Project 1 Scientific Computing 2021

Kimi Cardoso Kreilgaard (twn176)

November 6, 2021

Questions for Week 1

a(1)

The condition number of a matrix under the matrix infinity norm is given by eq. 1 ([1] p. 55)

$$\text{cond}_\infty(\mathbf{A}) = ||\mathbf{A}||_\infty \cdot ||\mathbf{A}^{-1}||_\infty \tag{1}$$

where $||\mathbf{A}||_\infty$ is the matrix infinity norm defined in eq. 2 below ([1] p. 54). Put simply, it is the maximum absolute row sum of the matrix.

$$||\mathbf{A}||_\infty = \max_i \sum_{j=1}^{n} |a_{ij}| \tag{2}$$

Putting these steps together, I have created a small function, `cond`, that produces the condition number when given a matrix.

a(2)

For three values of the frequency ω the condition number under the infinity norm is calculated using the method described in a(1). These are listed in table 1 along with the number of significant digits expected in $\mathbf{x}$ when the right hand side $\mathbf{z}$ is given with 8 significant digits and everything else is assumed exact. The condition number is an exact number and I have just chosen to display the first few digits. As for the number of significant digits in x, the decimals actually hold information in case we were to change base. As long as we work with ten based numbers we would however floor the number and chose that as the number of significant digits we report.

Frequency	**Condition Number**	**# Significant Digits in x**
1.300	303.0742	5.5185
1.607	327825.2278	2.4844
2.700	35.5520	6.4491

Table 1: Condition numbers and number of significant digits in $\mathbf{x}$ for three values of ω.

When assuming everything but $\mathbf{z}$ exact, the relative uncertainty on $\mathbf{x}$ is given by:

$$\frac{||\Delta x||_\infty}{||x||_\infty} = \text{cond}_\infty(\mathbf{E} - \omega\mathbf{S}) \frac{||\Delta z||_\infty}{||z||_\infty} \tag{3}$$

The number of significant digits corresponds to rewriting the relative error in $\mathbf{x}$ as a power of ten, then the exponent corresponds to the number of significant digits. We find this by taking:

$$-\log_{10}\left(\frac{||\Delta x||_\infty}{||x||_\infty}\right) \tag{4}$$

b(1)+b(2)

Now we assume that $\mathbf{z}$ is exact, but a pertubation on ω is introduced which contributes to an error on the matrix, $\delta\mathbf{A}$. Since $\mathbf{A} = \mathbf{E} - (\omega \pm \delta\omega)\mathbf{S} = \mathbf{A} \pm \delta\omega\mathbf{S}$, the equation from ([1] p. 59) becomes:

$$\frac{||\Delta x||_\infty}{||x||_\infty} = \text{cond}_\infty(\mathbf{A})\frac{||\delta\mathbf{A}||_\infty}{||\mathbf{A}||_\infty} = \text{cond}_\infty(\mathbf{E}-\omega\mathbf{S})\frac{||\delta\omega\mathbf{S}||_\infty}{||\mathbf{E}-\omega\mathbf{S}||_\infty} \tag{5}$$

The number of significant digits is determined with the same method as in a(2). The forward bounds and the number of significant digits it yields are listed in table 2 below

Frequency	**Relative Forward Error**	**# Significant Digits in x**
1.300	0.0048	2.3232
1.607	5.0900	-0.0767
2.700	0.0005	3.2726

Table 2: Relative forward error bounds due to a pertubation in the frequency.

c(1)+c(2)+c(3)

I implemented a function `lu_factorize` to perform LU factorisation by gaussian elimination, inspired by the pseudocode on p. 68 [1]. The function `forward_substitute` performs forward substitution to solve a lower triangular system. The function is inspired by the pseudocode on p. 64 [1]. The function `back_substitute` to perform back substitution solves an upper triangular system. The function is inspired by the pseudocode on p. 65 [1].

