

# SciComp Project 1

## Weeks 1+2

September 1, 2023

### 1 Background

Many of you will know the phenomenon that a prism refracts light, i.e. splits it up in different colors, because the refractive index of the prism varies with the frequency  $\omega$  of light. The underlying property of the molecules forming the material is the frequency dependent polarizability,  $\alpha(\omega)$ . The polarizability, like many other properties of molecules and materials, can be calculated from the basic laws of physics, here the time-dependent Schrödinger equation.

In the end, the polarizability for a given frequency  $\omega$  of the incoming light is obtained as the following scalar product of two column vectors  $\mathbf{z}$  and  $\mathbf{x}$ :

$$\alpha(\omega) = \mathbf{z}^T \mathbf{x} \quad (1)$$

where  $\mathbf{z}$  is a vector that can be calculated from the Schrödinger equation, and  $\mathbf{x}$  is the solution to the following system of linear equations:

$$(\mathbf{E} - \omega \mathbf{S}) \mathbf{x} = \mathbf{z} \quad (2)$$

Here,  $\mathbf{E}$  and  $\mathbf{S}$  are two square matrices, and  $\omega$  is the frequency of the incoming light. Like the column vector  $\mathbf{z}$ , the matrices  $\mathbf{E}$  and  $\mathbf{S}$  are calculated from the Schrödinger equation, which we will not discuss further in this course.

### 2 Data

In this project we consider the water molecule,  $\text{H}_2\text{O}$ , and its frequency dependent polarizability,  $\alpha(\omega)$ . It turns out that the matrices  $\mathbf{E}$  and  $\mathbf{S}$ , and the column vector  $\mathbf{z}$ , have a structure that lets us decompose the matrices into submatrices as follows:

$$\mathbf{E} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B} & \mathbf{A} \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & -\mathbf{I} \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} \mathbf{y} \\ -\mathbf{y} \end{bmatrix} \quad (3)$$

The Python-file `watermatrices.py` contains the submatrices  $\mathbf{A}$  and  $\mathbf{B}$ , and the subvector  $\mathbf{y}$  for a water molecule, obtained by an approximate solution to the Schrödinger equation. From this, you can construct the full matrices  $\mathbf{E}$  and  $\mathbf{S}$ , which in this approximation should be  $14 \times 14$ .

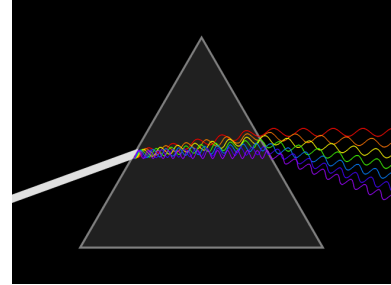


Figure 1: A dispersive prism slows light at different rates depending on the wave-length, causing refraction.

### 3 Questions for Week 1

- a. (1) Write a small function that computes the condition number of a matrix under the max-norm:

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

Use a library matrix inversion routine<sup>1</sup> for  $\mathbf{M}^{-1}$ , but do program the max-norm yourself using `sum`, `abs`, and `max`. (2) For three frequencies,  $\omega = \{0.800, 1.146, 1.400\}$ , calculate the condition number for the matrix  $\mathbf{E} - \omega\mathbf{S}$ . The right-hand-side  $\mathbf{z}$  is given with 8 significant digits. How many significant digits could we guarantee in the solution  $\mathbf{x}$  if everything else were assumed exact? Why?

- b. (1) For each of the three  $\omega$ , determine a bound on the relative forward error in the max-norm:

$$\frac{\|\Delta\mathbf{x}\|_{\infty}}{\|\hat{\mathbf{x}}\|_{\infty}} \leq \text{cond}_{\infty}(\mathbf{E} - \omega\mathbf{S}) \frac{\|\delta\omega\mathbf{S}\|_{\infty}}{\|\mathbf{E} - \omega\mathbf{S}\|_{\infty}}$$

for the perturbation that the frequency  $\omega$  is changed by  $\delta\omega = \frac{1}{2} \cdot 10^{-3}$ . Recall that  $\hat{\mathbf{x}}$  is the *computed* approximate value (known) of the exact  $\mathbf{x}$  (unknown), and  $\Delta\mathbf{x} \equiv \mathbf{x} - \hat{\mathbf{x}}$ . (2) As  $\omega$  is given with 3 digits after the comma, how many significant digits could we be guarantee in  $\mathbf{x}$  if everything else were exact? Why?

