Heat Equation / Diffusion Equation

Partial differential equation for $u(x,t)$ in one space dimension and time.

$u$ represents temperature in a 1-dimensional metal rod, for example.

Or concentration (density) of a chemical diffusing in a tube of water.

The PDE is

$$u_t(x,t) = Du_{xx}(x,t) + f(x,t)$$

where subscripts represent partial derivatives,

$D =$ diffusion coefficient,

$f(x,t) =$ source term.

Steady state diffusion

If $f(x,t) = f(x)$ does not depend on time and if the boundary conditions don't depend on time, then $u(x,t)$ will converge towards steady state distribution satisfying

$$0 = Du_{xx}(x) + f(x)$$

(by setting $u_t = 0$.)

This is now an ordinary differential equation (ODE) for $u(x)$.

We can solve this on an interval, say $0 \leq x \leq 1$ with

Boundary conditions:

$$u(0) = \alpha, \quad u(1) = \beta.$$
Steady state diffusion

More generally: Take $D = 1$ or absorb in $f$,

$$u_{xx}(x) = -f(x) \quad \text{for } 0 \leq x \leq 1,$$

Boundary conditions:

$$u(0) = \alpha, \quad u(1) = \beta.$$

Can be solved exactly if we can integrate $f$ twice and use boundary conditions to choose the two constants of integration.

Example: $f(x) = 0, \quad \alpha = 20, \quad \beta = 60$:

Solution: $u(x) = \alpha + x(\beta - \alpha)$.

No heat source $\implies$ linear variation in steady state ($u_{xx} = 0$).

Example: $f(x) = 100e^x, \quad \alpha = 20, \quad \beta = 60$:

Solution: $u(x) = -100e^x + (100e - 60)x + 120$.

For more complicated equations, numerical methods must generally be used, giving approximations at discrete points.
Finite difference method

Define grid points \( x_i = i\Delta x \) in interval \( 0 \leq x \leq 1 \), where
\[
\Delta x = \frac{1}{n+1}
\]
So \( x_0 = 0, \ x_{n+1} = 1 \), and the \( n \) grid points \( x_1, x_2, \ldots, x_n \) are equally spaced inside the interval.

Let \( U_i \approx u(x_i) \) denote approximate solution.

We know \( U_0 = \alpha \) and \( U_{n+1} = \beta \) from boundary conditions.

Idea: Replace differential equation for \( u(x) \) by system of \( n \) algebraic equations for \( U_i \) values (\( i = 1, 2, \ldots, n \)).

\[
U_i \approx u(x_i)
\]
\[
u_x(x_{i+1/2}) \approx \frac{U_{i+1}-U_i}{\Delta x}
\]
\[
u_x(x_{i-1/2}) \approx \frac{U_i-U_{i-1}}{\Delta x}
\]

So we can approximate second derivative at \( x_i \) by:
\[
u_{xx}(x_i) \approx \frac{1}{\Delta x^2} \left( \frac{U_{i+1}-U_i}{\Delta x} - \frac{U_i-U_{i-1}}{\Delta x} \right)
\]
\[
= \frac{1}{\Delta x^2} (U_{i-1} - 2U_i + U_{i+1})
\]

This gives coupled system of \( n \) linear equations:
\[
\frac{1}{\Delta x^2} (U_{i-1} - 2U_i + U_{i+1}) = -f(x_i)
\]
for \( i = 1, 2, \ldots, n \). With \( U_0 = \alpha \) and \( U_{n+1} = \beta \).

Tridiagonal linear system

For \( n = 5 \):

\[
\begin{bmatrix}
-2 & 1 & 0 & 0 & 0 \\
1 & -2 & 1 & 0 & 0 \\
0 & 1 & -2 & 1 & 0 \\
0 & 0 & 1 & -2 & 1 \\
0 & 0 & 0 & 1 & -2
\end{bmatrix}
\begin{bmatrix}
U_1 \\
U_2 \\
U_3 \\
U_4 \\
U_5
\end{bmatrix}
= -\Delta x^2
\begin{bmatrix}
f(x_1) \\
f(x_2) \\
f(x_3) \\
f(x_4) \\
f(x_5)
\end{bmatrix}
- \begin{bmatrix}
\alpha \\
0 \\
0 \\
0 \\
\beta
\end{bmatrix}.
\]

General \( n \times n \) system requires \( O(n^3) \) flops to solve.

Tridiagonal \( n \times n \) system requires \( O(n) \) flops to solve.

