MENG 21200 - Principles of Engineering Analysis II *Molecular Engineering*
Winter 2023 *University of Chicago*

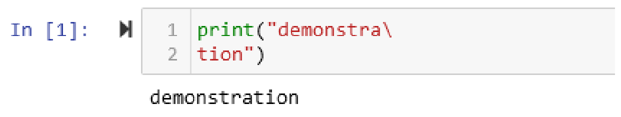
Problem Set 3

ASSIGNED: January 20, 2023
DUE: **Friday, January 27, 2023** at the beginning of class.

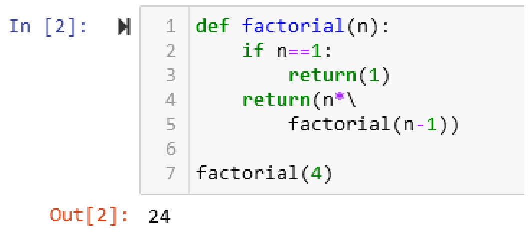
Problem set guidelines:

1. Each solution must be your own work.
2. All problems that involve python must be completed in Jupyter Notebooks**.** Some problems may not require python and may be better completed with pen and paper.
3. Highlight your final answer when providing numerical results. Provide plots, graphs, and tables of your results when appropriate.
4. **Submission Instructions:** This quarter we will use Gradescope to collect your submissions and grade them. We will only accept a single PDF file with your compiled solutions. Please follow the instructions below to obtain a PDF file from a Jupyter Notebook:
   i. Make sure all your code runs without error, and all figures (if any) show up where intended. We will not be running your code, therefore it is essential that your solutions output and highlight your results. Please be mindful of your line length so that it fits into the PDF layout and your results are clearly shown.
   ii. Select `File->Download as->HTML (.html)`. This will download an HTML version `your_homework.html` of your notebook to your computer, typically in your `Download` folder.
   iii. Open `your_homework.html` in your web browser (just double-click on it). ***Use the `File->Print` command of your browser to produce a PDF file.*** *(Do not produce the PDF in other ways, such as "Export to PDF" or other. Alternative methods will usually produce poor page margins, format headers differently, fail to incorporate figures, and so forth.)*
   iv. **Submit your PDF file** `your_homework.pdf` to Gradescope. Do *not* submit your HTML file.
5. Problem sets are due at the beginning of the class period on the due date listed. Late problem sets will not be accepted.

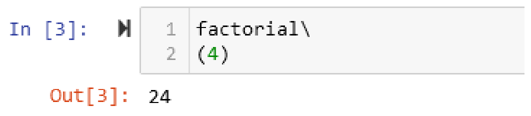
Note that you can wrap lines of python code using the "\" operator to ensure that all your code is visible within the width of the page. See the following examples:



```
In [1]:  print("demonstra\
         tion")

demonstration
```



```
In [2]:  def factorial(n):
             if n==1:
                 return(1)
             return(n*\
                 factorial(n-1))

         factorial(4)

Out[2]:  24
```



```
In [3]:  factorial\
         (4)

Out[3]:  24
```

Problem 1

Adapted from C&C problem 18.12

Employ inverse interpolation to determine the value of x that corresponds to $f(x) = 0.85$ for the following tabulated data:

x	0	1	2	3	4	5
$f(x)$	0	0.5	0.8	0.9	0.941176	0.961538

Note that the values in the table were generated with the function $f(x) = x^2/(1+x^2)$.

(a) Determine the correct value analytically.

(b) Use cubic interpolation of x versus $f(x)$. (Note the switching of axes here)

(c) Use inverse interpolation with quadratic interpolation and the quadratic formula.

(d) Use inverse interpolation with cubic interpolation and the bisection method. For parts (b) through (d) compute the true percent relative error.

(e) Plot on the same axes the data set as discrete points (in black), as well the analytical function (in red) and the interpolating functions from parts (b) through (d) in distinct colors. Be sure to add a legend to your plot to clearly identify the different functions that are included.

Problem 2

In lecture, we introduced the idea of using splines to fit a polynomial to a subset of the datapoints only. In this problem, you will need to write a python function that allows us to calculate the coefficients of quadratic splines given *n+1* data points. In this case, we have *n* intervals which means *n* 2nd-order polynomials resulting in *3n* parameters that must be determined.

The Chapra & Canale textbook provides a detailed explanation of the equations that you need to calculate these parameters (p. 512-515). In brief, the goal of quadratic splines is to interpolate between data points while fitting all the data perfectly using a subset of the points small just enough to make sure that the first derivative at the knots is continuous. The requirements are then:

- The function values of adjacent polynomials must be equal at the interior knots. Since we start with *n+1* points, we have *n-1* interior knots, so this gives us *2(n-1)* equations.
- The first and last functions must pass through the end points. This adds 2 more equations.
- The first derivatives at the interior knots must be equal which yields *n-1* equations for the *n-1* interior knots.
- And, because we need one more equation to solve for *3n* parameters, assume that the second derivative is zero at the first point.

These conditions will result in a system of *3n* equations to solve. To solve this system of equations, we recommend calling the `np.linalg.solve()` function from the `numpy` library after setting up your matrices in your function. We also recommend setting up the equations by hand first to see how they can be written in terms of matrices while keeping track of the indices!

(a) Write a function to calculate the coefficients of quadratic splines given 5 points (so for *n*=4 only, not the general case for an arbitrary *n* value).
(b) Call your function to determine the quadratic splines given 5 points: $x = [1, 2, 3, 4, 5]$ and $y = sin(x)$.
(c) Once you found the parameters of the *n* equations, you can use them to plot the parabolas in each of the *n* intervals. Plot on the same figure the discrete points and the true $y = sin(x)$ function to compare your result.
(d) Compare your fit to the one obtained by a cubic spline interpolation using the `scipy.interpolate.CubicSpline()` function and with the interpolating polynomial (Lagrange form) by plotting them along with the quadratic splines and the discrete data on a single set of axes. What do you observe about how these interpolations compare?