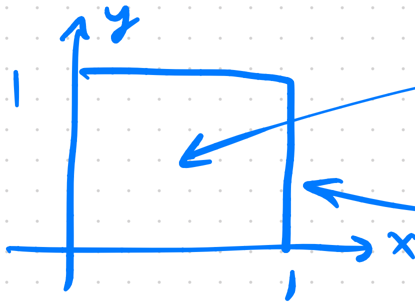
fast solvers for the Poisson equation

7 March 2024

- recall the most basic problem



$$\left.\begin{aligned} -\nabla^2 u &= f \\ u &= 0 \end{aligned}\right\}$$ Poisson equation strong form

- if we want an accurate solution then we will use a fine grid ...

Q1. what does it mean for the solver to be fast?

Q2. how do you get Firedrake to use a fast solver?

major stages of PDE FE solving:

you do

① strong form statement:

$$-\nabla^2 u = f$$

② weak form, into code:

$$\int_\Omega \nabla u \cdot \nabla v - \int_\Omega f v = 0 \rightarrow F = (\mathrm{dot}(\mathrm{grad}(u), \mathrm{grad}(v)) - \ldots) * dx$$

Firedrake

③ "F = 0" assembled into linear system:

$$Au = b$$

④ linear system is solved:

$$u = [\text{run code on } A, b]$$

you control this process through options

Paraview

⑤ visualize/analyze results

inside solve(F == 0, u, ...)

- u holds (unknown) nodal values
- Firedrake assembles a linear system

$$Au = b$$ } done <u>once</u> for the linear Poisson equation

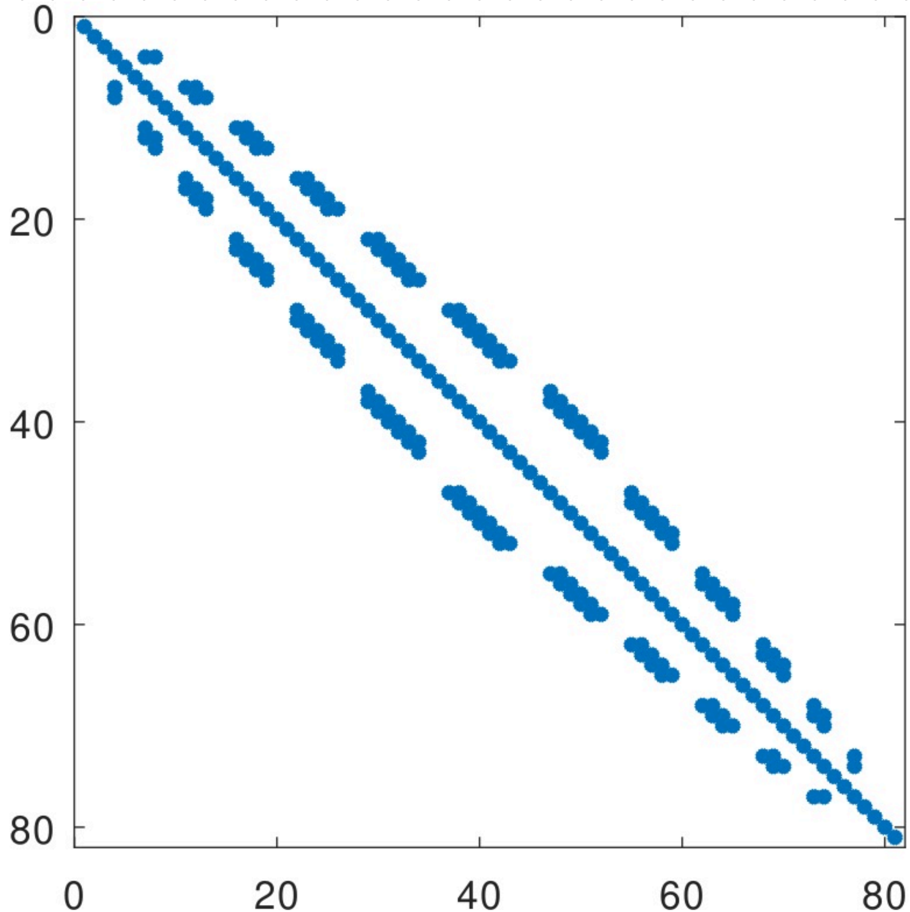
- A is $\underbrace{N \times N}$, $\underbrace{\text{symmetric}}$, and $\underbrace{\text{invertible}}$

