ENGSCI 233 - Lab Worksheet

Numerical Errors

Date: March 29, 2022

Lab Introduction

Your objective in this lab is to implement LU factorisation with partial pivoting. There are videos demonstrating the technique available from Canvas.

In order to complete the lab, you have been provided with the following files:

- *functions_errlab.py*: partially completed LU factorisation functions.
- *test_errlab.py*: partially completed tests for your functions.
- System of linear equations as data files in *system1.txt* and *system2.txt*
- *npdot_errlab.py*: file demonstrating usage of a useful command.

Any functions that you complete should include a suitable docstring.

Seeking Help

Learning a new language is challenging and Python is no exception. If you have been stuck on a particular issue for some time, then look for help. Read the official documentation for a command, which often includes examples of usage. Googling a command or error will often bring up an online forum post, where you can see how other people have approached a problem. Ask a friend how they dealt with the problem (but **don't** copy their code). Make a post on Piazza.

Task 1 - LU Factorisation without Partial Pivoting

Background

Our implementation of LU factorisation **without** partial pivoting will be divided into distinct steps, each implemented using a related function:

1. `def lu_read`

   This function is already complete and should **not** be modified.

Reads in from an external file the coefficient matrix, A, and vector of constants, $\vec{b}$, for a system of linear equations, $A\vec{x} = \vec{b}$.

2. `def lu_factor`

   This function will need to be completed in the lab.

   Factorises the matrix A into components L and U. By default, it does so **without** partial pivoting. For the purposes of being efficient in terms of memory, we can store both L and U in a single combined matrix. Consider the following LU factorisation, such that $A = LU$:

$$\begin{bmatrix} 1 & -1 & -3 \\ -2 & 4 & 9 \\ 2 & -4 & -8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -3 \\ 0 & 2 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

   L will **always** have all 1's on the diagonal, and all 0's in the upper triangular. We can therefore combine the lower triangular elements of L with the U matrix into a single combined matrix containing both L and U factors:

$$\begin{bmatrix} 1 & -1 & -3 \\ -2 & 2 & 3 \\ 2 & -1 & 1 \end{bmatrix}$$

   We can also aim for efficiency by updating/replacing the original A matrix with this combined LU matrix.

3. `def lu_forward_sub`

   This function will need to be completed in the lab.

   Performs forward substitution to solve $L\vec{y} = \vec{b}$. This will return the intermediate solution vector, $\vec{y}$.

4. `def lu_backward_sub`

   This function will need to be completed in the lab.

   Performs backward substitution to solve $U\vec{x} = \vec{y}$. This will return the solution vector, $\vec{x}$.

Note that the functions `def lu_factor` and `def lu_forward_sub` include the default arguments of `pivot=False` and `p=None`. If a function is called without the

argument, it will be assigned the default value. There are a few things to note about default arguments:

- When passing keyword arguments, the order of arguments is important.
- We should assign only one value for one parameter.
- The keyword that is passed should match the actual keyword name.
- If calling a function with non-keyword arguments, the order is important.

For Task 1, we can call these functions without these arguments i.e. the default values will be used. This ensures that we perform LU factorisation **without** partial pivoting.

Exercises

Complete the following exercises for this task:

1. Perform LU factorisation **without** partial pivoting by hand for the matrix (note: my video guide works through this exact example, both with and without pivoting, in case you get stuck):

$$A = \begin{bmatrix} 2 & 3 & -4 & 2 \\ -4 & -5 & 6 & -3 \\ 2 & 2 & 1 & 0 \\ -6 & -7 & 14 & -4 \end{bmatrix}$$

   This will give us an exact result that we can use to test our code. Think numerically about the algorithm you just used:

   - In what **order** did you work through the columns and rows?
   - How did you choose what the pivot **value** should be?
   - How did you choose what the pivot **position** should be?
   - How did you **calculate** the subtraction factor?
   - When reducing rows, what part of which row did you **add/subtract** from what part of which other row?

   It may be helpful to write out some pseudocode, based on the above questions, to plan out your LU factorisation code.

2. Complete the function `def lu_factor` to perform LU factorisation **without** partial pivoting. Run the simple test function `def test_lu_factor_nopivot`, provided in *test_errlab.py*, to check if the function output matches what you expect. Note: you will need to hard-code the exact answer to compare against.