These three functions are combined into a `linear_solver` that takes $\mathbf{A}$ and $\mathbf{b}$ as input and solves the system $\mathbf{Ax} = \mathbf{b}$ returning the solution vector $\mathbf{x}$. We test the function on the linear system of equations:

$$\begin{bmatrix} 2 & 1 & 1 \\ 4 & 1 & 4 \\ -6 & -5 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 11 \\ 4 \end{bmatrix} \tag{6}$$

Both the library routine `numpy.linalg.solve` and my solution yields the solution vector $\mathbf{x}$ to consist of $x_1 = -4$, $x_2 = 7$ and $x_3 = 5$.

d(1)+d(2)

The function `solve_alpha` is implemented in the attached .py file. It makes use of the `linear_solver` function to solve the equation $(\mathbf{E} - \omega\mathbf{S}) = \mathbf{z}$. Alpha is then found by computing: $\alpha(\omega) = \mathbf{z}^T\mathbf{x}$. For three values

of ω, the polarizability α is calculated along with the pertubated α. These values are listed in table 3 below.

ω	$\delta\omega$	$\alpha(\omega - \delta\omega)$	$\alpha(\omega)$	$\alpha(\omega - \delta\omega)$
1.300	$0.5 \cdot 10^{-3}$	-4.8934	-4.8759	-4.8586
1.607	$0.5 \cdot 10^{-3}$	151.0967	-434.9606	-91.0935
2.700	$0.5 \cdot 10^{-3}$	-0.3761	-0.3759	-0.3758

Table 3: Polarizabilities and their perturbations

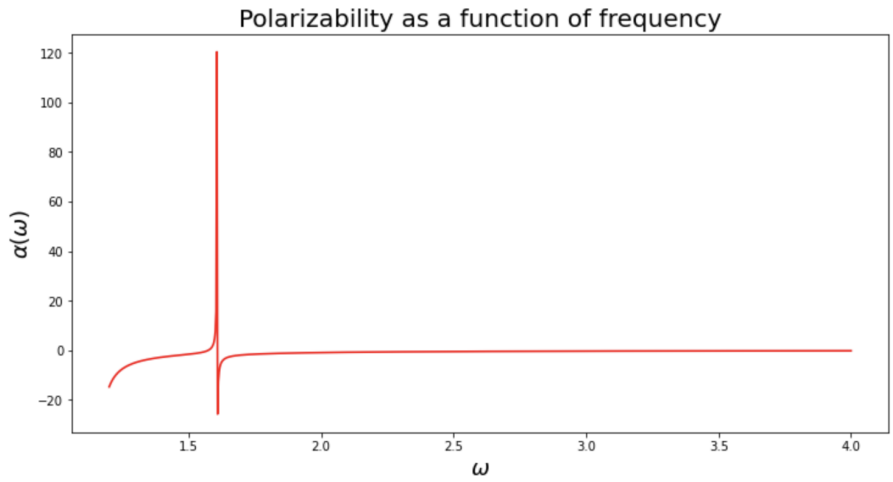
The variation of the calculated polarizailities arises due to a perturbation in the frequency ω. Since the frequency is part of the description of the matrix $\mathbf{A} = (\mathbf{E} - \omega\mathbf{S})$, and we still assume $\mathbf{z}$ to be exact, the error bound calculated in (b) is the correct one to understand the variation. Notice, however, that the error bound calculated in (b) is a bound on $\mathbf{x}$ while we want a bound on $\alpha = \mathbf{z}^T\mathbf{x}$. The bound for α is given by:

$$\frac{||\Delta\alpha||_\infty}{||\alpha||_\infty} = \frac{||\mathbf{z}^T\Delta x||_\infty}{||\mathbf{z}^T\hat{\mathbf{x}}||_\infty} \leq \frac{||\mathbf{z}||_\infty||\Delta x||_\infty}{||\mathbf{z}^T\hat{\mathbf{x}}||_\infty} \tag{7}$$