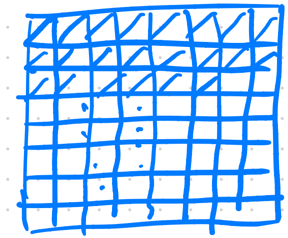
  $N = (m+1)^2$ for UnitSquareMesh(m, m)

  for Poisson at least

  because b.c.s are chosen correctly

- A is sparse ⟵ because FunctionSpace(mesh, 'CG', k) for mesh = UnitSquareMesh(8,8)





$m=8$

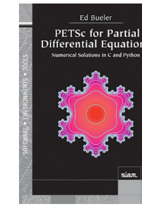
$N=(m+1)^2$

$=81$

$\therefore$ A is 81×81

solver possibilities for $Au=b$:

not a complete list!

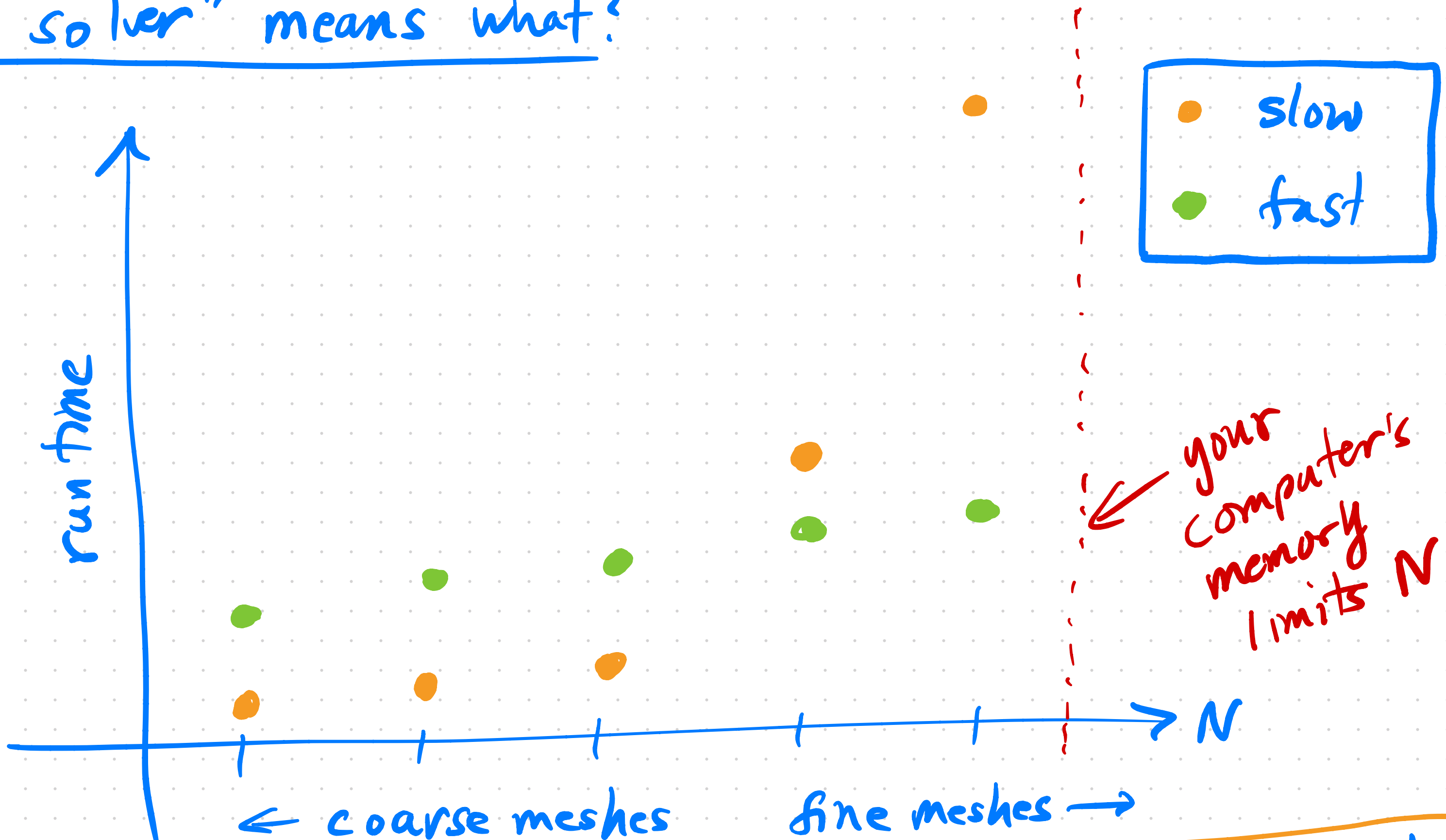


- LU decomposition = Gaussian elimination
- conjugate gradient (CG) iteration

suitable for Poisson

- preconditioned CG using an incomplete-factorization preconditioner
- preconditioned CG using a geometric multigrid preconditioner

"fast solver" means what?



what matters in practice is rate of growth!

test-case solver:

- solves Poisson on unit square
- user specifies mesh and solver parameters
- measures solve(...) time in seconds

```python
def solvecase(m,mesh,p):
    H = FunctionSpace(mesh,'CG',1)
    u = Function(H)
    v = TestFunction(H)

    x, y = SpatialCoordinate(mesh)
    dsqr = (x - 0.8)**2 + (y - 0.3)**2
    fsource = Function(H).interpolate(exp(-10.0*dsqr))

    F = ( dot(grad(u), grad(v)) - fsource * v ) * dx
    BCs = DirichletBC(H, Constant(0.0), (1, 2, 3, 4))

    t0 = time.time()
    solve(F == 0, u, bcs=[BCs], solver_parameters = p)
    t1 = time.time()

    dura = t1 - t0
    N = (m+1)**2
    print(f'  m = {m:5d},  N = {N:6.1e}:  {dura:7.2f} s; {1e6*dura/N:6.2f} mu s / N')
```

demo:

- run & read *.py
- at right: on my 128Gb desktop

```
(firedrake) ~/repos/fe-seminar/py/7mar[main*]$ python3 slowfish.py; python3 medfish.py; python3 fast
fish.py
```

slowfish.py

```
solve time for m x m meshes with N dofs:
  m =    64,  N = 4.2e+03:     0.14 s;  33.59 mu s / N
  m =   128,  N = 1.7e+04:     0.12 s;   6.93 mu s / N