- c. Implement three separate functions

1. `L,U = lu_factorize(M)`,<sup>2</sup> which takes a square matrix  $\mathbf{M}$  as input and returns two square matrices: A triangular matrix  $\mathbf{L}$  and upper triangular matrix  $\mathbf{U}$  such that  $\mathbf{M} = \mathbf{L}\mathbf{U}$ .
2. `y = forward_substitute(L,z)`, which takes a square lower triangular matrix  $\mathbf{L}$  and a vector  $\mathbf{b}$  as input, and returns the solution vector  $\mathbf{y}$  to  $\mathbf{L}\mathbf{y} = \mathbf{b}$ .
3. `x = back_substitute(U,y)`, which takes a square upper triangular matrix  $\mathbf{U}$  and a vector  $\mathbf{y}$  as input, and returns the solution vector  $\mathbf{x}$  to  $\mathbf{U}\mathbf{x} = \mathbf{y}$ .

and test them with the linear equation

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

You can test your solution against a library routine.<sup>3</sup>

If you know how, try to use vector operations instead of for-loops where possible (orders of magnitude faster in Python).

- d. Implement a function `alpha = solve_alpha(omega)` for calculating the frequency-dependent polarizability  $\alpha(\omega) = \mathbf{z}^T \mathbf{x}$  for water in the given approximation. This routine should solve Equation (2) by LU-factorization using your own three routines from (c). (1) Using your routine, make a table of the polarizabilities for the frequencies given in (a) and their perturbations, i.e. for  $\omega = \{0.800 \pm \delta\omega, 1.146 \pm \delta\omega, 1.400 \pm \delta\omega\}$ , with  $\delta\omega = \frac{1}{2} \cdot 10^{-3}$  as before. (2) Which error-bound is the correct one to understand the variation of the calculated polarizabilities due to the perturbation: (a) or (b) or both? Explain why. (3) Use this bound to derive an upper bound for  $\Delta\alpha(\omega) = \alpha(\omega + \delta\omega) - \alpha(\omega)$  of the form  $|\Delta\alpha(\omega)| \leq B(\omega)|\delta\omega|$ . Do your calculated values fall within this bound?
- e. (1) Compute a table of  $\alpha(\omega)$  for 1000 evenly spaced values in the interval  $[0.7, 1.5]$ <sup>4</sup> using your routine from (d), and plot the values. (2) Can you explain what happens to the linear system of Equation (2) around the frequency  $\omega = 1.146307999$ , and how is this reflected in  $\alpha(\omega)$ ?

<sup>1</sup>E.g. `inv` from `numpy.linalg` for Python. But `inv` is allowed only here.

<sup>2</sup> If you implement it with pivot, you will be able to solve a wider range of systems. However, it is not required.

<sup>3</sup>E.g. `numpy.linalg.solve` in Python

<sup>4</sup>Use `linspace` in Python.

## 4 Questions for Week 2

f. Implement and test the Householder and least-squares routines below.

