PHYS 331 – Introduction to Numerical Techniques in Physics

Homework 7: Matrix Conditioning, Gaussian Elimination and Eigenvalue Problems

Due Friday, Oct. 26, 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 cart on a track under constant acceleration. You plan to collect 3 measurements of position and time, x_i and t_i, i=1…3, where:

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

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? 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:

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

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, explore what happens with different ways of measuring the cart under acceleration. Since the cart runs out of track after 10 seconds, let us assume that the first and last measurements occur at t_1=0 and t_3 = 10s, respectively. The main question, then, is at what time, t_2 the second measurement should be taken? Using `checkConditioning` and any additional code needed, make a plot of the ratio 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 to within 0.01 seconds of accuracy.

(d) 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, 1s, 10s); (x_1,x_2,x_3) = (0.30m, 0.665m, 14.30m)

Strategy 2: (t_1,t_2,t_3) = (0s, 5s, 10s) ; (x_1,x_2,x_3) = (0.30m, 4.425m, 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 **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?

(e) Now to estimate the error, you know that your measurement accuracy in position is Δx = 0.005m. Using this information, estimate your 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 + Δx, x_3 + Δx)**. 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. Now, use your code developed in the previous section to compute the resulting best-fit values of (v_0, a). Make a table in your PDF file 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 would have 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 $\mathbf{A}\mathbf{x}_0 = \mathbf{b}$ with true solution $\mathbf{x}_0$, given an inexact solution $\mathbf{x}$ such that it has a residual defined by $\mathbf{r} = \mathbf{A}\mathbf{x} - \mathbf{b}$, we can put bounds on the error, $\mathbf{e} = \mathbf{x} - \mathbf{x}_0$ according to:

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

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 $C_A = \left\|\mathbf{A}^{-1}\right\|\|\mathbf{A}\|$. For extra credit, use properties of norms to prove *either* the right or the left inequality (this is tough, so, you do not need to do both!) 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.

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

(b) Test the code with the following example $\mathbf{A_1}\,\mathbf{x_1}=\mathbf{b_1}$ 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. You may use built-in matrix functions (such as `np.dot`) in this problem. 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.

$$\mathbf{A_1} = \begin{pmatrix} 4 & -2 & 1 \\ -3 & -1 & 4 \\ 1 & -1 & 3 \end{pmatrix}, \quad \mathbf{b_1} = \begin{pmatrix} 15 \\ 8 \\ 13 \end{pmatrix}, \quad \mathbf{x_1} = \begin{pmatrix} 2 \\ -2 \\ 3 \end{pmatrix} \tag{4}$$

(c) Test the code with the below example $\mathbf{A_2}\,\mathbf{x_2}=\mathbf{b_2}$. 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.

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

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?

Problem 4 – Using an LU Decomposition (10 points). Use the results of the LU decomposition:

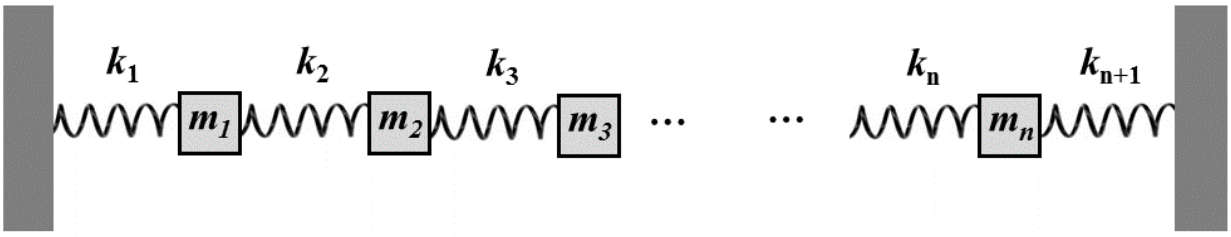
$$\mathbf{A} = \mathbf{LU} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & -2 \\ 0 & 0 & -5 \end{pmatrix}$$

To solve **Ax**=**b** for **x**, where

$$\mathbf{b} = \begin{pmatrix} 7 \\ -12 \\ -8 \end{pmatrix}$$

Do not use Python, instead work it out manually, showing all steps. You must use the proper method of solving the system from the LU decomposition (*i.e.*, the computationally efficient method) to receive full credit.

Problem 5 – Eigenvalue Problem (30 points). Consider the system of two coupled masses we discussed in lecture, and how we turned the equation of motion of each mass, using Newton's law, into a system of linear equations. Now I want you to consider the below system of n coupled masses m_i, connected by n+1 springs with spring constant k_i, with the displacements of each mass from equilibrium corresponding to x_i.



(a) Use the same method we used in class to write the equation of motion for each mass,

$$\sum F_i = m_i \ddot{x}_i ,$$

assuming that the solution for x takes the form

$$x = A\exp(i\omega t),$$

such that

$$\ddot{x} = -\omega^2 \left(A\exp(i\omega t) \right) = -\omega^2 x .$$

Write out the equations for the first 3-4 masses to see the trend. Then, write the system as an eigenvalue problem, showing at least the first 3 rows and the last row of the $n \times n$ system matrix. (This should all be in your PDF file).

(b) Write a function, `CreateSystem(kvec,mvec)`, that inputs an (n+1)×1 vector of spring constants k, and an $n \times 1$ vector of masses, m, and outputs the $n \times n$ system matrix that you solved for in part (a).

(c) Test that your function works for the scenario explored in class, for n=2, with k_1=k_2=k_3=1, and m_1=m_2=1. Do this by inputting the 2×2 array generated by `CreateSystem` into `np.linalg.eig()` to determine the eigenvalues and eigenvectors. Are the eigenvalues and eigenvectors what you expected? Explain the physical significance of the eigenvalues. Also, explain the physical significance of the two eigenvectors in terms of the relative motion of the masses (*i.e.*, whether they swing together or out-of-phase). Show your code, and also explain in your PDF file.

(d) Now use your code to determine the eigenvalues and eigenvectors for an n=3 system where all of the k's and m's equal 1 again. List each eigenvalue and its corresponding eigenvector. Draw a picture that illustrates the relative **directions and amplitudes** of the oscillatory motion for the 3 eigenmodes of the system (*i.e.*, the motion of the masses corresponding to the 3 eigenvalues/eigenvectors).

(e) Now let's consider a 1D crystal lattice with atoms of alternating masses (such as sodium chloride, NaCl). Each of the bonds between the atoms can be thought of as springs with identical spring constants. Let's consider a fairly large lattice with n=1000, and set odd $m_i = m_1 = m_3 = \ldots = 1$, and even $m_i = m_2 = m_4 = \ldots = 1.5$. Use your function to create the linear system, and solve for the eigenvalues and eigenvectors.

Make a histogram plot of the eigenvalues (make the bins narrow enough to observe relevant features). What do you notice about the eigenvalues, and what does this say about the possible vibrational frequencies of the system?

Make a second histogram plot for eigenvalues when the even masses are only 1.2 times the odd masses. What changes?

The features you are observing are related to the energy *band gap* of a condensed matter system (which occurs in semiconductors and insulators).

Extra Credit (up to 3 pts): For the even masses = 1.5 case, examine the eigenvectors corresponding to frequencies both above and below the band gap. Determine and explain what the different trends are for eigenvectors above or below the band gap – noting that we should consider the meaning of being above or below the band gap in terms of energy, which is proportional to frequency.