PHYS 331 – Introduction to Numerical Techniques in Physics

Homework 7: Matrix Conditioning, Gaussian Elimination and LU Decomposition

Due Friday, March 9, 2018, at 11:59pm.

Problem 1 – Real-World Exploration of Matrix Conditioning (25 points)

You are in PHYS 118 and arguing with your studio team about how best to take measurements of a system under constant acceleration. You plan to collect 3 measurements of position and time, x_i and t_i, i=1…3, where:

$$x(t) = x_0 + v_0 t + \frac{1}{2}at^2 \tag{1}$$

You plan to use these 3 measurements to determine the unknowns: x_0, v_0, and a, with a reasonable degree of accuracy. The controversy is at what times t should you take measurements to get the best accuracy? Some students think you should take measurements at even intervals **in time**, and other students think the measurements should be taken at even intervals **in position**. Using your advanced PHYS 331 knowledge, you will predict what set of measurements produce the most stable & accurate estimation of the parameters.

(a) First, write this system as a matrix equation $\mathbf{Ax} = \mathbf{b}$ in terms of the measurements x_i and t_i, i=1…3 and unknown parameters x_0, v_0, and a (this should be shown in your PDF). Is the system linear or nonlinear?

(b) Now, remembering what you learned in PHYS 331, you realize that this system will be ill-conditioned if the absolute value of the determinant of $\mathbf{A}$ is much smaller than the norm of $\mathbf{A}$. As such, the ratio, let's just call it R for convenience:

$$R = \frac{|\det(A)|}{\|A\|} \tag{2}$$

is a measure of "better" conditioning, *i.e.*, larger values should produce more accurate and stable results. Write a script, `checkConditioning(tvec)` that accepts a numpy array `tvec` that is (t_1,t_2,t_3), the time values at which you plan to take your measurements, computes the matrix you developed in part (a), and returns the ratio of Eqn. (2). Use the definition for the Euclidian norm given in the textbook. You may also use the built-in numpy function for computing the determinant, `np.linalg.det`.

(c) Using your `checkConditioning` function, we will explore 2 scenarios based on the students' different suggestions for how to collect the data. First let's explore what happens when the data is evenly spaced **in time**, *i.e.*, t_1=0, t_2 = Δt, and t_3 = 2Δt (where Δt is the time interval). Let's determine whether we want to use a short, medium, or long time interval. In the experiment, the maximum amount of time available to collect data before the mass moves out of range is 10 seconds, so the maximum Δt is 5s. Write some code as needed to compute the conditioning R as a function of Δt over the range (0.1s, 5s), and make a plot of R vs. Δt. What value of Δt suggests the best conditioning? Explain whether this is what you expected, and why you think the result is what you found.

(d) For the second scenario, we will assume that you will collect data at t_1=0 and t_3 = 10s, but that t_2 might be variable anywhere in between t_1 and t_3. (Although this isn't the same as saying the measurements were evenly spaced in position, it explores possibilities that include such a scenario). Using `checkConditioning` and any additional code needed, make a plot of R as a function of t_2 over the entire possible range of t_2. From your calculations, determine what value of t_2 suggests the best conditioning. Is this better than what you could obtain using an evenly spaced interval in part (c)?

(e) While all of this sounds neat in theory, let's verify our results in practice, using the upper-lower bound method** of error estimation you learned in PHYS 118. Let's compare two measurement strategies and their resulting x measurements:

Strategy 1: (t_1,t_2,t_3) = (0s, 5s, 10s); (x_1,x_2,x_3) = (0.30m, 4.425m, 14.30m)

Strategy 2: (t_1,t_2,t_3) = (0s, 1s, 10s) ; (x_1,x_2,x_3) = (0.30m, 0.665m, 14.30m)

Before we do any error estimation, first, solve the system to determine the best-fit parameters (x_0, v_0, a) for both strategies. In this problem you are allowed to use `np.linalg.solve`, and you can make a new function very similar to `checkConditioning` that returns the matrix $\mathbf{A}$ given the time values (t_1,t_2,t_3). Output the best-fit parameters to the screen for each scenario. Are they the same?

(f) Importantly, your measurement accuracy in position is $\Delta x = 0.005$m. So, let's estimate our worst-case scenario as the parameters obtained for a lower bound: (x_1, x_2 - Δx, x_3 - Δx) and an upper bound: (x_1, $x_2 + \Delta x$, $x_3 + \Delta x$). Note that you will NOT change x_1 in this scenario – it's a long story but the reason has to do with the fact that changes to x_0 dominate if you shift all values of x up simultaneously. In your PDF file, make a table of the values of (v_0, a) obtained for the lower bound, and the upper bound, for each strategy, and compute the difference (for example, $v_{0\text{upper}} - v_{0\text{lower}}$) as an estimate in the error of the parameters. Your table should contain a total of 12 numbers. (Note that you should find that x_0 remains unchanged, so there is no need to tabulate it).

