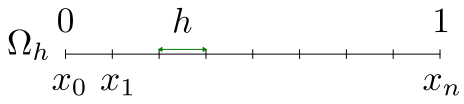
Practical work : A simple geometric multigrid method with python

C. Bovet

Introduction

The aim of this practical work is to implement a really simple multigrid method for the 1D Poisson problem.

$$\begin{aligned} -u''(x) &= f(x) \quad 0 < x < 1 \\ u(0) &= u(1) = 0 \end{aligned} \tag{1}$$



1 Technical information

You can work on your personal laptop, the only thing you need is python with the standard scientific libraries : numpy, scipy, matplotlib. If needed, you can use the virtual machine at `https://rep.mines-paristech.fr/home`.

1.1 Available documentation

This practical work make use of python packages, the documentation is online.

2 Required work

2.1 Model problem

1. Discretize problem (1) using classical finite differences with a constant length step $h = 1/n$ (n being the number of segments). It leads to the linear system :

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f} \tag{2}$$

   To fix the idea, we will use $n = 64$ in the following (but feel free to use a lower value for debugging).

2. Complete the `laplace` function in order to create the operator $\boldsymbol{A}$.

2.2 Smoother

We recall that multigrid algorithm make use of stationary iterative methods such as smoothers.

1. The preconditioner for Jacobi over relaxation (JOR) is $\boldsymbol{M} = \frac{1}{\omega}\text{diag}(\boldsymbol{A})$, $0 < \omega \leq 1$ Complete the `JOR` function.

2. The preconditioner for Successive Over Relaxation (SOR) is $\boldsymbol{M} = \frac{1}{\omega}\text{diag}(\boldsymbol{A}) - \boldsymbol{E}$ where $(-\boldsymbol{E})$ is the strict lower triangular part of $\boldsymbol{A}$ and $0 < \omega < 2$. Complete the `SOR` function.

Algorithm 1: Stationary iterative method

Initial guess $\boldsymbol{u}_0$
for $i = 0$ *to convergence* **do**
  $\boldsymbol{r}_i = \boldsymbol{f} - \boldsymbol{A}\boldsymbol{u}_i$
  $\boldsymbol{z}_i = \boldsymbol{M}^{-1}\boldsymbol{r}_i$
  $\boldsymbol{u}_{i+1} = \boldsymbol{u}_i + \boldsymbol{z}_i$
end

3. The use of a null right-hand-side and a custom initial guess gives some insights about the convergence of the method. Can you explain why ?

4. Plot few first approximate solutions with a high frequency initial guess $\boldsymbol{u}_0(x) = \sin(k_h \pi x)$.

5. Do the same thing for a low frequency initial guess $\boldsymbol{u}_0(x) = \sin(k_l \pi x)$ and draw some conclusions.

2.3 Restriction & prolongation

We consider the standard coarsening $H = 2h$.

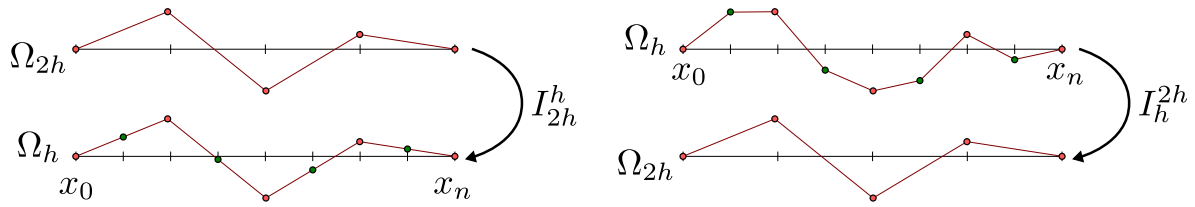


Figure 1: Classical injection and linear interpolation operators

1. The classical restriction is injection. As described in Figure 1, with the injection operator $\boldsymbol{I}_h^{2h}$, the value of a coarse node is directly the value of the corresponding fine node. Complete the `injection` function.

