

Exercise: Iterative Solver and Gradient Decent Methods

In this exercise class, based on the morning lecture, we discuss the basics of how to solve linear equation systems (LES). In the first task, we will discuss how to implement iterative solvers and in the second exercise, we will discuss the conjugate gradient method. Finally, we provide a Bonus advanced task.

Motivation: The Poisson Problem

Many problems in physics can be described by so-called partial differential equations (PDE). One simple example of a PDE is the Poisson problem and our aim on this exercise sheet is to solve the Laplace problem numerically.

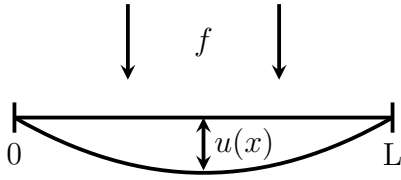


Fig. 1: The clothesline problem (Poisson problem in 1D): A clothesline (a 1D line) is held in place at the points $x = 0$ and $x = L$ everywhere else it is pulled down by gravity, where we consider gravity to be the uniform force $f = -1$. The displacement of the clothesline at point x is given by $u(x)$.

The mathematical description of the problem described in Fig. 1 is given by: Let $f \in C^0([0, L])$ be given then find $u \in C^2([0, L])$ such that

$$\begin{cases} -u''(x) = f & \forall x \in (0, L), \\ u(0) = 0, \\ u(L) = 0. \end{cases}$$

To solve this equation numerically we first of all need to discretize the problem. This will yield the following LES:

$$\underbrace{\frac{1}{h^2} \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & -1 & 2 & -1 & 0 \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & \dots & 0 & 0 & 0 & 1 \end{pmatrix}}_{=A} \cdot \underbrace{\begin{pmatrix} U_0 \\ U_1 \\ \vdots \\ \vdots \\ U_n \\ U_{n+1} \end{pmatrix}}_{\mathbf{u}} = \underbrace{\begin{pmatrix} 0 \\ f_1 \\ \vdots \\ \vdots \\ f_n \\ 0 \end{pmatrix}}_{\mathbf{f}} \quad (1)$$

Where n is the number of discretization steps and $h = \frac{L}{n+1}$ is the size of each discretization interval. The derivation exceeds the scope of this exercise, but this can be done for example with finite differences or a finite element approach, please see for details [1, Chapter 6.3] or [1, Chapter 8], respectively.

For example if we choose $n = 3$ and $L = 1$ equation (1) becomes

$$\begin{pmatrix} 16 & 0 & 0 & 0 & 0 \\ -16 & 32 & -16 & 0 & 0 \\ 0 & -16 & 32 & -16 & 0 \\ 0 & 0 & -16 & 32 & -16 \\ 0 & 0 & 0 & 0 & 16 \end{pmatrix} \cdot \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 0 \\ f_1 \\ f_2 \\ f_3 \\ 0 \end{pmatrix} \quad (2)$$

Exercise 1: Basic Iterative Solvers based on Fixed-Point Schemes

To solve such a linear system as given in equation (1) we have seen in the lecture today different methods. Here, we want to solve the LES with different iterative schemes and compare those iterative schemes.

For this exercise, you will need the file *exercise_1*.

(a) Jacobi Method

Read the code in the provided file *exercise_1.py*. After you have read the code complete the *jacobi* function by adding the iteration step while using index-notation.

Reminder (from slide 32 of the class lecture notes), the index notation of the Jacobi method is

$$x_i^k = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^n a_{ij} x_j^{k-1} \right), \quad i = 1, \dots, n.$$

(b) Gauss-Seidel Method

Implement the Gauss-Seidel method, while using index notation.

