PERSONAL NOTES

Mathematical Modeling

Hanfeng Zhai

2023

CEE 6736: HW #1

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

September 6, 2022

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

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

(1) Use Python to draw the "vector plot" (i.e. slopes within the velocity-time space) for the drone free fall ODE that we derived in class (Lesson 3. . . we sketched out the idea in there). Assume a mass, $m = 10kg$ and drag coefficient, $\gamma = 2\frac{kg}{sec}$. See the web link, under "Homework" within the course website, for guidance on how to draw the vector field plot using *Python* and *matplotlib*. Discuss the results.

The easiest way to create the Python environment for this course is to install Anaconda (see web link under "Software resources" on course website)

Solution:

According to the lecture notes, the governing equation of a pendulum follows (assuming g possesses a positive acceleration, a negative sign are added before it):

$$\frac{dv(t)}{dt} = -g - \frac{\gamma}{m}v \tag{1}$$

where

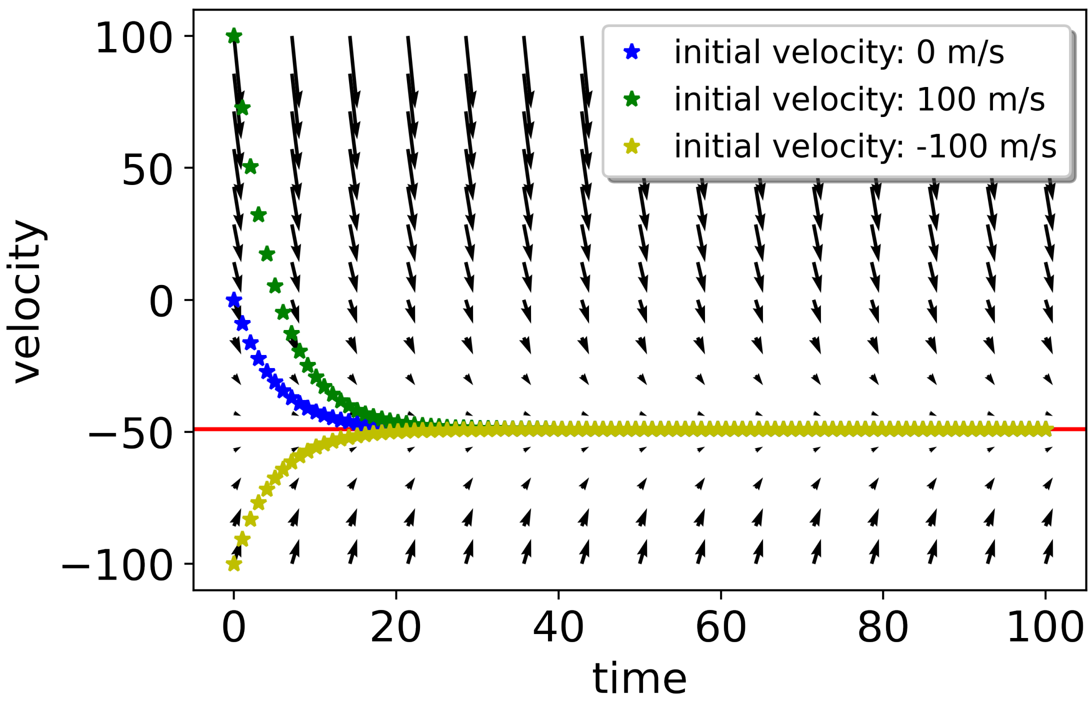
$$\begin{aligned} \gamma &\equiv \text{drag of coefficient} \\ \gamma v &\equiv \text{wind of resistance} \\ m &\equiv \text{mass of quad copter} \end{aligned} \tag{2}$$

By just observing the equation one can discern that when $\dot{v}(t) = 0$, $v_{eq} = 49\text{m/s}$, in which we denote v_{eq} as the equilibrium velocity. By plotting the vector field without solving the equation and mark the equilibrium velocity we observe that the field cnverges to this value.

Further, solving this equation using *Python*, we employ the ODE solver in `scipy`. We first define a function then apply the `odeint` for solution visualizations. Visualizing numerical solutions we observe that the solutions with different ICs also converge to the equilibrium velocity.

[8]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy.integrate import odeint
```

```
t,v = np.meshgrid(np.linspace(0,100,15),np.linspace(-100,100,15)) # generate mesh

# define parameters
gamma = 2
m = 10

# define equations
drag = (gamma/m) * v
g = 9.8
RHS = - g - drag

# define accurate form of the ODE
def drone_freefall(v, t):
  dvdt = - g - (gamma/m) * v
  return dvdt

# solving the equation(s) using odeint
t_dis = np.linspace(0,100,100)
v0_1 = 0
v0_2 = 100
v0_3 = -100
v_sol_1 = odeint(drone_freefall, v0_1, t_dis)
v_sol_2 = odeint(drone_freefall, v0_2, t_dis)
v_sol_3 = odeint(drone_freefall, v0_3, t_dis)

# plotting
plt.quiver(t,v,3,RHS)
plt.axhline(y=-49, color='r', linestyle='-')
plt.plot(t_dis, v_sol_1, 'b*', label="initial velocity: 0 m/s")
plt.plot(t_dis, v_sol_2, 'g*', label="initial velocity: 100 m/s")
plt.plot(t_dis, v_sol_3, 'y*', label="initial velocity: -100 m/s")
plt.xlabel('time')
mpl.rcParams.update({'font.size': 16})
plt.ylabel('velocity')
plt.legend(shadow=True, handlelength=1, fontsize=12)
plt.rcParams['figure.dpi'] = 500
plt.show()
plt.figure(figsize=(5, 3))
```