2. The classical prolongation is linear interpolation. Complete the `interpolation` function.

2.4 Two-grid algorithm

1. Complete the function `tgcyc` according to Algorithm 2 in order to get a working Two-grid cycle.

2. Replace the coarse grid solve by 3 iterations of smoothing. Is the convergence really affected ?

3. Adapt `tgcyc` to get a V-cycle according to Algorithm 3.

Algorithm 2: Two-grid cycle $\boldsymbol{u}_{k+1}^h = TGCYC(\boldsymbol{u}_k^h, \boldsymbol{A}^h, \boldsymbol{f}^h, \nu_1, \nu_2)$

`// Presmoothing`

Compute $\bar{\boldsymbol{u}}_k^h$ by applying $\nu_1 (\geq 0)$ steps of the smoothing procedure to $\boldsymbol{u}_k^h$:
$\bar{\boldsymbol{u}}_k^h = \text{Smooth}^{\nu_1}(\boldsymbol{u}_k^h, \boldsymbol{A}^h, \boldsymbol{f}^h)$

`// Coarse grid correction (CGC)`

Compute the defect $\bar{\boldsymbol{d}}_k^h = \boldsymbol{f}^h - \boldsymbol{A}^h \bar{\boldsymbol{u}}_k^h$

Restrict the defect $\bar{\boldsymbol{d}}_k^H = \boldsymbol{I}_h^H \bar{\boldsymbol{d}}_k^h$

Solve on Ω^H $\boldsymbol{A}^H \hat{\boldsymbol{v}}_k^H = \bar{\boldsymbol{d}}_k^H$

Interpolate the correction $\hat{\boldsymbol{v}}_k^h = \boldsymbol{I}_H^h \hat{\boldsymbol{v}}_k^H$

Update the approximation $\boldsymbol{u}_{k,afterCGC}^h = \bar{\boldsymbol{u}}_k^h + \hat{\boldsymbol{v}}_k^h$

`// Postsmoothing`

Compute $\boldsymbol{u}_{k+1}^h$ by applying $\nu_2 (\geq 0)$ steps of the smoothing procedure to $\boldsymbol{u}_{k,afterCGC}^h$:
$\boldsymbol{u}_{k+1}^h = \text{Smooth}^{\nu_2}(\boldsymbol{u}_{k,afterCGC}^h, \boldsymbol{A}^h, \boldsymbol{f}^h)$

Algorithm 3: Multigrid cycle $\boldsymbol{u}_{k+1}^h = MGCYC(\ell, \boldsymbol{u}_k^\ell, \boldsymbol{A}^\ell, \boldsymbol{f}^\ell, \nu_1, \nu_2)$

Apply $\nu_1 (\geq 0)$ presmoothing steps to $\boldsymbol{u}_k^\ell$: $\bar{\boldsymbol{u}}_k^\ell = \text{Smooth}^{\nu_1}(u_k^\ell, \boldsymbol{A}^\ell, \boldsymbol{f}^\ell)$

`// Coarse grid correction (CGC)`

Compute the defect $\bar{d}_k^\ell = f^\ell - A^\ell \bar{u}_k^\ell$

Restrict the defect $\bar{d}_k^{\ell-1} = I_\ell^{\ell-1} \bar{d}_k^\ell$

`// Compute an approx. sol.` $\hat{\boldsymbol{v}}_k^{\ell-1}$ `of the defect eq. on` $\Omega^{\ell-1}$

if $\ell = 1$ **then** Use a direct solver $\boldsymbol{A}^{\ell-1} \hat{\boldsymbol{v}}_k^{\ell-1} = \bar{\boldsymbol{d}}_k^{\ell-1}$

else

Perform one grid cycle using the zero grid function as a first approximation
$\hat{\boldsymbol{v}}_k^{\ell-1} = \text{MGCYC}(\ell - 1, 0, \boldsymbol{A}^{\ell-1}, \bar{\boldsymbol{d}}_k^{\ell-1}, \nu_1, \nu_2)$

end

Interpolate the correction $\hat{\boldsymbol{v}}_k^\ell = I_{\ell-1}^\ell \hat{\boldsymbol{v}}_k^{\ell-1}$

Update the approximation $\boldsymbol{u}_{k,afterCGC}^\ell = \bar{\boldsymbol{u}}_k^\ell + \hat{\boldsymbol{v}}_k^\ell$

Apply $\nu_2 (\geq 0)$ postsmoothing steps to $\boldsymbol{u}_{k,afterCGC}^\ell$: $u_{k+1}^\ell = \text{Smooth}^{\nu_2}(\boldsymbol{u}_{k,afterCGC}^\ell, \boldsymbol{A}^\ell, \boldsymbol{f}^\ell)$