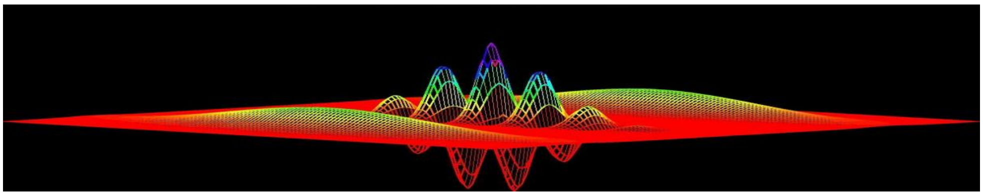
Computational Physics

numerical methods with C++ (and UNIX)



Fernando Barao
Instituto Superior Tecnico, Dep. Fisica
email: barao@lip.pt

Computational Physics
Numerical methods

Fernando Barao, Phys Department IST (Lisbon)

Numerical methods

- ✔ System of linear equations
  - ► Gauss elimination
  - ► LU decomposition
  - ► Gauss-Seidel method
- ✔ Interpolation
  - ► Lagrange interpolation
  - ► Newton method
  - ► Neville method
  - ► Cubic spline

LU decomposition

✔ Any square matrix $\mathbf{A}$ can be expressed as the product of a lower triangular matrix $\mathbf{L}$ and an upper trinagular matrix $\mathbf{U}$

$\boxed{\mathbf{A} = \mathbf{L}\ \mathbf{U}}$

☞ *the computation of $\mathbf{L}$ and $\mathbf{U}$ is known as LU decomposition or $\mathbf{LU}$ factorization*

☞ *the factorization is not unique unless constraints on L and U are applied*

✔ common decompositions :

Decomposition	Constraints
Doolittle	$L_{ii} = 1$ with $i = 1, 2, ..., n$
Crout	$U_{ii} = 1$ with $i = 1, 2, ..., n$
Choleski	$\mathbf{L} = \mathbf{U}^{\mathbf{T}}$

After decomposing $\mathbf{A}$:

$\mathbf{Ax} = \mathbf{b} \Rightarrow \mathbf{LUx} = \mathbf{b}$

We have :

$\mathbf{Ly} = \mathbf{b}$ with $(\mathbf{Ux} = \mathbf{y})$

Therefore : we start getting $\mathbf{y}$ and then $\mathbf{x}$

Doolittle decomposition

✔ Considere a 3×3 $\mathbf{A}$ matrix and the respective triangular lower and upper matrices $\mathbf{L}$ and $\mathbf{U}$

$$[\mathbf{A}] = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \quad [\mathbf{L}] = \begin{pmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{pmatrix} \quad [\mathbf{U}] = \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix}$$

✔ Making the operation : $\mathbf{A} = \mathbf{LU}$

$$[\mathbf{A}] = \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ U_{11}L_{21} & U_{12}L_{21} + U_{22} & U_{13}L_{21} + U_{23} \\ U_{11}L_{31} & U_{12}L_{31} + U_{22}L_{32} & U_{13}L_{31} + U_{23}L_{32} + U_{33} \end{pmatrix}$$

Doolittle decomposition (cont.)

✔ Applying Gauss elimination : eliminating elements below pivot $(LU)_{11}$

$(\text{Row}_2 - L_{21}\text{Row}_1 \rightarrow \text{Row}_2)$ to eliminate $(\text{LU})_{21}$

$(\text{Row}_3 - L_{31}\text{Row}_1 \rightarrow \text{Row}_3)$ to eliminate $(\text{LU})_{31}$

$$[\mathbf{A}'] = \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & U_{22}L_{32} & U_{23}L_{32} + U_{33} \end{pmatrix}$$

✔ Applying Gauss elimination : eliminating element below pivot $(LU)_{22}$

$(\text{Row}_3 - L_{32}\ \text{Row}_2 \rightarrow \text{Row}_3)$ to eliminate $(\text{LU})_{32}$

$$[\mathbf{A}''] = \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix}$$

Gauss elimination method provided us with $\mathbf{U}$ and $\mathbf{L}$ matrices !

Doolittle decomposition (cont.)

