Problem Set 7

Exercises for the Lecture Fundamentals of Simulation Methods (WS24/25)

Michela Mapelli and Cornelis Dullemond (Lecture Tuesday and Thursday 14h - 16h)
Lucia Haerer (Tutor Group T1 on Wednesday 11h - 13h)
Stefano Rinaldi (Tutor Group T2 on Thursday 11h - 13h)
Nicholas Storm (Tutor Group T3 on Friday 11h - 13h)

Submit the solution in electronic by the **Friday 6pm**, **December 6, 2024**.

7. Diffusion and Poisson equation in one and more dimensions

7.1. Forward elimination, backward substitution (*10 points*)

We consider the following set of n coupled linear equations, written in matrix form:

$$\mathbf{A}\mathbf{y} = \mathbf{r} \tag{1}$$

with $n \times n$ matrix

$$\mathbf{A} = \begin{pmatrix} b_0 & c_0 & & & & & \\ a_1 & b_1 & c_1 & & & & \\ & a_2 & b_2 & c_2 & & & \\ & & \cdots & & & & \\ & & a_{n-3} & b_{n-3} & c_{n-3} & & \\ & & & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & & a_{n-1} & b_{n-1} \end{pmatrix} \tag{2}$$

and right hand side vector

$$\mathbf{r} = \begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_{n-2} \\ r_{n-1} \end{pmatrix} \tag{3}$$

Note the special form of the matrix **A**: it is *tridiagonal*.

1. Write your own Python[1] function `solve_tridiagonal(a,b,c,r)`, where `a`, `b`, `c` and `r` are 1D arrays of n elements, and the result is a 1D array `y` with the solution to the above equation. Note that `a[0]` and `c[n-1]` will have no meaning, as they are outside of the matrix. As always, test your function with some simple test problems by going the reverse way: enter the solution `y` into the equation, carry out the matrix multiplication, and verify that the original `r` values come out again.

[1] Or in C or Fortran if you prefer.

2. Now consider the following 1D "gravity" problem:

$$\frac{d^2\Phi(x)}{dx^2} = \rho(x) \tag{4}$$

(where we set the gravitational constant $4\pi G = 1$ for convenience). The density function $\rho(x)$ is given by

$$\rho(x) = \begin{cases} 1 & \text{for} \quad |x| \leq 1 \\ 0 & \text{for} \quad |x| > 1 \end{cases} \tag{5}$$

Let us choose our domain from:

$$-x_{\max} \leq x \leq x_{\max} \tag{6}$$

with, for now, $x_{\max} = 3$. At the boundaries of this domain ($x \pm x_{\max}$) we set the *boundary conditions*

$$\Phi(\pm x_{\max}) = 0 \tag{7}$$

Of course, this particular problem can be solved analytically. But we will do this numerically here.

a) Write a program that solves this problem on a grid of n gridpoints, using the `solve_tridiagonal(a,b,c,r)` function you made above.

b) Solve this problem on a grid of $n = 100$ gridpoints, and plot the result as a figure of y against x.

c) Now solve it for $n = 10000$ and overplot the result over the $n = 100$ case.

7.2. LU decomposition *(5 points)*

Let us now consider the same problem as above, but now without taking into account that the matrix has a tridiagonal shape. We will set up the matrix as a 2D array and use the LU decomposition method to solve the matrix equation $\mathbf{Ay} = \mathbf{r}$. We will use the `scipy.linalg` library for that. Here is an example of how to use `scipy.linalg` to solve a matrix equation (taken from the SciPy documentation):

```
import numpy as np
from scipy.linalg import lu_factor, lu_solve
A = np.array([[2, 5, 8, 7],\
              [5, 2, 2, 8],\
              [7, 5, 6, 6],\
              [5, 4, 4, 8]])
r = np.array([1, 1, 1, 1])
lu, piv = lu_factor(A)
y = lu_solve((lu, piv), r)
np.allclose(A @ y - r, np.zeros((4,)))
```

1. Solve the "gravity" problem of the previous exercise for $n = 100$ using LU decomposition instead of forward elimination, backward substitution.

