• Explicit method and implicit method.
• Tri-diagonal banded matrix.
• LU decomposition.
• Successive over relaxation (SOR).
• Ising model revisited.
**Explicit vs Implicit scheme in solving diffusion is contrasted**

- A diffusion type equation reads

\[
\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2}.
\]

- Forward Time Central Space (FTCS) **explicit** scheme

\[
\frac{u(x, t + k) - u(x, t)}{k} = \frac{u(x + h, t) - 2u(x, t) + u(x - h, t)}{h^2}.
\]

which reads (here \(\sigma = k/h^2\))

\[
u(x, t + k) = \sigma u(x + h, t) + (1 - 2\sigma)(x, t)\sigma u(x - h, t).
\]

- As a contrast, the Crank Nicolson (from 4/12) is an **implicit** scheme

\[
\frac{u(x, t) - u(x, t - k)}{k} = \frac{u(x + h, t) - 2u(x, t) + u(x - h, t)}{h^2}.
\]

Note \(u(x, t - k)\) on the right hand side is the only term from the previous step (we call this implicit).
Diffusion equation is solved for one step using $A \cdot x = b$ solve.
Let us apply Gaussian elimination to a $4 \times 4$ tri-diagonal matrix

- Exercise with a simple one:

  $$
  \begin{pmatrix}
  1 & 0 & 0 & 0 \\
  1 & -2 & 1 & 0 \\
  0 & 1 & -2 & 1 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4
  \end{pmatrix}
  =
  \begin{pmatrix}
  0 \\
  1 \\
  1 \\
  0
  \end{pmatrix}
  $$

- Subtract 1st row $\times 2$ from 2nd. No need for 3rd and 4th row.

  $$
  \begin{pmatrix}
  1 & 0 & 0 & 0 \\
  0 & -2 & 1 & 0 \\
  0 & 1 & -2 & 1 \\
  0 & 0 & 0 & 1
  \end{pmatrix}
  \begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4
  \end{pmatrix}
  =
  \begin{pmatrix}
  0 \\
  1 \\
  1 \\
  0
  \end{pmatrix}
  $$

  Then, add ”2nd $\times 1/2$” on 3rd.
• Add ”2nd ×1/2” on 3rd. Finally,

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & -2 & 1 & 0 \\
0 & 0 & -3/2 & 1 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{pmatrix}
= 
\begin{pmatrix}
0 \\
1 \\
3/2 \\
0
\end{pmatrix}
\]

We obtain an upper triangular matrix.

• Back substitution gives

\[
x_4 = 0, (B.C.)
\]

\[
x_3 = \left( \frac{3}{2} - x_4 \right) \times \frac{-2}{3} = \frac{3}{2} \times \frac{-2}{3} = -1,
\]

\[
x_2 = 2 \times \frac{-1}{2} = -1,
\]

\[
x_1 = 0 (B.C.)
\]
A scheme in solving tri-diagonal system is given

- The 1st to 2nd transformation is given by

\[ d_2 \leftarrow d_2 - \frac{a_1}{d_1} c_1 \]

\[ b_2 \leftarrow b_2 - \frac{a_1}{d_1} b_1 \]

- In general,

\[ d_i \leftarrow d_i - \frac{a_{i-1}}{d_{i-1}} c_{i-1} \]

\[ b_i \leftarrow b_i - \frac{a_{i-1}}{d_{i-1}} b_{i-1} \]
• At the end of the forward elimination phase, we have

\[
\begin{pmatrix}
d_1 & c_1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & d_2 & c_2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & d_3 & c_3 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & d_i & c_i & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & d_{n-1} & c_{n-1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & d_n
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
\vdots \\
x_i \\
\vdots \\
x_{n-1} \\
x_n
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
b_2 \\
b_3 \\
\vdots \\
b_i \\
\vdots \\
b_{n-1} \\
b_n
\end{pmatrix}
\]

• Back substitution is given by

\[
x_i \leftarrow \frac{b_i - c_i x_{i+1}}{d_i}
\]
A pseudo code for tri-diagonal solver is given

**Forward elimination**

```
do i = 2, n - 1
    x_{mult} = a_{i-1}/d_{i-1}
    d_i = d_i - x_{mult} \cdot c_{i-1}
    b_i = b_i - x_{mult} \cdot b_{i-1}
endo
do i = n - 2, 1, -1
    x_i = (b_i - c_i \cdot x_{i+1})/d_i
endo```

**Back substitution**