- ✔ The matrix $\mathbf{U}$ is the one that results from the Gauss elimination
- ✔ The off-diagonal elements of matrix $\mathbf{L}$ correspond to the multipliers used during Gauss elimination
- ✔ It is current pratice to store in a matrix both the upper triangular matrix and the lower triangular matrix

  the diagonal elements of the $\mathbf{L}$ matrix are not stored...

$$[\mathbf{L} \setminus \mathbf{U}] = \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ L_{21} & U_{22} & U_{23} \\ L_{31} & L_{32} & U_{33} \end{pmatrix}$$

```
// matrix A(nxn)

// Gauss elimination

loop on pivot row (k): k = 0, n-2

  loop on rows below pivot:
                     i = k+1, n-1

     - for every row:
       compute multiplier
                     A(i,k)/A(k,k)

     - transform row i:
       only elements (i, k+1:n)
                      are stored

     - store mutipliers on A(i,k)

// solution now...
```


Doolittle : solution

- ✔ We have to solve the system $\mathbf{Ly} = \mathbf{b}$ by forward substitution

$$\begin{pmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

- ✔ forward substitution :

$$\begin{pmatrix} y_1 & = b_1 \\ L_{21}y_1 + y_2 & = b_2 \\ L_{k1}y_1 + L_{k2}y_2 + \cdots + L_{k,k-1}y_{k-1} + y_k & = b_k \end{pmatrix}$$

The solution of the equation for a generic k row :

$$y_k = b_k - \sum_{j=1}^{k-1} L_{kj} y_j \qquad (k = 2, 3, ...n(\text{rows}))$$

Doolittle decomp : example

Solve the following system using Doolittle decomposition

$$[\mathbf{A}] = \begin{pmatrix} 1 & 4 & 1 \\ 1 & 6 & -1 \\ 2 & -1 & 2 \end{pmatrix} \qquad [\mathbf{b}] = \begin{pmatrix} 7 \\ 13 \\ 5 \end{pmatrix}$$

Choleski decomposition

- ✔ This method uses the decomposition : $\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathbf{T}}$
- ✔ The nature of the decomposition ($\mathbf{L}\mathbf{L}^{\mathbf{T}}$) requires a symmetric $\mathbf{A}$ matrix
- ✔ It envolves the using of square root function

  ☞ to avoid square roots of negative numbers the matrix must be *positive definite* $\Rightarrow \mathbf{x}^{\mathbf{T}}\mathbf{A}\mathbf{x} > \mathbf{0}$

$$[\mathbf{A}] = LL^T = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{pmatrix}$$

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} L_{11}^2 & L_{11}L_{21} & L11L_{31} \\ L_{11}L_{21} & L_{21}^2 L_{22}^2 & L_{21}L_{31} + L_{22}L_{32} \\ L_{11}L_{31} & L_{21}L_{31} + L_{22}L_{32} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{pmatrix}$$

Choleski decomposition (cont.)

✔ Symmetric matrix $\Rightarrow$ $n!$ equations to solve ($n = 3 \Rightarrow$ 6eqs)

$$L_{11} = \sqrt{A_{11}}$$
$$L_{21} = A_{21}/L_{11}$$
$$L_{31} = A_{31}/L_{11}$$
$$L_{22} = sqrtA_{22} - L_{21}^2$$
$$L_{32} = (A_{32} - L_{21}L_{31})/L_{22}$$
$$L_{33} = \sqrt{A_{33} - L_{31}^2 L_{32}^2}$$

Matrix inversion

✔ To invert the matrix $\mathbf{A}$ we have to solve the equation :

$$\mathbf{AX} = \mathbf{I} \Rightarrow \mathbf{A}^{-1}\mathbf{AX} = \mathbf{A}^{-1}\mathbf{I} \Rightarrow \mathbf{X} = \mathbf{A}^{-1}$$

$\mathbf{I} \equiv$ is the identity matrix

$\mathbf{X} \equiv$ is the inverse of $\mathbf{A}$

✔ For inverting $\mathbf{M}$ we have to solve :

$\mathbf{Ax_i} = \mathbf{b_i} \qquad \mathbf{i = 1, 2, ...n}$

$\mathbf{b_i}$ = ith column of I

$\mathbf{x_i}$ = ith column of $\mathbf{A}^{-1}$

Banded matrices

- ✔ In case a matrix present its non-zero members all grouped around the main diagonal, it is said to be of the **banded** type (common to scientific problems)

  - ☞ a **tridiagonal matrix** presents a **bandwidth=3**, i.e., at most three nonzero elements in each row (or column)
  - ☞ some of the elements in the populated diagonals can be zero (of course !)

$$[\mathbf{A}] = \begin{pmatrix} A_{11} & A_{12} & 0 & 0 & 0 \\ A_{21} & A_{22} & A_{23} & 0 & 0 \\ 0 & A_{32} & A_{33} & A_{34} & 0 \\ 0 & 0 & A_{43} & A_{44} & A_{45} \\ 0 & 0 & 0 & A_{43} & A_{55} \end{pmatrix}$$

- ✔ The banded structure of a coefficient matrix can be exploited to save storage space and computation time

Banded matrices : LU decomposition

- ✔ Let's use the Doolittle scheme to decompose the triadiagonal matrix $\mathbf{A}$
- ✔ To reduce the row $\mathbf{k}$, i.e., to eliminate the $\mathbf{a_{k-1}}$ element we do (pivot $\rightarrow$ $\mathbf{Row_{k-1}}$) :

$$\mathbf{Row_k} - \mathbf{Row_{k-1}} \times \left(\frac{\mathbf{a_{k-1}}}{\mathbf{b_{k-1}}}\right) \rightarrow \mathbf{Row_k}$$
$$\mathbf{k} = \mathbf{2}, \mathbf{3}, \cdots, \mathbf{n}$$

