Prepared by Julian García

Workshop 6
Iterative Methods

FIT 3139

Computational Modelling and Simulation

MONASH University
Information Technology

Outline

- What are **Iterative Methods**
- Iterative methods:
  - Solving linear systems — **Jacobi iteration**.
  - ~~For eigenvalues/eigenvectors — **Power method**.~~
- Comparing **Iterative vs Direct** methods.

Direct method:
Produces an exact solution (assuming exact arithmetic) in a finite number of steps.

Iterative method:
Starts with “a guess” that is *refined* via operations until a *desired accuracy* is reached.

Potentially an infinite number of steps

$$A\vec{x} = \vec{b}$$

Direct method: LU decomposition with pivoting.

Iterative method: Start with a *guess* $\mathbf{x}_0$ *and refine.*

Find an operation, that produces a sequence….

$$\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots\ldots, \mathbf{x}_k$$

such that $\mathbf{x}_k$ is *close enough* to the solution.

?

$$A\vec{x} = \vec{b}$$

$A = A - B + B$ assuming B has an inverse.

$$(A - B + B)\vec{x} = \vec{b}$$

$$B\vec{x} = \vec{b} - (A - B)\vec{x}$$

$$\vec{x} = B^{-1}\vec{b} - B^{-1}(A - B)\vec{x}$$

$$\vec{x} = \vec{c} + C\vec{x}$$

$$C = B^{-1}(B - A)$$
$$c = B^{-1}b$$

$$\vec{x}^{(i)} = \vec{c} + C\vec{x}^{(i-1)}$$

It is easy to show that the fixed point of this iteration must be a solution.

$$A\vec{x} = \vec{b}$$

$$A = A - B + B$$

$$C = B^{-1}(B - A)$$

$$c = B^{-1}b$$

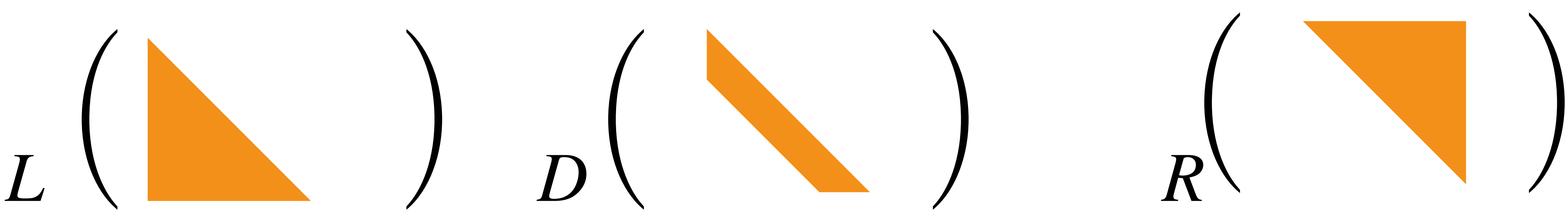
$$\vec{x}^{(i)} = \vec{c} + C\vec{x}^{(i-1)}$$

- How to choose B so that the iteration converges from an arbitrary $\vec{x}^{(0)}$

- Assume A has no zeroes in the diagonal

(we can always swap rows to achieve this if A is invertible)

$$A = L + D + R$$



- B = D is a good choice.

(it is possible to show why, but out of unit scope)

$$A\vec{x} = \vec{b}$$

$$A = A - B + B$$

$$C = B^{-1}(B - A)$$

$$c = B^{-1}b$$

$$A = L + D + R$$

$$B = D$$

Make it a more concrete iteration with B= D

$$\vec{x}^{(i)} = \vec{c} + C\vec{x}^{(i-1)}$$

$$C = I - B^{-1}A$$

$$C = I - D^{-1}(L + D + R)$$

$$C = I - D^{-1}L - I - D^{-1}R$$

$$C = -D^{-1}(L + R)$$

$$c = D^{-1}b$$

$$\boxed{\vec{x}^{(i)} = D^{-1}b + (-D^{-1}(L + R))\vec{x}^{(i-1)}}$$

$$A\vec{x} = \vec{b}$$

$$A = L + D + R$$

$$\boxed{\vec{x}^{(i)} = D^{-1}b + (-D^{-1}(L+R))\vec{x}^{(i-1)}}$$

Jacobi iteration

$$A = L + D + R$$

$$\boxed{\vec{x}^{(i)} = D^{-1}b + (-D^{-1}(L+R))\vec{x}^{(i-1)}}$$

Element-wise

(you will do this in your tutorial)

$$\vec{x}_j^{(i)} = \frac{1}{a_{jj}}\left(b_j - \sum_{k \neq j}^{n} a_{jk} x_k^{(i-1)}\right)$$

```python
import numpy as np


def jacobi(A, b, number_of_iterations):
    assert type(A) == np.ndarray
    assert type(b) == np.ndarray
    assert len(b.shape) == 1
    assert A.shape[0] == A.shape[1]

    n = A.shape[0]
    x = np.ones(n)
    y = np.ones(n)

    for i in range(number_of_iterations):
        print(x)
        for j in range(0, n):
            y[j] = b[j]
            for k in range(0, j):
                y[j] = y[j]-A[j, k]*x[k]
            for k in range(j+1, n):
                y[j] = y[j]-A[j, k]*x[k]
            y[j] = y[j]/A[j, j]
        x = np.copy(y)
```

$$\vec{x}_j^{(i)} = \frac{1}{a_{jj}}\left(b_j - \sum_{k\neq j}^{n} a_{jk} x_k^{(i-1)}\right)$$

Jacobi iteration does not always converge….

$$\boxed{\vec{x}_j^{(i)} = \frac{1}{a_{jj}}\left(b_j - \sum_{k\neq j}^{n} a_{jk} x_k^{(i-1)}\right)}$$

- The Jacobi iteration is slow to converge, and does not always converge.

- Note that the computation of $x_i^{(j+1)}$ is independent of any other $x_l^{(j+1)}$

- This may be good for parallelism, but also hints at improvement potential.

Gauss-Seidel iteration

- **Idea**: To calculate $x_2^{(j+1)}$ you can already use $x_1^{(j+1)}$. To calculate $x_3^{(j+1)}$ you can use newly computed $x_2^{(j+1)}$ and $x_1^{(j+1)}$; and so on.

$$\boxed{\vec{x}_j^{(i)} = ??????}$$

(left as an exercise)

- Gauss-Seidel has weaker convergence conditions and is often faster than Jacobi.

LUP vs Jacobi

(Time complexity)

<u>LU with pivoting:</u> $\mathcal{O}(n^3)$

target
number of iterations

<u>Iterative method:</u> $\mathcal{O}(n^2) \times \mathcal{O}(n)$

per iteration

This is achievable **in some applications.** If the matrix is sparse (lots of zeros); e.g., constant number of zeroes per row.

Per iteration complexity is customary for iterative methods.

Direct vs Iterative

- Direct methods require no initial estimate and are accurate (assuming exact arithmetic).

- Iterative methods:

    - Convergence may depend on special properties. Often problem-specific.

    - May profit from specific representations or data structures that are problem-specific.

    - Practical in many cases.

Read More…

- Heath, Michael T. **Scientific computing: an introductory survey**. Vol. 80. SIAM, 2018.
  - Chapter 4.5
  - Chapters 11.5
- Wendland, Holger. **Numerical linear algebra: an introduction**. Vol. 56. Cambridge University Press, 2017.
  - Chapter 4.

Thank you

Up next: Iterative methods for solving non-linear equations