + hf(t_n, y_n) \\ &= y_n + hry_n \\ &= y_n(1 + rh) \end{aligned}$$

(2) Re-use your previous Euler solver source code (from HW3) to investigate the utility of the theoretical stability criterion that we arrived at for (1): $h < \frac{2}{|r|}$. (hint I just want you to do this by trial and error, as you adjust step size, h.)

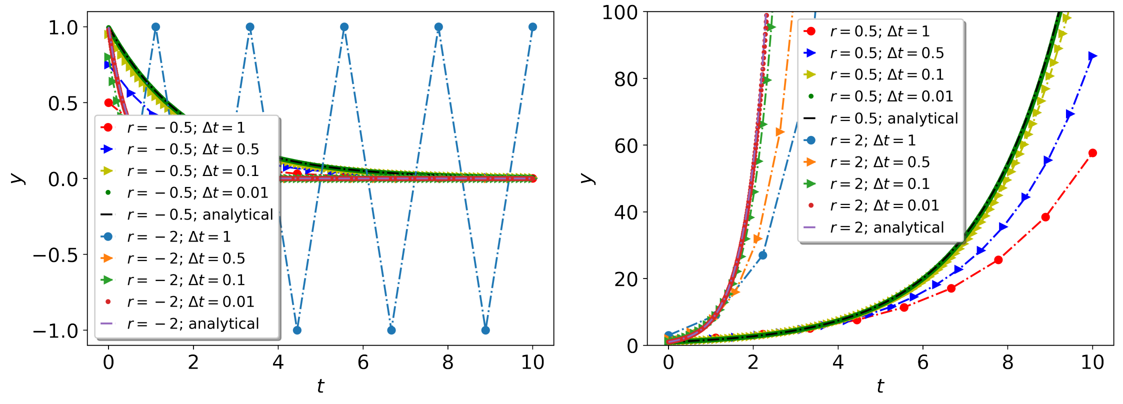
Solution:

In this problem, I set up 4 rs, $r = \pm\frac{1}{2}$ and $r = \pm 2$. I approximate the given ODE of these four rs with different h to check the theoretical stability criterion. The application codes are shown as below:

[52]:
```
time = 20.0
q_t1 = np.ones(20)
q_t2 = np.ones(40)
q_t3 = np.ones(200)
q_t4 = np.ones(2000)
```

```
r = 0.5

q0 = 1
i=0

def obtain_discretized(r,time):
  h_1 = 1;h_2 = .5;h_3 = .1;h_4 = .01
  q_t1 = np.ones(int(time/h_1));q_t2 = np.ones(int(time/h_2))
  q_t3 = np.ones(int(time/h_3));q_t4 = np.ones(int(time/h_4))
  t_1 = np.linspace(0,time,int(time/h_1));t_2 = np.linspace(0,time,int(time/h_2))
  t_3 = np.linspace(0,time,int(time/h_3));t_4 = np.linspace(0,time,int(time/h_4))
  t = np.linspace(0,time,100)
  for i in range(0,int(time/h_1)):
    q_t1[i] = q_t1[i-1] * (1 + r * h_1);i += 1
  for i in range(0,int(time/h_2)):
    q_t2[i] = q_t2[i-1] * (1 + r * h_2);i += 1
  for i in range(0,int(time/h_3)):
    q_t3[i] = q_t3[i-1] * (1 + r * h_3);i += 1
  for i in range(0,int(time/h_4)):
    q_t4[i] = q_t4[i-1] * (1 + r * h_4);i += 1
  q_analy = np.exp(r * t)
  return q_t1, q_t2, q_t3, q_t4, q_analy, t_1, t_2, t_3, t_4, t

r1_sol1,r1_sol2,r1_sol3,r1_sol4,r1_analy,t1,t2,t3,t4,t = obtain_discretized(-0.
 ↪5,10)
r2_sol1,r2_sol2,r2_sol3,r2_sol4,r2_analy,t1,t2,t3,t4,t = obtain_discretized(0.
 ↪5,10)
r3_sol1,r3_sol2,r3_sol3,r3_sol4,r3_analy,t1,t2,t3,t4,t =␣
 ↪obtain_discretized(-2,10)
r4_sol1,r4_sol2,r4_sol3,r4_sol4,r4_analy,t1,t2,t3,t4,t = obtain_discretized(2,10)
# plt.plot(t1, r1_sol1, 'ro-.', label='Euler\'s Method: $\Delta t = 1$')
# plt.plot(t2, r1_sol2, 'b>-.', label='Euler\'s Method: $\Delta t = 0.5$')
# plt.plot(t3, r1_sol3, 'y>-.', label='Euler\'s Method: $\Delta t = 0.1$')
# plt.plot(t4, r1_sol4, 'g.', label='Euler\'s Method: $\Delta t = 0.01$')
# plt.plot(t,r1_analy, 'k-.', label='Analytical Solution')
# plt.xlabel("$t$")
# plt.ylabel("$y$")

fig, axs = plt.subplots(1, 2, figsize = (15, 5))
axs[0].plot(t1, r1_sol1, 'ro-.', label='$r = -0.5$; $\Delta t = 1$')
axs[0].plot(t2, r1_sol2, 'b>-.', label='$r = -0.5$; $\Delta t = 0.5$')
axs[0].plot(t3, r1_sol3, 'y>-.', label='$r = -0.5$; $\Delta t = 0.1$')
axs[0].plot(t4, r1_sol4, 'g.', label='$r = -0.5$; $\Delta t = 0.01$')
axs[0].plot(t, r1_analy, 'k-.', label='$r = -0.5$; analytical')
axs[0].plot(t1, r3_sol1, 'o-.', label='$r = -2$; $\Delta t = 1$')
axs[0].plot(t2, r3_sol2, '>-.', label='$r = -2$; $\Delta t = 0.5$')
axs[0].plot(t3, r3_sol3, '>-.', label='$r = -2$; $\Delta t = 0.1$')
```

```python
axs[0].plot(t4, r3_sol4, '.', label='$r = -2$; $\Delta t = 0.01$')
axs[0].plot(t, r3_analy, '-.', label='$r = -2$; analytical')
axs[0].set_xlabel('$t$')
axs[0].set_ylabel('$y$')
axs[0].legend(shadow=True, handlelength=1, fontsize=12)
# axs[0].set_yscale('log')
axs[1].plot(t1, r2_sol1, 'ro-.', label='$r = 0.5$; $\Delta t = 1$')
axs[1].plot(t2, r2_sol2, 'b>-.', label='$r = 0.5$; $\Delta t = 0.5$')
axs[1].plot(t3, r2_sol3, 'y>-.', label='$r = 0.5$; $\Delta t = 0.1$')
axs[1].plot(t4, r2_sol4, 'g.', label='$r = 0.5$; $\Delta t = 0.01$')
axs[1].plot(t, r2_analy, 'k-.', label='$r = 0.5$; analytical')
axs[1].plot(t1, r4_sol1, 'o-.', label='$r = 2$; $\Delta t = 1$')
axs[1].plot(t2, r4_sol2, '>-.', label='$r = 2$; $\Delta t = 0.5$')
axs[1].plot(t3, r4_sol3, '>-.', label='$r = 2$; $\Delta t = 0.1$')
axs[1].plot(t4, r4_sol4, '.', label='$r = 2$; $\Delta t = 0.01$')
axs[1].plot(t, r4_analy, '-.', label='$r = 2$; analytical')
axs[1].set_xlabel('$t$')
axs[1].set_ylabel('$y$')
# axs[1].set_yscale('log')
axs[1].set_ylim(0,100)
axs[1].legend(shadow=True, handlelength=1, fontsize=12)
# set plotting
plt.rcParams['figure.dpi'] = 500
plt.show()
plt.figure(figsize=(5, 3))
mpl.rcParams.update({'font.size': 16})
```



```
<Figure size 2500x1500 with 0 Axes>
```

