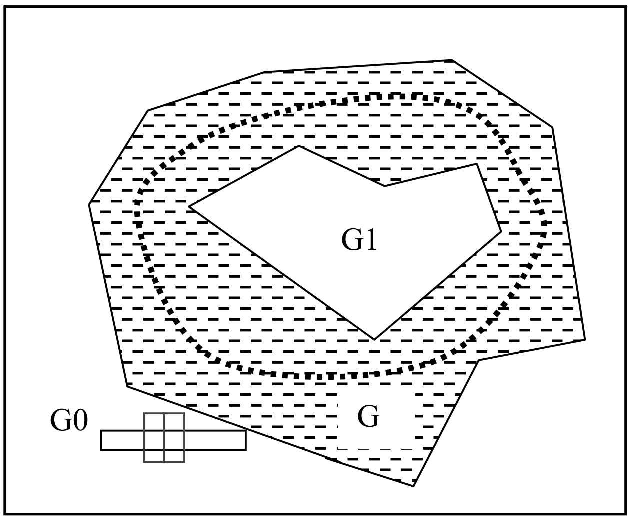
Multigrid for Poisson's Equation

(Inspired by Ch x in "*Computational Electromagnetics*", Bondeson & al., Springer)



The task is to compute the capacitance per unit length of a wave-quide with an outer conductor and an inner. We compute the potential as the solution to Laplace's equation over the annular domain between them, with Dirichlet conditions, $V = 1$ on the inner (G1) and $V = 0$ on the outer (G0). Then, the capacitance is

$$C = \oint_{\Gamma} \nabla V \cdot \mathbf{n} ds$$

with Γ any closed curve encircling the inner conductor, like the dashed curve.

The unit square is discretized by an n x n grid, $\Delta x = \Delta y = 1/(n-1)$, and we propose to solve the discretized equations by explicit time-stepping:

$$\frac{\partial V}{\partial t} = \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial x^2}, (x,y) \in G$$

$$V = \begin{cases} 0, (x,y) \in G0 \\ 1, (x,y) \in G1 \end{cases} \Rightarrow$$

```
for k = 0,1,...
   for $i,j = 2,...,n-1$
      $V_{ij}^{(k+1)} = V_{ij}^{(k+1)} + \frac{\Delta t}{\Delta x^2}\omega_{ij}\left(V_{i,j+1}^{(k)} + V_{i,j-1}^{(k)} + V_{i+1,j}^{(k)} + V_{i-1,j}^{(k)} - 4V_{i,j}^{(k)}\right)$
   end
end
```

$$\omega_{ij} = \begin{cases} 0, (i,j) \in G0 \cup G1 \\ \omega^*, \text{ elsewhere} \end{cases}$$

with initial values

$$V_{ij}^{(0)} = \begin{cases} 0, & (i,j) \in G0 \cup G \\ 1, & (i,j) \in G1 \end{cases}$$

Notes

1. The gridpoints with $\omega = 0$ are unchanged
2. ω^* = const. recovers the explicit time stepping scheme; $\omega^* < 1/4$ is necessary for stability (see below)
3. The boundary conditions are approximated with $O(\Delta x)$ error, since the boundary may be Δx from G0 and G1.
4. The central difference approximation to the Laplace operator is second order accurate.
5. G0 includes the outermost gridpoint layers ($i = 1$ and n, $j = 1$ and n)

How many steps are necessary to obtain an accurate solution?
The answer is produced most expeditiously by Fourier analysis. In order to illustrate with simpler formulas we consider the 1D problem with solution $u = 0$,

$$u_t = u_{xx}, u(0) = u(1) = 0$$
$$u(x,0) = f(x) \Rightarrow$$

```
for $k = 0,1,...$
    $u_j^{k+1} = u_j^{k+1} + \omega \frac{\Delta t}{\Delta x^2}\left(u_{j-1}^k - 2u_j^k + u_{j+1}^k\right), j = 1,...,n-1$
end
```

$$\Delta x = 1/n, u_j^0 = f(j\Delta x), \sigma = \omega \frac{\Delta t}{\Delta x^2}$$

where the initial guess is $u = f(x)$ and we have defined the "Courant number" σ.
By separation of variables, the solution u_j can be written as a linear combination of exponential functions,

$$u_j^k = \sum_{m=0}^{n-1} \tilde{u}_m^k W_n^{mj}, W_n = e^{\frac{2\pi i}{n}}, j = 0,1,..,n$$

the Discrete Fourier Transform. Let us see what happens to one of the exponentials, say

