One- and Two-Dimensional Poisson's Equation

Pierce Hunter, Nick Kuckuck, Haoran Wang

CIS 410/510 Final Presentation

8th June 2020

Outline

1. Theory
   - Poisson in 1D
   - Poisson in 2D
   - Equations
   - Discretizations
   - Matrix formulations
2. Code samples
   - A matrices and array set-up
   - The four tests we run
   - How we check for convergence
3. Results
   - One dimensional with convergence test
   - Two dimensional with error analysis

One-dimensional model

We first solved Poisson's equation in 1D

$$\frac{\partial^2 u}{\partial z^2} = -1; \quad 0 \leq z \leq 1$$

with boundary conditions

$$\frac{\partial u}{\partial z}(1) = 0; \quad u(0) = 0$$

in the four following ways (all using built-in functions):

- Direct solve on the CPU
- CG on the CPU
- Direct solve on the GPU
- CG on the GPU

1D Discretization

Use centered difference for z

$$\frac{u_{j-1} - 2u_j + u_{j+1}}{\Delta z^2} = -1 \quad \text{for} \quad 1 \leq j \leq N$$

Apply boundary conditions giving the system of equations

$$\begin{cases} \dfrac{-2u_2 + u_3}{\Delta z^2} = -1 \\ \dfrac{u_{j-1} - 2u_j + u_{j+1}}{\Delta z^2} = -1; \quad 3 \leq j \leq N-1 \\ \dfrac{u_{N-1} - u_N}{\Delta z^2} = -1 \end{cases}$$

Matrix formulation

We can represent this system of equations as a matrix (A) of the form

$$A = \frac{1}{\Delta z^2} \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -1 \end{bmatrix}$$

with the u-column vector and b-solution vector as

$$u = \begin{bmatrix} u_2 \\ \vdots \\ u_N \end{bmatrix} \qquad b = \begin{bmatrix} -1 \\ \vdots \\ -1 \end{bmatrix}$$

such that $Au = b$

Two-dimensional approach

In 2D Poisson's equation is

$$\frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = -1; \quad \begin{cases} 0 \le y \le 1 \\ 0 \le z \le 1 \end{cases}$$

and we now apply boundary conditions on edges as

$$\frac{\partial u}{\partial y} = 0 \text{ at } y = 1$$

$$\frac{\partial u}{\partial z} = 0 \text{ at } z = 1$$

$$u = 0 \text{ at } y = 0$$

$$u = 0 \text{ at } z = 0$$

2D Discretization

Using centered-difference for y and z

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta y^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta z^2} = -1$$

which simplifies when $\Delta y = \Delta z$ to

$$u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1} = -\Delta y^2$$

$$2 \leq i \leq N; \qquad 2 \leq j \leq N$$

2D Matrix formulation

Define a global index k as

$$k = (i-2)(N-1) + (j-1)$$

Then create

- $(N-1)^2 \times (N-1)^2$ matrix A
- $(N-1)^2 \times 1$ array b

Solve for

- $(N-1)^2 \times 1$ solution vector u

2D Matrix A

The matrix A contains

- -4 along the main diagonal, except
  - -3 for row $\alpha(N-1)$, $1 \leq \alpha \leq N-2$
  - -3 for row $(N-2)(N-1)+\beta$, $1 \leq \beta \leq N-2$
  - -2 for row $(N-1)^2$
- 1 along four subdiagonals
  - $j-1$ except $(\alpha(N-1)+1, \alpha(N-1))$, $1 \leq \alpha \leq N-2$
  - $j+1$ except $(\alpha(N-1), \alpha(N-1)+1)$, $1 \leq \alpha \leq N-2$
  - $i-1$ (i.e. starting at $(N,1)$)
  - $i+1$ (i.e. starting at $(1,N)$)

Code: Sparse Matrix A in 1D

```
# create sparse matrix A
Il = 2:N-1
Jl = 1:N-2
Iu = 1:N-2
Ju = 2:N-1

dl = ones(N-2)
du = ones(N-2)
d = -2*ones(N-1)
A = sparse(Il,Jl,dl,N-1,N-1) +
    sparse(Iu,Ju,du,N-1,N-1) +
    sparse(1:N-1,1:N-1,d,N-1,N-1)
A[N-1, N-1] = -1
A .= A / (Δz^2)
```

Code: Direct Solve on CPU

```
u_dummy_DSCPU = A \ b
@time u_int_DSCPU = A \ b
u_DSCPU = [0; u_int_DSCPU]
@printf "norm between our solution and the exact solution =
    \x1b[31m %e \x1b[0m\n" sqrt(Δz) * norm(u_DSCPU - exact(z))
```

Code: CG Solve CPU

```
u_dummy_CGCPU = cg(-A, -b)
@time u_int_CGCPU = cg(-A, -b)
u_CGCPU = [0; u_int_CGCPU]
@printf "norm between our solution and the exact solution =
        \x1b[31m %e \x1b[0m\n" sqrt(Δz) * norm(u_CGCPU - exact(z))
```

Code: Direct Solve GPU

```
d_A = CuArray(A)
d_b = CuArray(b)

u_dummy_DSGPU = d_A \ d_b
@time u_int_DSGPU = d_A \ d_b
u_int_DSGPU_reg = Array(u_int_DSGPU)
u_DSGPU = [0; u_int_DSGPU_reg]
@printf "norm between our solution and the exact solution =
         \x1b[31m %e \x1b[0m\n" sqrt(Δz) * norm(u_DSGPU - exact(z))
```

Code: CG Solve GPU

```julia
d_A = CuArrays.CUSPARSE.CuSparseMatrixCSC(A)
d_b = CuArray(b)
u_cg = CuArray(zeros(size(d_b)))
u_dummy = CuArray(zeros(size(d_b)))

cg!(u_dummy, d_A, d_b)
@time cg!(u_cg, d_A, d_b)
u_int_CGGPU_reg = Array(u_cg)
u_CGGPU = [0; u_int_CGGPU_reg]
@printf "norm between our solution and the exact solution =
      \x1b[31m %e \x1b[0m\n" sqrt(Δz) * norm(u_CGGPU - exact(z))
```

Code: Sparse Matrix A in 2D

```
Imain = 1:(N-1)^2
Jmain = 1:(N-1)^2

Il = 2:(N-1)^2
Jl = 1:((N-1)^2-1)
Iu = 1:((N-1)^2-1)
Ju = 2:(N-1)^2

il = N:(N-1)^2
jl = 1:((N-1)^2-(N-1))
iu = 1:((N-1)^2-(N-1))
ju = N:(N-1)^2

dbig = ones((N-1)^2-1)
dsmall = ones((N-1)^2-(N-1))
dmain = -4*ones((N-1)^2)
```

Code: Sparse Matrix A in 2D

```
A = (sparse(Il,Jl,dbig,(N-1)^2,(N-1)^2)
    + sparse(Iu,Ju,dbig,(N-1)^2,(N-1)^2)
    + sparse(Imain,Jmain,dmain,(N-1)^2,(N-1)^2)
    + sparse(il,jl,dsmall,(N-1)^2,(N-1)^2)
    + sparse(iu,ju,dsmall,(N-1)^2,(N-1)^2))
```

Code: Sparse Matrix A in 2D

```
for α=1:N-2
    A[α*(N-1),α*(N-1)] = -3
end

for β=1:N-2
    A[(N-2)*(N-1)+β,(N-2)*(N-1)+β] = -3
end

A[(N-1)^2,(N-1)^2] = -2

for α=1:N-2
    A[α*(N-1)+1,α*(N-1)] = 0
    A[α*(N-1),α*(N-1)+1] = 0
end
```

