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

Problem Set 1

ASSIGNED: January 6, 2023
DUE: **Friday, January 13, 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.

Problem 1
Code the bisection method for finding roots in a well-defined interval as introduced in Lecture 2. Be sure to comment and document your code thoroughly, as well as demonstrate their operation on the example function.

(a) Write a function "`def f(x)`" that defines the nonlinear function $f(x) = xe^x - \sin(8x) - 0.5$.

(b) Plot the function $f(x)$ and determine its root(s) in the interval $[0, 2]$.

(c) Write a function "`def bisection(f, x1, x2, tol)`" that describes the main code of the bisection method, finding the root of $f(x)$ on the interval $[x_1, x_2]$ with a specified tolerance tol.

Approach: Implement a `while` loop based on the following pseudo-code:

```
while (absolute error > tol)
    x3 = (x1 + x2)/2
    if f(x1)*f(x3) < 0:
    x2 ← x3
    f(x2) ← f(x3)
    if not:
    x1 ← x3
    f(x1) ← f(x3)
```

(d) Employ your function bisection that calls the function $f(x)$ you implemented to find the root on the interval $[0, 2]$ with a tolerance of 0.0001.

(e) Now choose a few different intervals that begins on the negative values of x and find its roots (e.g., try $[-10, 0]$, $[-10, -8]$, and $[-8.8, -8.6]$). Did you get the same results? Why or why not?

Problem 2

Adapt your `bisection` function so that when it provides the root for function $f(x)$ it also provides a single plot of the (i) true percent relative error and (ii) approximate percent relative error versus iteration number. Make sure to plot the errors versus iteration number on the appropriate semilog axes.

Provide the plot of errors versus iteration number that results for the root found for $f(x) = xe^x - \sin(8x) - 0.5$ on the interval $[0, 2]$ with a tolerance of 1×10^{-5}.

Problem 3

Modify your `bisection` function to account for the deficiencies identified in Problem 1. In other words, how can you modify the code to find more than one root? One approach is to divide the interval into subintervals and search for roots in each subinterval. Therefore, the smaller the subintervals, the more accurate your root finding will be.

(a) Write a function "`def BisectMany(f, x1, x2, tol, subintervals)`" that is able to find multiple roots within $[x_1, x_2]$, where `subinterval` defines the number of the subintervals in which you will search for roots. Call your previously defined `bisection` function within this function.

(b) Employ your `BisectMany` function for the interval $[-10,2]$ with the same tolerance as before, and subinterval sizes of 2, 1, 0.1, and 0.01. What do you find?

Problem 4

Carbon-dating was developed by Prof. Willard Libby of the University of Chicago in 1949. The method uses the radioactive decay of the carbon-14 (^{14}C) isotope to determine the age of carbon-containing materials. During their lives, plants and animals exchange ^{14}C with the environment via respiration or diet such that they have the same proportion of ^{14}C as the atmosphere or oceans for terrestrial or marine life, respectively. Once an organism dies, it stops being in equilibrium with the environment and the ^{14}C within the biological material begins to decay such that the ratio of ^{14}C to ^{12}C in the remains decreases. By measuring the ratio of ^{14}C to ^{12}C, it is therefore possible to estimate the age of archaeological finds.

Originally it had been assumed that the concentration of atmospheric carbon had remained constant over time and was independent of geography. In reality this is a poor assumption, due in part to variations in the production rate of ^{14}C, changes caused by glaciation, and changes induced by human activity (*e.g.*, the burning of fossil fuels and testing of nuclear weapons).

It also has been noted that the production rate of ^{14}C has regular fluctuations with an oscillatory behavior related to the strength of the earth's magnetic field (*i.e.*, the cosmic rays that produce ^{14}C are deflected by magnetic fields). These fluctuations occur with a period of ~9,000 yrs. A second set of shorter terms oscillations in the production rate of ^{14}C arise due to sunspot activity (that reduce the cosmic ray flux) and occur with periods of 200 and 11 years. A simple model for the radiocarbon age (Age_{radio}) as a function of calendar age (Age_{cal}) may therefore be approximated as

$$\begin{aligned}(Age_{radio}) = {} & 0.92\,Age_{cal} - 240 \\ & + 250\cos\big((Age_{cal} + 700) * 2\pi/9000\big) \\ & + 50\sin(Age_{cal} * 2\pi/200) \\ & + 20\cos(Age_{cal} * 2\pi/11)\end{aligned}$$

Based on this calibration model, if you have measured an archaeological find to have a radiocarbon age of 4518 years, what is the true calendar age of the artifact? Provide your answer as a range of values.

Problem 5 – Taylor series & approximate relative error

(a) Starting with the notebook provided for Discussion 1, define a function $g(x) = \sin(x) + \cos(3x)$. Plot the function on the domain $[-1, 2]$.