$$u_j = \tilde{u}\exp\left(i\xi x_j\right), j = 0,1,..,n;$$
$$u_{j-1} - 2u_j + u_{j+1} = ... = (2\cos\xi\Delta x - 2)u_j$$

so

$$\tilde{u}^{(k+1)} = G(\sigma,\theta)\tilde{u}^{(k)}, G = (1 + \sigma(2\cos\theta - 2)), \theta = \xi\Delta x$$

which defines the "von Neumann amplification factor" G for the wave-number ξ, which has phase-shift per cell θ.
For the boundary conditions here, $u(0) = u(1) = 0$, the smallest θ is π/n, and the largest is π. So, we have

$$1 - 4\sigma \le G \le 1 - \sigma\left(\frac{\pi}{n}\right)^2, \sigma = \omega\frac{\Delta t}{\Delta x^2}$$

It is clear that
1. $\sigma < 1/2$ is necessary for the iteration to decrease *all* the harmonics. Then

$$\max|G| \ge 1 - \left(\frac{\pi}{n}\right)^2 \quad (*)$$

For the 2D case the condition is $\sigma < ¼$.
Exercise: Prove!

2. The *smooth*, long-wavelength harmonics with small θ are damped much more slowly than the *oscillatory*, short-wavelength ones.

The number of steps N to reduce all harmonics (initially assumed to have amplitude $O(1)$) to an amplitude d is given by

$$G^N < d \text{ or } N > \frac{\ln d}{\ln(1-\frac{\pi^2}{n^2})} \approx n^2 \frac{-\ln d}{\pi^2}$$

so the iteration is *extremely* slow for large n. It is the

smooth components, say with $\theta < \pi/2$,

which are responsible for the slow convergence, whereas the

oscillatory components with $\theta > \pi/2$,

are quickly damped.

This is where the multi-grid idea enters: The time-stepping *is* efficient at reducing the short harmonics ($\theta > \pi/2$) if $\sigma < 1/2$. But the long harmonics can be well represented on a coarser grid on which they will appear more oscillatory. Indeed, half of them will be oscillatory according to the definition.

So the following is a natural attempt at a two-grid iteration to solve $\mathbf{Au} + \mathbf{f} = 0$ where $\mathbf{A}$ corresponds to an elliptic operator like the Laplace operator, with all negative eigenvalues.

Indeed, we have

$$\mathbf{v}^T\mathbf{A}\mathbf{v} \le -\mu\mathbf{v}^T\mathbf{v}$$

and $\mu = \pi^2/n^2$ in the above 1D example.

0. $\mathbf{v} = 0$ - take a better initial guess if it exists

1. Run $N1$ timesteps (= iterations) $\mathbf{v} = \mathbf{v} + \alpha(\mathbf{Av} + \mathbf{f})$. This reduces the oscillatory components of the residual $\mathbf{r} = \mathbf{Av}-\mathbf{f}$:

even if the solution *u* is oscillatory due to oscillatory *f* !!

2. Solve the correction equation $\mathbf{Ae} = \mathbf{r}$ on a coarser grid:

restrict $\mathbf{r}$ onto $\mathbf{r}_C$;

(*) solve $\mathbf{A}_C\mathbf{e}_C = \mathbf{r}_C$;

3. Interpolate (prolong) the coarse correction $\mathbf{e}_C$ onto the fine grid as $\mathbf{e}$;

correct $\mathbf{v}$: $\mathbf{v} = \mathbf{v} + \mathbf{e}$;

If not converged, goto 1.

When the exact solve step (*) is replaced by an approximate iterative solution by recursive use of even coarser grids the multi-grid V-cycle algorithm appears (Briggs p. 48) on a hierarchy of grids which we label 0 for finest and lmax for coarsest. The grids here will be obtained by successively doubling the mesh-width, and the number of gridpoints on the finest level ($2^M + 1$) is such that all gridpoints on level m appear also on all finer levels.

There remains to define the restriction and prolongation operators, and how to construct the matrix $\mathbf{A}_C$ from $\mathbf{A}$, generally, how to construct $\mathbf{A}$ for a given grid level. At least two ideas come to mind,

1. set $\mathbf{A}$ = difference approximation to d^2/dx^2 on the current grid, ie.

$$\mathbf{A} = \frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \dots & 0 \\ 0 & 1 & -2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 1 & -2 \end{pmatrix}, \quad \mathbf{f} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \dots \\ f(x_{n-1}) \end{pmatrix}$$