Code: Direct Solve 2D

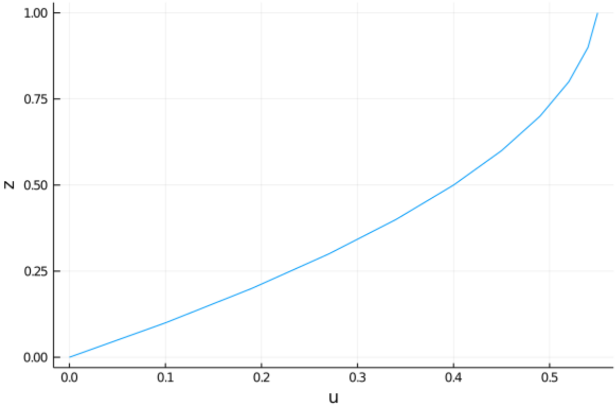
```
u_dummy_DSCPU = A \ b
@time u_int_DSCPU = A \ b
u_DSCPU = reshape(u_int_DSCPU, (N-1, N-1))
U_DSCPU = zeros(N,N)
U_DSCPU[2:N,2:N] = u_DSCPU[:,:]

@printf "norm between AU and b =
        \x1b[31m %e \x1b[0m\n" norm(A*u_int_DSCPU - b)
```

Code: CG Solve 2D

```
u_dummy_CGCPU = cg(-A, -b)
@time u_int_CGCPU = cg(-A, -b)
u_CGCPU = reshape(u_int_CGCPU, (N-1, N-1))
U_CGCPU = zeros(N,N)
U_CGCPU[2:N,2:N] = u_CGCPU[:,:]
@printf "norm between AU and b =
        \x1b[31m %e \x1b[0m\n" norm(A*u_int_CGCPU - b)
```

1D Solution Vector

Our model solves for velocity $u(z)$



- Velocity decreases with depth
- The boundary conditions are satisfied
  - $u(0) = 0$
  - $u'(1) = 0$

1D Convergence Test

Δz	$\varepsilon_{\Delta z} = \sqrt{z}\|u - e\|$	$\Delta\varepsilon = \dfrac{\varepsilon_{2\Delta z}}{\varepsilon_{\Delta z}}$	$r = \log_2(\Delta\varepsilon)$
0.1	3.102×10^{-2}	N/A	N/A
0.05	1.497×10^{-2}	2.072	1.051
0.025	7.352×10^{-3}	2.036	1.026
0.0125	3.642×10^{-3}	2.019	1.014

1D Time Analysis

Using $\Delta z = 5 \times 10^{-5}$

Device	**Method**	**Time [s]**	**Error** $\sqrt{z}\|u - e\|$
CPU	Direct	0.0426	1.44×10^{-5}
	CG	15.1	1.44×10^{-5}
GPU	Direct	1.13	1.44×10^{-5}
	CG	3.19	1.44×10^{-5}