```
x_{n-1} = b_{n-1}/d_{n-1}
endo```

**LU factorization is based on Gaussian elimination but useful if** $A$ **of** $Ax = b$ **is fixed**

- For example Poisson equation in PIC/Vlasov simulation. Only the right hand side changed every time step.
- **We consider** $A = LU$ **where**

$$
A = \begin{pmatrix}
    a_{1,1} & \ldots & \ldots & \ldots & a_{1,n} \\
    \ldots & \ldots & \ldots & \ldots & \ldots \\
    \ldots & a_{i,i-1} & a_{i,i} & a_{i,i+1} & \ldots \\
    \ldots & \ldots & \ldots & \ldots & \ldots \\
    a_{n,1} & \ldots & \ldots & \ldots & a_{n,n}
\end{pmatrix}
$$

$$
L = \begin{pmatrix}
    1 & \ldots & \ldots & \ldots & 0 \\
    \ldots & \ldots & \ldots & \ldots & \ldots \\
    \ldots & l_{i,i-1} & 1 & 0 & \ldots \\
    \ldots & \ldots & \ldots & \ldots & \ldots \\
    l_{n,1} & \ldots & \ldots & \ldots & 1
\end{pmatrix}
$$

$$
U = \begin{pmatrix}
    u_{11} & \ldots & \ldots & \ldots & u_{1n} \\
    \ldots & \ldots & \ldots & \ldots & \ldots \\
    \ldots & 0 & u_{i,i} & u_{i,i+1} & \ldots \\
    \ldots & \ldots & \ldots & \ldots & \ldots \\
    0 & \ldots & \ldots & \ldots & u_{nn}
\end{pmatrix}
$$
Recall the Gaussian elimination problem

- After the elimination the matrix equation
  \[
  \begin{pmatrix}
  6 & -2 & 2 & 4 \\
  12 & -8 & 6 & 10 \\
  3 & -13 & 9 & 3 \\
  -6 & 4 & 1 & -18 \\
  \end{pmatrix}
  \cdot
  \begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  \end{pmatrix}
  =
  \begin{pmatrix}
  16 \\
  26 \\
  -19 \\
  -34 \\
  \end{pmatrix}
  \]
  ended up in an upper triangular matrix.

  \[
  \begin{pmatrix}
  6 & -2 & 2 & 4 \\
  0 & -4 & 2 & 2 \\
  0 & 0 & 2 & -5 \\
  0 & 0 & 0 & -3 \\
  \end{pmatrix}
  \cdot
  \begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  \end{pmatrix}
  =
  \begin{pmatrix}
  16 \\
  -6 \\
  -9 \\
  -3 \\
  \end{pmatrix}
  \]

- The forward elimination phase can be interpreted as starting from 1st form and proceeding to

  \[ MAx = Mb \]

  where \( MA \) is the matrix of the final form.
• Remind the first step of elimination results in
\[
\begin{pmatrix}
6 & -2 & 2 & 4 \\
0 & -4 & 2 & 2 \\
0 & -12 & 8 & 1 \\
0 & 2 & 3 & -14
\end{pmatrix} \cdot \begin{pmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4
\end{pmatrix} = \begin{pmatrix}
16 \\
-6 \\
-27 \\
-18
\end{pmatrix}
\]

• This step can be obtained by multiplying a lower triangular matrix "\(M_1\),"

\[M_1Ax = M_1b\]

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
-2 & 1 & 0 & 0 \\
-1/2 & 0 & 1 & 0 \\
1 & 0 & 0 & 1
\end{pmatrix}
\]

• In the similar manner

\[M_3M_2M_1Ax = M_3M_2M_1b\]

where \(M = M_3M_2M_1\).
• Since we found \( MA = U \),

\[
A = M^{-1}U = M_1^{-1}M_2^{-1}M_3^{-1}U = LU
\]

where

\[
L = \begin{pmatrix}
1 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 \\
1/2 & 0 & 1 & 0 \\
-1 & 0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 3 & 1 & 0 \\
0 & -1/2 & 0 & 1
\end{pmatrix} \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 2 & 1
\end{pmatrix} = \begin{pmatrix}
1 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 \\
1/2 & 3 & 1 & 0 \\
-1 & -1/2 & 2 & 1
\end{pmatrix}
\]

• Then we can easily verify

\[
LU = \begin{pmatrix}
1 & 0 & 0 & 0 \\
2 & 1 & 0 & 0 \\
1/2 & 3 & 1 & 0 \\
-1 & -1/2 & 2 & 1
\end{pmatrix} \begin{pmatrix}
6 & -2 & 2 & 4 \\
0 & -4 & 2 & 2 \\
0 & 0 & 2 & -5 \\
0 & 0 & 0 & -3
\end{pmatrix} = \begin{pmatrix}
6 & -2 & 2 & 4 \\
12 & -8 & 6 & 10 \\
3 & -13 & 9 & 3 \\
-6 & 4 & 1 & -18
\end{pmatrix} = A
\]