```
[8]: <Figure size 2500x1500 with 0 Axes>
```

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

More physical intuition regarding this plot is that no matter which initial velocity the object was assigned (or given), it will always converge to a fixed velocity value. However, the converging times are different regarding different initially set velocities. One **Important Point** to note is that the horizontal-directional (i.e. time) portion of the acceleration should be a constant since there is no time term in the RHS of Equation (1). However, selecting the appropriate constant regarding time "direction" is crucial. Based on the fitting with the numerical solution of the colored dots, we think "3" is a decent value. Also, we observe that at a fixed velocity the slope remains the same, as the value can be calculated from Equation (1).

CEE 6736: HW #1

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

September 16, 2022

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

Problem 1

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

(1) Pick a population (e.g. New York City in March of 2020, or a city in the state of Florida in August 2021, etc.) and find the COVID data from the relevant health department. Assume defensible values for X and I in your populations. Use these data to test our COVID transmission model. Discuss the results.

Solution:

Recall the COVID transmission model discussed in class. First defining: $\mathbb{N}_i \equiv$ Number of infections on a given day i; $\mathbb{X} \equiv$ Expected numbers of daily contacts per infected person; $\mathbb{I} \equiv$ Probability of infection for each contact. The infection for the next day writes:

$$\mathbb{N}_{i+1} = (\mathbb{I}\mathbb{X} + 1)\mathbb{N}_i$$

This model can be further expressed as, at a certain day i, the infectious number follows:

$$\mathbb{N}_i = (1 + \mathbb{I}\mathbb{X})^i \mathbb{N}_0$$

where $\mathbb{N}_0$ denotes the initial values at $t = 0$. Promoting this to a continuous model:

$$\frac{d\mathbb{N}}{dt} = (\mathbb{I}\mathbb{X})\,\mathbb{N}$$

where $\mathbb{I}\mathbb{X}$ is identified as the $\mathcal{R}$ factor.

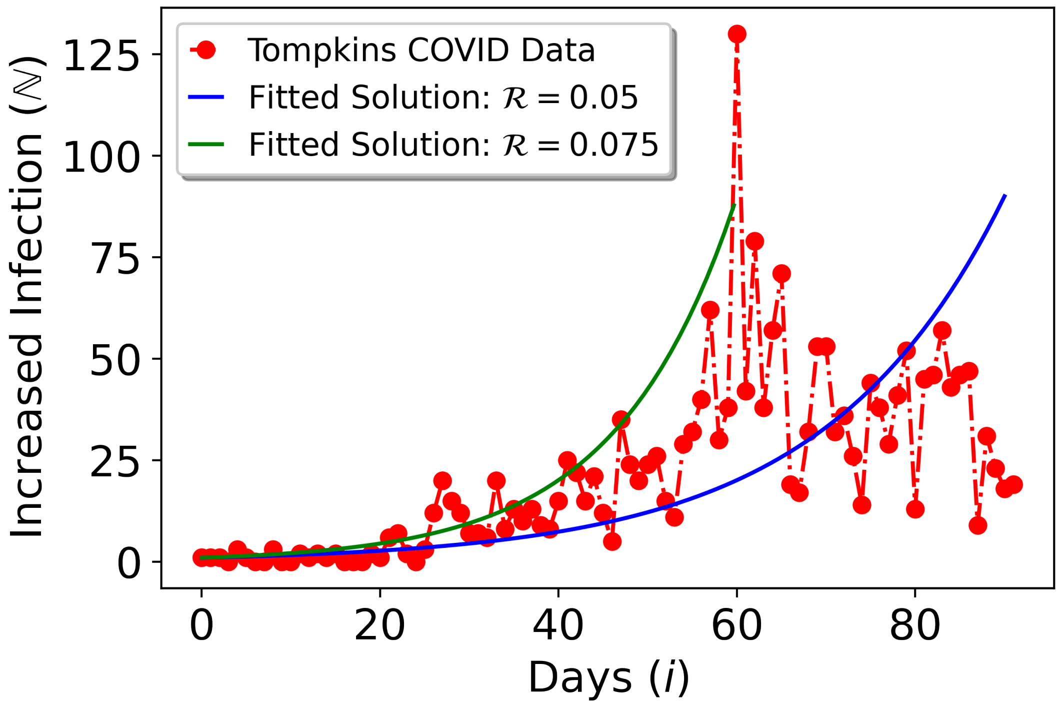
This is a special case that follows the linear ODE form when the bias term equals zero. Hence, the solution ought to follow the basic form: $\mathbb{N} = Ce^{\mathcal{R}t}$, where C is a constant. Based on the data fitting, it seems like a good fit for $\mathcal{R}$ is 0.05. Assuming an (reasonable) $\mathbb{I}$ of 0.001, the $\mathbb{X}$ is hence 50.

We now apply the data of Tompkins county (from July 2021 to Oct 2021) to test the model, starting from loading the data from `.txt` file:

```
[ ]: import numpy as np
     import matplotlib.pyplot as plt
     import matplotlib as mpl
```

```
from scipy.integrate import odeint

t_discretized = np.linspace(0,90,90)
N_real = np.exp(.05*t_discretized)
N_real_2 = np.exp(.075*t_discretized[0:60])

tompkins_data = np.loadtxt("tompkins_data.txt")
num_infec = np.flip(tompkins_data.T[2])
daily_text = np.flip(tompkins_data.T[1]);daily_text_identity = daily_text/np.
 ↪mean(daily_text)
total_infec = np.flip(tompkins_data.T[3]);total_infec_identity = total_infec/np.
 ↪mean(total_infec)
plt.plot(num_infec,'ro-.',label='Tompkins COVID Data')
plt.plot(t_discretized,N_real,'b-',label='Fitted Solution: $\mathcal{R} = 0.05$')
plt.plot(t_discretized[0:60],N_real_2,'g-',label='Fitted Solution: $\mathcal{R}␣
 ↪= 0.075$')
plt.xlabel("Days ($i$)")
plt.ylabel("Increased Infection ($\mathbb{N}$)")

plt.legend(shadow=True, handlelength=1, fontsize=12)
```

```
[ ]: <matplotlib.legend.Legend at 0x7f445026c390>
```



Problem 2

Determine if the equation of a line ($y = mx + b$) satisfies the *Axioms of Linearity*. Discuss the results.

Solution:

Let $f(x) = y = mx + b$, recall the definition of axioms of linearity,

$$cf(x_1 + x_2) = cf(x_1) + cf(x_2)$$

Substitute it into $f(x)$ we have:

LHS $= c(m(x_1 + x_2) + b) = cmx_1 + cmx_2 + cb$RHS $= c(mx_1 + b) + c(mx_2 + b) = cmx_1 + cmx_2 + 2cb$

It is observed that LHS $\neq$ RHS. Hence the line does not satisfy the axiom of linearity. What's more, it can be determined if the bias term b is eliminated, the line hence obeys the axiom of linearity.

Problem 3

Use the ODE solution method discussed in class (*i.e.* $\frac{d}{dt} ln|\cdot|$) to exactly solve the drone ODE from HW1 using mass, $m = 10kg$ and the drag coefficient, $\gamma = 2kg/sec$ and the initial condition: $v(0) = 0$. Please plot the exact solution within the vector plot from HW1 and discuss.

Solution:

Recall the governing equation of the drone free fall, the velocity writes:

$$\frac{dv(t)}{dt} = -g - \frac{\gamma}{m} v$$

where

$$\gamma \equiv \text{drag of coefficient}$$
$$\gamma v \equiv \text{wind of resistance}$$
$$m \equiv \text{mass of quad copter}$$

written in the standard form

$$\frac{dv(t)}{dt} + \frac{\gamma}{m} v = -g$$

applying the integration factor $\mu(t) = e^{\frac{\gamma}{m}t}$, the equation can be rewrite in the form given the fact that $\frac{d\mu(t)}{dt} = \frac{\gamma}{m}\mu$:

$$\frac{d}{dt}\left[e^{\frac{\gamma}{m}t} v(t)\right] = -e^{\frac{\gamma}{m}t} g$$

the solution further writes:

$$v(t) = e^{-\frac{\gamma}{m}t} \int_{t_0}^{t} e^{\frac{\gamma}{m}s}(-g) \cdot ds + Ce^{-\frac{\gamma}{m}t}$$

skipping the detailed derivation, we directly land in the final solution form of $v(t)$:

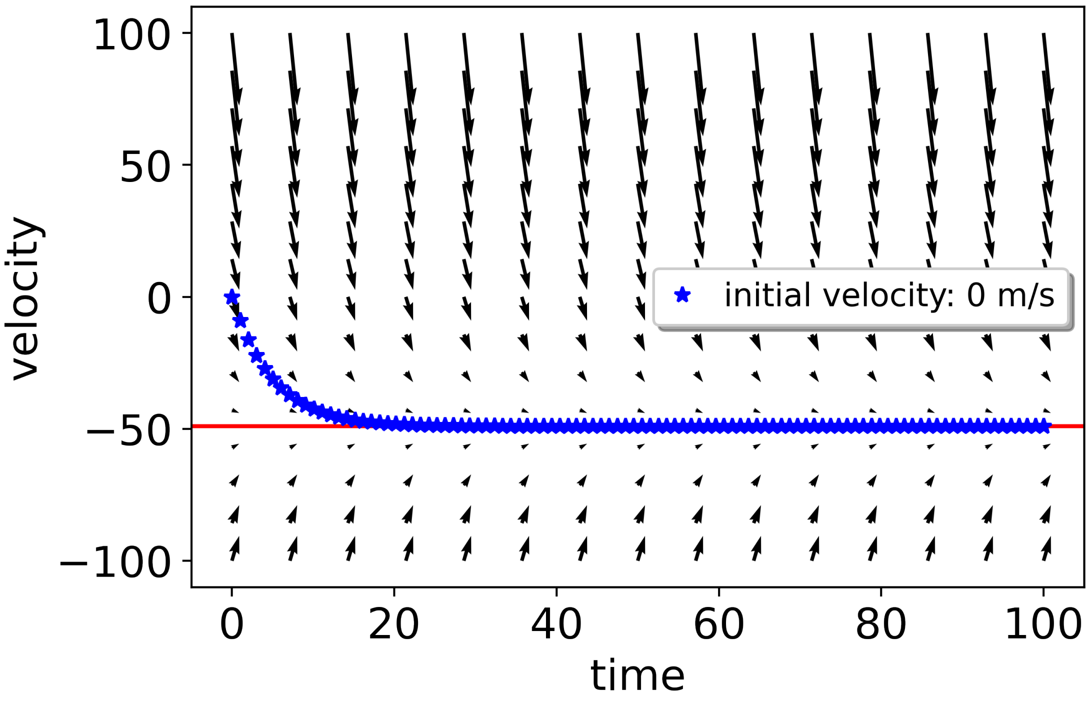
$$v(t) = \frac{1}{\mu(t)} \left[\int_{t_0}^{t} \mu(s)(-g) \cdot ds + C\right]$$

To determine the constant C, we substitute the given initial values: $v(0) = 0 \rightarrow C = 5g$. We, therefore, obtain the solution:

$$v(t) = \frac{1}{e^{\frac{\gamma}{m}t}}\left[-g\int_{t_0}^{t} e^{\frac{\gamma}{m}s}\cdot ds + 5g\right] = \frac{1}{e^{\frac{\gamma}{m}t}}\left[-g\frac{m}{\gamma}e^{\frac{\gamma}{m}t} + 5g\right] = \frac{1}{e^{0.2t}}\left[-5ge^{0.2t} + 5g\right]$$

We, therefore, plot the graph:

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

     t,v = np.meshgrid(np.linspace(0,100,15),np.linspace(-100,100,15)) # generate mesh

     # define parameters
     gamma = 2
     m = 10

     # define equations
     drag = (gamma/m) * v
     g = 9.8
     RHS = - g - drag

     sol_exact = (1/np.exp(0.2*t_dis)) * (-5*g*np.exp(0.2*t_dis) + 5*g)
     # print(np.shape(sol_exact))
     # solving the equation(s) using odeint
     t_dis = np.linspace(0,100,100)

     # # plotting
     plt.quiver(t,v,3,RHS)
     plt.axhline(y=-49, color='r', linestyle='-')
     plt.plot(t_dis, sol_exact, 'b*', label="initial velocity: 0 m/s")
     plt.xlabel('time')
     mpl.rcParams.update({'font.size': 16})
     plt.ylabel('velocity')
     plt.legend(shadow=True, handlelength=1, fontsize=12)
     plt.rcParams['figure.dpi'] = 500
     plt.show()
     plt.figure(figsize=(5, 3))
```


