Multigrid

optimal solvers for linear and nonlinear elliptic PDEs

Ed Bueler

UAF Math 692 Scalable Seminar

Spring 2023

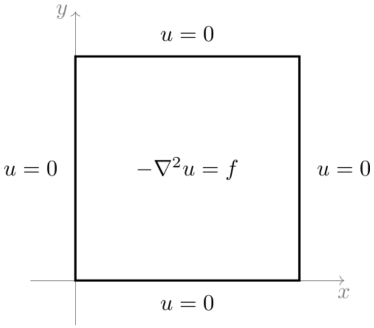
Outline

a linear PDE problem

- I will consider only two PDEs today
  1. Poisson problem
  2. minimal surface problem
- elliptic boundary-value problems
- recall Laplacian

$$\nabla^2 u = \nabla \cdot (\nabla u)$$

  - in 2D: $\nabla^2 u = u_{xx} + u_{yy}$
  - ∇^2 is a negative-definite operator



1. Poisson problem

for a given source function $f(x, y)$, find $u(x, y)$ so that

$$-\nabla^2 u = f \text{ on } \Omega, \qquad u\big|_{\partial\Omega} = 0$$

a nonlinear PDE problem

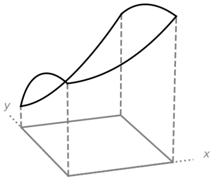
- the second problem is nonlinear, but still elliptic
- **claim 1.** the area of surface $z = v(x,y)$ over Ω is

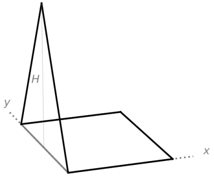
$$I[v] = \int_\Omega \sqrt{1 + |\nabla v|^2}\, dx\, dy$$

- **claim 2.** for given g, continuous along $\partial\Omega$,

$$u = \min_{\{v\,:\,v|_{\partial\Omega}=g\}} I[v]$$

solves the boundary value problem below





2. minimal surface problem

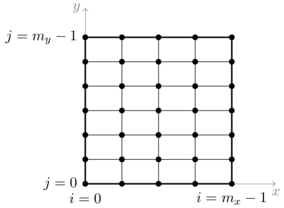
for given boundary function (wire frame) $g(x,y)$, find $u(x,y)$ so that

$$-\nabla \cdot \left(\frac{\nabla u}{\sqrt{1 + |\nabla u|^2}} \right) = 0 \text{ on } \Omega, \qquad u\big|_{\partial\Omega} = g$$

finite difference discretization

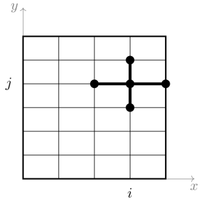
- PDEs are infinite-dimensional problems, but **discretization** yields finite systems of equations
- for example, finite differences (FD):

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2)$$



- assume: $\Omega = (0,1) \times (0,1)$
- grid spacings: $h_x = \frac{1}{m_x - 1}$, $h_y = \frac{1}{m_y - 1}$
- grid points: $(x_i, y_j) = (ih_x, jh_y)$ for $i = 0, \dots, m_x - 1$ and $j = 0, \dots, m_y - 1$
- 5-point stencil for the Laplacian:

$$\nabla^2 u(x_i, y_j) \approx \frac{u_{i-1,j} - 2u_{ij} + u_{i+1,j}}{h_x^2} + \frac{u_{i,j-1} - 2u_{ij} + u_{i,j+1}}{h_y^2}$$

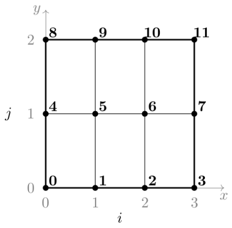


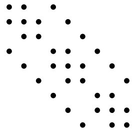
PDE $\implies$ linear system $A\mathbf{u} = \mathbf{b}$

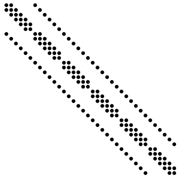
- FD discretize $-\nabla^2 u = f$:
  - note we keep the boundary values in the system
  - $N = m_x m_y$ total unknowns on the grid
    - $m_x = 4$, $m_y = 3$ shown at right
  - unknowns $\{u_{ij}\}$ are globally ordered: $u_\ell = u_{ij}$
- get a linear system for $\mathbf{u} \in \mathbb{R}^N$:

$$A\mathbf{u} = \mathbf{b}$$

  - $A \in \mathbb{R}^{N\times N}$ is **sparse**
  - A has positive diagonal
  - A is symmetric positive definite
  - A has $O(1)$ nonzeros per row $\leftarrow$ *at most 5, actually*
  - $\mathbf{b} \in \mathbb{R}^N$ has entries $b_\ell = f(x_i, y_j)$
- for $m_x = m_y = 5$: 9 non-trivial eqns (middle right)
- for $m_x = m_y = 8$: 36 non-trivial eqns (lower right)







goal: solve $A\mathbf{u} = \mathbf{b}$ in $O(N)$ work

- the linear system $A\mathbf{u} = \mathbf{b}$ can be set-up using $6N$ memory locations
  - A has 5 nonzero entries per row, plus one entry of $\mathbf{b}$
- can we solve $A\mathbf{u} = \mathbf{b}$ in $O(N)$ work and time as $N \to \infty$?
  - if so, the solver is *optimal*
- A is banded, with bandwidth $\approx \min\{m_x, m_y\} \approx \sqrt{N}$
  - so banded Gaussian elimination will solve in $O(N^2)$
- let's back up and ask:

  what operations using our A are obviously $O(N)$?

goal: solve $A\mathbf{u} = \mathbf{b}$ in $O(N)$ work

- the linear system $A\mathbf{u} = \mathbf{b}$ can be set-up using $6N$ memory locations
  - A has 5 nonzero entries per row, plus one entry of $\mathbf{b}$
- can we solve $A\mathbf{u} = \mathbf{b}$ in $O(N)$ work and time as $N \to \infty$?
  - if so, the solver is *optimal*

- A is banded, with bandwidth $\approx \min\{m_x, m_y\} \approx \sqrt{N}$
  - so banded Gaussian elimination will solve in $O(N^2)$

- let's back up and ask:

  what operations using our A are obviously $O(N)$?

goal: solve $A\mathbf{u} = \mathbf{b}$ in $O(N)$ work

- the linear system $A\mathbf{u} = \mathbf{b}$ can be set-up using $6N$ memory locations
  - A has 5 nonzero entries per row, plus one entry of $\mathbf{b}$
- can we solve $A\mathbf{u} = \mathbf{b}$ in $O(N)$ work and time as $N \to \infty$?
  - if so, the solver is *optimal*

- A is banded, with bandwidth $\approx \texttt{min}\{m_x, m_y\} \approx \sqrt{N}$
  - so banded Gaussian elimination will solve in $O(N^2)$

- let's back up and ask:

  what operations using our A are obviously $O(N)$?

cheap $O(N)$ operations: mat-vec and residual

- for A from PDE discretization, we can compute $A\mathbf{v}$ in $O(N)$ flops
  - this is because A has $O(1)$ nonzero entries per row
- also the residual is $O(N)$

Definition

given $\mathbf{v} \in \mathbb{R}^N$, the *residual* for the linear system $A\mathbf{u} = \mathbf{b}$ is

$$\mathbf{r}(\mathbf{v}) = \mathbf{b} - A\mathbf{v}$$

observe that:

$$A\mathbf{u} = \mathbf{b} \quad \iff \quad \mathbf{r}(\mathbf{u}) = 0$$

are simple iterations cheap $O(N)$ operations?

Definition

for the linear system $A\mathbf{u} = \mathbf{b}$ and an invertible matrix $M \in \mathbb{R}^{N\times N}$, *simple iteration* is

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \alpha\, M^{-1} \underbrace{(\mathbf{b} - A\mathbf{v}_k)}_{=\mathbf{r}(\mathbf{v}_k)}$$

- $\alpha \in \mathbb{R}$ is a tuning parameter, with default $\alpha = 1$
- if $M = A$ and $\alpha = 1$ then $\mathbf{v}_{k+1} = \mathbf{u}$, so one iteration solves it!
  - ...not practical, and usually not $O(N)$
- *Richardson iteration* $\quad \mathbf{v}_{k+1} = \mathbf{v}_k + \alpha\, \mathbf{r}(\mathbf{v}_k) \quad$ is the $M = I$ case
  - observation: Richardson iteration for the Poisson equation is *gradient descent* for the quadratic functional $\quad I[v] = \alpha \int_\Omega \frac{1}{2}|\nabla v|^2 - fv$
  - each Richardson iteration is clearly $O(N)$ work
- a simple iteration is $O(N)$ ... *if* M is the right kind of matrix!

preconditioned linear systems

Definition

if a matrix or linear map $M \in \mathbb{R}^{N\times N}$ is invertible, we say

$$M^{-1}A\mathbf{u} = M^{-1}\mathbf{b}$$

is a *(left-) preconditioned* system for $A\,\mathbf{u} = \mathbf{b}$

- the preconditioned system has the same solutions as before
- the new residual is $\tilde{\mathbf{r}}(\mathbf{v}) = M^{-1}\mathbf{b} - M^{-1}A\mathbf{v}$
- observation: Richardson iteration on the new system is equivalent to simple iteration

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \alpha\,\tilde{\mathbf{r}}(\mathbf{v}) \quad \iff \quad \mathbf{v}_{k+1} = \mathbf{v}_k + \alpha\,M^{-1}(\mathbf{b} - A\mathbf{v}_k)$$

fast preconditioners?

Definition

a matrix or linear map M is a *fast preconditioner* if solving $M\mathbf{z} = \mathbf{c}$ requires $O(N)$ work

- example: if A has nonzero diagonal entries, the diagonal $M = D$ is a fast preconditioner
- do *not* actually form the matrix M^{-1} when solving $M\mathbf{z} = \mathbf{c}$
- warning: a preconditioner M can be fast without being a useful tool!
- preconditioning is an apparently simple idea, but in the 21st century it is used all over the space of solvers

$O(N)$ simple iterations: Jacobi and Gauss-Seidel

- suppose we split A into diagonal and triangular parts:

$$A = D + L + U$$

- the linear system can be rearranged using the splitting:

$$\begin{aligned} A\mathbf{u} = \mathbf{b} \quad &\iff \quad D\mathbf{u} = \mathbf{b} - (L + U)\mathbf{u} \\ &\iff \quad \mathbf{u} = \mathbf{u} + D^{-1}(\mathbf{b} - A\mathbf{u}) \end{aligned}$$

- solving $D\mathbf{z} = \mathbf{c}$ is $O(N)$

Definition

a *Jacobi iteration* applies a fast preconditioner:

$$\mathbf{v}_{k+1} = \mathbf{v}_k + D^{-1}(\mathbf{b} - A\mathbf{v}_k)$$

$O(N)$ simple iterations: Jacobi and Gauss-Seidel

- suppose we split A into diagonal and triangular parts:

$$A = D + L + U$$

- the linear system can be rearranged using the splitting:

$$\begin{aligned} A\mathbf{u} = \mathbf{b} \quad &\iff \quad (D+L)\mathbf{u} = \mathbf{b} - U\mathbf{u} \\ &\iff \quad \mathbf{u} = \mathbf{u} + (D+L)^{-1}(\mathbf{b} - A\mathbf{u}) \end{aligned}$$

- solving $(D+L)\mathbf{z} = \mathbf{c}$ is $O(N)$ (for our A)

Definition

a *Gauss-Seidel (GS) iteration* applies a fast preconditioner:

$$\mathbf{v}_{k+1} = \mathbf{v}_k + (D+L)^{-1}(\mathbf{b} - A\mathbf{v}_k)$$

example of Gauss-Seidel iteration

- the matrix-splitting view obscures the simplicity of Gauss-Seidel?
- *example*: consider the linear system $A\mathbf{u} = \mathbf{b}$ with

$$A = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_N \end{bmatrix}$$

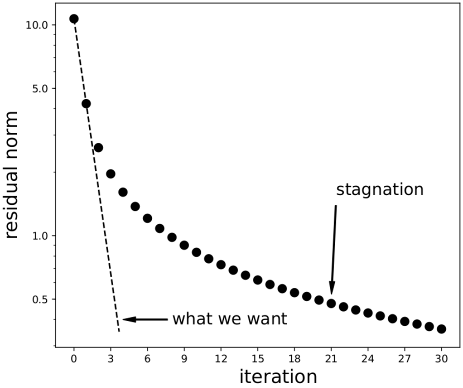
  - in this case, Gauss-Seidel iteration computes

$$v_j^{[k+1]} = \frac{b_j}{2} + \frac{1}{2}\left(v_{j-1}^{[k+1]} + v_{j+1}^{[k]}\right)$$

  - this is a *relaxation* method ... update v_j using average of neighbors
  - one can prove this method converges
  - this example is relevant because $A \sim -\nabla^2$ in 1D

Jacobi and Gauss-Seidel iterations *as solvers*

- for the Poisson problem linear system $A\mathbf{u} = \mathbf{b}$, one can prove that Gauss-Seidel and Jacobi converge
- but, after initial progress, residual norm decrease is *agonizingly slowly* on fine grids
- these simple iterations stagnate
- an iteration $\mathbf{v}_{k+1} = \phi(\mathbf{v}_k)$ *stagnates* or *stalls* if the ratio of successive residual norms $\{\|\mathbf{r}(\mathbf{v}_{k+1})\|/\|\mathbf{r}(\mathbf{v}_k)\|\}$ goes to one

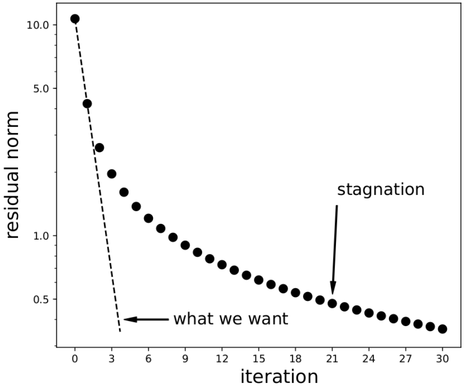


- Achi Brandt, an inventor of multigrid:

  Stalling numerical processes must be wrong. Whenever the computer grinds very hard for small or slow effect, there must be a better way to achieve the same goal.

Jacobi and Gauss-Seidel iterations *as solvers*

- for the Poisson problem linear system $A\mathbf{u} = \mathbf{b}$, one can prove that Gauss-Seidel and Jacobi converge
- but, after initial progress, residual norm decrease is *agonizingly slowly* on fine grids
- these simple iterations stagnate
- an iteration $\mathbf{v}_{k+1} = \phi(\mathbf{v}_k)$ *stagnates* or *stalls* if the ratio of successive residual norms $\{\|\mathbf{r}(\mathbf{v}_{k+1})\|/\|\mathbf{r}(\mathbf{v}_k)\|\}$ goes to one

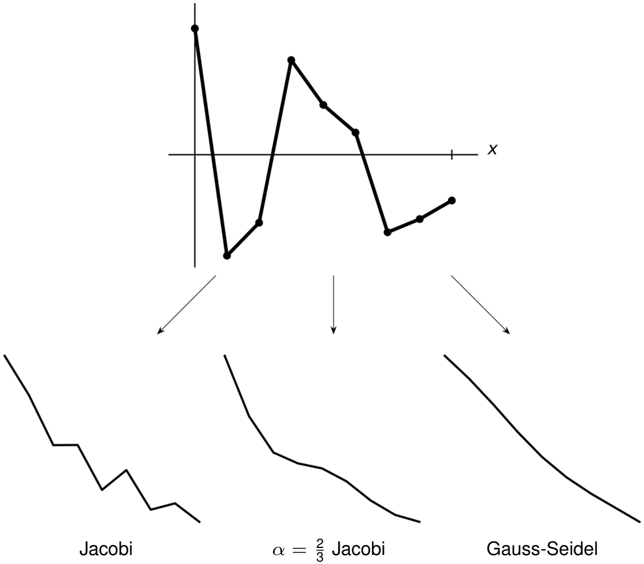


- Achi Brandt, an inventor of multigrid:

  Stalling numerical processes must be wrong. Whenever the computer grinds very hard for small or slow effect, there must be a better way to achieve the same goal.

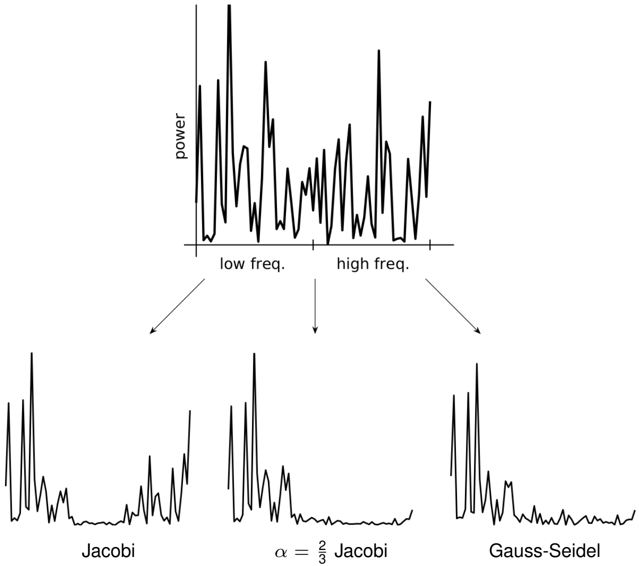
Jacobi and Gauss-Seidel iterations *as smoothers*

- observation: functions become *much smoother* after a few iterations
- the first multigrid paper (Federenko, 1961) observed this?



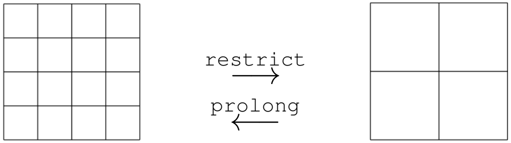
Jacobi and Gauss-Seidel iterations *as smoothers*

- observation: functions become *much smoother* after a few iterations
- the first multigrid paper (Federenko, 1961) observed this?



grid transfers

- multiple grid resolutions will allow us to exploit smoothers to generate fast solutions, but we need *grid transfer* operators between the different grids



- for a function **v** defined on a finer grid, define its *restriction R***v** to a coarser grid to be its value on the coarse grid points (*injection*), or restrict by averaging onto the coarser grid (*full weighting*)
- for a function **w** defined on the coarser grid, define its *prolongation P***w** to a finer grid by (linear) *interpolation*
- R, P are linear operators
  - R, P are rectangular (non-square) matrices
  - $RP = I$, or approximately so

2-grid method

- put these ideas together!
- the *2-grid method* approximately solves the PDE on the finer grid:

```
def twogrid(v,pre=2,post=2):
    for k in range(pre):
        v = smooth(A,b,v)
    rc = restrict(b - A*v)
    ec = solve(Ac,rc)           # solve error equation
    v = v + prolong(ec)
    for k in range(post):
        v = smooth(A,b,v)
    return v
```

- where

```
A,b  = discretize(m,m)        # fine grid
Ac,_ = discretize(m/2,m/2)    # coarse grid
```

- `smooth()` does one Jacobi or GS iteration
- `solve()` might be Gaussian elimination, etc., for $A^c \mathbf{e}^c = \mathbf{r}^c$

the error equation

- what do I mean by the *error equation*?
- for the linear system $A\mathbf{u} = \mathbf{b}$, consider some $\mathbf{v}$ which is *not* a solution

Definition

for any vector $\mathbf{v}$, the *error equation* corresponding to the linear system $A\mathbf{u} = \mathbf{b}$ is the equation

$$A\mathbf{e} = \mathbf{r}(\mathbf{v})$$

- here's the logic:

$$\begin{aligned}
\mathbf{e} &= \mathbf{u} - \mathbf{v} && \textit{definition of the error} \\
A\mathbf{e} &= A\mathbf{u} - A\mathbf{v} && \textit{multiply by } A \\
A\mathbf{e} &= \mathbf{b} - A\mathbf{v} = \mathbf{r}(\mathbf{v}) && \textit{error equation}
\end{aligned}$$

the coarse grid correction

- the essential 3 lines in `twogrid()` form a *coarse-grid correction*:

```
rc = restrict(b - A*v)       # restrict the residual
ec = solve(Ac,rc)            # coarse-grid solve
v = v + prolong(ec)          # add back as correction
```

- this is a kind of simple iteration: $\mathbf{v} \leftarrow \mathbf{v} + P(A^c)^{-1} R(\mathbf{b} - A\mathbf{v})$
- define the *coarse-grid correction matrix*:

$$B^c = P(A^c)^{-1} R$$

- so `twogrid()` mixes two flavors of simple iteration:

$$\mathbf{v} \leftarrow \mathbf{v} + M^{-1}(\mathbf{b} - A\mathbf{v}) \qquad \textit{the smoother}$$
$$\mathbf{v} \leftarrow \mathbf{v} + B^c(\mathbf{b} - A\mathbf{v}) \qquad \textit{the coarse-grid correction}$$

- Q: is $B^c \approx A^{-1}$?

the coarse grid correction

- the essential 3 lines in `twogrid()` form a *coarse-grid correction*:

```
rc = restrict(b - A*v)      # restrict the residual
ec = solve(Ac,rc)           # coarse-grid solve
v = v + prolong(ec)         # add back as correction
```

- this is a kind of simple iteration: $\mathbf{v} \leftarrow \mathbf{v} + P(A^c)^{-1}R(\mathbf{b} - A\mathbf{v})$
- define the *coarse-grid correction matrix*:

$$B^c = P(A^c)^{-1}R$$

- so `twogrid()` mixes two flavors of simple iteration:

$$\mathbf{v} \leftarrow \mathbf{v} + M^{-1}(\mathbf{b} - A\mathbf{v}) \qquad \textit{the smoother}$$
$$\mathbf{v} \leftarrow \mathbf{v} + B^c(\mathbf{b} - A\mathbf{v}) \qquad \textit{the coarse-grid correction}$$

- Q: is $B^c \approx A^{-1}$? A: yes, but only for smooth inputs

2-grid method: the effect on error

- look at `twogrid()` again:

```
def twogrid(v,pre=2,post=2):
    for k in range(pre):
        v = smooth(A,b,v)
    rc = restrict(b - A*v)
    ec = solve(Ac,rc)
    v = v + prolong(ec)
    for k in range(post):
        v = smooth(A,b,v)
    return v
```

- a fairly-quick calculation shows that `twogrid()` applies a linear operator, which is close to the zero operator, to $\mathbf{e} = \mathbf{u} - \mathbf{v}$:

$$\mathbf{e} \leftarrow (I - M^{-1}A)^{\text{post}}(I - B^c A)(I - M^{-1}A)^{\text{pre}}\mathbf{e}$$

- Q: how should we solve the coarse-grid problem $A^c\mathbf{e}^c = \mathbf{r}^c$?

2-grid method: the effect on error

- look at `twogrid()` again:

```
def twogrid(v,pre=2,post=2):
    for k in range(pre):
        v = smooth(A,b,v)
    rc = restrict(b - A*v)
    ec = solve(Ac,rc)
    v = v + prolong(ec)
    for k in range(post):
        v = smooth(A,b,v)
    return v
```

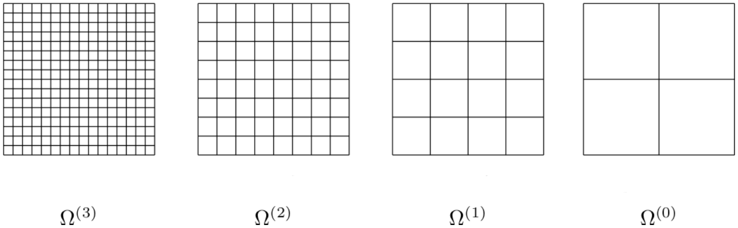
- a fairly-quick calculation shows that `twogrid()` applies a linear operator, which is close to the zero operator, to $\mathbf{e} = \mathbf{u} - \mathbf{v}$:

$$\mathbf{e} \leftarrow (I - M^{-1}A)^{\texttt{post}}(I - B^c A)(I - M^{-1}A)^{\texttt{pre}}\mathbf{e}$$

- Q: how should we solve the coarse-grid problem $A^c\mathbf{e}^c = \mathbf{r}^c$?

hierarchy of grids

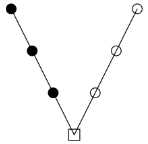
- A: if we can have two levels of grids, we can have many!
- when faced with a coarse-grid solve, just do another 2-grid . . . and keep going down to some really easy and cheap coarse grid

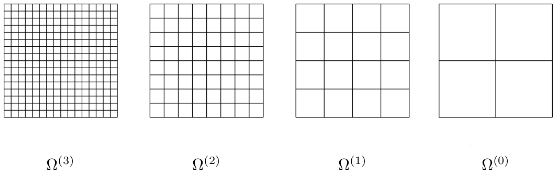


- restrictions *R* and prolongations *P* are needed in this *grid hierarchy*

recursive V-cycle

```
def vcycle(b,v,lev,pre=2,post=2):
    A,_ = discretize(lev)
    if lev == 0:
        return solve(A,b)     # the buck stops here
    for k in range(pre):
         v = smooth(A,b,v)
    rc = restrict(b - A*v)
    ec = vcycle(r,0,lev-1)    # descend a grid level
    v = v + prolong(ec)
    for k in range(post):
         v = smooth(A,b,v)
    return v
```

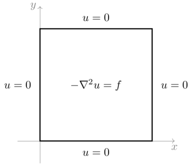


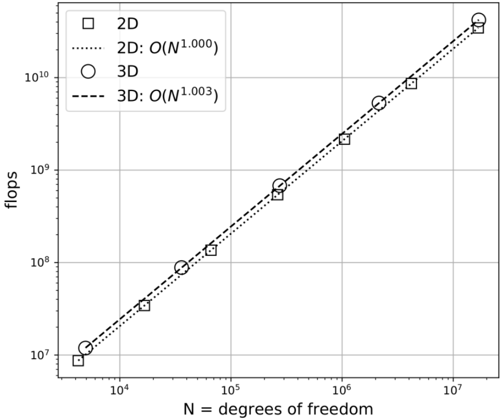


how well does it work?

- so, how well does it work on our Poisson problem $-\nabla^2 u = f$?

- absurdly well!
- here is scaling out to $m = 4097$, when $N = 1.6 \times 10^7$

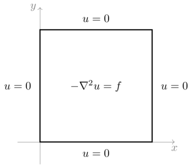


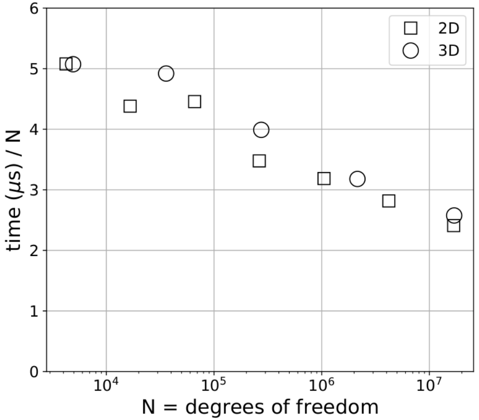


how well does it work?

- so, how well does it work on our Poisson problem $-\nabla^2 u = f$?

- absurdly well!
- here is scaling out to $m = 4097$, when $N = 1.6 \times 10^7$





on multigrid costs: single V-cycle

- let us analyze the work (flops) of applying a single V-cycle
  - note: multiple V-cycles are generally needed to solve the problem

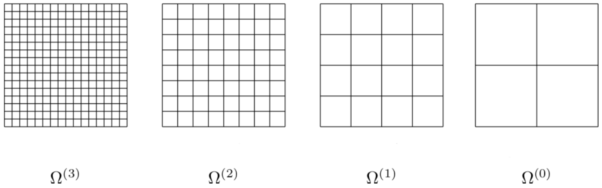
Definitions

$$|\Omega^{(k)}| = \text{(number of grid points (unknowns) on grid } \Omega^{(k)}\text{)}$$

$$W_k^{k-1} = \left(\begin{array}{c}\text{smoother work done on grid } \Omega^{(k)}\text{, plus cost of} \\ \text{restriction/prolongation to/from next-coarser grid } \Omega^{(k-1)}\end{array}\right)$$

$$W_0 = \text{(solver work done on the coarsest level)}$$

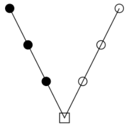
- $K = 3$ case:



on multigrid costs: single V-cycle

- total cost of a single V-cycle:

$$\overline{W} = W_K^{K-1} + W_{K-1}^{K-2} + \dots + W_1^0 + W_0$$



- for 2D grids, each coarse grid is 4 times smaller:

$$|\Omega^{(k-1)}| \approx \tfrac{1}{4}|\Omega^{(k)}|$$

- since smoothers and restriction/prolongation are $O(1)$ per grid point:

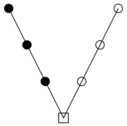
$$W_k^{k-1} \le C|\Omega^{(k)}|$$

  - for some C independent of k
- since $N = |\Omega^{(K)}|$ is the number of points in the finest grid,

$$\begin{aligned}\overline{W} &\le C|\Omega^{(K)}| + C|\Omega^{(K-1)}| + \dots + C|\Omega^{(1)}| + W_0 \\ &\approx CN\left(1 + \tfrac{1}{4} + \dots + \tfrac{1}{4^{K-1}}\right) + W_0 \\ &\approx CN\frac{1}{1-(1/4)} = \tfrac{4}{3}CN\end{aligned}$$

optimal

on multigrid costs: single V-cycle



- total cost of a single V-cycle:

$$\overline{W} = W_K^{K-1} + W_{K-1}^{K-2} + \dots + W_1^0 + W_0$$

- for 2D grids, each coarse grid is 4 times smaller:

$$|\Omega^{(k-1)}| \approx \tfrac{1}{4}|\Omega^{(k)}|$$

- since smoothers and restriction/prolongation are $O(1)$ per grid point:

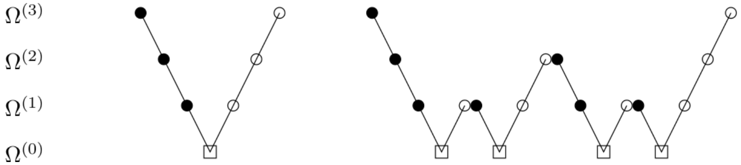
$$W_k^{k-1} \le C|\Omega^{(k)}|$$

  - for some C independent of k
- since $N = |\Omega^{(K)}|$ is the number of points in the finest grid,

$$\begin{aligned}\overline{W} &\le C|\Omega^{(K)}| + C|\Omega^{(K-1)}| + \dots + C|\Omega^{(1)}| + W_0 \\ &\approx CN\left(1 + \tfrac{1}{4} + \dots + \tfrac{1}{4^{K-1}}\right) + W_0 \\ &\approx CN\tfrac{1}{1-(1/4)} = \tfrac{4}{3}CN \qquad \text{optimal}\end{aligned}$$

multigrid variations

- there are many variations on linear multigrid:
  - choose different smoothers (● is pre-smoother, ○ is post-smoother)
  - choose different values for `pre` and `post` smoother iterations
  - choose different coarse-grid solvers (□)
  - repeat the coarse-grid correction a couple of times (*W cycles*)



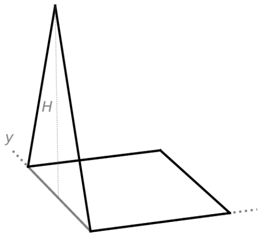
summary so far

- multigrid combines three conceptual threads:
  1. a few classical iterations, such as Jacobi and GS, are **cheap smoothers** of the residual and the error
  2. a **coarse-grid correction** does a good job of updating the fine-grid solution when acting on a smooth residual
  3. the coarse-grid correction is cheap because **restriction and prolongation are cheap**

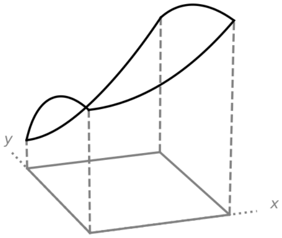
- but the Poisson problem is too easy!

minimal surfaces, a nonlinear problem

- recall . . .



tent



catenoid

minimal surface problem

for given boundary function (wire frame) $g(x, y)$, find $u(x, y)$ so that

$$-\nabla \cdot \left(\frac{\nabla u}{\sqrt{1 + |\nabla u|^2}} \right) = 0 \text{ on } \Omega, \qquad u\big|_{\partial\Omega} = g$$

discretization gets you . . . more general equations

- at each point (x_i, y_j) on the target (finest) grid $\Omega^{(K)}$ we discretize to get an FD approximation of the PDE:

$$f_{ij}(u_{i-1,j},\, u_{i+1,j},\, u_{i,j-1},\, u_{i,j+1},\, u_{i,j}) = 0$$

  - roughly-speaking, anyway ... see details next slide
  - unknowns must be globally-ordered into a vector $\mathbf{u} \in \mathbb{R}^N$:

$$u_\ell = u_{i,j}$$

    where $\ell = \ell(i,j)$ is a global-to-local indexing function

nonlinear discretization principle

enforcing the PDE at grid point (x_i, y_j) gives one scalar equation $f_{ij}(\mathbf{u}) = 0$

- also globally-order the equations (functions), $f_\ell(\mathbf{u}) = f_{ij}(\mathbf{u})$, to get a nonlinear system of N scalar equations in N scalar unknowns:

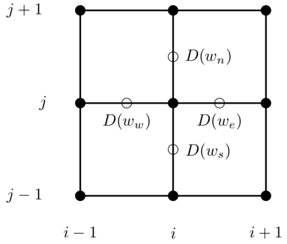
$$\mathbf{F}(\mathbf{u}) = \mathbf{0}$$

- $\mathbf{F}$ is a sparse function, as each f_ℓ depends on only $O(1)$ entries of $\mathbf{u}$

details: 9-point stencil with staggered diffusivity

- how do you discretize $\nabla \cdot \left(\frac{\nabla u}{\sqrt{1+|\nabla u|^2}}\right)$?
- technique: generalize first!
- discretize $\nabla \cdot (D(w)\nabla u)$ where

$$D(w) = (1+w)^{-1/2}$$
$$w = |\nabla u|^2$$



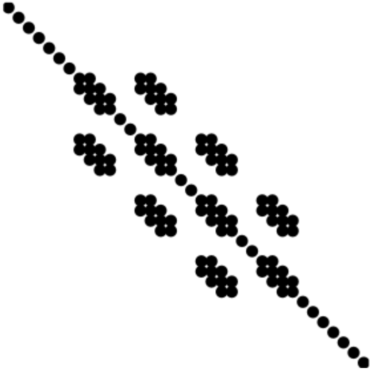
- centered FD, using *staggered* values of $D(w)$, gets $O(h_x^2+h_y^2)$ truncation error and symmetry:

$$\nabla\cdot(D(w)\nabla u) \approx \frac{D(w_e)(u_{i+1,j}-u_{i,j}) - D(w_w)(u_{i,j}-u_{i-1,j})}{h_x^2} + \frac{D(w_n)(u_{i,j+1}-u_{i,j}) - D(w_s)(u_{i,j}-u_{i,j-1})}{h_y^2}$$

$$w_e = \left[|\nabla u|^2\right]_{i+\frac{1}{2},j} \approx \left(\frac{u_{i+1,j}-u_{i,j}}{h_x}\right)^2 + \left(\frac{u_{i,j+1}+u_{i+1,j+1}-u_{i,j}-u_{i+1,j}}{4h_y}\right)^2$$

9-point stencil Jacobian: sparsity

- on a 6 $\times$ 6 grid, the Jacobian $J(\mathbf{v})$ has this sparsity pattern:



Newton's method

- Q: how do we solve our nonlinear system $\mathbf{F}(\mathbf{u}) = \mathbf{0}$?
  A: Newton's method!:

$$\begin{aligned} J(\mathbf{u}^p)\,\mathbf{s} &= -\mathbf{F}(\mathbf{u}^p) \\ \mathbf{u}^{p+1} &= \mathbf{u}^p + \mathbf{s} \end{aligned}$$

  where $J(\mathbf{v})$ is the Jacobian

$$J(\mathbf{v})_{r,s} = \left[\frac{\partial f_r(\mathbf{v})}{\partial v_s} \right]$$

- Q: how do you calculate the Jacobian? (*cause it's a pain in the . . .*)
  A: more finite differencing:

$$J(\mathbf{v})_{r,s} \approx \frac{f_r(\mathbf{v} + \epsilon \mathbf{1}_s) - f_r(\mathbf{v})}{\epsilon}$$

  A$'$: symbolically?

Newton's method with finite-differenced Jacobian

- Q: how do we solve our nonlinear system $\mathbf{F}(\mathbf{u}) = \mathbf{0}$?
  A: Newton's method!:

$$\begin{aligned} J(\mathbf{u}^p)\,\mathbf{s} &= -\mathbf{F}(\mathbf{u}^p) \\ \mathbf{u}^{p+1} &= \mathbf{u}^p + \mathbf{s} \end{aligned}$$

  where $J(\mathbf{v})$ is the Jacobian

$$J(\mathbf{v})_{r,s} = \left[\frac{\partial f_r(\mathbf{v})}{\partial v_s} \right]$$

- Q: how do you calculate the Jacobian? (*cause it's a pain in the ...*)
  A: more finite differencing:

$$J(\mathbf{v})_{r,s} \approx \frac{f_r(\mathbf{v} + \epsilon \mathbf{1}_s) - f_r(\mathbf{v})}{\epsilon}$$

  A′: symbolically?

Newton's method with finite-differenced Jacobian

- Q: how do we solve our nonlinear system $\mathbf{F}(\mathbf{u}) = \mathbf{0}$?
  A: Newton's method!:

$$J(\mathbf{u}^p)\,\mathbf{s} = -\mathbf{F}(\mathbf{u}^p)$$
$$\mathbf{u}^{p+1} = \mathbf{u}^p + \mathbf{s}$$

  where $J(\mathbf{v})$ is the Jacobian

$$J(\mathbf{v})_{r,s} = \left[\frac{\partial f_r(\mathbf{v})}{\partial v_s} \right]$$

- Q: how do you calculate the Jacobian? (*cause it's a pain in the ...*)
  A: more finite differencing:

$$J(\mathbf{v})_{r,s} \approx \frac{f_r(\mathbf{v} + \epsilon \mathbf{1}_s) - f_r(\mathbf{v})}{\epsilon}$$

  A$'$: symbolically?

- Q: how do you *efficiently* calculate the FD Jacobian? A: graph coloring

Newton-multigrid

- Q: how do you solve each linear system in the Newton iteration?

  A: solve $J(\mathbf{u}^p)\,\mathbf{s} = -\mathbf{F}(\mathbf{u}^p)$ using multigrid:

```
def newtonmultigrid(v,lev,maxnewts=50,cycles=1):
    for p in range(maxnewts):
        b = -F(v)
        s = 0
        for _ in range(cycles):
            s = vcycle(b,s,lev)
        v = v + s
    return v
```

- details:
  - inside `vcycle()`, the matrix $A = A^{(k)}$ on each grid level $\Omega^{(k)}$ is computed using the Jacobian on that grid level (*rediscretization*)
  - the finest-level iterate $\mathbf{u}^p$ must be restricted (injected) down to $\Omega^{(k)}$:

$$A^{(k)} = J^{(k)}(R^{K-k}\mathbf{u}^p)$$

  - $J^{(k)}$ is approximated using FD and graph coloring on $\Omega^{(k)}$

Newton-multigrid

- Q: how do you solve each linear system in the Newton iteration?

  A: solve $J(\mathbf{u}^p)\,\mathbf{s} = -\mathbf{F}(\mathbf{u}^p)$ using multigrid:

```
def newtonmultigrid(v,lev,maxnewts=50,cycles=1):
    for p in range(maxnewts):
        b = -F(v)
        s = 0
        for _ in range(cycles):
            s = vcycle(b,s,lev)
        v = v + s
    return v
```

- details:
  - inside `vcycle()`, the matrix $A = A^{(k)}$ on each grid level $\Omega^{(k)}$ is computed using the Jacobian on that grid level (*rediscretization*)
  - the finest-level iterate $\mathbf{u}^p$ must be restricted (injected) down to $\Omega^{(k)}$:

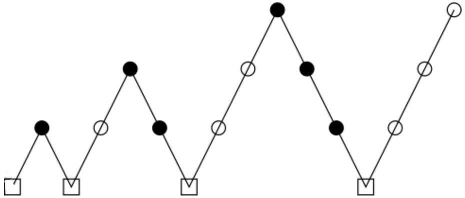
$$A^{(k)} = J^{(k)}(R^{K-k}\mathbf{u}^p)$$

  - $J^{(k)}$ is approximated using FD and graph coloring on $\Omega^{(k)}$

nonlinear multigrid F-cycle solvers

- but wait, ... there's more!
- Q: how do you find a good initial iterate $\mathbf{u}^0$ for the Newton iteration?

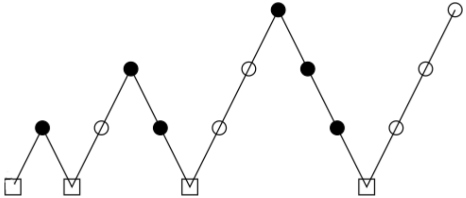
  A: by solving the problem on a coarser grid, and prolonging
- justification: the domain of Newton convergence is larger on the smaller (= coarser) version of the PDE
- this strategy is called *nested iteration* or *grid sequencing*
- if you also solve at each level with Newton-multigrid, then this is a *nonlinear multigrid F-cycle* ... the most powerful solver you've seen!



nonlinear multigrid F-cycle solvers

- but wait, . . . there's more!
- Q: how do you find a good initial iterate $\mathbf{u}^0$ for the Newton iteration?

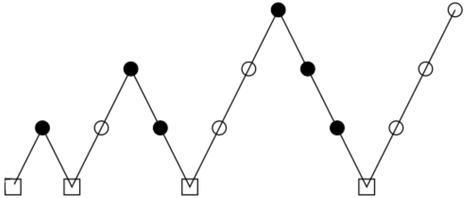
  A: by solving the problem on a coarser grid, and prolonging
- justification: the domain of Newton convergence is larger on the smaller (= coarser) version of the PDE
- this strategy is called *nested iteration* or *grid sequencing*
- if you also solve at each level with Newton-multigrid, then this is a *nonlinear multigrid F-cycle* . . . the most powerful solver you've seen!



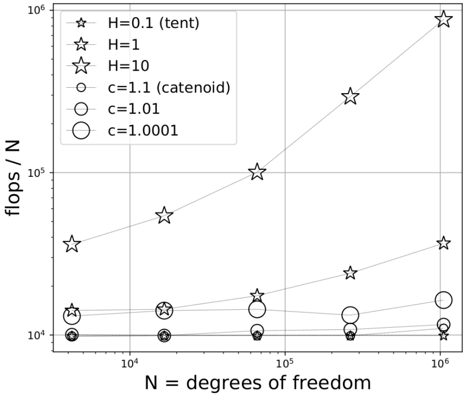
nonlinear multigrid F-cycle solvers

- but wait, . . . there's more!
- Q: how do you find a good initial iterate $\mathbf{u}^0$ for the Newton iteration?

  A: by solving the problem on a coarser grid, and prolonging
- justification: the domain of Newton convergence is larger on the smaller ($=$ coarser) version of the PDE
- this strategy is called *nested iteration* or *grid sequencing*
- if you also solve at each level with Newton-multigrid, then this is a *nonlinear multigrid F-cycle* . . . the most powerful solver you've seen!



minimal surface PDE problem: results

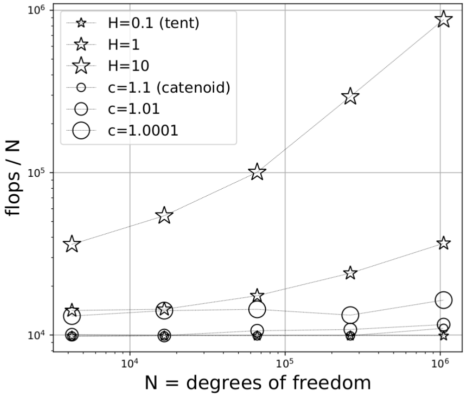


- run-time demo:

```
$ cd p4pdes/c/ch7/
$ make minimal
$ mpiexec -n 6 ./minimal -snes_fd_color -pc_type mg \
    -{snes,ksp}_converged_reason -snes_grid_sequence 10
```

-snes_monitor_solution draw -mg_levels_{snes,ksp}_converged_reason

minimal surface PDE problem: results



- run-time demo:

```
$ cd p4pdes/c/ch7/
$ make minimal
$ mpiexec -n 6 ./minimal -snes_fd_color -pc_type mg \
  -{snes,ksp}_converged_reason -snes_grid_sequence 10
```

```
-snes_monitor_solution draw -mg_levels_{snes,ksp}_converged_reason
```

summary

- a multigrid V-cycle combines three conceptual threads to build an optimal solver for linear elliptic PDEs:
  1. classical iterations $=$ cheap smoothers
  2. coarse-grid correction effective, if starting from a smooth residual
  3. restriction and prolongation are cheap
- there is also **algebraic multigrid**, but that is a different talk . . .
- for nonlinear elliptic PDEs:
  1. wrap a Newton iteration around multigrid V-cycles: **Newton-multigrid**
  2. **grid sequencing** generates a high-quality finest-grid initial iterate
  3. thus: a **nonlinear multigrid F-cycle solver**
- Newton-multigrid is not the only nonlinear option . . . there is also **full approximation scheme** multigrid, but that is a different talk . . .

summary

- a multigrid V-cycle combines three conceptual threads to build an optimal solver for linear elliptic PDEs:
    1. classical iterations $=$ cheap smoothers
    2. coarse-grid correction effective, if starting from a smooth residual
    3. restriction and prolongation are cheap
- there is also **algebraic multigrid**, but that is a different talk . . .
- for nonlinear elliptic PDEs:
    1. wrap a Newton iteration around multigrid V-cycles: **Newton-multigrid**
    2. **grid sequencing** generates a high-quality finest-grid initial iterate
    3. thus: a **nonlinear multigrid F-cycle solver**
- Newton-multigrid is not the only nonlinear option . . . there is also **full approximation scheme** multigrid, but that is a different talk . . .

references

A. Brandt (1977). *Multi-level adaptive solutions to boundary-value problems*, Mathematics of Computation 31 (138), 333–390

- the guru of multigrid

W. Briggs, V. E. Henson, & S. McCormick (2000). *A Multigrid Tutorial*, 2nd ed., SIAM Press, Philadelphia

- straightforward introduction

E. Bueler (2021). *PETSc for Partial Differential Equations*, SIAM Press, Philadelphia

- preconditioning ideas and diverse multigrid examples

H. Elman, D. Silvester, & A. Wathen (2014). *Finite Elements and Fast Iterative Solvers: With Applications to Incompressible Fluid Dynamics*, 2nd ed., Oxford U. Press

- multigrid in FE and fluids contexts

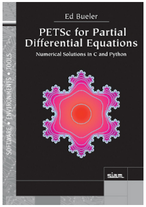
R. Fedorenko (1961). *A relaxation method for solving elliptic difference equations*, USSR Comput. Math. Math. Phys. 1, 922–927

- the first multigrid paper

U. Trottenberg, C. Oosterlee, & A. Schuller (2001). *Multigrid*, Elsevier

- comprehensive view, and theory

Achi Brandt


Ed Bueler
PETSc for Partial Differential Equations
Numerical Solutions in C and Python
SOFTWARE • ENVIRONMENTS • TOOLS
siam