- ✔ In the decomposition process, the reduced $\mathbf{a_i}$ elements are replaced by the multipliers $\left(\frac{\mathbf{a_{k-1}}}{\mathbf{b_{k-1}}}\right)$

$$\mathbf{a_{k-1}} = \left(\frac{\mathbf{a_{k-1}}}{\mathbf{b_{k-1}}}\right)$$
$$\mathbf{b_k} = \mathbf{b_k} - \left(\frac{\mathbf{a_{k-1}}}{\mathbf{b_{k-1}}}\right) \times \mathbf{c_{k-1}}$$
$$\mathbf{c_k} = \text{not affected}$$

$$[\mathbf{A}] = \begin{pmatrix} b_1 & c_1 & 0 & 0 & \cdots & 0 \\ a_1 & b_2 & c_2 & 0 & \cdots & 0 \\ 0 & a_2 & b_3 & c_3 & \cdots & 0 \\ 0 & 0 & a_3 & b_4 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & a_{n-1} & b_n \end{pmatrix}$$

```
The vectors to store are:

   a = a_1, a_2, ..., a_{n-1}

   b = b_1, b_2, ..., b_{n}}

   c = c_1, c_2, ..., c_{n-1}
```

Banded matrices : LU solution

✔ Now we have to solve the equation $\mathbf{Ax} = \mathbf{d}$, there are two equations to solve :

1) $\mathbf{Ly} = \mathbf{d}$

2) $\mathbf{Ux} = \mathbf{y}$

by respectively forward and back substitution

$$[\mathbf{L}|\mathbf{d}] = \left(\begin{array}{cccccc|c} 1 & 0 & 0 & 0 & \cdots & 0 & d_1 \\ a_1 & 1 & 0 & 0 & \cdots & 0 & d_2 \\ 0 & a_2 & 1 & 0 & \cdots & 0 & d_3 \\ 0 & 0 & a_3 & 1 & \cdots & 0 & d_4 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & \cdots & 0 & a_{n-1} & 1 & d_n \end{array}\right) \quad [\mathbf{U}|\mathbf{y}] = \left(\begin{array}{cccccc|c} b_1 & c_1 & 0 & \cdots & 0 & 0 & y_1 \\ 0 & b_2 & c_2 & \cdots & 0 & 0 & y_2 \\ 0 & 0 & b_3 & \cdots & 0 & 0 & y_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & b_{n-1} & c_{n-1} & y_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & b_n & y_n \end{array}\right)$$

Iterative methods

✔ In iterative methods, we start with an initial guess for the solution $\mathbf{x}$ and then we iterate over solutions until changes are negligible

✔ The convergence of the iterative methods is only guaranteed if the coefficient matrix is diagonally dominant

- ▸ The number of iterations depend on the initial guess
- ▸ Convergence will be attained independently of the initial guess

Gauss-Seidel method

✔ Let's write the equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ in scalar notation :

$$\sum_{j=1}^{n} A_{ij}\, x_j = b_i \qquad (i = 1, 2, \cdots, n)$$

✔ Extracting the term containing $\mathbf{x_i}$:

$$A_{ii}x_i + \sum_{\substack{j=1\\(i\neq j)}}^{n} A_{ij}\, x_j = b_i \quad \Rightarrow \quad x_i = \frac{1}{A_{ii}}\left(b_i - \sum_{\substack{j=1\\(i\neq j)}}^{n} A_{ij}\, x_j\right)$$

Gauss-Seidel method (cont.)

✔ The convergence of the method can be improved using *relaxation*

✔ the iterated $\mathbf{x_i}$ value is obtained from a weighted (ω) average of its previous value and the iterative formula shown before

$$x_i^{(k+1)} = \frac{\omega}{A_{ii}}\left(b_i - \sum_{\substack{j=1\\(i\neq j)}}^{n} A_{ij}\, x_j^{(k)}\right) + (1-\omega)x_i^{(k)}$$

ω is the *relaxation factor*

✔ Defining the change on x on the kth iteration without relaxation mechanism as,
$\Delta x^{(k)} = |\mathbf{x}^{(k-1)} - \mathbf{x}^(k)|$

A good estimate of ω can be computed at run time as,
$\omega \simeq \frac{2}{1+\sqrt{1-(\Delta x^{(k+p)}/\Delta x^{(k)})^{1/p}}}$

algorithm

```
- realize k iterations (~10)
  without weighting and record
  after the kth iteration the
  change on x

- realize additional p iterations
  and record the change on x for
  the last iteration

- from that iteration on, introduce
  weighting on x calculation
```

C++ class scheme