100
50
0
−50
−100
velocity
0
20
40
60
80
100
time
initial velocity: 0 m/s

```
[ ]: <Figure size 2500x1500 with 0 Axes>

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

CEE 6736: HW #1

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

September 22, 2022

[65]:
```
# !sudo apt install cm-super dvipng texlive-latex-extra texlive-latex-recommended
```

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

(1) Please revisit our example from class, where the pond experienced salt water runoff from a bridge crossing over it. Please use Python to plot the salt concentration, $q(t)$, given in metric tons, as it fluctuates during the first 20 years that the bridge is in service. I am simply asking you to plot the solution to our math model that we obtained in class.

Solution:

Recall the governing equation for pond runoff discussed in class (in metric tons case):

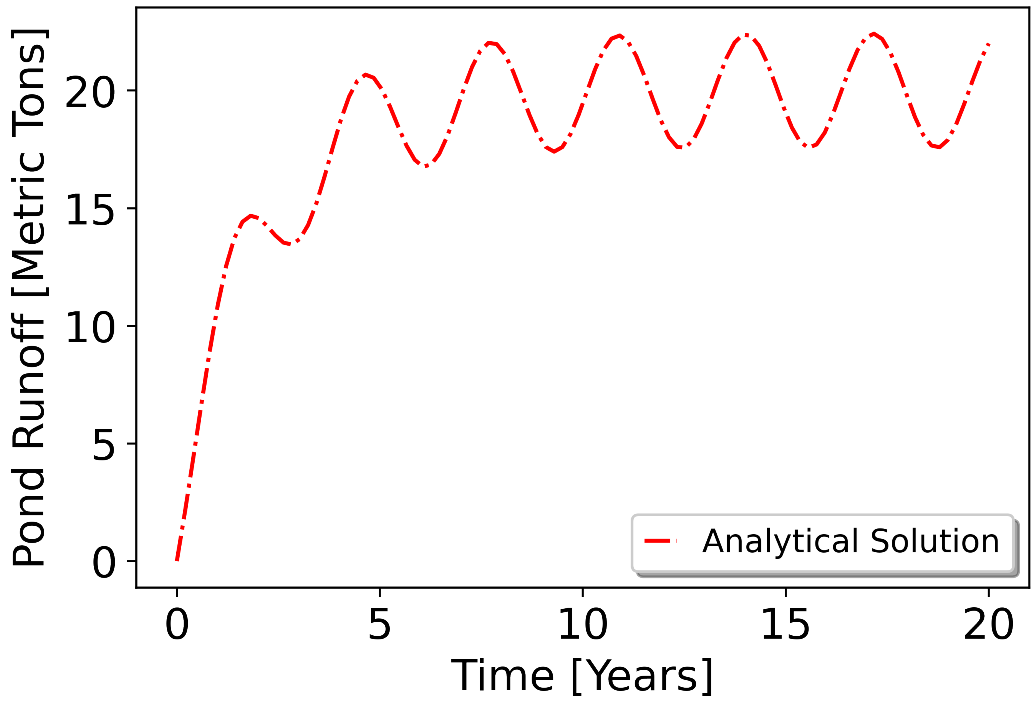
$$\frac{dq}{dt} + \frac{1}{2}q = 10 + 5\sin(t)$$

Consider there is no runoff at the "very beginning": $q(0) = 0$. Recall the analytical solution given in the lecture:

$$q(t) = 20 - \frac{40}{17}\cos(2t) + \frac{10}{17}\sin(2t) - \frac{300}{17}e^{-\frac{t}{2}}$$

Plotting this:

[5]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from scipy.integrate import odeint

t = np.linspace(0,20,100)
q_analy = 20 - (40/17)*np.cos(2*t) + (10/17)*np.sin(2*t) - (300/17)*np.exp(-t/2)

plt.plot(t,q_analy,'r-.',label='Analytical Solution')
plt.xlabel("Time [Years]")
plt.ylabel("Pond Runoff [Metric Tons]")

# set plotting
plt.legend(shadow=True, handlelength=1, fontsize=12)
plt.rcParams['figure.dpi'] = 500
plt.show()
plt.figure(figsize=(5, 3))
mpl.rcParams.update({'font.size': 16})
```



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

(2) Please write a small Python program that uses Euler's method to approximate the same response of our pond system during the first 20 years. Vary time step size, Δt, and compare the approximate solutions against the closed form solution for this problem (given in class, and used in (1)).