Reminder (from slide 33 of the class lecture notes), Index-notation of the Gauss-Seidel method:

$$x_i^k = \frac{1}{a_{ii}} \left(b_i - \sum_{j < i} a_{ij} x_j^k - \sum_{j > i} a_{ij} x_j^{k-1} \right), \quad i = 1, \dots, n.$$

(c) Comparison between the Jacobi and the Gauss-Seidel Method

Consider different n (for example $n = 4, 8, 16$) and compare the results between the two different methods. Specifically, please compare the number of iteration steps of both methods. What do you observe? What is your interpretation?

Exercise 2: Conjugate Gradients

The system matrix of the Poisson problem is symmetric and positive definite, the LES provided in equation (1) can be solved with the *conjugate gradient* (CG) method, which is faster (please make yourself clear what ‘faster’ means: less iterations? Less wall time?) than the methods we have seen in the first exercise.

For this exercise you will need the file *exercise_2*.

(a) Implement the CG Method

Implement the *conjugate gradient* (CG) method. You can use the following pseudo-code as a starting point.

Algorithm 1: CG Method

```
Given: Matrix  $A$ , Vector  $\mathbf{b}$ , Vector  $\mathbf{x}^0$  ( $= 0$ , initial solution)
Vector  $\mathbf{x}_{\text{old}} = \mathbf{x}^0$ ;
Vector  $\mathbf{r}_{\text{old}} = \mathbf{b} - A \cdot \mathbf{x}_{\text{old}}$ ;
Vector  $\mathbf{d}_{\text{old}} = \mathbf{r}_{\text{old}}$ ;
int  $k = 0$ ;
while  $k < n$  do
    Vector  $A\mathbf{d}^k = A \cdot \mathbf{d}_{\text{old}}$ ;
     $\alpha_k = \frac{\mathbf{r}_{\text{old}}^T \cdot \mathbf{d}_{\text{old}}}{\mathbf{d}_{\text{old}}^T \cdot A\mathbf{d}^k}$ ;
    Vector  $\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} + \alpha^k * \mathbf{d}_{\text{old}}$ ;
    Vector  $\mathbf{r}_{\text{new}} = \mathbf{r}_{\text{old}} - \alpha^k * A\mathbf{d}^k$ ;
    if  $(|\mathbf{r}_{\text{new}}| < \text{tol})$  then
         $\mathbf{x}_{\text{old}} = \mathbf{x}_{\text{new}}$ ;
        exit loop;
    end
     $\beta^k = \frac{\mathbf{r}_{\text{new}}^T \cdot A\mathbf{d}^k}{\mathbf{d}_{\text{old}}^T \cdot A\mathbf{d}_{\text{old}}}$ ;
     $\mathbf{d}_{\text{new}} = \mathbf{r}_{\text{new}} - \beta^k * \mathbf{d}_{\text{old}}$ ;

    //update values
     $k = k + 1$ ;
     $\mathbf{x}_{\text{old}} = \mathbf{x}_{\text{new}}$ ;
     $\mathbf{r}_{\text{old}} = \mathbf{r}_{\text{new}}$ ;
     $\mathbf{d}_{\text{old}} = \mathbf{d}_{\text{new}}$ ;
end
return  $\mathbf{x}_{\text{old}}$ ;
```

(b) Comparison between previous methods and the CG method

Again, consider different n (for example $n = 8, 16, 32$) and compare the results between the two different methods. Specifically, compare the number of iteration steps of both methods. What do you observe? What is your interpretation?

Exercise 3: (Bonus) Preconditioned Conjugate Gradients

For this exercise you will need the file *exercise_3.py*.

(a) Implement the SSOR Preconditioner

Implement the SOR Preconditioner as described on slide 63.

(b) Implement the Preconditioned CG (PCG) Method and perform Comparisons

Implement the PCG method as described on slide 60. Please compare now Jacobi, Gauss-Seidel, CG, SSOR-PCG.

References

- [1] T. Wick, *Numerical methods for partial differential equations*, Hannover : Institutionelles Repositorium der Leibniz Universität Hannover, DOI: <https://doi.org/10.15488/11709>, 2022. DOI: <https://doi.org/10.15488/11709>.