Direct CPU solve is by far the fastest method at this problem size

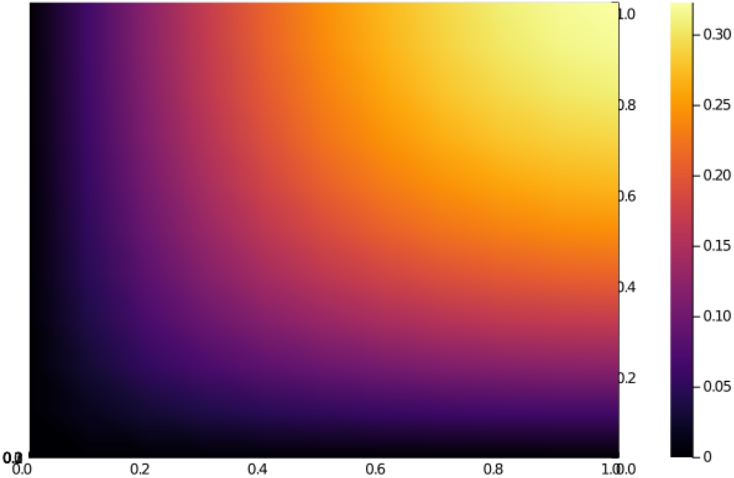
2D Results

No analytical solution, so we test for convergence

1. Visually
2. Via $Au - b \approx 0$

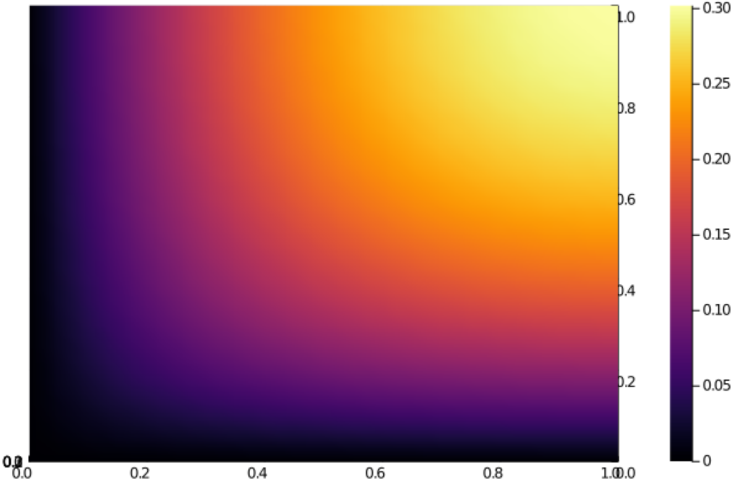
2D Visual Comparison

Low resolution ($\Delta z = 0.1$)



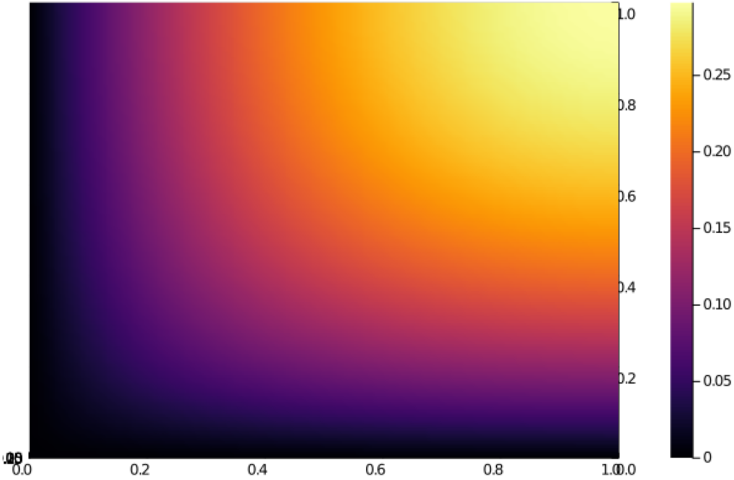
2D Visual Comparison

Mid resolution ($\Delta z = 0.025$)



2D Visual Comparison

High resolution ($\Delta z = 0.01$)



2D Error Analysis

Solve for u, and check $Au - b \approx 0$

Δz	$Au - b$
0.1	8.02×10^{-16}
0.05	1.8×10^{-15}
0.025	4.6×10^{-15}
0.0125	1.02×10^{-14}

The error seems to be something on the order of $(N-1)^2\varepsilon$ with N as $1/\Delta z$ and ε machine precision.

2D Time Analysis

Device	**Method**	**Time [s]**	**Error** $Au - b$
CPU	Direct	0.027	1.31×10^{-14}
	CG	0.025	1.49×10^{-10}
GPU	CG	0.2	1.49×10^{-10}

- CG is less precise
- The CPU methods are faster
- Likely due to problem size limitations
- GPU should surpass CPU eventually, but limited by GPU memory on Talapas