(b) Define a second function, `taylor_g(x, nth_order)`, which calculates the nth-order Taylor polynomial approximation **centered at $\boldsymbol{x = 0.25}$** for $g(x)$ by adding together the requisite number of consecutive terms in the Taylor series, evaluated at the specified x value. Plot $g(x)$, along with the Taylor polynomials (centered about 0.25) of order 1, 2, 3, and 4. Plot on a domain of $[-0.5, 1]$, and include a legend which labels each Taylor polynomial by its order.

(c) Calculate the true relative error of the Taylor approximations at $x = 0.1$. **What order**

Taylor polynomial is required to achieve a true relative error below 0.1%, one tenth of one percent?

Often in numerical methods, we will not know beforehand the true value of the function that we are estimating (in other words, we will not know the true value of the function "g" at $x = 0.1$ hence the need to use the Taylor series approximation). In these cases, we must approximate the error by using our current best approximation for the true value of the function:

$$\varepsilon_a = \frac{current\ approximation - previous\ approximation}{current\ approximation} 100\%$$

(d) Here you will write an algorithm to approximate the value of $g(0.1)$ to a desired accuracy, without knowing the "true" value of $g(0.1)$. Begin by calculating the first-order Taylor polynomial approximation for $g(0.1)$. Use this as your "best current approximation" for the true value in order to calculate the relative approximate error of the second-order Taylor polynomial.
If the relative approximate error is not less than 1%, continue by calculating the third-order Taylor polynomial, and estimate the error using the second-order polynomial as the new "best approximation".
Continue iterating in this manner, using the most recent Taylor polynomial as the best approximation for the true value of the function, until you reach a relative approximate error below 1%. **What order Taylor polynomial** is required to achieve a true relative error below 1%, one percent?

Problem 6 – Taylor series & true absolute error

The Taylor series for $\mathbf{sin}(\boldsymbol{x})$ is:

$$\mathbf{sin}(\boldsymbol{x}) = \boldsymbol{x} - \frac{\boldsymbol{x}^3}{\mathbf{3}!} + \frac{\boldsymbol{x}^5}{\mathbf{5}!} - \frac{\boldsymbol{x}^7}{\mathbf{7}!} + \frac{\boldsymbol{x}^9}{\mathbf{9}!} + \cdots$$

Or in summation form,

$$\mathbf{sin}(\boldsymbol{x}) = \sum_{\boldsymbol{n}=\mathbf{0}}^{\infty} (-\mathbf{1})^{\boldsymbol{n}} \frac{\boldsymbol{x}^{\mathbf{2}\boldsymbol{n}+\mathbf{1}}}{(\mathbf{2}\boldsymbol{n}+\mathbf{1})!}$$

(a) Write a function `taylor_sin(x, nterms)` which calculates the nth-order Taylor polynomial approximation for $\mathbf{sin}(\boldsymbol{x})$ by adding together `n` consecutive terms in the Taylor series, evaluated at the specified `x` value.

(b) Plot $\mathbf{sin}(\boldsymbol{x})$ on the domain $[-\boldsymbol{\pi},\ \boldsymbol{\pi}]$ in black. In the same figure the Taylor polynomial approximations of sin(x) with **n = 0, 1, 2, and 3** (*i.e. polynomial orders 1, 3, 5, 7).*

(c) Calculate the true absolute error between $\mathbf{sin}(\boldsymbol{x})$ evaluated at $\boldsymbol{\pi}$ and the nth-term

Taylor polynomial evaluated at $\boldsymbol{\pi}$, where $\boldsymbol{n}$ ranges from $[\mathbf{0},\mathbf{20}]$ ***(i.e. up to polynomial of order 41)***. Plot the true absolute error as a function of Taylor polynomial order on a semilog y plot (x-axis is linear, y-axis is logarithmic). Is there a limit to how precisely you can approximate $\mathbf{sin}(\boldsymbol{x})$ at $\boldsymbol{\pi}$? If so, why does this limit exist?

(d) Now calculate the true absolute error between $\mathbf{sin}(\boldsymbol{x})$ evaluated at $\mathbf{2}\boldsymbol{\pi}$ and the nth-order Taylor polynomial evaluated at $\mathbf{2}\boldsymbol{\pi}$, where $\boldsymbol{n}$ ranges from $[\mathbf{0},\mathbf{20}]$ $(\boldsymbol{i.e. up\ to\ polynomial\ of\ order}\ \mathbf{41})$. Plot the true absolute errors as a function of the Taylor polynomial order on a semilog y plot (x-axis is linear, y-axis is logarithmic), alongside the error from part (c) calculated at $\boldsymbol{\pi}$. Are the Taylor polynomials more accurate at $\boldsymbol{\pi}$ or $\mathbf{2}\boldsymbol{\pi}$, and why is that the case?