Project 2

Deadline: Tuesday, 18 May 2021

In this project you will implement a multigrid method for the numerical solution of the following elliptic boundary value problem in $\mathbb{R}^d$, $d = 1, 2$.
Let $\Omega = (-1, 1)^d \subset \mathbb{R}^d$. Given the functions $c, f \in C^0(\overline{\Omega})$ find a function $u \in C^2(\overline{\Omega})$ such that

$$\begin{aligned} -\Delta u + cu &= f \quad \text{in } \Omega\,, \\ u &= 0 \quad \text{on } \partial\Omega\,. \end{aligned}$$

Throughout this project we fix

$$c(x) = |x|^2\,.$$

Let $x_0 = -1 < x_1 < x_2 < \ldots < x_{N-1} < 1 = x_N$ be a uniform partitioning of the interval $[-1, 1]$ with mesh size $h = \frac{2}{N}$, $N = 2^\ell$, $\ell \geq 1$. We define the uniform grids $\Omega_\ell = \{x_0, x_1, \ldots, x_N\}$ in 1d and similarly $\Omega_\ell = \{(x_0, x_0), (x_0, x_1), \ldots, (x_0, x_N), \ldots, (x_N, x_0), \ldots, (x_N, x_N)\}$ in 2d.
Let

$$A_h u_h = f_h, \qquad \text{where } A_h \in \mathbb{R}^{(N-1)^d \times (N-1)^d}, \tag{1}$$

be the standard finite difference approximation to the given boundary value problem.
We also consider the sequence of grids $\Omega_1, \ldots, \Omega_\ell$, where Ω_1 is the trivial grid with $x = 0$ as the only interior node.
For the implementation of the following tasks you should use Python, and you are free to use all available standard types and methods. You are only allowed to import the external package NumPy, as well as Matplotlib for the preparation of your report.

Multigrid in 1d

For the one-dimensional simulations, we choose

$$f(x) = (x - \tfrac{1}{2})^4\,.$$

1. Implement a function `jacobi_step_1d(uh, fh, omega)` that performs one step of the weighted Jacobi method with weight ω for the linear system (1). The inputs are `uh`, the initial guess $u_h^{(k)}$, `fh`, the right hand side from (1), and `omega`, the weight ω. The outputs are the new iterate $u_h^{(k+1)}$ (in place of `uh`), and the pseudo-residual $|u_h^{(k+1)} - u_h^{(k)}|_\infty$, which acts as the return value of the function.

   Here, even though $u_h, f_h \in \mathbb{R}^{N-1}$, `uh` and `fh` are implemented as arrays of length $N + 1$, with `fh[i]` $= f(x_i)$ for $i = 1, \ldots, N - 1$, while `fh[0]` and `fh[N]` can hold the boundary conditions for u, which in our case are zero. You may assume that `uh[0]` and `uh[N]` are zero.

   Note that the function should infer the value of N, and hence $h = \frac{2}{N}$, from the length of the arrays `uh` and `fh`.

2. Investigate the number of iterations needed for the weighted Jacobi method with $\omega = \frac{2}{3}$, and the effect on the CPU time, applied to the linear system (1) for the stated model problem, as h decreases.

   Here the iteration, starting from $u_h^{(0)} = 0$, should stop when the pseudo-residual satisfies $|u_h^{(k+1)} - u_h^{(k)}|_\infty <$ `tol` $= 10^{-8}$. Your sequence of grid sizes should include as least $N = 2^\ell$, $\ell = 3, \ldots, 7$.

3. Implement the two-grid correction scheme, where instead of solving $A_{2h}e_{2h} = r_{2h}$, you perform ten weighted Jacobi steps on the coarse grid. Investigate the improvement on the number of iterations, and on the CPU time, compared to the standard Jacobi iterations you have used above.