This requires computation to set up $\mathbf{A}_C$ from $\mathbf{A}$. If we multiply the equations by Δx^2 the matrices look the same on all grid levels, but the right-hand side **f** has to be scaled:

$$\mathbf{A} = \begin{pmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \dots & 0 \\ 0 & 1 & -2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 1 & -2 \end{pmatrix}, \quad \mathbf{f} = \Delta x^2 \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \dots \\ f(x_{n-1}) \end{pmatrix}$$

The intergrid transfer operators are defined next (see Briggs). The prolongation from stepsise $2h$ to h will be done by linear interpolation, and the restriction can be done by simple injection, or by the Galerkin construction. The gridlevels are denoted by superscripts and component numbers are subscripts,

$$\text{Prolong}, I_{2h}^{h} : \begin{cases} v_{2j}^{h} := v_j^{2h} \\ v_{2j+1}^{h} := \frac{1}{2}(v_j^{2h} + v_{j+1}^{2h}) \end{cases}, \quad \text{Restrict}, I_h^{2h} : v_j^{2h} := v_{2j}^{h}$$

$$\mathbf{I}_{2h}^{h} = \mathbf{P}_n = 0.5 \begin{pmatrix} 2 & 0 & 0 & \dots \\ 1 & 1 & 0 & \dots \\ 0 & 2 & 0 & \dots \\ 0 & 1 & 1 & \dots \\ 0 & 0 & 2 & \dots \\ 0 & 0 & 1 & \dots \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 2 \end{pmatrix}, \quad n \times (n+1)/2,$$

$$\mathbf{I}_h^{2h} = \mathbf{R}_{(n+1)/2} = \begin{cases} \text{Injection}: \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, (n+1)/2 \times n \\ \text{Galerkin}: 0.5\mathbf{P}_n^T \end{cases}$$

The Galerkin restriction is also called "full weighting". The recipes for constructing $\mathbf{A}_C$ from $\mathbf{A}$ may be explained as follows:
Find an approximate solution to $\mathbf{Au} + \mathbf{f} = 0$ in the subspace $\mathbf{u} = \mathbf{Pv}$, i.e., with **u** a linear interpolant to a coarse grid function, such that the restriction of the residual vanishes:

$$\mathbf{RAPv} + \mathbf{Rf} = 0$$

With **R** as injection, we simply pick every second of the equations. The Galerkin recipe is symmetric,

$$\mathbf{P}^T\mathbf{APv} + \mathbf{P}^T\mathbf{f} = 0$$

If **A** is a symmetric definite matrix (such as the Laplace matrix) the Galerkin **v** is the best in the sense of minimizing the “energy error” $(\mathbf{v}-\mathbf{u})^T\mathbf{A}(\mathbf{v}-\mathbf{u})$ over the subspace **P**. This property enables error estimates. Note, however, that the MG never solves the equation exactly on any grid, so the benefit in actual computation may be limited.

Exercise: Prove that for the Laplace matrix in 2D, the three variants:

i) Difference approximation on the current grid
ii) Restriction as Injection
ii) Restriction as full weighting

give the same matrices (possibly with different scalings)

The MG algorithm needs a simple interface to the difference equations. The scheme requires *no* coefficient matrices, only an evaluation of the difference equations for a candidate solution **v** on a grid hierarchy.

1. A basic iterative smoother for the residual $\mathbf{v}^{(n+1)} := \text{smooth}(\mathbf{v}^{(n)})$
2. The residual $\mathbf{r} = \mathbf{A}\mathbf{v}+\mathbf{f}$ itself

When the basic smoother is a time-stepper,

$$\mathbf{v}^{(n+1)} = S(\mathbf{v}^{(n)}), S(\mathbf{x}) = \mathbf{x} + \alpha\mathbf{r},$$
$$\mathbf{r}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{f};$$
$$\because \mathbf{r}(\mathbf{x}) = (S(\mathbf{x}) - \mathbf{x})/\alpha$$

we may choose S as the only interface. α is called the *relaxation* parameter.

The lab. represents the solutions etc. as 2D arrays. Let **U** be m x n. A few notes:

• The Laplace operator may be evaluated by matrix multiplication without BIG matrices:

$$\Delta_5\mathbf{U} = \frac{1}{\Delta x^2}\mathbf{T}_m\mathbf{U} + \frac{1}{\Delta y^2}\mathbf{U}\mathbf{T}_n, \quad \mathbf{T}_k = \begin{pmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \dots & 0 \\ 0 & 1 & -2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 1 & -2 \end{pmatrix}, \quad k \times k$$

• A `matlab` implementation can use `diff(U,n,k)` which computes the n:th order differences along direction k, the array `diff(U,2,1)` is $(m-2)$ x n and `diff(U,2,2)` is m x $(n-2)$.