Project 2

Antonio Gómez Garrido and David Martínez Hernández

(Dated: September 30, 2022)

https://github.com/GatoMartinez/Pachuco-Gavotte.git

PROBLEM 1

We define $\hat{x}$ as: $\hat{x} \equiv \frac{x}{L}$. Therefore, what we've got here are two functions: $\hat{x}(x) = \frac{x}{L}$ and u which, when composed, give our original function $u(x)$:

$$u(x) = (u \circ \hat{x})(x) = u[\hat{x}(x)]$$

If we now apply the rules for the derivation of compositions of functions, we get that:

$$\frac{d^2u(x)}{dx^2} = \frac{d}{dx}\left(\frac{du[\hat{x}(x)]}{dx}\right) \tag{1}$$

$$\frac{d^2u(x)}{dx^2} = \frac{d}{dx}\left(\frac{du}{d\hat{x}} \cdot \frac{d\hat{x}}{dx}\right) \tag{2}$$

$$\frac{d^2u(x)}{dx^2} = \frac{d^2u}{d\hat{x}^2} \cdot \left(\frac{d\hat{x}}{dx}\right)^2 + \frac{du}{d\hat{x}} \cdot \frac{d^2\hat{x}}{dx^2} \tag{3}$$

If we now evaluate the expressions $\frac{d\hat{x}}{dx}$ and $\frac{d^2\hat{x}}{dx^2}$ and introduce them in the expression, we can see that:

$$\frac{d\hat{x}}{dx} = \frac{d}{dx}\left(\frac{x}{L}\right)$$
$$\frac{d\hat{x}}{dx} = \frac{1}{L}$$

$$\frac{d^2\hat{x}}{dx^2} = \frac{d}{dx}\left[\frac{d}{dx}\left(\frac{x}{L}\right)\right]$$
$$\frac{d^2\hat{x}}{dx^2} = \frac{d}{dx}\left(\frac{1}{L}\right)$$
$$\frac{d^2\hat{x}}{dx^2} = 0$$

And, therefore, expression 3 stays as follows:

$$\frac{d^2u(x)}{dx^2} = \frac{1}{L^2} \cdot \frac{d^2u(\hat{x})}{d\hat{x}^2} \tag{4}$$

Finally, the only thing we have to do is change the argument of our function as indicated in the deffinition of $\hat{x}$ and introduce the result from equation 4 in our differential equation. Once we have done this, such equation adquires the form:

$$\gamma\frac{d^2u(x)}{dx^2} = -Fu(x) \tag{5}$$

$$\gamma\frac{1}{L^2}\frac{d^2u(\hat{x})}{d\hat{x}^2} = -Fu(\hat{x}) \tag{6}$$

$$\frac{d^2u(\hat{x})}{d\hat{x}^2} = -\frac{FL^2}{\gamma}u(\hat{x}) \tag{7}$$

$$\frac{d^2u(\hat{x})}{d\hat{x}^2} = -\lambda u(\hat{x}) \tag{8}$$

PROBLEM 2

The code for this problem is stored inside the 'eignevecsP2.cpp' file, and the display of the comparison between the analytic solution and the one obtained from our program is shown in ' .txt' file.

If we check this last file we will see that, in some cases, the eigenvectors from our program might not be exactly equal but, instead, they will be the same as the analytic ones multiplied by a certain constant (-1 in this case). This is perfectly fine, as these are eigenvectors too.

PROBLEM 3

3.a)

The first part of the file called 'fulljacobi.hpp' corresponds to the definition of this function. Instead of returning just the maximum value of the elements outside of the diagonal, our program returns an element of the type std::map, where the entries "maxval", "row", "col" correspond to the maximum value, the index k of the row where it is located inside the matrix and the index l of the column respectively.

3.b)

This exercise corresponds to the program contained in the file 'maxvaltest.cpp', where we have included the .hpp file mentioned in the previous section and we have defined the test function in order to check the wellfunctioning of the program. After compiling and linking it, we get a file which, when executed, gives an output in our terminal where we can see the maximum value outside of the diagonal as well as the row and column where it is located.

PROBLEM 4

A. 4.a)

Following the recomendation from the problem's statement, we have created a code with a similar structure to that of the snippet which can be seen in the file 'fulljacobi.hpp'.

The first one of the functions defined in it is usefull for finding the maximum value outside of the diagonal of our matrix. The next one is called jacobi _ rotation and the only thing it does is performing one rotation as described in the jacobi method to whatever squared symmetric matrix we introduce as an argument for the function, as well as returning the new maximum value of the matrix after performing such rotation.

Finally, the function jacobi _ eigensolver is a function, in our case, with 5 arguments:

- A: this is the matrix of our problem.
- iter: this is a variable that we will use to count the number of transformations performed until reaching convergence.
- eps: this is what we would call "tolerance". It is just a refference that we use to know when our algorithm has reached a form of the matrix A which is "diagonal enough".
- eval: this is a vector where we will store all of the eigenvalues of our matrix.
- evec: this matrix will contain the eigenvectors of this problem as column vectors.

The main feature of this function is a while loop which will perform transformations to our matrix A using the function jacobi _ rotate until reaching convergence. This is why we chose the current maximum value of the matrix as the output for the previously mentioned function, as it will help us to keep track of it in order to know when the algorithm has already converged.