In the left sub-figure, we can observe that at the bound for stability criterion the approximation solution is evidently not accurate. It is also observed that when $\Delta t = 0.1$ the approximated solution is generally accurate. When the r value is positive, the approximated solutions propagate the errors along with increasing time.

(3) Using the solver from (2), please reduce the step size, h, to determine the approximate step size where accumulated round-off error becomes important, with respect to solution accuracy, within your particular computational environment (i.e. the combination of hardware, OS, language choice, and selected numerical precision.) Please indicate what all four of these computational environmental parameters are, as part of your answer to this question.

Solution:

In this problem, we first enforce the data type to be `float16`, then decrease the order of the value of h and observe the approximated solution after one step benchmarked by the exact solution. From the output data, we can deduce that when $h = 10^{-3}$ the approximated value diverges to a non-accurate solution. The hardware is a MacBook Pro M1 chip, with a macOS system, using the python language implemented in Google Colab. The numerical precision is float16.

```
[4]: def obtain_discretized_h(r,time,h):
       q_r = np.ones(int(time/h));q_r = q_r.astype('float16')
       t_r = np.linspace(0,time,int(time/h));t_r = t_r.astype('float16')
       t = np.linspace(0,time,100)
       for i in range(0,int(time/h)):
         q_r[i] = q_r[i-1] * (1 + r * h);i += 1
       q_analy = np.exp(r * t)
       return q_r,t_r,q_analy,t

     r1_sol1,r1_t,q_analy,t_a = obtain_discretized_h(1,1,1e-2)
     r2_sol1,r2_t,q_analy,t_a = obtain_discretized_h(1,1,1e-3)
     r3_sol1,r3_t,q_analy,t_a = obtain_discretized_h(1,1,1e-4)
     r4_sol1,r4_t,q_analy,t_a = obtain_discretized_h(1,1,1e-5)
     r5_sol1,r5_t,q_analy,t_a = obtain_discretized_h(1,1,1e-6)
     r6_sol1,r6_t,q_analy,t_a = obtain_discretized_h(1,1,1e-7)
     # rinf_sol1,rinf_t1,q_analy,t_a = obtain_discretized_h(1,10,1e-10)
     # plt.yscale('log')\
     print(q_analy[-1],r1_sol1[-1],r2_sol1[-1],r3_sol1[-1],r4_sol1[-1],r5_sol1[-1],r6_sol1[-1])
```

```
22026.465794806718 20910.0 10190.0 1.0 1.0 1.0 1.0
```