Solution:

Recall the Euler method discretization taught in class, the derivative is discretized in the form (lecture notes):

$$\frac{q_1 - q_0}{\Delta t} = 10 + 5\sin(2t) - 0.5q_0$$

where the current point q_1 can be computed as

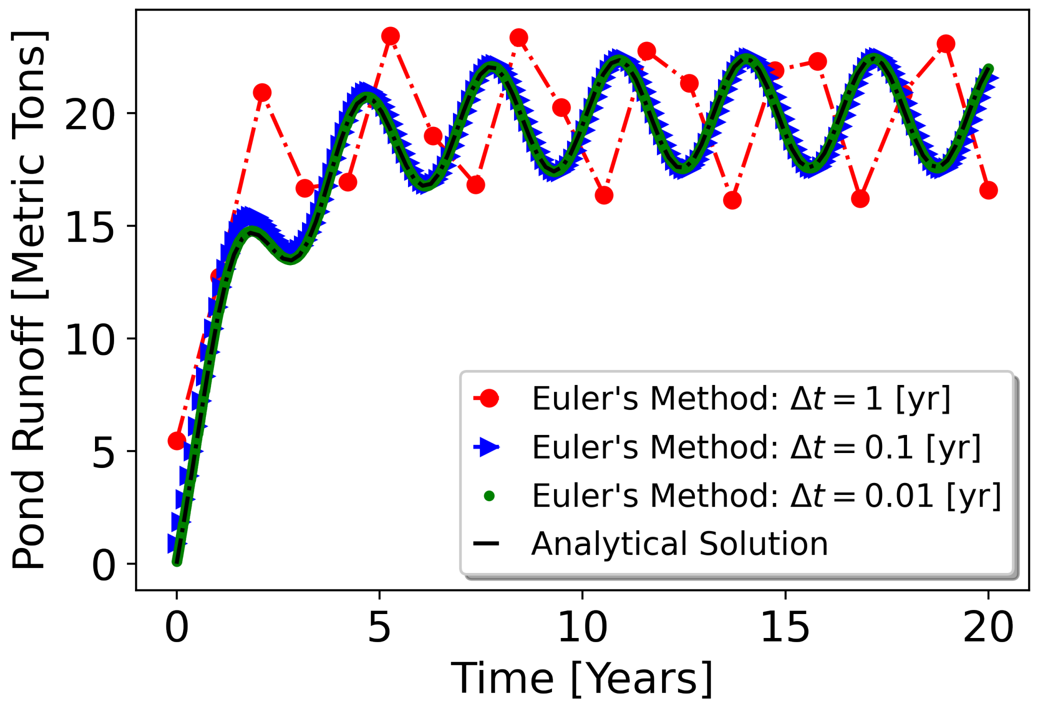
$$q_1 = q_0 + \underbrace{(10 + \sin(2t) - 0.5q_0)}_{f(t,q_0)}(t - t_0)$$

This can be promoted to a more general case as

$$\underbrace{q_i}_{\text{current step}} = \underbrace{q_{i-1}}_{\text{previous step}} + \left(10 + \sin(\underbrace{t_{i-1}}_{\text{previous time}}) - 0.5\underbrace{q_{i-1}}_{\text{previous step}}\right)\Delta t$$

We hence write a loop to solve this:

[4]:
```
year = 20.0
q_t1 = np.zeros(20)
q_t2 = np.zeros(200)
q_t3 = np.zeros(2000)
Delta_t1 = 1
Delta_t2 = .1
Delta_t3 = .01
q0 = 0
i=0
for i in range(0,int(year/Delta_t1)):
  q_t1[i] = q_t1[i-1] + np.float64((10 + 5*np.sin(2*((i-1)/1)) - .5*q_t1[i-1]) *␣
 ↪Delta_t1)
  i += 1
for i in range(0,int(year/Delta_t2)):
  q_t2[i] = q_t2[i-1] + np.float64((10 + 5*np.sin(2*((i-1)/10)) - .5*q_t2[i-1])␣
 ↪* Delta_t2)
  i += 1
for i in range(0,int(year/Delta_t3)):
  q_t3[i] = q_t3[i-1] + np.float64((10 + 5*np.sin(2*((i-1)/100)) - .5*q_t3[i-1])␣
 ↪* Delta_t3)
  i += 1

t_1 = np.linspace(0,20,20)
t_2 = np.linspace(0,20,200)
t_3 = np.linspace(0,20,2000)
plt.plot(t_1, q_t1, 'ro-.', label='Euler\'s Method: $\Delta t = 1$ [yr]')
plt.plot(t_2, q_t2, 'b>-.', label='Euler\'s Method: $\Delta t = 0.1$ [yr]')
plt.plot(t_3, q_t3, 'g.', label='Euler\'s Method: $\Delta t = 0.01$ [yr]')
plt.plot(t,q_analy, 'k-.', label='Analytical Solution')
plt.xlabel("Time [Years]")
plt.ylabel("Pond Runoff [Metric Tons]")

# set plotting
plt.legend(shadow=True, handlelength=1, fontsize=12)
plt.rcParams['figure.dpi'] = 500
plt.show()
plt.figure(figsize=(5, 3))
mpl.rcParams.update({'font.size': 16})
```



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

(3) Please discuss your results from (1) and (2)

Solution:

As we are discretizing the solutions, when $\Delta t = 1$ [yr], the discretized solution is highly inaccurate — the solution points are observed to be "lagged behind" the standard analytical solution. Increasing discretization fidelity to $\Delta t = 0.1$ and 0.01 the approximated solutions are generally accurate — observed to be agreeing well with the analytical solution.

(4) Please analytically solve the following ODE using separation of variables: $\frac{dy}{dx} = \frac{4x-x^3}{4+y^3}$

Solution:

Rewrite the equation in the form:

$$(4+y^3)dy = (4x-x^3)dx$$

Integrate both sides:

$$\int (4+y^3)dy = \int (4x-x^3)dx$$

The general solution writes:

$$4y + \frac{1}{4}y^4 + C = 2x^2 - \frac{1}{4}x^4$$

Or written in the form:

$$2x^2 - \frac{1}{4}x^4 - 4y - \frac{1}{4}y^4 + C = 0$$

which is an implicit solution.

Now, let $x^2 = \mathcal{X}$, the equation writes:

$$-\frac{1}{4}\mathcal{X}^2 + 2\mathcal{X} = 4y + \frac{1}{4}y^4 + C$$

The explicit solution can be obtained via the quadratic equation:

$$\mathcal{X}_1 = 4 - 2\sqrt{4 + 4y + \frac{1}{4}y^4 + C}$$
$$\mathcal{X}_2 = 4 + 2\sqrt{4 + 4y + \frac{1}{4}y^4 + C}$$

The solutions of the overall equation hence write:

$$x_1 = \sqrt{4 - 2\sqrt{4 + 4y + \frac{1}{4}y^4 + C}}$$
$$x_2 = -\sqrt{4 - 2\sqrt{4 + 4y + \frac{1}{4}y^4 + C}}$$
$$x_3 = \sqrt{4 + 2\sqrt{4 + 4y + \frac{1}{4}y^4 + C}}$$
$$x_4 = -\sqrt{4 + 2\sqrt{4 + 4y + \frac{1}{4}y^4 + C}}$$

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.)