4. Implement a function `v_cycle_step_1d(uh, fh, omega)` that performs one V-cycle for the linear system (1), using one weighted Jacobi step with weight ω as pre- and post-smoothing. On the coarsest grid Ω_1 use an exact solve. The function should return the pseudo-residual of the post-smoothing Jacobi step, unless it is called on the coarsest grid Ω_1, when it can return zero.

   You can make the same assumptions on `uh` and `fh` as for the function `jacobi_step_1d()`. But here, in addition, you may assume that $N = 2^\ell$, for some $\ell \geq 1$.

5. Investigate the number of iterations needed for the implemented V-cycle with $\omega = \frac{2}{3}$, and the effect on the CPU time, applied to the linear system (1) for the stated model problem, as h decreases. Here the iteration, starting from a zero initial guess as usual, should stop when the pseudo-residual satisfies $|s_h^{(k+1)}|_\infty <$ `tol` $= 10^{-8}$ for the post-smoothing step on the finest grid. Your sequence of grid sizes should include as least $N = 2^\ell$, $\ell = 3, \ldots, 14$.

6. Implement a function `full_mg_1d(uh, fh, omega)` that performs a full multigrid step, utilizing your V-cycle implementation to perform a single V-cycle step in each grid, for the linear system (1). You can make the same assumptions on `uh` and `fh` as for the function `v_cycle_step_1d()`. The function should return the result of the call to `v_cycle_step_1d()` on the finest grid.

7. Investigate the CPU time needed, and the sizes of the final pseudo-residual $|s_h^{(k+1)}|_\infty$ as well as the size of the residual $|r_h|_\infty = |f_h - A_h u_h|_\infty$ after one full multigrid step with $\omega = \frac{2}{3}$, as h decreases. Your sequence of grid sizes should include as least $N = 2^\ell$, $\ell = 3, \ldots, 14$.

8. Investigate the number of additional V-cycles needed, and the effect on the total CPU time, by using the method `full_mg_1d()` to provde an initial guess for your V-cycle multigrid method. That is, after calling `full_mg_1d()`, use the produced approximation u_h as the initual guess for your V-cycle multigrid method. The total CPU time includes the time for `full_mg_1d()` and all the successive V cycles. Here the iteration should stop when the pseudo-residual satisfies $|s_h^{(k+1)}|_\infty <$ `tol` $= 10^{-8}$ for the post-smoothing step on the finest grid. Your sequence of grid sizes should include as least $N = 2^\ell$, $\ell = 3, \ldots, 14$.

9. Plot an approximation to the solution u over $\overline{\Omega}$ and provide an accurate estimate of $\max_{x \in \overline{\Omega}} u(x)$.

Multigrid in 2d

For the two-dimensional simulations, we choose

$$f(x) = f(x_1, x_2) = x_1 \, e^{2\,x_2} \, .$$

1. Implement the natural 2d variants of the previously discussed functions:

   - `jacobi_step_2d(uh, fh, omega)`,
   - `v_cycle_step_2d(uh, fh, omega)`,
   - `full_mg_2d(uh, fh, omega)`.

   In each case you can assume that `uh` and `fh` are two-dimensional NumPy arrays of size $(N+1) \times (N+1)$, where for the latter two functions $N = 2^\ell$ for some $\ell \geq 1$.

2. Repeat the analogous numerical investigations for the 1d model problem now in 2d. For the sequences of mesh sizes you should include at least $N = 2^\ell$, $\ell = 2, \ldots, 6$ for the weighted Jacobi iterations, and at least $N = 2^\ell$, $\ell = 2, \ldots, 8$ for the multigrid methods.

3. Plot an approximation to the solution u over $\overline{\Omega}$ and provide an accurate estimate of $\max_{x \in \overline{\Omega}} u(x)$.

Submission via Moodle

Prepare a single Python source file called `project2.py` that contains your implemented functions, together with any routines or classes that they depend on, but nothing else. The docstring of the file `project2.py` must contain your full name. In addition, prepare a short report in PDF format, called `project2.pdf`, that summarizes your investigations and reports on your results.