As every argument except for A and eps is an "empty" argument which is then redefined inside the function itself, we can already declare these variables in the header file so that we only need to define A and eps in the source files where we will use such function.

4.b)

The file 'p2e42.cpp' contains a very simple program which generates the 6×6 matrix needed for this problem and then introduces it into the jacobi method's algorithm in order to know the eigenvalues and eigenvectors of A, which are then displayed as an output into the terminal so that we can compare them to the analytic results already stored inside the .txt file from problem 2.

PROBLEM 5

5.a)

The code for this exercise is inside the file 'p2e5.cpp'. This creates another file 'n _ iter.txt' where are stored the values of the size of the matrix and the number of transformations performed in our algorithm before reaching convergence for each one of these sizes.

This .txt file can then be used in a python script to produce a plot which shows the growth in the number of iterations against the growth in the size of our matrix A.

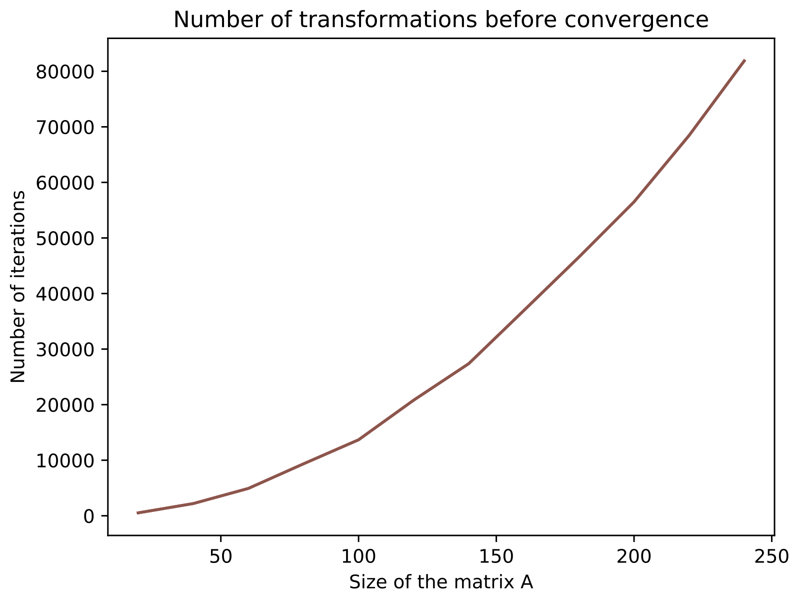

Figure 1. The graphic shows the number of transformations performed until convergence against the size of the matrix A. There is a total of 12 different values of size going from 20 to 240

5.b)

The reason why the algorithm is so slow is the fact that, when transforming the matrix A, due to the properties of matrix multiplication, not every value that was 0 in the beginning remains 0 after transforming it, so the number of 0 elements of the original matrix is, actually, irrelevant, and the scaling behaviour is very similar. We would therefore expect the algorithm to be just as slow for dense matrices.

PROBLEM 6

The plots that we have generated for this problem are the following:

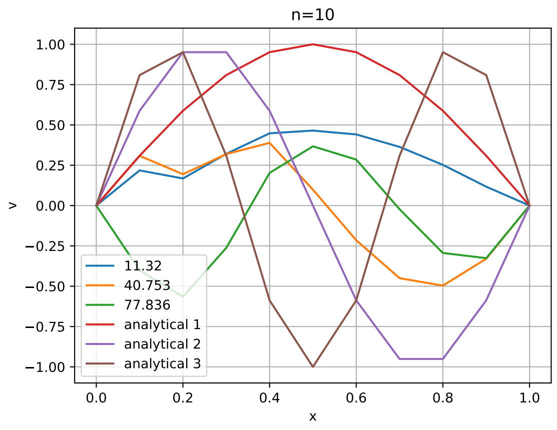

Figure 2. Representation of the eigenvectors (both the program-generated ones and the analytical solutions) for the case of a discretization of n steps

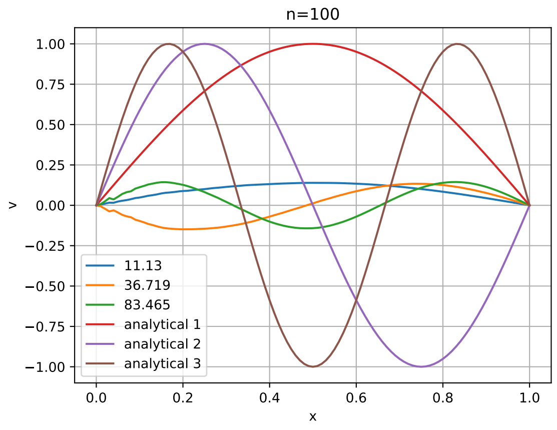

Figure 3. Same case as the one in figure 2 but for the case of n=100

As we can appreciate (especially in the figure for n=100), there is a difference in the scale of the vectors (they are not normalized) and, on top of that, the third eigenvector in analytical form is different from the same one generated by the program. This last thing is because, as we have already seen in problem 2, some of the eigenvectors that our program generates are equal to the analytical ones but multiplied by the constant -1.