  m =   256,  N = 6.6e+04:     0.64 s;   9.71 mu s / N
  m =   512,  N = 2.6e+05:     4.30 s;  16.34 mu s / N
  m =  1024,  N = 1.1e+06:    32.52 s;  30.96 mu s / N
```

medfish.py

```
solve time for m x m meshes with N dofs:
  Linear firedrake_0_ solve converged due to CONVERGED_RTOL iterations 46
  m =    64,  N = 4.2e+03:     0.12 s;  28.99 mu s / N
  Linear firedrake_1_ solve converged due to CONVERGED_RTOL iterations 91
  m =   128,  N = 1.7e+04:     0.09 s;   5.59 mu s / N
  Linear firedrake_2_ solve converged due to CONVERGED_RTOL iterations 179
  m =   256,  N = 6.6e+04:     0.47 s;   7.17 mu s / N
  Linear firedrake_3_ solve converged due to CONVERGED_RTOL iterations 359
  m =   512,  N = 2.6e+05:     3.21 s;  12.22 mu s / N
  Linear firedrake_4_ solve converged due to CONVERGED_RTOL iterations 720
  m =  1024,  N = 1.1e+06:    24.13 s;  22.96 mu s / N
```

fastfish.py

```
solve time for m x m meshes with N dofs:
  Linear firedrake_0_ solve converged due to CONVERGED_RTOL iterations 4
  m =    64,  N = 4.2e+03:     2.09 s; 493.87 mu s / N
  Linear firedrake_1_ solve converged due to CONVERGED_RTOL iterations 4
  m =   128,  N = 1.7e+04:     0.42 s;  25.35 mu s / N
  Linear firedrake_2_ solve converged due to CONVERGED_RTOL iterations 4
  m =   256,  N = 6.6e+04:     1.01 s;  15.27 mu s / N
  Linear firedrake_3_ solve converged due to CONVERGED_RTOL iterations 4
  m =   512,  N = 2.6e+05:     3.21 s;  12.19 mu s / N
  Linear firedrake_4_ solve converged due to CONVERGED_RTOL iterations 4
  m =  1024,  N = 1.1e+06:    12.70 s;  12.09 mu s / N
  Linear firedrake_5_ solve converged due to CONVERGED_RTOL iterations 4
  m =  2048,  N = 4.2e+06:    50.53 s;  12.04 mu s / N
  Linear firedrake_6_ solve converged due to CONVERGED_RTOL iterations 4
  m =  4096,  N = 1.7e+07:   210.73 s;  12.55 mu s / N
```