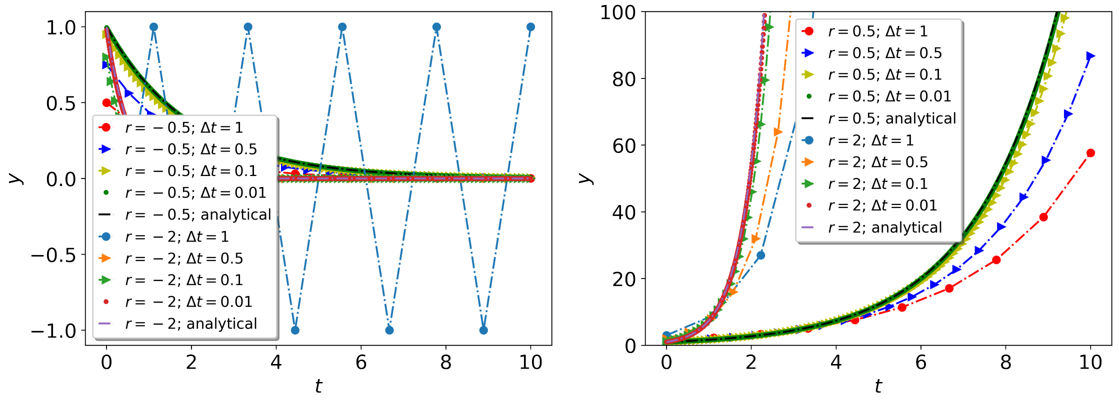
Solution:

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
```

Learning the stress mapping of composite materials with Fourier Neural Operator in the context of small data*

Hanfeng Zhai

Sibley School of Mechanical and Aerospace Engineering
Cornell University
Ithaca, New York 14853
Email: hz253@cornell.edu

Composite and porous materials have been widely applied in various industries including aerospace, civil infrastructure, automobile, bioengineering, electronics, etc., where the microstructural and topological design for tailored properties has been of importance. Towards this goal, the first step is usually to build up a surrogate model for fast evaluation of the physical fields. In this work, we employed the developed Fourier Neural Operator (FNO) architecture to do real-time inference of the von Mises' stress field of set digital composite materials. A hypothesis is set between the mapping of the input materials representation to its corresponding von Mises' stress distribution calculated from linear elastic solid mechanics for surrogate modeling with FNO. The inference results show satisfactory accuracy, and further, push up toward our future work of inverse design and neural operator benchmarking.

1 Introduction

Obtaining decent designs of materials, structures, and systems, via probing the design variables is a long-standing problem in various scientific and engineering fields. Recently, with the rapid developments of machine learning (ML) and its applications in optimization and computing, many researchers have applied various ML frameworks for different design problems, such as CNN for porous graphene design [1], Gaussian process for antibiofilm nanosurfaces design [2], CNN for composites design [3]. All these works rely on utilizing state-of-the-art machine learning (ML) structures for constructing a surrogate for materials properties evaluation for the inverse problem. In this course project, I mainly investigate how to construct such forward surrogate models using the recently developed neural operator methods.

There have been rapid developments in applying ML for mechanics problems, mainly on how to construct *surrogate models* efficiently. From the multiscale perspective, *ab initio* based machine learning potential bridges the scale from quantum (or more rigorously in the scale of density functional theory) to the molecular regime [4, 5]. There are follow-up works adopting the same ideology bridging the molecular scale to the mesoscale, i.e. the coarse-grained regime [6] by E and coworkers. Up to the micro- and macroscale, there are many works that leverage high-resolution simulations and use various ML tools to build up surrogate models, like predicting bubble dynamics [7]. In general, the key idea is to train a surrogate model using the regressor of convenience with the corresponding optimization algorithms, i.e. GD, SGD, RMSProp, and Adam.

There is a rise in the efforts to integrate known theories in ML since around 2015. Some representative works include using sparse regression to identify dynamics systems' mathematical expression, and encoding dynamical systems by Brunton and coworkers [8–10]; encoding known equations into neural networks via automatic differentiation for more accurate inference of physical systems by Karniadakis and coworkers [11–13]; and most importantly and recently, applying universal approximation theory to map functional spaces and extended to deep nets called operator regression [14]. There are some major debates on the performances and structures of different operator networks, more specifically debates over deep operator networks and Fourier neural operator [15, 16]. Despite the plural network architectures, we here are interested in how the Fourier Neural Operator (FNO), and future benchmarking of different neural operators, perform on predictive materials design and deployment applications.

Thanks to their tunable and flexible properties, composites, have been widely employed in various industries. To rapidly assess the properties and design composite ma-

*Submitted as a course paper for *Mathematical Modeling of Natural and Engineered Systems* terials, the first step is usually to obtain the stress distribution on the materials under a certain loading. Calculating this stress distribution can somehow be time-consuming. In other senses, even if we can fast evaluate the stress mapping, it is still computationally burdensome to obtain the stress mapping for the vast amounts of different composite structures. Here, we leverage the general FNO architecture to map the structure of the composite materials to its corresponding von Mises' stress distribution for "on-the-fly" materials evaluation.

This short report is arranged as follows: In section 2 we briefly introduce how the problem is set up, including basic theoretical and numerical setup for the solid mechanics problems (Sec. 2.1 & Sec. 2.2) and the materials choices (Sec. 2.3). In section 3 we introduce how to generate the training data (Sec. 3.1) and review the formulation of FNO (Sec. 3.2). In section 4 we present the results by FNO predictions by analyzing the data distribution and showed some stress mapping comparison results. Eventually, we concluded the paper and present some future plans in section 5. Based on my future plans, eventually, I hope these efforts could lead to a publication.

2 Problem Formulation

The key and simple problem we are interested in is solving the stress distribution on composite materials based on different geometries. To obtain the stress field, one would need the materials' structures, know the corresponding initial and boundary conditions, generate computational meshes for spatial distribution, and composite the corresponding stress distribution based on finite element simulations. To generate different geometries, we set up a so-called "materials basis" on a 5×5 grid, where we generate 10 random locations within the basis. These 10 pores are set as carbon, with the rest material matrix set as Poly(methyl methacrylate) (PMMA), together composing the composite material unit volume. Here, we use linear elastic solid mechanics (COMSOL Multiphysics®) to solve for the stress distributions. We denote the representation of the materials as $a \in \mathscr{A}$ where $\mathscr{A}$ are the totally different materials combination possibilities, and our targeted physical property, i.e., the stress field as $u \in \mathscr{U}$.

2.1 Linear Elastic Solid Mechanics

The stress distributions on the composite materials are solved based on the linear elasticity model. In the closed physical system, i.e. composite materials computational domain, the equations for equilibrium shall be satisfied:

$$\nabla \cdot \sigma + \mathbf{F}\mathbf{v} = 0 \tag{1}$$

where σ is the stress field to be computed, $\mathbf{F}$ is the external forces applied to the body and $\mathbf{v}$ is the object's velocity. Here, since we are investigating a static problem, Equation (1) can be reduced to $\nabla \cdot \sigma = 0$

The stresses σ and strains ϵ distributions on the body obeys the linear elasticity rule:

$$\sigma = \mathbf{C} : \epsilon_{\text{elastic}} \tag{2}$$

where ϵ is the elastic strain and $\mathbf{C}$ is the elasticity matrix.

The overall strains take the form of the gradients of the displacement fields $\mathbf{u} = [u_1, u_2]$:

$$\epsilon = \frac{1}{2}\left[(\nabla \mathbf{u})^\top + \nabla \mathbf{u}\right] \tag{3}$$

And the elasticity matrix is a function of the Poisson's ratio ν and elastic modulus E:

$$\mathbf{C} = \mathbf{C}(E, \nu) \tag{4}$$

In our approach, we use a plane strain approach with homogeneous isotropic materials, where the general linear elasticity relation between the stresses and strain can be expanded to

$$\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{12}\end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix}1-\nu & \nu & 0\\ \nu & 1-\nu & 0\\ 0 & 0 & 1-2\nu\end{bmatrix}\begin{bmatrix}\epsilon_{11}\\ \epsilon_{22}\\ \epsilon_{12}\end{bmatrix} \tag{5}$$

In the computation, we obtain the von Mises' stress based on the full field stress data, written as (in the case of plane strain):

$$\sigma_{\text{vM}} = \sqrt{\frac{1}{2}\left[(\sigma_{11}-\sigma_{22})^2 + (\sigma_{22}-\sigma_{33})^2 + (\sigma_{33}-\sigma_{11})^2\right] + 3\sigma_{12}^2} \tag{6}$$

where the normal stress in the 3-direction is solved from the other normal stresses and Poisson's ratio:

$$\sigma_{33} = \nu(\sigma_{11} + \sigma_{22}) \tag{7}$$

2.2 Bidirection Tensile-Compression Tests

In this problem, we avoid periodic boundary conditions to make the problem more complex, i.e., harder for the ML model to learn. We hence applied loading in 2 directions and fix the bottom side.

Assuming the two directions in the horizontal and vertical axes are x_1 and x_2, the exact boundary conditions are hence written as (in SI unit):

$$\begin{aligned} x_2 = 0 &: \mathbf{u} = 0,\\ x_1 = 0 &: \sigma_{xx} = -100,\\ x_1 = 0.1 &: \sigma_{xx} = 100,\\ x_2 = 0.1 &: \sigma_{yy} = -100. \end{aligned} \tag{8}$$