• Solving is easy since \( Ax = b \) (or \( LUx = b \)) can be separated into

\[
Lz = b \quad \text{and} \quad Ux = z.
\]

This form is useful for problems that involve the same matrix \( A \) and may differ ent right hand side vector \( b \).
Let us restate/recap the formal derivation of LU decomposition

• Now imagine an \( n \times n \) matrix \( M_{qp} \) which subtracts \( \lambda \) times row \( p \) from row \( q \). Assume that \( p < q \) (for in the naive algorithm, this is always true).

• Then the elements of \( M_{qp} = (m_{ij}) \) are

\[
m_{ij} = \begin{cases} 
1 & \text{if } i=j \\
-\lambda & \text{if } i=q \text{ and } j=p \\
0 & \text{otherwise}
\end{cases}
\]

• Therefore, the elements of \( M_{qp}A \) are given by

\[
(M_{qp}A)_{ij} = \sum_{s=1}^{n} m_{is} a_{sj} = \begin{cases} 
a_{ij} & \text{if } i \neq q \\
a_{qj} - \lambda a_{pj} & \text{if } i=q
\end{cases}
\]

The \( q \)-th row of \( M_{qp}A \) is the sum of the \( q \)-th row of \( A \) and \( \lambda \) times the \( p \)-th row of \( A \).
• The \( k \)-th step of Gaussian elimination corresponds to the matrix \( M_k \), which is the product of \( nk \) elementary matrices:

\[
M_k = M_{nk}M_{n1,k}...M_{k+1,k}
\]

Notice that each elementary matrix \( M_{ik} \) is lower triangular \((i > k)\), and therefore, \( M_k \) is also lower triangular.

• If we carry out the Gaussian forward elimination process on \( A \), the result will be an upper triangular matrix \( U \).

• On the other hand, the result is obtained by applying a succession of factors such as \( M_k \) to the left of \( A \). The entire process is summarized by writing

\[
M_{n-1}...M_2M_1A = U
\]

Since each \( M_k \) is invertible, we have

\[
A = M_1^{-1}M_2^{-1}...M_{n-1}^{-1}U.
\]

Each \( M_k \) is lower triangular having ones on its main diagonal (unit lower triangular). Each inverse \( M1 \) has the same property, and the same is true of their product. Hence, the matrix \( L = M_1^{-1}M_2^{-1}...M_{n-1}^{-1} \) is unit lower triangular, and we have \( A = LU \).
A pseudo code for LU decomposition called Doolittle factorization is given

- The forward elimination phase looks like

```plaintext
for k = 1 to n do
    l_{kk} ← 1
    for j = k to n do
        u_{kj} ← a_{kj} - \sum_{s=1}^{k-1} l_{ks}u_{sj}
    end for
    for i = k + 1 to n do
        l_{ik} ← \left( a_{ik} - \sum_{s=1}^{k-1} l_{is}u_{sk} \right) / u_{kk}
    end for
end for
```
• Now we solve the linear system using the relation $Ax = LUx = b$
  $\rightarrow Lz = b$ and $Ux = z$.

• The two phase of back substitution phase looks like

  1st phase
  
  $z_1 \leftarrow b_1$
  
  for $i = 2$ to $n$ do
  
  $z_1 \leftarrow b_i - \sum_{j=1}^{i-1} l_{ij} z_j$
  
  end for

  2nd phase

  $x_n \leftarrow z_n / u_{nn}$
  
  for $i = n - 1$ to $1$ do
  
  $x_i \leftarrow \left( z_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) / u_{ii}$
  
  end for
**1D Poisson equation can be solved by the iteration methods**

- Numerical discretization of 1D Poisson reads

\[
\Phi_{i-1} - 2\Phi_i + \Phi_{i+1} = -h^2 \bar{\rho}_i,
\]

where the subscript \( i \) is the index of the grid points. \(^a\)

- As before, Jacobi method reads

\[
\Phi^{(\text{new})}_i = (1/2) \left( \Phi^{(\text{old})}_{i-1} + \Phi^{(\text{old})}_{i+1} + h^2 \bar{\rho}_i \right),
\]

and Gauss-Seidel method reads

\[
\Phi^{(\text{new})}_i = (1/2) \left( \Phi^{(\text{new})}_{i-1} + \Phi^{(\text{old})}_{i+1} + h^2 \bar{\rho}_i \right),
\]

You can directly solve Eqs.(1) and (2) or use matrix. Note that Eqs.(1) and (2) are equivalent to solving the tri-diagonal band matrix equation.

