CS 111 (S22): Homework 5

Due Monday, May. 9th by 11:59 PM

NAME and PERM ID No.: Gaucho Olé, 1234567 (replace with yours)

UCSB EMAIL: GauchoOle@ucsb.edu (replace with yours)

Homework Buddy: (remove entire line if not applicable)

Additional Instructions: Please enter the LaTeX command *newpage* at the end of each of your answers in your LaTeX source code. This will put each answer on at least one page and make it easier to grade on Gradescope.

1. The moral of this problem is twofold: First, linear least squares can be used to fit data with polynomials, not just with straight lines or planes. (As we've seen, it can be used to fit data with lots of different kinds of models.) Second, fitting data with a very high-degree polynomial is not always a good idea.

   The data for this problem is the population of the United States at each 10-year census from 1900 to 2020. Here is the data.

```
date = np.array(range(1900,2030,10)) - 1900
population = 1000 * np.array([ 75995,  91972, 105711, 123203, 131669,
                              150697, 179323, 203212, 226505, 249633,
                              281422, 308746, 329168])
for (d,p) in zip(date, population):
    print(d, ':', p)
```

   Notice that we are writing the date as "years since 1900." How come? It turns out to make the fitting computation more stable.

   (a) Express the problem of fitting a straight line of the form

$$p = x_0 + x_1 d$$

   to the data as a linear least squares problem

$$Ax \approx b$$

   where $x = (x_0, x_1)^T$ is the vector of coefficients of the line. (The answer to this question consists just of the 13-by-2 matrix A and the vector b.)

   (b) Solve the least squares problem in (1a) for x using the routine `npla.lstsq()`. Show the Python code and also a plot that shows the original population data as (little) circles, and the least squares fit as a line.

(c) Use x to compute the US population in the year 2020, as predicted by the straight-line fit. What is that number and how far is it from the "observed" (i.e. given) number for 2020? Also, use x to compute the US population in the year 2030.

(d) Express the problem of fitting a quadratic polynomial of the form

$$p = x_0 + x_1 d + x_2 d^2$$

to the data as a linear least squares problem

$$Ax \approx b$$

where $x = (x_0, x_1, x_2)^T$ is the vector of coefficients of the polynomial. This time the matrix A will be 13-by-3. Again, solve the least squares problem, make a plot of the resulting parabola, and use the new x to predict the 2020 population (and tell us how far it is from the observed number in the data set). What does the model predict the 2030 population to be? Finally, report what the relative residual norm of this model vs. observed data is.

(e) Revise the code you used in (1d) to take the polynomial degree as a parameter. Repeat the fitting, plotting, and prediction for polynomials of degrees 3, 4, 5, 6, 7, and 8. Turn in your plots and the values of your predictions for this part, but not your Python code. Challenge: arrange your plots on this one page (maybe 3 of them in 2 lines, or 2 of them in 3 lines? Either case, the plots don't have to be big ones) With each plot, show what the relative residual norm is. What do you notice about the last few predictions?

2. These questions are all about IEEE standard 64-bit floating-point arithmetic, which is behind both numpy's `float64` type and C's `double` type. You can use `fprint()` shown in Lecture 9 to see the actual bits that represent any number. Recall that one hexadecimal digit stands for 4 bits.

   (a) *Machine epsilon*, or just ϵ for short, is defined as the largest floating-point number x such that $x + 1 = 1$ in floating-point arithmetic. The experiment we did in class showed that ϵ is approximately 10^{-16}, which is why we say that IEEE floating-point can be accurate to about 16 decimal digits.

   What is the exact value of ϵ? Your answer should be an exact arithmetic expression, not a decimal expansion. You should also explain (briefly in words or show supporting code) how you got this answer. What is the 16-digit hex representation of ϵ in the IEEE standard? How close is ϵ to 10^{-16}?

   (b) What is the 16-digit hex representation of $1/\epsilon$? What is its approximate value in base 10?

   (c) What is the largest non-infinite positive number that can be represented exactly in IEEE floating-point? Give your answer in 2 ways: As the 16-hex-digit IEEE representation, and as an approximate value in base 10. (Hint: Consider the largest possible value of the 11-bit exponent field in the IEEE standard, but remember that value is reserved to represent "infinity".)

(d) A common mistake some people make (but not you!) is to think that ϵ is the smallest positive (non-zero) floating-point number. It's not, by a long shot. Consider for example $x = \epsilon^{10}$. What is the approximate value of x in base 10? Is x an exact floating-point number? If so, give its IEEE 16-hex-digit representation; if not, give an IEEE 16-hex-digit representation of a floating-point number as close to x as you can.

(e) How many different floating-point numbers x are there with $1 < x < 2$?

3. Consider each of the following Python loops. For each loop, answer: How many iterations does it do before halting? What are the last two values of x it prints (both as decimals printed by Python, and as IEEE standard 16-hex-digit representations)? Explain in one sentence what property of the floating-point system the loop's behavior demonstrates.

(a)

```
x = 1.0
while 1.0 + x > 1.0:
    x = x / 2.0
    print(x)
```

(b)

```
x = 1.0
while x + x > x:
    x = 2.0 * x
    print(x)
```

(c)

```
x = 1.0
while x + x > x:
    x = x / 2.0
    print(x)
```