PROJECT 2

Computational Physics - FYS3150

September 27, 2021

Duncan Wilkins | Fuad Dadvar | Don Phillip Bullecer | Erling Nupen
https://github.com/dondondooooon/FYS3150

Problem 1

In order to make the equation $\gamma\frac{d^2u(x)}{dx^2} = -Fu(x)$ dimensionless, we need to re-define our differential operator $\frac{d}{dx}$ in terms of a differential operator which differentiates with respects to $\tilde{x}$.

$$(\frac{d}{dx})^2 = (\frac{d\tilde{x}}{dx}\frac{d}{d\tilde{x}})^2 = \frac{1}{L^2}\frac{d^2}{(d\tilde{x})^2} \tag{1}$$

Inserting this new differential operator into our original equation, we get:

$$\frac{d^2u(\tilde{x})}{d\tilde{x}^2} = -\frac{L^2F}{\gamma}u(\tilde{x}) = -\lambda u(\tilde{x}) \tag{2}$$

Problem 2

In order for $\vec{w}$ to be an orthonormal set, it must satisfy the relation $\vec{w}_j\vec{w}_i = \delta_{ji}$. If we now insert what we know $\vec{w}$ to be, $\vec{w}_i = \mathbf{U}\vec{v}_i$ we get:

$$\vec{w}_j^T\vec{w}_i = \vec{v}_j^T\mathbf{U}^T\mathbf{U}\vec{v}_i = v_jv_i = \delta_{ji} \tag{3}$$

And we have therefor proven $\vec{w}$ is also an set of orthonorma. vectors. In the equation above the relations $(\mathbf{U}\vec{v}_j)^T = \vec{v}_j^T\mathbf{U}^T$ was used for the first step whilst $\mathbf{U}\mathbf{U}^T = I$ was used in the second step.

Problem 3

Before we can implement our Jacobi rotation algorithm, we must be able to set up a tridiagonal matrix A. View github *(add specific file)* to view our code which does exactly that for N = 6. We must check with our analytical

result to see if they match. We are already given the analytical results from the introduction to this task, and the eigenvalues and eigenvectors are equal to:

$$\lambda^i = d + 2aCos(\frac{i\pi}{N+1}) \tag{4}$$

$$\vec{v}^{(i)} = [sin(\frac{i\pi}{N+1}), sin(\frac{2i\pi}{N+1}) \dots sin(\frac{Ni\pi}{N+1})] \tag{5}$$

We are to compare the analytical and numerical results for N = 6. The analytical results for i = 1 to N, is:

Analytical		**Numerical**
	λ^i	λ^i
(i = 1)	9.7098	9.7051
(i = 2)	3.6892·10	3.6898 ·10
(i = 3)	7.6181·10	7.6193·10
(i = 4)	$1.1981 \cdot 10^2$	$1.1981 \cdot 10^2$
(i = 5)	$1.5991 \cdot 10^2$	$1.5910 \cdot 10^2$
(i = 6)	$1.8663 \cdot 10^2$	$1.8629 \cdot 10^2$

Table 1: This table shows us the analytical eigenvalues compared with the numerical eigenvalues.

The analytical eigenvectors are:

v^1	v^2	v^3	v^4	v^5	v^6
0.2319	0.4179	0.5211	0.5211	0.4179	0.2319
0.4179	0.5211	0.2319	-0.2319	-0.5211	-0.4179
0.5211	0.2319	-0.4179	-0.4179	0.2319	0.5211
0.5211	-0.2319	-0.4179	0.4179	0.2319	-0.5211
0.4179	-0.5211	0.2319	0.2319	-0.5211	0.4179
0.2319	-0.4179	0.5211	-0.5211	0.4179	-0.2319

Table 2: These shows the analytical eigenvectors

The numerical eigenvectors are

v^1	v^2	v^3	v^4	v^5	v^6
0.2319	0.4179	0.5211	0.5211	0.4179	0.2319
0.4179	0.5211	0.2319	-0.2319	-0.5211	-0.4179
0.5211	0.2319	-0.4179	-0.4179	0.2319	0.5211
0.5211	-0.2319	-0.4179	0.4179	0.2319	-0.5211
0.4179	-0.5211	0.2319	0.2319	-0.5211	0.4179
0.2319	-0.4179	0.5211	-0.5211	0.4179	-0.2319

Table 3: These shows the numerical eigenvectors

As we can see, they are exactly the same. If you run the code for this problem, you would get that the eigenvectors are flipped, aka the numerical eigenvectors are multiplied with minus one, but the eigenvalue equation obviously still holds since you can just factor out the -1 from the vector.

Problem 4

We wrote a C++ function "maxoffdiagsymmetric" that takes in an matrix input, references to two integers, and identifies the largest off-diagonal element (in absolute value) in the matrix. We assume that the matrix is symmetric ($k < l$), so we can evaluate only the lower triangle to make the program run faster. The function then writes down the matrix indices for this element to the two integer references. And finally, returns the value of this matrix element.

We ran a small testcode for the following matrix A, which is given as:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 1 & -0.7 & 0 \\ 0 & -0.7 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

The result being that the function returns max = 0.7 and the indices k and l is (1,2) respectively.