Notice that the bound involves the computed solution $\hat{\mathbf{x}}$ instead of the unknown, real solution. If we use the unknown solution $\hat{\mathbf{x}}$ and it turns out to be orthogonal to $\mathbf{z}$ the bound will approach ∞. We therefor use the computed solution to avoid this problem, and since we actually know that value. To obtain the bound we use a triangle inequality. This means that the bound on α will be proportional to Δx.

e(1)+e(2)

In the graph below, the polarizability α for 1000 evenly spaced values of $\omega \in [1.2, 4]$ is plotted. The



condition number of a nonsingular matrix is ∞, we can therefor check if we obtain a very large number when computing the conditionnumber of the matrix $\mathbf{E} - \omega\mathbf{S}$ with $\omega = 1.60686978$.

$$\text{cond}(\mathbf{E} - 1.60686078\mathbf{S}) = 1.73 \cdot 10^{10} \tag{8}$$

We see that the matrix "approaches" singularity in this point, which is reflected in $\alpha(\omega)$ by a discontinuity in the function.

Questions for Week 2

f(1)+f(2)

I implemented the function `householder_qr` in the .py file which computes the QR decomposition of a rectangular matrix. The function is based on the description of the method found on `http://rosettacode.org/wiki/QR_decomposition`. In the code, we see that $\mathbf{Q}$ is orthogonal by checking whether $\mathbf{Q}^T\mathbf{Q} = \mathbf{Q}\mathbf{Q}^T = \mathbf{I}$. We also find that $\mathbf{R}$ satisfies $\mathbf{A} = \mathbf{QR}$.

We combine the function above with the `back_substitute` function to create a `least_squares` routine. It uses the QR decomposition method to transform $\mathbf{A}$ into an upper triangular matrix $\mathbf{R}$. The matrix $\mathbf{Q}$ is used to find the transformed right hand side vector $\mathbf{b}' = \mathbf{Q}^T\mathbf{b}$. We can now use back substitution to solve the upper triangular system $\mathbf{Rx} = \mathbf{b}'$ and find $\mathbf{x}$ in from the original system $\mathbf{Ax} = \mathbf{b}$.

We test the function on the system:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1237 \\ 1941 \\ 2417 \\ 711 \\ 1177 \\ 475 \end{bmatrix} \tag{9}$$

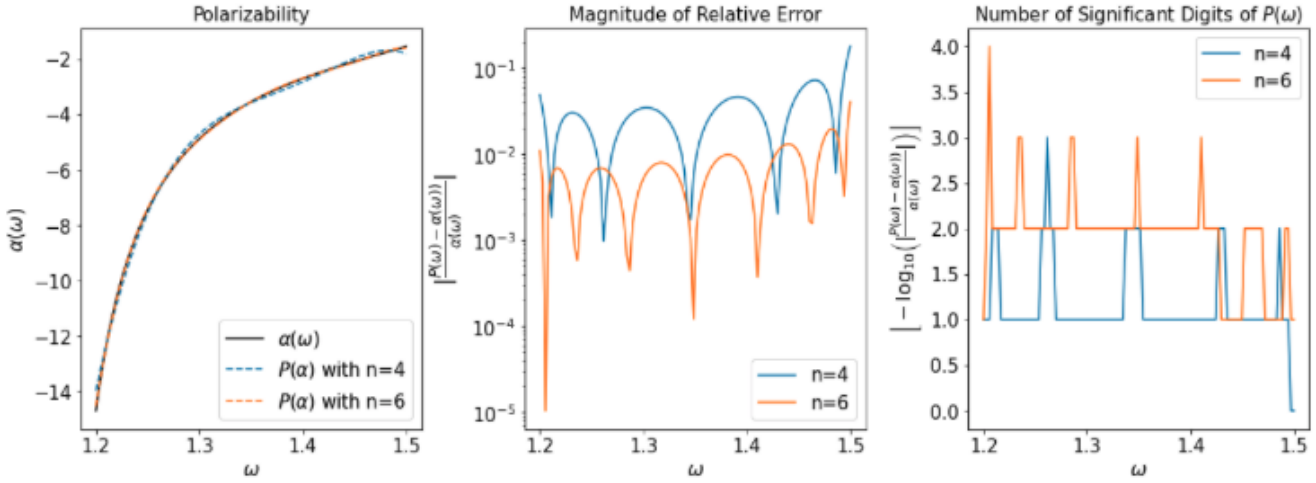
The library routine `np.linalg.lstsq` and my own routine both obtain the solution vector $\mathbf{x}$ consisting of $x_1 = 1236$, $x_2 = 1943$ and $x_3 = 2416$.

