Please provide complete and well-written solutions to the following exercises.

Due September 12, 4PM PST, to be uploaded as a single PDF document to brightspace. It is acceptable to instead upload a Jupyter Notebook, assuming you write in complete sentences where appropriate, and format your responses to be easily readable (i.e. if you only submit one big block of code with nothing written about what you did, then many points will be deducted from your score).

Homework 2

Exercise 1. Suppose we want to solve the linear system of equations

$$17x_1 + 5x_2 = 22,$$
$$1.7x_1 + .5x_2 = 2.2.$$

Note that $(x_1, x_2) = (1, 1)$ is a solution to this system of equations.

Python can numerically solve this system with the following program

```
A = np.array([ [ 17, 5], [1.7, .5] ])
b = np.array([22, 2.2])
x = np.linalg.solve(A, b)
```

- What is the solution x that is output from the program?
- Is the output of the program an actual solution of the original system of equations?
- What is the determinant of A? What does Python output from the command `np.linalg.det(A)`?

Warning: for a 2×2 matrix A and a scalar $t > 0$, we have $\det(tA) = t^2 \det(A)$. So, the value of a determinant does not necessarily say anything about how well we can solve a linear system of equations of the form $Ax = b$.

Exercise 2. The sin function, like other special functions such as $\cos, \exp, \log$, etc., cannot be computed exactly on a computer. A common way to compute these special functions is via power series. Recall that sin has the following power series that is absolutely convergent for all $x \in \mathbf{R}$:

$$\sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}.$$

With this power series in mind, run the following program when $x = \pi/2, 11\pi/2, 21\pi/2$ and $31\pi/2$. (Before you run the program, set x to a specific value.)

```
s = 0
t = x
```

```
n = 1
while s + t != s:
    s = s + t
    t = -(x**2) * t / ((n + 1) * (n + 2))
    n = n + 2
print(s)
```

When the program terminates, the value of s is the computed value of $\sin(x)$. For each value of x stated above, answer the following:

- What is the absolute error of the computation of $\sin(x)$?
- How many terms of the power series were used in the computation of $\sin(x)$?
- What is the largest term in the power series expansion of $\sin(x)$? (Hint: consider using the `numpy.max` command)

Exercise 3. This exercise examines an unstable recurrence computation.

Consider the following recursion with $x_0 := 1$ and $x_1 := 1/3$.

$$x_{n+1} = \frac{13}{3}x_n - \frac{4}{3}x_{n-1}, \qquad \forall\, n \geq 1.$$

- Verify that the recurrence is solved by $x_n := (1/3)^n$ for all $n \geq 0$.
- Using Python, solve for x_{40}. For example, use

  ```
  x = np.array([1, 1/3])
  for i in range(2,41):
      x = np.append(x, (13/3)*x[i-1]  - (4/3)*x[i-2])
  print(x[40])
  ```

  Is the answer what you expected to get? (Hint: examine a logarithmically scaled plot in the y-axis, using `matplotlib.pyplot.semilogy`.)
- With a different initial condition, the above recurrence can have other solutions. To find them, rewrite the recurrence as

  $$\begin{pmatrix} 13/3 & -4/3 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_n \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} x_{n+1} \\ x_n \end{pmatrix}, \qquad \forall\, n \geq 1.$$

  Then note that the eigenvalues of the matrix $A := \begin{pmatrix} 13/3 & -4/3 \\ 1 & 0 \end{pmatrix}$ are $1/3$ and 4, so iterating the recurrence shows that

  $$\begin{pmatrix} 13/3 & -4/3 \\ 1 & 0 \end{pmatrix}^n \begin{pmatrix} x_1 \\ x_0 \end{pmatrix} = \begin{pmatrix} x_{n+1} \\ x_n \end{pmatrix}, \qquad \forall\, n \geq 0.$$

  Since A has two distinct eigenvalues, it is diagonalizable, so if $\begin{pmatrix} x_2 \\ x_1 \end{pmatrix}$ is written as a linear combination $a_1 v_1 + a_2 v_2$ of the corresponding eigenvectors $v_1, v_2 \in \mathbf{R}^2$ of A, the recurrence becomes

  $$a_1(1/3)^{n0} v_1 + a_2 4^n v_2 = \begin{pmatrix} x_{n+1} \\ x_n \end{pmatrix}, \qquad \forall\, n \geq 0.$$

- Show that in the case $x_1 = 1$ and $x_2 = 1/3$, we have $a_2 = 0$. However, small numerical errors that occur in the computation of the recurrence correspond to a_2 being computed to be nonzero. Explain how this relates to the logarithmic plot you examined above.

Exercise 4 (Numerical Integration)**.** Consider the function

$$f(t) := t^3 + 1.$$

In this case, we can easily compute

$$\int_0^1 f(t)dt = \frac{5}{4}.$$

Sometimes, especially in computer graphics applications, integrals are too complicated to compute directly, so we instead use randomness to estimate the integral. That is, we pick n random points in $[0, 1]$, and average the values of f at these points, as in the following program.

```
n = 10**5
t = np.random.rand(n)
f = t**3 +1
np.mean(f)
```

Using this program with $n = 10^5, 10^6, 10^7$ and 10^8, report the estimated values for the integral of f, along with their relative errors.

Now, compute the exact value of $\int_3^5 \log x dx$, and modify the above program to give estimates for the value of this integral and report relative errors, using a number of points n where $n = 10^5, 10^6, 10^7$ and 10^8.

Exercise 5. In this exercise, we will compare the run time of built-in vectorized functions versus a naive for loop

- Compare the time it takes to compute a dot product using numpy's `np.dot` function, versus using a for loop. More specifically, use `x=np.random.randn(10**k)` and `y=np.random.randn(10**k)` for $k = 3, 4, 5, 6, 7$, and compute the dot product of x and y
- Compare the time it takes to compute a matrix product using numpy's `np.dot` function, versus using a for loop. More specifically, use `A=np.random.randn(10**k, 10**k)` and `B=np.random.rand(10**k, 10**k)` for $k = 1, 2, 3, 4$, and compute the matrix product AB using `A @ B`, versus a for loop.

Exercise 6. The links below contain `.csv` files, each with 1000 (pseudo) random samples from a Gaussian distribution with variance one and unknown mean $\mu \in \mathbf{R}$

gaussian data

gaussian data v2

Recall that a basic question in parametric statistics is to estimate the unknown mean μ. From statistics class, we know that a good estimator for the mean will be the sample mean

(since e.g. it is the MLE for the mean). Using the following commands, we can import the first `.csv` file into a Numpy array, and then take the sample mean to estimate μ.

```
x = np.genfromtxt("gaussian_data.csv", delimiter=",")
np.mean(x)
```

The output is $-.00968$. I used $\mu = 0$ to generate these samples, so the mean estimate is pretty close to reality. However, the second file is exactly the same as the first, but I intentionally created two outliers to skew the final result. The output of the above program for the second file is 11371.66, which is quite far from the true value $\mu = 0$. With this example in mind, we ask: what is a good estimate of the unknown mean μ that is robust to noise (or robust to outliers)? There are many possible good answers, and one such answer is the median. The following program

```
x = np.genfromtxt("gaussian_datav2.csv", delimiter=",")
np.median(x)
```

has output $-.03$.

Can you think of a better way to remove the outliers and estimate the unknown mean? (This question is intentionally open ended.)