Looking at your table, which strategy offers better accuracy, and was this what you predicted based on your findings in the earlier part of this problem? *You do not need to submit any Python for this part of the problem, although you will want to use your earlier code to do these computations.*

**There are better methods than upper-lower bound for estimating parameter error based on Jacobians.

Extra Credit (up to 5 points). As discussed in class, for a linear system of equations $\mathbf{Ax}_0 = \mathbf{b}$ with true solution $\mathbf{x}_0$, given an inexact solution $\mathbf{x}$ such that it has a residual defined by $\mathbf{r} = \mathbf{Ax} - \mathbf{b}$, we can put bounds on the error, $\mathbf{e} = \mathbf{x} - \mathbf{x}_0$ according to:

$$\frac{1}{C_A}\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \le \frac{\|\mathbf{e}\|}{\|\mathbf{x}_0\|} \le C_A \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}, \qquad (3)$$

where $\dfrac{\|\mathbf{e}\|}{\|\mathbf{x}_0\|}$ can be thought of as a fractional error in the estimate of $\mathbf{x}_0$, and C_A is the condition number. We proved the left-hand inequality using properties of norms in class. For extra credit, use properties of norms to prove the right-hand inequality. The amount of points awarded will be based on the thoroughness of your proof.

Problem 2 – Gaussian Elimination (10 points). Instead of writing new code (since you all did very well in writing `TriSolve` in the last homework set), this problem focuses on testing your understanding of the Gaussian elimination method. Upload the file `HW7p2template.py` which contains a function `GaussElimin`. **Do not modify this function**; show all your work in the lines below it. In this problem, you may use built-in matrix functions (such as `np.dot`) to answer the questions.

(a) In your own words, explain what the elimination phase of the Gauss elimination does, and why.

(b) Test the code with the following example $\mathbf{A_1 x_1}$=$\mathbf{b_1}$ below. The solution $\mathbf{x_1}$ is provided so you can compare. Show that the output **A** and **b** matrices have the same solution as the input matrices. Explain how you are showing this using comments in your code. Two warnings: 1) Pay attention to column versus row vectors. 2) Pay attention to variable **type** in this problem; the function is perfectly happy to operate on variables of whatever type it is passed.

$$\mathbf{A}_1 = \begin{pmatrix} 4 & -2 & 1 \\ -3 & -1 & 4 \\ 1 & -1 & 3 \end{pmatrix}, \quad \mathbf{b}_1 = \begin{pmatrix} 15 \\ 8 \\ 13 \end{pmatrix}, \quad \mathbf{x}_1 = \begin{pmatrix} 2 \\ -2 \\ 3 \end{pmatrix} \qquad (4)$$

(c) Test the code with the below example $\mathbf{A_2 x_2}$=$\mathbf{b_2}$. The solution $\mathbf{x_2}$ is provided so you can compare. What happens when you try to input these matrices as written, and why? Rewrite the matrices as needed so that they can be passed to `GaussElimin` without errors. (Explain what you did in your PDF file). Show that the output **A** and **b** matrices have the same solution as the input matrices.

$$\mathbf{A}_2 = \begin{pmatrix} 0 & 2 & 0 & 1 \\ 2 & 2 & 3 & 2 \\ 4 & -3 & 0 & 1 \\ 6 & 1 & -6 & -5 \end{pmatrix}, \quad \mathbf{b}_2 = \begin{pmatrix} 0 \\ -2 \\ -7 \\ 6 \end{pmatrix}, \quad \mathbf{x}_2 = \begin{pmatrix} -0.5 \\ 1 \\ 1/3 \\ -2 \end{pmatrix} \qquad (5)$$

Problem 3 – LU Decomposition (15 points). As discussed in class, if we need to solve a linear equation system with many different right-hand sides **b**, the *LU*-decomposition is much cheaper (less computationally expensive) than full Gaussian elimination, because it only requires a forward- and back-substitution for each **b**, once the matrix **A** has been decomposed. With the Gaussian elimination function already provided above, the *LU* decomposition can be retrieved with simple modifications. While I think there is value in learning to read and modify existing code, you may, if you wish, completely rewrite the function in your own style.

(a) Make a copy of the `GaussElimin` function from Problem 2 into a new .py file and rename the function `LUdecomp`. Modify the function so that it now **only** inputs an n × n matrix, and only outputs its decomposition matrices `L` and `U` that are also n × n. Within this function you are **not** allowed to use any other built-in functions that perform operations on matrices; you may only perform arithmetic operations on elements of the matrices themselves.

(b) Show that the outputs from `LUdecomp` satisfy **A** = **LU** when inputting the arrays $\mathbf{A_1}$ and the modified $\mathbf{A_2}$ from Problem 2 (you may use `np.dot` in this part of the problem). Are there any differences between the input **A** and the dot product of **L** and **U**? If there are differences, what do you think caused them?