2. Now do this for $n = 10000$. Notice that it takes a lot longer than with the tridiagonal method. If it takes *too* long on your laptop, try with $n = 3000$ or $n = 1000$.

7.3. 2D problem *(10 points)*

Let us consider the 2D version of the "gravity" problem:

$$\frac{\partial^2\Phi(x,y)}{\partial x^2} + \frac{\partial^2\Phi(x,y)}{\partial y^2} = \rho(x,y) \tag{8}$$

on a 2D grid with `nx` gridpoints in x-direction and `ny` gridpoints in y-direction. $\rho(x,y)$ is given as

$$\rho(x,y) = \begin{cases} 1 & \text{for} \quad \sqrt{x^2+y^2} \le 1 \\ 0 & \text{for} \quad \sqrt{x^2+y^2} > 1 \end{cases} \tag{9}$$

The function $\rho(x,y)$ will then become an array `rho[0:nx,0:ny]`, meaning the x-index `ix` has values from `0` to `nx-1`, and similar for `iy`. The same will be true for the solution $\Phi(x,y)$ becoming an array `Phi[0:nx,0:ny]`.
In the computer memory such a 2D array is, in fact, a big 1D array with length `nx*ny`. It is stored row-wise:

```
rhoflat = [ rho[0,0],...,rho[0,ny-1] ,
            rho[1,0],...,rho[1,ny-1] ,
             ... ,
            rho[nx-1,0],...,rho[nx-1,ny-1] ]
```

The Numpy-command `rhoflat = rho.flatten()` will return the 1D array `rhoflat` corresponding to the 2D array.
We can now again write the "gravity" problem in matrix form:

$$\mathbf{A}\mathbf{s} = \mathbf{r} \tag{10}$$

where $\mathbf{r}$ is now a vector with length `nx*ny` which, for all non-boundary locations, equals the `rhoflat` array. The matrix $\mathbf{A}$ is now an array with `nx*ny` rows and `nx*ny` columns. For all non-boundary locations, the matrix $\mathbf{A}$ contains the Laplace operator in discrete form. Due to the row-wise storage of the 2D grid, the matrix $\mathbf{A}$ will have not only a tridiagonal diagonal shape, but will also have side-diagonals that are `ny` to the left and right of the diagonal. The boundary conditions are $\Phi = 0$ at $x = \pm x_{\text{max}}$ and $y = \pm y_{\text{max}}$. These are implemented on the diagonal of the matrix $\mathbf{A}$.

1. This problem is implemented for you in the program `exercise_2d_1.py`. A small portion is left out: where the solution is obtained using LU decomposition.
   a) Use `plt.imshow(A,origin='lower')` to show that the matrix indeed has side-diagonals. Try this for different numbers of grid cells.
   b) Complete the program, solve the problem with `scipy.linalg.lu_factor` and `scipy.linalg.lu_solve`, and show the solution for a grid of 60x40 gridpoints (`nx=60, ny=40`) (take a smaller number of cells if your laptop cannot handle 60x40). Tip: your solution is a flattened array, which you can reshape back onto a 2D array with `Phi = s.reshape((nx,ny))`.

2. Since the matrix **A** can become really large, while containing mostly zeros, it is better to use a *sparse matrix storage* form of this matrix. Also, instead of using LU decomposition we will now use the iterative method `BiCGStab`. This is implemented in `exercise_2d_2.py`, with, again, the solution step left out.

    a) Complete the program, solve the problem with `scipy.sparse.linalg.bicgstab`, and show the solution for a grid of 60x40 gridpoints (`nx=60`, `ny=40`).

    b) Using the keyword `maxiter=10` you can stop the `BiCGStab` iteration after 10 iteration steps. Analyse how good or bad the solutions are for 1, 3, 10, 30, 100 iteration steps, by overplotting a 1D cross-section of the 2D solution near $y = 0$ (i.e., `iy=ny//2`). This figure should look roughly like this: