# Jacobi Introduction

Intro to HPC - Lab 2

# Iterative methods

- “Too hard” to analytically solve a problem
- Estimate an answer and iteratively refine
- e.g. Newton-Raphson method for finding roots

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

# Linear solvers

- Used to solve a system of linear equations

$$
\begin{aligned}
ax + by + cz &= d \\
ex + fy + gz &= h \\
ix + jy + kz &= l
\end{aligned}
$$

- Can express as a matrix equation

$$
\begin{pmatrix} a & b & c \\ e & f & g \\ i & j & k \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} d \\ h \\ l \end{pmatrix}
$$

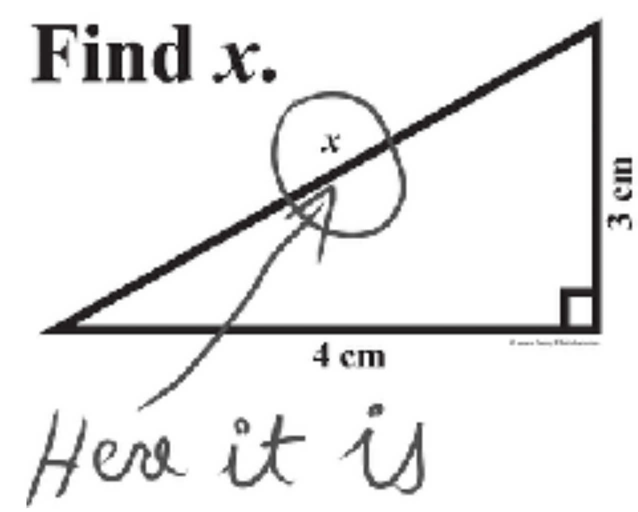
$$
Ax = b
$$

# Linear solvers (matrix)

- The matrix A and the right hand side b is **known**

$$Ax = b$$

- **Goal:** find x




- **But A is "too hard" to just invert**

$$x = A^{-1}b$$

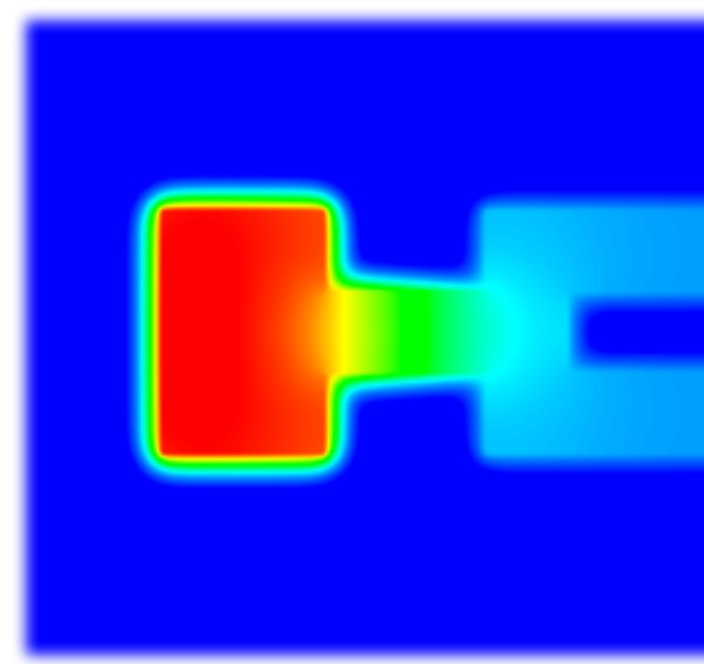
# Motivation

- Linear systems come up in many physical settings described by (partial-)differential equations

- E.g. Diffusion equation, describing the temperature u of $\Omega$ an object over time

$$\frac{\partial u}{\partial t} - \nabla \cdot (a \nabla u) = f \text{ in } \Omega$$

- Linear system can be formed $Au = f$



# General iterative methods

- Start with an estimate (guess) $x_0$
- Evaluate $Ax_0$ and calculate error (residual) $b - Ax_0$
- Use this to improve the guess of x, and keep going until convergence

**Key point: haven't had to invert the matrix**

# Jacobi method

- Reduce the number of iterations to convergence with the Jacobi method
- Split A into diagonal plus "the rest" $A = D + R$
- Idea is $D^{-1}A \approx I$
- So solving $D^{-1}Ax = D^{-1}b$ is easier as inverting a diagonal matrix is trivial (entries are 1/entry)
- Apply iterative method: $x_{k+1} = D^{-1}(b - Rx_k)$

# Jacobi iterations

**New estimate**

**Initial estimate**

$$x_{k+1}(i) = \frac{1}{a_{ii}} \left( b(i) - \underbrace{\sum_{j \neq i} a_{ij} x_k(j)}_{} \right)$$

**Matrix-vector multiply**

# Convergence checking and solution error

- A convergence check is used to watch for when x changes less than a given tolerance
  - If x doesn't change significantly, then finish iterations
- By default, the new x is within 0.0001 Euclidian distance of the old x

$$\sqrt{(x_1 - x_0)^2}$$

- Program prints out solution error (residual) as Euclidian distance of Ax and b

$$\sqrt{(b - Ax_k)^2}$$

# Coursework

- Given an unoptimised, serial implementation of Jacobi
- Coursework 1: optimise it
- Coursework 2: parallelise it