g(1)

Since there is a discontinuity in the data around $\omega = 1.60686978$ I choose to work with the interval $[1.2, 1.5]$. Using the points we plotted in e(1) this yields 108 points to fit the polynomium to.

g(2)+g(3)+g(4)

I created a function `get_P_coef` which computes the coefficients a_j of the polynomial approximation (eq. 4 in the assignment) for a given n and when supplied with the coordinates of a number of points to fit the polynomium to. This routine was repeated for $n = 4$ and $n = 6$, with the results plotted in the figure below along with the original function we are trying to approximate. In the center panel the magnitude of the relative error is plotted. Here it is evident that the polynomium with $n = 6$ is a more accurate approximation, with the relative error becoming almost one order of magnitude smaller than in $n = 4$. The number of significant digits in α varies with ω and with the approximation used. This is plotted on the right panel, computed by taking the negative log10 to the magnitude of the relative error (similar to what we did in eq. 4). Typically the $n = 4$ approximation yields 1 significant digit while $n = 5$ typically yields 2 significant digits.

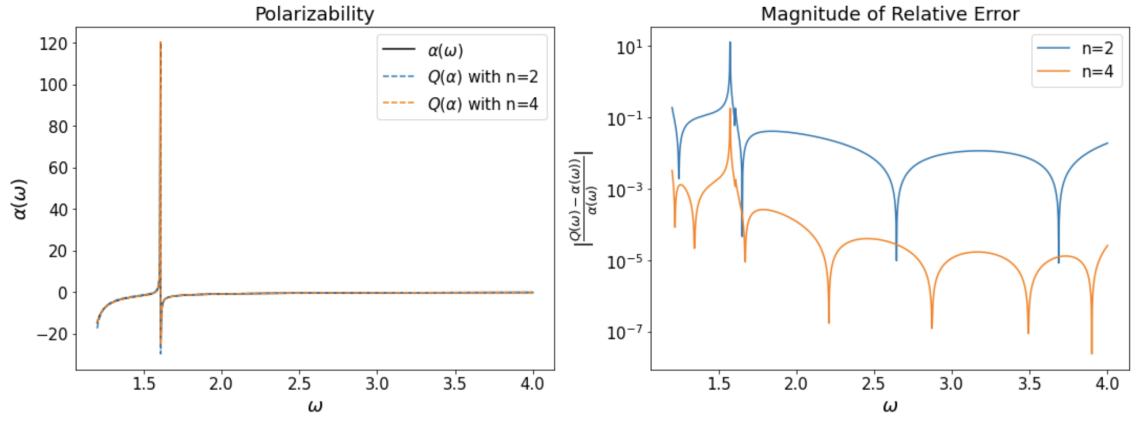


h(1)+h(2)

We now extend the interval of frequencies to [1.2,4]. We rewrite the rational approximating function Q to a linear expression as is outlined in the hint for problem (h):