\(^a\)Boundary conditions, for example, take \( \Phi(0) = 0 \) and \( \Phi(1) = 0 \) within the domain \( 0 \leq x \leq 1 \).
Succesive over relaxation (SOR) is yet another iteration method speeding up the convergence.

- First, let us realize the Jacobi and the Gauss-Seidel method can be casted into a general matrix form. Continue taking the notation of Poisson eq.

- In a matrix form, as before, Jacobi method reads

\[
\Phi_i^{(k)} = - \sum_{j=1, j \neq i}^n \left( \frac{a_{ij}}{a_{ii}} \right) \Phi_j^{(k-1)} + \left( \frac{b_i}{a_{ii}} \right),
\]

while the Gauss-Seidel method reads

\[
\Phi_i^{(k)} = - \sum_{j=1}^{i-1} \left( \frac{a_{ij}}{a_{ii}} \right) \Phi_j^{(k)} - \sum_{j=i+1}^n \left( \frac{a_{ij}}{a_{ii}} \right) \Phi_j^{(k-1)} + \left( \frac{b_i}{a_{ii}} \right),
\]

- Then, with a parameter \(0 < \omega < 2\), Succesive over relaxation (SOR) reads

\[
\Phi_i^{(k)} = \omega \left[ - \sum_{j=1}^{i-1} \left( \frac{a_{ij}}{a_{ii}} \right) \Phi_j^{(k)} - \sum_{j=i+1}^n \left( \frac{a_{ij}}{a_{ii}} \right) \Phi_j^{(k-1)} + \left( \frac{b_i}{a_{ii}} \right) \right] + (1 - \omega) \Phi_i^{(k-1)}
\]
As a reminder, the matrix equation for 1D Poisson solve is given by

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
. & . & . & . & . & . & . & . & . \\
0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\
. & . & . & . & . & . & . & . & . \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
\Phi_0 \\
\Phi_1 \\
. \\
\Phi_{i-1} \\
\Phi_i \\
. \\
\Phi_{i+1} \\
\Phi_{n-1} \\
\Phi_n
\end{pmatrix}
= 
\begin{pmatrix}
0 \\
h^2 \rho_1 \\
. \\
h^2 \rho_{i-1} \\
h^2 \rho_i \\
. \\
h^2 \rho_{i+1} \\
h^2 \rho_{n-1} \\
0
\end{pmatrix}
\]

The components multiplied by zero’s are practically of no use (note the computational memory for the storage).
Vector norms can be used for the checking the convergence

- $L_1$ vector norm:

$$||x||_1 = \sum_{i=1}^{n} |x_i|$$

- $L_2$ vector norm:

$$||x||_2 = \left( \sum_{i=1}^{n} x_i^2 \right)^{1/2}$$

- $L_\infty$ vector norm:

$$||x||_\infty = \max_{1 \leq i \leq n} |x_i|$$
A pseudo code for Gauss-Seidel method is given

- In Gauss-Seidel, if \( x^{(k-1)} \) is not saved, then we can dispense with the superscripts in the pseudocode as follows:

\[
\text{for } k = 1 \text{ to } k_{\text{max}} \text{ do }
\text{for } i = 1 \text{ to } n \text{ do }
\quad x_i \leftarrow \frac{b_i - \sum_{j=1, j \neq i}^{n} a_{ij} x_j}{a_{ii}}
\text{end for}
\text{end for}
\]

- Note that the convergence criteria (e.g. L2-norm) can be included into iteration process process.
**Ising model simulation is made available in C**

- Take a look at isingc.tar of the homepage.
- Let us remind ”magnets”. Ferromagnetic and anti-ferromagnetic (MnO or NiO).
- Curie temperatures: Iron (Fe) 770 C, Cobalt (Co) 1130 C, Nickel (Ni) 358 C.
- Flipping the coins with \( s > \epsilon \) (0, 0.05, 0.2).

- Long range correlation:
  \[
  C = \frac{\sum S_i^{(0)} S_i}{\left( \sum S_i^{(0)} S_i^{(0)} \right)}
  \]

- Short range correlation can be considered as well. Consider the correlation of only the locally neighboring spins:
  \[
P(k) = \left[ \frac{1}{M(k)} \right] \sum_{<i,j>}^k < 1 - S_i S_j >
  \]
Ising model considers spin Hamiltonian

- The Hamiltonian is given by
  \[ H = -\sum_{i,j} \varepsilon_{ij} S_i S_j = -\varepsilon \sum_{i,j} S_i S_j \]

- With external magnetic field
  \[ H = -\varepsilon \sum_{i,j} S_i S_j - B \sum_i S_i \]

- Spins take on values of ±1.

