One- and Two-Dimensional Poisson's Equation

Pierce Hunter, Nick Kuckuck, Haoran Wang

CIS 410/510 Final Presentation
8th June 2020

Outline

One dimensional model

We solved the 1D equation

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

with boundary conditions

$$\frac{\partial u}{\partial z}(1) = 0; \quad u(0) = 0 \tag{2}$$

in the four following ways:

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

1D Discretization

For the one-dimensional problem we can utilize the straight-forward centered-difference discretization from class, namely

$$\frac{u_{j-1} - 2u_j + u_{j+1}}{\Delta z^2} = -1 \quad \text{for} \quad 1 \leq j \leq N. \tag{3}$$

We discretize on the boundaries as well, giving the final system of equations

$$\begin{cases} \dfrac{-2u_2 + u_3}{\Delta z^2} = -1 \\ \dfrac{u_{j-1} - 2u_j + u_{j+1}}{\Delta z^2} = -1; \quad 3 \leq j \leq N-1 \\ \dfrac{u_{N-1} - u_N}{\Delta z^2} = -1. \end{cases} \tag{4}$$

Matrix formulation

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

$$A = \frac{1}{\Delta z^2} \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -1 \end{bmatrix} \tag{5}$$

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

$$u = \begin{bmatrix} u_2 \\ \vdots \\ u_N \end{bmatrix} \qquad b = \begin{bmatrix} -1 \\ \vdots \\ -1 \end{bmatrix} \tag{6}$$

such that $Au = b$.

Two-dimensional approach

We then expanded the problem to two-dimensions utilizing the same techniques. In 2D eq. (1) becomes

$$\frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = -1; \quad \begin{cases} 0 \le y \le 1 \\ 0 \le z \le 1 \end{cases} \tag{7}$$

and we expand the boundary conditions in (2) as

$$\frac{\partial u}{\partial y} = 0 \text{ at } y = 1 \tag{8}$$

$$\frac{\partial u}{\partial z} = 0 \text{ at } z = 1 \tag{9}$$

$$u = 0 \text{ at } y = 0 \tag{10}$$

$$u = 0 \text{ at } z = 0. \tag{11}$$

2D Discretization

We transform (7) into a system of ODE's by discretizing in space—both y and z—using centered difference, such that

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{\Delta y^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{\Delta z^2} = -1. \tag{12}$$

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

$$u_{i-1,j} + u_{i,j-1} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1} = -\Delta y^2. \tag{13}$$

This discretization works when $2 \leq i \leq N$ and $2 \leq j \leq N$, but we need to solve on the boundaries, of which there are many. The boundary conditions discretize separately for the edges and corners, which expands our boundary conditions to eight separate cases

2D Matrix formulation

In order to convert this discretization to a matrix that can be used for a direct solve we need to define a new indexing convention. For this we calculate a global index k as

$$k = (i-2)(N-1) + (j-1). \tag{14}$$

We can then translate our discretization into this new system, giving nine total cases.

2D Matrix formulation Cont.

So what we end up with is an $(N-1)^2 \times (N-1)^2$ matrix A and a solution vector b with $(N-1)^2$ entries. Moving across a row we start at $(2,2)$, to increase j by one we move to the right 1 entry, to increase i by 1 we move the right $(N-1)$ entries, such that we hit every value of j first, then move to the next i.
Along the diagonals of the matrix A we have -4 except in the following locations:

- rows $\alpha(N-1)$ the diagonal entry is -3 for $1 \leq \alpha \leq N-2$
- rows $(N-2)(N-1)+\beta$ the diagonal is -3 for $1 \leq \beta \leq N-2$
- row $(N-1)^2$ the diagonal is -2

2D Matrix formulation Cont.

We also have four other sub-diagonals that will all contain ones except where noted. These represent the following location:

- $j-1$ which is directly below the diagonal
  In Julia these are the locations: `[2:(N-1)`^{2}`, 1:(N-1)`^{2}`-1]`
  Exception: the locations $(\alpha(N-1)+1, \alpha(N-1))$ should be zero for $1 \leq \alpha \leq N-2$
- $j+1$ which is directly above the diagonal
  In Julia these are the locations: `[1:(N-1)`^{2}`-1, 2:(N-1)`^{2}`]`
  Exception: the locations $(\alpha(N-1), \alpha(N-1)+1)$ should be zero for $1 \leq \alpha \leq N-2$
- $i-1$ which are exactly $N-1$ below the diagonal
  In Julia these are the locations: `[N:(N-1)`^{2}`, 1:(N-1)`^{2}`-(N-1)]`
- $i+1$ which are exactly $N+1$ above the diagonal
  In Julia these are the locations: `[1:(N-1)`^{2}`-(N-1), N:(N-1)`^{2}`]`

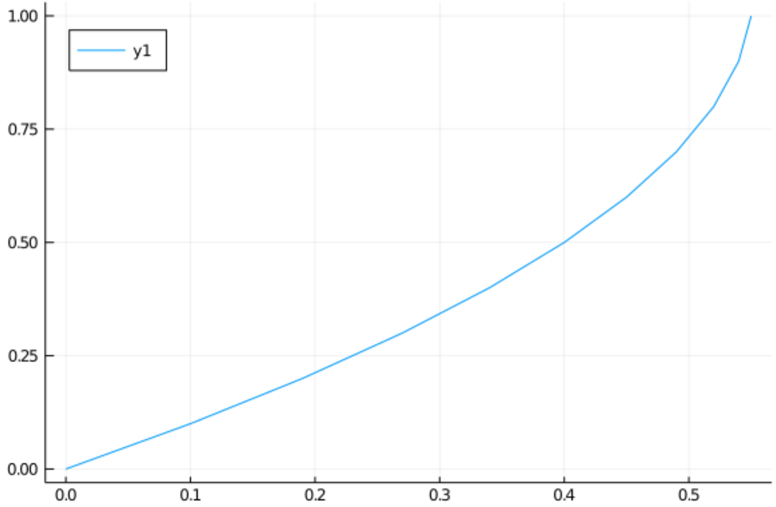
2D Matrix formulation Cont.

Once A is created it is probably easier to create a column vector of length $(N-1)^2$ in which every location contains $-\Delta y^2$. Once a solution is found via $u = A\backslash b$ or CG, then u can be reshaped to the correct dimensions either manually—`i = floor((k-1)/(N-1)) + 2; j = mod(k-1,N-1) + 2`—or via the reshape function transposed—`U = reshape(u,N-1,N-1)'`. Using reshape without the transpose puts the solution in meshgrid format (with y as the columns and z as the rows) similar to looking at a cross-section.

Code Samples

One dimensional with convergence tests

Our model solves for the velocity u as a function of depth z for a semi-random nondimensional problem. We plot u on the x-axis and z as the y-axis below.



1D Results

It is noted that velocity increases as a function of z until reaching a maximum such that the upper boundary condition is satisfied.

We then ensured our solution was converging as expected by decreasing Δz by orders of 2, and checking to make sure the error was decreasing at a constant rate.

Δz	$\varepsilon_{\Delta z} = \sqrt{z}\|u - e\|$	$\Delta\varepsilon = \frac{\varepsilon_{2\Delta z}}{\varepsilon_{\Delta z}}$	$r = \log_2(\Delta\varepsilon)$
0.1	3.102×10^{-2}	Empty Entry	Empty Entry
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

We do see the error decreasing linearly with decreases in Δz, which leads us to believe we do not have instability in our system.

1D Time

We timed the one-dimensional problem using the four methods listed above and depict the results in the table below, keeping Δz constant at 5×10^{-5} throughout.

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}

We note the direct solve on the CPU is the fastest by far. We expect this to be the case as we do not have any time dependence in our model. We do see, when using CG, that the GPU is significantly faster than the CPU, which would be helpful if this problem were altered to include time dependence.

2D Results

We ensured our solution was converging as expected by decreasing $\Delta z = \Delta y$ by orders of 2, and checking to make sure the error was still within the machine precision limit. We did this by solving for u and then checking to ensure $Au - b \approx 0$. We show the results for four values of Δz in the table below.

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}

2D Time

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}

We see the CPU methods are much faster than the GPU methods, but we do expect that if we had been able to run a large enough problem (we were limited by GPU memory) the CG solver on the GPU would have overtaken both methods in terms of both efficiency and time. Unfortunately, in the time remaining we don't really have a way to evidence that claim, it is just our groups expectation that eventually the GPU code would be faster.