Note that the unit for the geometry, i.e., the square lengths can be neglected since the geometry is treated as a unitless computational domain in the simulations. Hence, as long as the ratio of x_1/x_2 remains the same the results should be the same. But the unit has to satisfy the continuum assumption in solid mechanics. For example, if one assumes $x_1 = 0.1$ nm or Ångstrom should not be valid.

2.3 Materials Properties

The composite materials are assumed to be a fiber-reinforced microstructure with carbon fiber embedded in the soft PMMA matrix. The carbon is considered a simple time- and temperature-independent linear elastic material, with given properties shown in Table 1

E	2.5×10^{12}
ν	0.188
ρ	2.267×10^{3}

Table 1. The materials properties for carbon, in SI unit. E is the elastic modulus, ν is the Poisson's ratio and ρ is the density.

For the PMMA, both the elastic moduli, Poisson's ratio, and density are functions of temperatures T. Here, the temperature is fixed at 293.15K during the computation process, the corresponding materials properties for PMMA are shown in Table 2.

E	6.1742×10^{9}
ν	0.3216
ρ	1.900×10^{3}

Table 2. The materials properties for PMMA, in SI unit.

Given the properties of our materials, and based on the mentioned computation process in sections 2.1 and 2.2, we can write our formulated problem simplified as

$$a \xrightarrow{h,p_h} [x_1, x_2] \xrightarrow{E,\nu,\rho,p_s} [\sigma_{11}, \sigma_{12}, \sigma_{22}] \rightarrow \sigma_{\mathrm{vM}} = u \tag{9}$$

where h stands for the mesh sizes, and p_h is the related meshing parameters; p_s stands for the parameters in the solvers employed. This whole process can be further simplified to a mapping

$$a \in \mathscr{A} \xrightarrow{FEM} u \in \mathscr{U} \tag{10}$$

where the main task here is to find a surrogate for FEM so that can infer u "on-the-fly" from a as a forward problem. Here, we instead approximate the mapping of $\mathscr{A} \rightarrow \mathscr{U}$, with the developed FNO method by approximating in a high-dimensional space, so the maps between any subsets a and u are theoretically to be covered and predicted.

3 Methodology

3.1 Data Generation

To solve for the stress based on given representations, a learning task to approximate the mapping is formulated with small data. The general workflow for generating the training data is shown in Figure 1. Since the problem is formulated as the location randomization of the 10 carbon fiber locations on the 5×5 materials basis, there are expected to have $C_5^5 = 3268760$ possible materials combinations. As to challenge the approximation & representation capability of the neural operator, we only run 100 FEM simulations, approximately 0.0031% of the full possible materials representations. With the small dataset, we use the classic 80/20 split to generate the training and testing data, i.e., the FNO is trained on 80 FEA simulations that map the materials' representations to their von Mises' stress and infer the stress distribution of the rest 20 simulations. Here, based on the computed Gauss von Mises' stress distribution, we map the field data on a 512×512 matrix or can be treated as an image representation, i.e., 512×512 pixels.

There are two main challenges in the learning task: (1) the dataset is very small, makes it difficult for any learners to approximate the operator; (2) The stress distribution is computed based on non-periodic boundary conditions, making it more challenging to approximate the intrinsic trend in the data, as a little variation of carbon location may induce a nonlinearly increase of the stress that is very difficult to be captured.

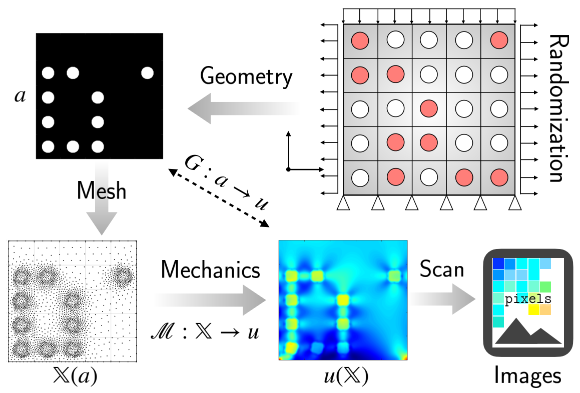


Fig. 1. The general schematic of this project. Random locations are generated based on the materials basis grid for a digital composite structure representation. Its corresponding meshes are then generated for linear elastic solid mechanics calculation of the von Mises' stress field. The field can then be scanned as an image for training.

3.2 Fourier Neural Operator

Different from the traditional numerical solver or recently developed data-driven methods (e.g., Gaussian process regression or neural networks), the neural operator strives to approximate the mapping between infinite dimensional spaces, theoretically, given any input-output pairs (i.e., the training data). Li et al. [17] propose the approximation of the mapping between two separable Banach spaces $\mathscr{A} = \mathscr{A}(D;\mathbb{R}^{d_a})$, $\mathscr{U} = \mathscr{U}(D;\mathbb{R}^{d_u})$: there exists a possible operator projection $G : \mathscr{A} \to \mathscr{U}$, where $D \subset \mathbb{R}^d$ is a bounded open set and numerical values live in spaces $\mathbb{R}^{d_a}$ and $\mathbb{R}^{d_u}$. The authors assume the map can be parameterized by $\theta \in \Theta$ based on given N observed data $\{a_j, u_j\}_j^N$. Given the observations, the materials representations $a_j \sim \mu$ is an i.i.d. sequence from the probability measure μ. The operator mapping thence writes:

$$G : \mathscr{A} \times \Theta \to \mathscr{U} \tag{11}$$

where in our problem the materials representation writes $a \in \mathscr{A}$ maps to the von Mises' stress $u \in \mathscr{U}$, where the observed data $a, u \in \mathbb{R}^{512\times512}$. In the learning task, the loss function seeks to minimize the expectation of the errors between the observations and operator-inferred data in the stochastic process for searching the optimal parameters, $\mathscr{U} \times \mathscr{U} \to \mathbb{R}$:

$$\min_{\theta\in\Theta} \mathbb{E}_{a\sim\mu} \left[\mathscr{L}(\hat{G}(a;\theta), G(a)) \right] \tag{12}$$

Li et al. [17] propose an iterative update process in a higher dimensional space to update the function that lives in the so-called Fourier layer, $v(x)$, where x can be treated as the discretized observations: $v_0 \longmapsto v_1 \longmapsto \dots \longmapsto v_T$ where v are the functions taking values from $\mathbb{R}^{d_v}$. This iterative process is illustrated in Figure 2, combined with schematics for mapping different dimensions from the training images. The iterative updates take the form:

$$v_{t+1}(x) := \sigma\left(W v_t(x) + \mathscr{F}^{-1}\left(\mathscr{F}(\kappa_\phi) \cdot \mathscr{F}(v_t) \right)(x) \right), \quad \forall x \in D \tag{13}$$

where $\mathscr{F}^{-1}\left(\mathscr{F}(\kappa_\phi) \cdot \mathscr{F}(v_t) \right)(x)$ is the Fourier integral operator by applying convolution in the Fourier space. Note that $\mathscr{F}$ and $\mathscr{F}^{-1}$ are the Fourier and inverse Fourier transform, respectively, on a function $v : D \to \mathbb{R}^{d_v}$

$$\begin{aligned} (\mathscr{F} v)_j(k) &= \int_D v_j(x) e^{-2i\pi\langle x,k\rangle} dx, \\ (\mathscr{F}^{-1} v)_j(k) &= \int_D v_j(k) e^{2i\pi\langle x,k\rangle} dk \end{aligned} \tag{14}$$

Since the learning is conducted in a higher dimensional space, i.e., $\mathbb{R}^{d_v}$ for the operator approximation as iterating functions v, the authors apply two shallow neural networks, P and Q, that maps the observation data into and out of the Fourier space $\mathbb{R}^{d_v}$, respectively. We can simplify these processes as $v_0(x) = P(a(x))$ and $u(x) = Q(v_T(x))$ that map $\mathbb{R}^{d_a} \to \mathbb{R}^{d_v}$ and $\mathbb{R}^{d_v} \to \mathbb{R}^{d_u}$, respectively.

The overall architecture of FNO can then be simplified to a combination of two shallow networks P and Q for dimensional mapping and an iterative algorithm for function update in the Fourier spaces. Suppose there are T hidden layers in the Fourier spaces, by adding the biases to the iteration algorithm, the FNO architecture writes:

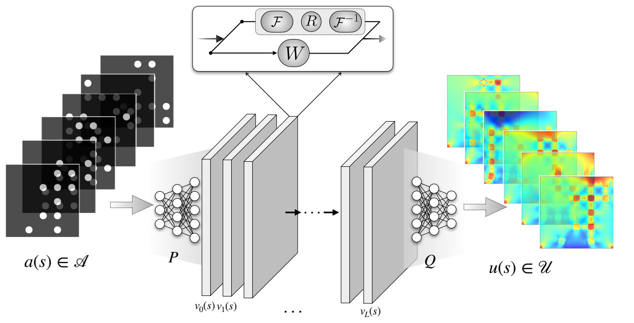


Fig. 2. The schematic and architecture for learning the structure-stress mapping with Fourier Neural Operator. The materials representation is input as images for the combined linear and Fourier convolution with an output stress field for the supervised learning task. The two neural networks represent the P and Q networks, respectively.

$$\begin{aligned} v_0(x) &= P(a(x), x), \\ v_{t+1}(x) &:= \sigma\left(W v_t(x) + \mathscr{F}^{-1}\left(\mathscr{F}(\kappa_\phi) \cdot \mathscr{F}(v_t) \right)(x) + b_t(x) \right), \\ u(x) &= Q(v_T(x)). \end{aligned} \tag{15}$$

FNO is then trained on FEM simulations data of $\{a_j, u_j\}_{j=1}^{80} \in \mathbb{R}^{512\times512} \to \mathbb{R}^{512\times512}$ for the inference of testing data $\{a_k, u_k\}_{k=1}^{20}$.

4 Results and Discussion

Figure 3 shows the prediction accuracy for the 20 cases as to normalize the stress distribution in the range of [0,1]. The blue dots are the inference-testing data points and the red line benchmarks the perfect fit, i.e., 100% accuracy. Most cases exhibit satisfactory inference accuracy as they generally match the trend of the benchmark red line. Both cases exhibit an interesting "upward" trend: the predicted stress distribution is slightly smaller than the training data when the stress values are high. Some cases are not so accurate with obvious convex-shaped data distribution, e.g., cases VIII, XIII, XIV, etc. This could be induced by the non-periodic boundary conditions induced stress value variation.