1. `Q,R = householder_QR_slow(A)`, which takes as input a rectangular matrix  $\mathbf{M}$ :  $m \times n$  and uses the Householder method to compute its QR decomposition. Check that  $\mathbf{Q}$ :  $m \times m$  is orthogonal, i.e.,  $\mathbf{Q}^T \mathbf{Q} = \mathbf{Q} \mathbf{Q}^T = \mathbf{I}$ , and that the upper triangular matrix  $\mathbf{R}$ :  $m \times n$  satisfies  $\mathbf{M} = \mathbf{Q} \mathbf{R}$ .
2. A more efficient version of (f.1) that doesn't compute the  $m \times m$  matrix  $\mathbf{Q}$ , but just stores the reflection vectors  $\mathbf{v}$ . You can either give it the interface `V,R = householder_fast(A)` or (as is done in practice) let the result be a combined  $(m+1) \times n$  matrix  $\mathbf{VR}$ , in which the upper  $n \times n$  triangle contains  $\mathbf{R}$ , and the  $k$ th column below the diagonal is  $\mathbf{v}_k$  (of length  $m-k$ ).
3. `tilde_x,r = least_squares(A,b)`, which combines this routine with your back-substitution from (c.3) to compute a linear least squares fitting. It should take as input a rectangular  $m \times n$  matrix  $\mathbf{A}$  and an  $m \times 1$  right-hand-side vector  $\mathbf{b}$ , returning an  $n \times 1$  approximate solution vector  $\tilde{\mathbf{x}}$  to  $\mathbf{A}\tilde{\mathbf{x}} \simeq \mathbf{b}$  as output, and the residual. If you did not get (f.2) to work, you can just use your solution to (f.1).

You can use the linear system in Heath's Example 3.1 & 3.8 to debug individual steps of your computation. Test your routines against `A1,b1` from `HHexamples.py`, and print the resulting upper triangular  $\mathbf{R}$  and solution  $\mathbf{x}$  rounded to 3 digit accuracy.

g. We now want to approximate  $\alpha(\omega)$  in the interval  $[0.7, \omega_p]$  using a polynomial

$$P(\omega) = \sum_{j=0}^n a_j \omega^{2j} \quad (4)$$

(1) Suggest a suitable value of  $\omega_p < 1.5$ . What makes this choice reasonable? (2) Find the coefficients of the polynomial using your linear least squares routine from (f)<sup>5</sup>, the table computed above, and  $n = 4$ . (3) Repeat the computation for  $n = 6$  and compare the accuracies of the two polynomials: Plot the magnitude of the relative error (in a  $\log_{10}$ -scale) of the polynomial approximation as the difference between the  $P(\omega)$  and  $\alpha(\omega)$  values. (4) How many significant digits does each approximation yield?

h. We would now like to approximate  $\alpha(\omega)$  in the entire interval  $[0.7, 1.5]$ , and choose the *rational* approximating function, which is able to represent singularities:

$$Q(\omega) = \frac{\sum_{j=0}^n a_j \omega^j}{1 + \sum_{j=1}^n b_j \omega^j} \quad (5)$$

(1) Why will this fail with the polynomial approximation of Problem (g)? And why can Eq. (5) do a better job? (2) Find the coefficients  $a_j$  and  $b_j$  using your linear least squares routine, the table of  $\alpha$ -values computed above, and  $n = 2$ . You need to reformulate the expression as an linear approximation so that you can use a linear least squares fitting. Plot the error of the the rational-function approximation  $Q(\omega)$  compared to  $\alpha(\omega)$  calculated by Equations (1) and (2).<sup>6</sup> (2) Repeat the computation for  $n = 4$  and compare the accuracies of the two approximations quantitatively. (3) If you look at  $\alpha(\omega)$  in the extended interval  $\omega \in [-4; 4]$ , you will notice that  $\alpha(\omega)$  has multiple singularities. Are you able to modify your approximation to accurately approximate the full interval  $[-4; 4]$ , in particular so that it reproduces the singularities correctly? Explain your solution.

<sup>5</sup>If you could not get your own least squares solver to work, you can use a library version for the remaining problems - but make sure to write clearly that you have done so.

<sup>6</sup> Once you have computed the coefficients  $\mathbf{a}$  and  $\mathbf{b}$ , be sure to finish the construction of  $Q(\omega)$  using Eq. (5), and to use this for the error. It is tempting (but wrong) to just use the linear approximation you used to find  $\mathbf{a}$  and  $\mathbf{b}$ , but that is linear and hence cannot represent singularities.