3. Perform forward and backward substitution by hand for the system of linear equations:

$$\begin{bmatrix} 2 & 3 & -4 & 2 \\ -4 & -5 & 6 & -3 \\ 2 & 2 & 1 & 0 \\ -6 & -7 & 14 & -4 \end{bmatrix} \vec{x} = \begin{bmatrix} 4 \\ -8 \\ 9 \\ 6 \end{bmatrix}$$

   It uses the same $A = LU$ as above. Think about the algorithm you just used for the forward substitution:

   - In what **order** did you work through the rows?
   - If the **index**, i, refers to the current row, then to calculate each component of $\vec{y}$, which other components of $\vec{y}$ and $\vec{b}$ were used?

   And for the backward substitution:

   - Was the **order** that you worked through the rows different to the forward substitution?
   - Calculating each component of $\vec{x}$ was similar **except** for an additional **division** step.

4. Complete the functions `def lu_forward_sub` and `def lu_backward_sub` to solve a system $A\vec{x} = \vec{b}$ via the intermediate solution $\vec{y}$.

   If as part of your code you want to take the dot product of two vectors, or perform matrix multiplication, the `np.dot()` function may be worth reading up on. A file demonstrating its usage has been provided in *npdot_errlab.py*.

   Complete two test functions, `def test_lu_forward_sub_nopivot` and `def test_lu_backward_sub_nopivot`, in *test_errlab.py* to check if the output of the functions matches what we expect.

Task 2 - Partial Pivoting

Background

Now that we have a working implementation of LU factorisation **without** partial pivoting, we can modify the existing functions to implement LU factorisation **with** partial pivoting.

The default arguments in `def lu_factor` and `def lu_forward_sub` are related to implementing LU factorisation **with** partial pivoting:

- We can call `def lu_factor` with `pivot=True` to perform LU factorisation **with** partial pivoting, rather than **without**.
- We can call `def lu_forward_sub` with `p` equal to a 1D NumPy array to solve the system $A\vec{x} = \vec{b}$ **with** partial pivoting, rather than **without**. Note that this 1D array includes information on the row swaps required.

Exercises

Complete the following exercises for this task:

1. For the example from Task 1, perform LU factorisation, forward and backward substitution by hand using partial pivoting. Think numerically about how your algorithm differed from last time:
   - How did you **choose** the pivot value in each column?
   - What did you need to change in your L and U matrix?
   - What do the numbers in the row swap vector mean and how did you obtain these?
2. Modify the functions `def lu_factor` and `def lu_forward_sub` to implement partial pivoting. Note that we are using the form of partial pivoting discussed in my video recording (there are other forms of partial pivoting strategy that we are not considering here).

   A common mistake when searching for the pivot value is use `np.max(*)` instead of `np.max(abs(*))`. What is the difference? Before doing forward substitution, you need to apply the appropriate row swaps to $\vec{b}$, as encoded in the row swap vector, $\vec{p}$. In the function `def lu_forward_sub`, add the appropriate

commands under `if p is not None:`.

You may wish to write additional test functions for these modified functions, but this is considered **optional**.

Task 3: Numerical Errors

Background

Open and inspect the (contrived) system of linear equations, $A\vec{x} = \vec{b}$, in *system2.txt*. Use your implementation of LU factorisation to solve this matrix equation both **with** and **without** partial pivoting.

The **correct solution** is a vector of 1's:

$$\vec{x} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

However, this result can only be obtained using partial pivoting.

Exercises

Answer the following questions on floating point error, how it occurs, and how partial pivoting can address the issue.

1. What type of floating point error has been introduced?
2. How did this floating point error occur?
3. How does partial pivoting address this issue?

It may help to inspect the intermediate solution vector, $\vec{y}$. Please keep your answer as brief as possible i.e. a few sentences at most.

Submission Instructions

For this lab, you should submit the files *functions_errlab.py*, *test_errlab.py* and a PDF or Word file containing your answers to the Task 3 questions. You do **not**

need to submit any other files. Do **not** modify or vary the names of the functions file (but don't worry if Canvas appends a `-1` or `-2` to the end).

Remember, all submissions are compared against each other and those from previous years. Copying someone else's code and changing the variable names constitutes academic misconduct by *both* parties.