Problem 5

a

Jacobi's algorithm is an algorithm which repeatedly performs rotations on a matrix, A, until it becomes almost diagonal. This new matrix will have the approximations of real eigenvalues of A, on its diagonal. Writing the algorthim out on an matrix form, it looks like equation (7).

$$A' = SAS^T \tag{7}$$

Where S is a Givens rotation matrix, on the form:

$$\begin{vmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & -s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{vmatrix}$$

Our program applies this rotation matrix N times and for each time it is applied, the biggest off-diagonal element is found, in order to maximize our algorithm.

b

The numerical eigenvectors calculated using Jacobi-algorithm is:

v^1	v^2	v^3	v^4	v^5	v^6
0.2319	0.4179	0.5211	0.5211	0.4179	0.2319
0.4179	0.5211	0.2319	-0.2319	-0.5211	-0.4179
0.5211	0.2319	-0.4179	-0.4179	0.2319	0.5211
0.5211	-0.2319	-0.4179	0.4179	0.2319	-0.5211
0.4179	-0.5211	0.2319	0.2319	-0.5211	0.4179
0.2319	-0.4179	0.5211	-0.5211	0.4179	-0.2319

Table 4: Eigenvectors from Jacobi algorithm

The eigenvalues calculated by Jacobi's algorithm is:

And as we can see, the Jacobi algorithm gives us exactly the same as the analytical and the numerical values calculated from armadillo.

λ^i
(i = 1) 9.7098
(i = 2) 3.6892·10
(i = 3) 7.6181·10
(i = 4) $1.1981 \cdot 10^2$
(i = 5) $1.5991 \cdot 10^2$
(i = 6) $1.8663 \cdot 10^2$

Table 5: Numerical values calculated from Jacobi algorithm

Problem 6

a

The number of required transformations scale with a given matrix size N is given on the table below:

N-size matrix	Number of transformation
2.0E0	1
3.0E0	9
4.0E0	6
5.0E0	31
6.0E0	35
7.0E0	69
8.0E0	92
9.0E0	120
10.0E0	153

Table 6: Shown number of tranformations and the value that are given out by each transformation as N-size matrix

We see a general trend where the larger the given matrix size N, the more transformation needs to be performed on the matrix before we reach a result where all non-diagonal matrix elements are close to zero. However, for unknown reasons there is a weird dip when $N = 4$.

b

It is easier to see the scaling behaviour if we plot number of symmetry transformations vs N. From the plot below 1 we can see that the scaling behaviour is exponential.

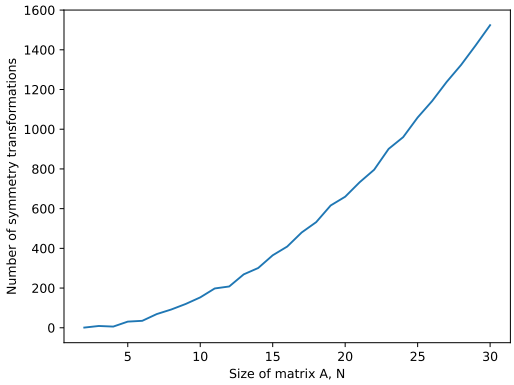

Figure 1: This figure illustrates how the numbers of transformations required N - scalings.

Problem 7

a and b

Using the Jacobi Rotation method we plotted the three eigenvectors that corresponds to the three lowest eigenvalues, for both $n = 10$ and $n = 100$.

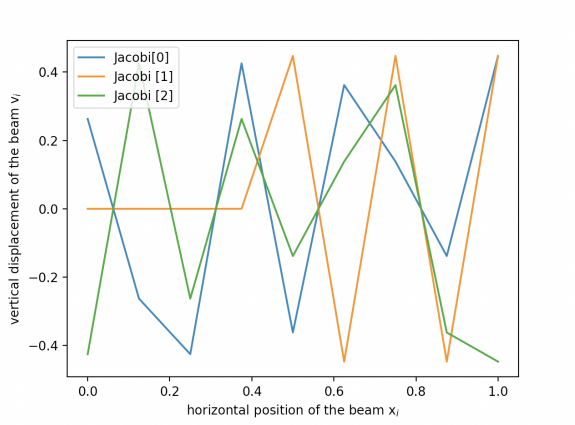

Figure 2: This figure illustrates the vector elements $\vec{v}_i$ corresponding to the positions $\hat{x}$.

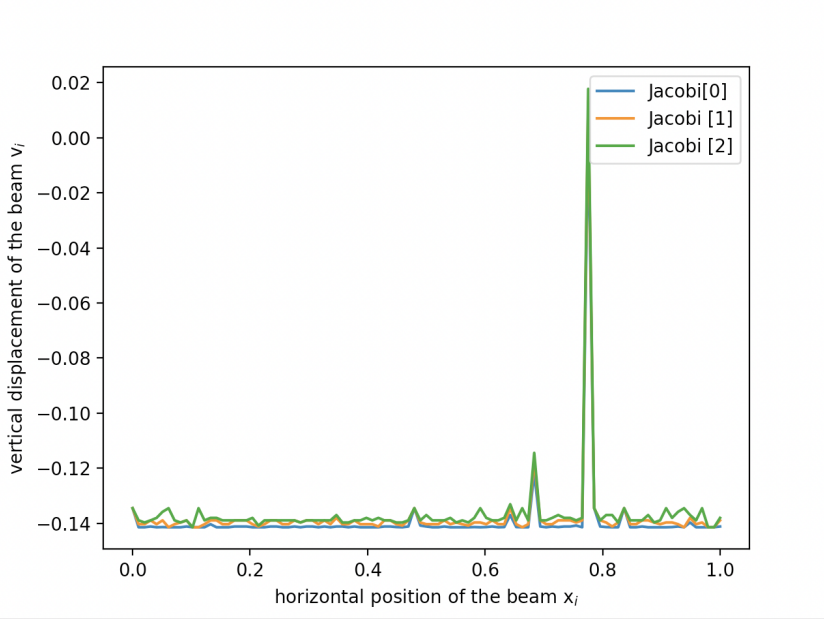

Figure 3: This figure illustrates the same as figure(2), but this time with 100 n - steps.