To better explain and unveil a deeper outlook for the inferences in Figure 3, we generate Figure 4 to show the probability distribution combined with a Gaussian fit of both the testing data and the inferences. It can be observed that cases VIII and XIII both show decent matching of the probability distribution, whereas XIV shows an evident data mismatch between the benchmark and the inference. Also, although observed to be acceptably accurate, case XX's data distribution shows that the inference exhibits a pretty high variability.

To verify the observations and primary conclusions drawn from comparing Figures 3 and 4, we further generate Table 3 to show the actual relative errors $|\bar{\mathscr{E}}|$ between

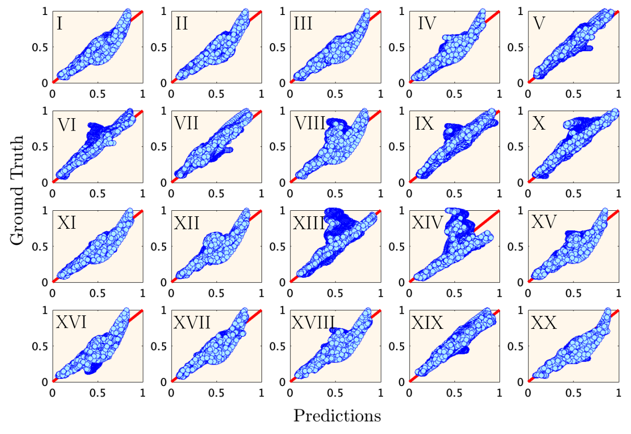


Fig. 3. The 1D data linear fit for the 20 test cases. The blue dots represent the actual data mapping between the test data (vertical axis) and the FNO-predicted stress data (horizontal axis). The red lines represent the ground truth mapping. Note that the data are represented in the normalized range of 0 to 1.

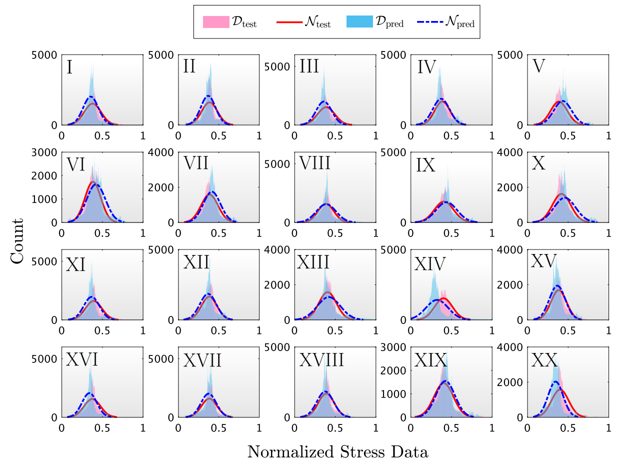


Fig. 4. The data distribution and the corresponding Gaussian fit for both the normalized stresses of the predicted and ground truth for the 20 test cases. The shallow pink ($\mathcal{D}_{\text{test}}$) and blue ($\mathcal{D}_{\text{pred}}$) data bars stand for the ground truth and predictions, respectively. The red lines ($\mathcal{N}_{\text{test}}$) and blue dotted lines ($\mathcal{N}_{\text{pred}}$) stand for the Gaussian distribution fit for $\mathcal{D}_{\text{test}}$ and $\mathcal{D}_{\text{pred}}$, respectively.

the inferences and the testing data computed from

$$|\bar{\mathscr{E}}| = \left|\frac{u_{pred} - u_{test}}{u_{test}}\right| \tag{16}$$

where u_{pred} are the von Mises' stresses inferred by FNO and u_{test} are the stresses in the test data. From Table 3 we deduce that two observed cases with high data distribution variances, XIV and XX, do have a higher $|\bar{\mathscr{E}}|$. What's more, case V with a higher $|\bar{\mathscr{E}}|$ also has a higher data distribution mismatch in Figure 4, even though it is observed to be relatively accurate from Figure 3. Case XIII, although observed to be extremely inaccurate in Figure 3, is estimated to have an acceptable accuracy of 6.61% (Tab. 3), due to the good

6.2928%	4.5463 %	7.8800%	2.7298%	10.5378%
9.7442%	6.8194%	3.1333%	7.7651%	9.4791%
6.6065%	2.8239%	6.6120%	15.2119%	4.0497%
9.8786%	1.7947%	2.7175%	4.5316%	15.2800%

Table 3. The relative errors for the test cases based on the predictions from FNO.

data distribution match (Fig. 4).

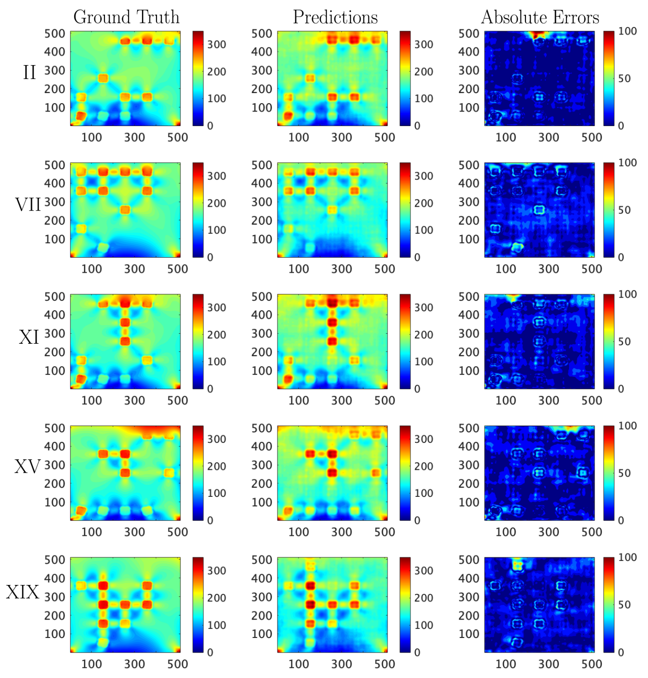


Fig. 5. Some preliminary prediction results for the von Mises' stress distribution. The first column is the ground truth, i.e., the stress field calculated by FEM. The second column is the inference results by FNO. The third column is the absolute errors. The axial marks are the pixel numbers.

Eventually, we generate Figure 5 to visualize the "good" prediction results, i.e., the data fit agrees well with the linear correlation, and the data distribution matches well. The relative errors are small, by comparing them with the testing data and showing the absolute errors, i.e., $|\mathscr{E}| = |u_{pred} - u_{test}|$. It is visually verified that the FNO inferred stress fields from small data training are pretty accurate. Judging from $|\mathscr{E}|$ visually we can deduce there are two main sources of the prediction errors: (1) The non-periodic boundary conditions, as one observes high error distribution along the upper boundary; (2) Geometry induced variability, as one observes error distribution along the carbon fiber areas. Note that these phenomena are also observed for other deep learning-based stress predictions, which can be viewed as a drawback for data-driven approximations.

5 Conclusion and Outlook

In this report, we present efforts utilizing Fourier neural operator to infer the von Mises' stress data based on given material microstructure representation from small data. The training data is generated by solving for the stress field from linear elastic solid mechanics given a non-periodic compression loading. Based on the prediction results from FNO, we have the following main conclusions:

- Based on training from the limited dataset, we generate good predictive accuracy on the test dataset of the von Mises' stress distribution from FNO.
- Although a scattered data plot offers a good sense of how the inference fit the benchmarks, it may not accurately describe the detailed prediction accuracy.
- The probability distribution offers more insight into the prediction accuracy if taking the relative errors as the benchmark criterion.
- The two main error sources from FNO on mapping stress fields from material geometries are proposed to be the non-periodic boundary conditions and the variations within the geometries.

Based on our present efforts, there are certain steps that can push this work forward. We make the following plan for future work hoping to compile a full manuscript:

- Incorporate more complicated physical models for computing the corresponding physical properties, i.e., multiphysics models, with more complex constitutive models to generate the corresponding stress fields.
- Compare DeepONet and FNO on the training process to offer better insights into the two models, as they are treated as canonical models in operator regressions.
- Study how pixel resolutions affect prediction accuracy.
- Investigate how network volume/parameter size affects prediction accuracy.
- Employ the trained models with optimization frameworks for inverse design.

Acknowledgements

The author appreciates the valuable discussion with Professor Christopher Earls.

References

[1] Wan, J., Jiang, J.-W., and Park, H. S., 2020. "Machine learning-based design of porous graphene with low thermal conductivity". *Carbon,* **157**, Feb., pp. 262–269.

[2] Zhai, H., and Yeo, J., 2022. Computational design of antimicrobial active surfaces via automated bayesian optimization.

[3] Gu, G. X., Chen, C.-T., and Buehler, M. J., 2018. "De novo composite design based on machine learning algorithm". *Extreme Mechanics Letters,* **18**, Jan., pp. 19–28.

[4] Behler, J., and Parrinello, M., 2007. "Generalized neural-network representation of high-dimensional potential-energy surfaces". *Physical Review Letters,* **98**(14), Apr.

[5] Zhang, L., Han, J., Wang, H., Car, R., and E, W., 2018. "Deep potential molecular dynamics: A scalable model with the accuracy of quantum mechanics". *Phys. Rev. Lett.,* **120**, Apr, p. 143001.

[6] Zhang, L., Han, J., Wang, H., Car, R., and E, W., 2018. "DeePCG: Constructing coarse-grained models via deep neural networks". *The Journal of Chemical Physics,* **149**(3), July, p. 034101.

[7] Zhai, H., Zhou, Q., and Hu, G., 2022. "Predicting micro-bubble dynamics with semi-physics-informed deep learning". *AIP Advances,* **12**(3), Mar, p. 035153.

[8] Brunton, S. L., Proctor, J. L., and Kutz, J. N., 2016. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". *Proceedings of the National Academy of Sciences,* **113**(15), Mar., pp. 3932–3937.

[9] Rudy, S. H., Brunton, S. L., Proctor, J. L., and Kutz, J. N., 2017. "Data-driven discovery of partial differential equations". *Science Advances,* **3**(4), Apr.

[10] Lusch, B., Kutz, J. N., and Brunton, S. L., 2018. "Deep learning for universal linear embeddings of nonlinear dynamics". *Nature Communications,* **9**(1), Nov.

[11] Raissi, M., Perdikaris, P., and Karniadakis, G., 2019. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". *Journal of Computational Physics,* **378**, Feb., pp. 686–707.

[12] Raissi, M., Yazdani, A., and Karniadakis, G. E., 2020. "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations". *Science,* **367**(6481), Feb., pp. 1026–1030.

[13] Yang, L., Zhang, D., and Karniadakis, G. E., 2018. Physics-informed generative adversarial networks for stochastic differential equations.

[14] Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E., 2021. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators". *Nature Machine Intelligence,* **3**(3), Mar., pp. 218–229.

[15] Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G. E., 2022. "A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data". *Computer Methods in Applied Mechanics and Engineering,* **393**, Apr., p. 114778.

[16] Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., and Anandkumar, A., 2021. Neural operator: Learning maps between function spaces.

[17] Li, Z., Kovachki, N. B., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. M., and Anandkumar, A., 2020. "Fourier neural operator for parametric partial differential equations". *CoRR,* **abs/2010.08895**.