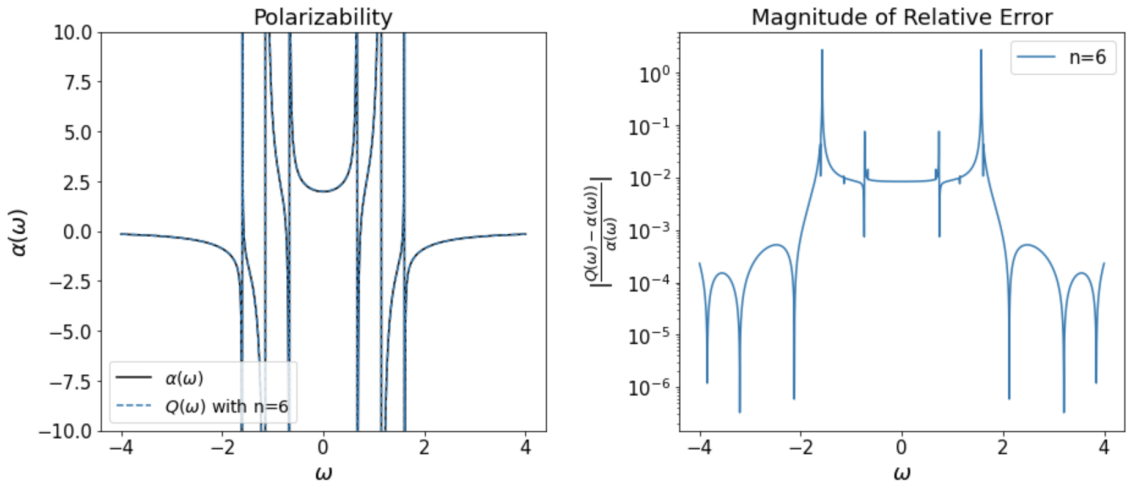
$$Q(\omega) = \frac{\sum_{j=0}^{n} a_j \omega^j}{1 + \sum_{j=1}^{n} b_j \omega^j} \approx \sum_{j=0}^{n} a_j \omega^j - \sum_{j=1}^{n} b_j (\alpha(\omega)\omega^j) \tag{10}$$

The function `get_Q_coef` computes the coefficients a_j and b_j with a least squares routine solving the system $\mathbf{Ax} = \mathbf{c}$. Where $\mathbf{A}$ is a matrix containing the terms inside the sums, respectively ω^j and $\alpha(\omega)\omega^j$. $\mathbf{x}$ is a columnvector with the coefficients $[a_0, a_1, ..., a_n, b_1, ..., b_n]$ and $\mathbf{c}$ is a columnvector with the y-values of the function we are trying to approximate $c_j = \alpha(\omega_j)$. When the coefficents are determined, we can insert these into the function `Q_func`, which takes a list of ω-values, the order n which we will approximate with and the coefficients of the fit and returns a corresponding list of Q-values approximating the true α-values. Q is calculated from the rational function displayed in (the middle part of) eq. 10. The approximated function Q for $n = 2$ and $n = 4$ is plotted along with the original function $\alpha(\omega)$ in the left panel in the figure below. In the right panel the magnitude of the relative error for each of the rational functions used is plotted. From this it is clear that the accuracy of Q with $n = 4$ is much better, the accuracy of course varies with ω but generally $n = 4$ seems to be approximately two orders of magnitude more accurate than $n = 2$.



h(3)

Now that we can approximate singularities, lets expand the range of frequencies to [-4,4]. We use the same functions as before and similar principles to approximate the polarizability in this larger range of frequencies. The least squares method that is employed to fit the rational approximating function is, however, quite sensitive to extreme outliers, that is points in the singularities where α is diverging. To make sure these points don't have too much influence on the fit, I use 3000 evenly spaced values of ω instead of only 1000 as we have done so far. In this range there are now 6 singularities, so we use $n = 6$ to be able to approximate all singularities that arise when the polynomium containing the b coefficients in the denominator is equal to -1. Using $n = 6$ will give this polynomium 6 roots (at least in the complex plane) according to the fundamental theorem of algebra. I obtain the approximating function plotted in the left panel below. In the right panel the magnitude of the relative error is displayed.



References

[1] Michael T. Heath, *Scientific Computing: An Introductory Survey*, 2nd Ed.