- Nature finds its own way to minimum energy state.

- If \( \varepsilon > 0 \), then when \( S \cdot S > 1 \), Hamiltonian will be negative (preferred) \( \rightarrow \) Ferromagnetic.

- If \( \varepsilon < 0 \), then when \( S \cdot S < 1 \), Hamiltonian will be negative \( \rightarrow \) Anti-ferromagnetic.
Ising model (2) with Monte-Carlo method

• The probability (lower the better) of a state given by a Hamiltonian $H_m$ is given by
  \[ f_m = \exp \left(-\frac{H_m}{T}\right) \]

• The basic is to one spin and compare it to old ($H_1$) and new ($H_2$) state. If $H_2$ is more probable, then change it to that new spin. This can be given by
  \[ \frac{f_2}{f_1} = \frac{\exp \left(-\frac{H_2}{T}\right)}{\exp \left(-\frac{H_1}{T}\right)} = \exp \left[\left(\frac{H_1 - H_2}{T}\right)\right] > s \]
  with $0 \leq s \leq 1$ being a random number, which is nothing but the Metropolis algorithm.
Ising model simulation at various $\varepsilon$

- Here, $\varepsilon$ represents the ratio between spin and thermal energy.

- Let us estimate the $H_1 - H_2$ term. Note that each spin has four neighbors. Denoting $S_0$ as the center and $S_1$ to $S_4$ as four neighbors,

\[
H_1 - H_2 = -\varepsilon S_0^{(1)} (S_1 + S_2 + S_3 + S_4) - [-\varepsilon S_0^{(2)} (S_1 + S_2 + S_3 + S_4)]
\]

\[
= -2\varepsilon S_0^{(1)} (S_1 + S_2 + S_3 + S_4) = -2\varepsilon S_0^{(1)} g
\]

since $S_0^{(1)} = -S_0^{(2)}$.

- Practically there are only five values of $g$ that are $\pm 4$, $\pm 2$, and 0.

- In the examples below we prepared $10 \times 10$ spins, and start flipping the spins from $i_x = 1, i_y = 1, i_x = 2, i_y = 1 ...$ and $i_x = 10, i_y = 10$, and repeat that process. Note we take periodic boundary conditions.
• Case with $\varepsilon = 1$. 
Case with $\varepsilon = 0.25$. Thermal fluctuation is dominant over spin energy.
Phase transition can be seen at Curie temperature

- Correlation like function $C = (\sum S)/N$ is estimated for different values of $\varepsilon$. 

![Graph of $C$ against $t$ for different $\varepsilon$ values](image)
Summary of discussions

• Explicit vs implicit in diffusion equation.

• Tri-diagonal banded matrix.

• LU decomposition.

• Thursday: some more exercise. Conjugate gradient method (end of linear algebra).

• Possible Mid-term exam on May 3rd?
Use subroutines and functions to modularize your program

- When the program becomes long we can use subroutines (unique to f90) and functions (both for c and f90).

- module and use (f90)

- dynamical allocation

- We can use Makefile to organize codes.\(^\text{a}\)

- Note that excessive use of subroutines and functions can slow down computing.

\(^\text{a}\)make remembers when last modified the source codes. If not modified uses the old object file.
Good practices for programming

- Use **Pseudocode** and **Modularize**: if you are not sure about the implementation of your algorithm, first cut them into pieces. You can never write a long code from scratch without trial and error. **Show intermediate results**.

- **Generalize**: a little effort can help you in the future.

- Comment for yourself not really for somebody else but you will read it in the future (but not too excessive).

- **Declare all variables**: implicit none (f90) : can avoid mistype.

- **Use clean loops**: Loop indentation.

- **Mathematical constants**: Use precision dependent
  \[ \pi = \text{acos}(-1.0) \text{ or } e = \text{exp}(1.0). \]
A few more practice for programming
(personal experiences)

- Don’t generate integers from reals.
- Correctness first, efficiency second.
- Variables names should be concise. It’s mathematics after all.
- Open editor and console separately.

**Most importantly:** Is your calculation correct?

- Use physical intuition (energy conservation for example) Physically and mathematically.
- **Visualize.**

- Ironically, manual writers are slow in producing science.