lecture: Krylov space methods

$Au=b$ ← too expensive by LU or other "direct" solver

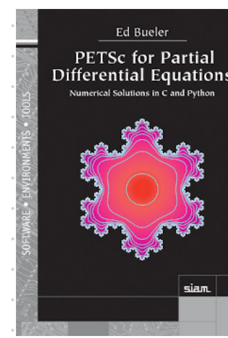
• try

$$u \overset{\circledast}{\approx} c_0 b + c_1 Ab + c_2 A^2 b + \dots + A^{n-1} b$$

(why? because "Aw" is cheap if A is sparse)

A. Krylov

Chapter 2 ↓



def: $K_n(A,b) = \mathrm{span}\{b, Ab, \dots, A^{n-1}b\}$

• conjugate gradients finds c_j so that error in $\circledast$ is minimized in $\|w\|_A = \sqrt{\langle Aw, Aw\rangle}$ norm

u lecture: preconditioning

for $Au = b$,

def: if M is an invertible linear map then

$$(M^{-1}A)u = M^{-1}b$$

is the <u>left-preconditioned</u> <u>system</u> and

$$(AM^{-1})(Mu) = b$$

is the <u>right-preconditioned system</u>

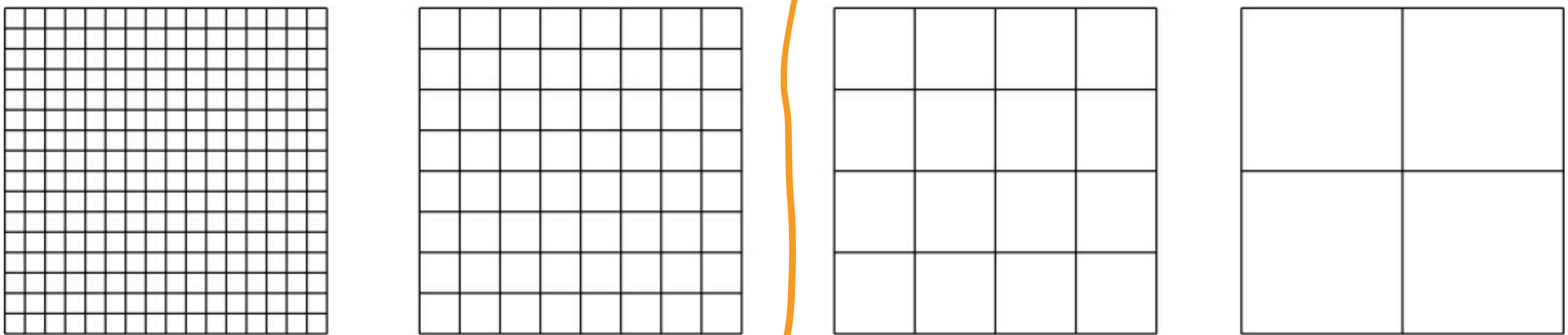
M is only <u>useful</u> if
① M^{-1} fast
② $M^{-1}A$, AM^{-1} has better spectrum

- <u>idea</u>: $M^{-1}A$ or AM^{-1} can have better-behaved Krylov space $K_n(M^{-1}A, M^{-1}b)$ or $K_n(AM^{-1}, b)$

μ lecture: multigrid

`MeshHierarchy(mesh, k)`

in Firedrake



$\Omega^{(3)}$ $\Omega^{(2)}$ $\Omega^{(1)}$ $\Omega^{(0)}$

idea: ① build finest mesh via hierarchy

② multigrid solvers fix low-frequency errors over $\Omega^{(0)}$

A. Brandt

chapter 6 →