Could use LAPACK routine \texttt{dgtsv}.
Heat equation in 2 dimensions

One-dimensional equation generalizes to

$$u_t(x, y, t) = D(u_{xx}(x, y, t) + u_{yy}(x, y, t)) + f(x, y, t)$$

on some domain in the $x$-$y$ plane, with initial and boundary conditions.

We will only consider rectangle $0 \leq x \leq 1$, $0 \leq y \leq 1$.

**Steady state problem (with $D = 1$):**

$$u_{xx}(x, y) + u_{yy}(x, y) = -f(x, y)$$

Finite difference equations in 2D

Let $U_{ij} \approx u(x_i, y_j)$.

Replace differential equation

$$u_{xx}(x, y) + u_{yy}(x, y) = -f(x, y)$$

by algebraic equations

$$\frac{1}{\Delta x^2} (U_{i-1,j} - 2U_{i,j} + U_{i+1,j}) + \frac{1}{\Delta y^2} (U_{i,j-1} - 2U_{i,j} + U_{i,j+1}) = -f(x_i, y_j)$$

If $\Delta x = \Delta y = h$:

$$\frac{1}{h^2} (U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}) = -f(x_i, y_j).$$

On $n \times n$ grid ($\Delta x = \Delta y = 1/(n + 1)$) this gives a linear system of $n^2$ equations in $n^2$ unknowns.

The above equation must be satisfied for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$.

Matrix is $n^2 \times n^2$.

- e.g. on 100 by 100 grid, matrix is $10,000 \times 10,000$.
- Contains $(10,000)^2 = 100,000,000$ elements.

Matrix is sparse: each row has at most 5 nonzeros out of $n^2$ elements! But structure is no longer tridiagonal.
Finite difference equations in 2D

Matrix has block tridiagonal structure:

\[ A = \frac{1}{h^2} \begin{bmatrix} T & I & & & \\ I & T & I & & \\ & I & T & I & \\ & & I & T & \\ & & & & I \end{bmatrix} \]

\[ T = \begin{bmatrix} -4 & 1 & & & \\ 1 & -4 & 1 & & \\ & 1 & -4 & 1 & \\ & & 1 & -4 & \\ & & & & 1 \end{bmatrix} \]

Iterative methods

Back to one space dimension first...

Coupled system of linear equations:

\[(U_i - 2U_i + U_{i+1}) = -\Delta x^2 f(x_i)\]

for \(i = 1, 2, \ldots, n\). With \(U_0 = \alpha\) and \(U_{m+1} = \beta\).

Iterative method starts with initial guess \(U^{[0]}\) to solution and then improves \(U^{[k]}\) to get \(U^{[k+1]}\) for \(k = 0, 1, \ldots\).

**Note:** Generally does not involve modifying matrix \(A\).

Do not have to store matrix \(A\) at all, only know about stencil.

Jacobi iteration

\[(U_i - 2U_i + U_{i+1}) = -\Delta x^2 f(x_i)\]

**Solve for** \(U_i\):

\[ U_i = \frac{1}{2} \left( U_{i-1} + U_{i+1} + \Delta x^2 f(x_i) \right). \]

**Note:** With no heat source, \(f(x) = 0\), the temperature at each point is average of neighbors.

Suppose \(U^{[k]}\) is a approximation to solution. Set

\[ U_i^{[k+1]} = \frac{1}{2} \left( U_{i-1}^{[k]} + U_{i+1}^{[k]} + \Delta x^2 f(x_i) \right) \quad \text{for} \quad i = 1, 2, \ldots, n. \]

**Repeat** for \(k = 0, 1, 2, \ldots\) until convergence.

Can be shown to converge (eventually... very slow!)
Slow convergence of Jacobi
Iterative methods

Jacobi iteration is about the worst possible iterative method. But it’s very simple, and useful as a test for parallelization.

Better iterative methods:

- Gauss-Seidel
- Successive Over-Relaxation (SOR)
- Conjugate gradients
- Preconditioned conjugate gradients
- Multigrid

Speedup for problems like steady state heat equation

Source: SIAM Review

Iterative methods – initialization

! allocate storage for boundary points too:
allocate(x(0:n+1), u(0:n+1), f(0:n+1))

dx = 1.d0 / (n+1.d0)

!$omp parallel do
do i=0,n+1
! grid points:
x(i) = i*dx
! source term:
f(i) = 100.*exp(x(i))
! initial guess (linear function):
u(i) = alpha + x(i)*(beta-alpha)
enddo
Jacobi iteration in Fortran

```fortran
uold = u ! starting values before updating
do iter=1,maxiter
  dumax = 0.d0
  do i=1,nu(i)
    u(i) = 0.5d0*(uold(i-1) + uold(i+1) + dx**2*f(i))
    dumax = max(dumax, abs(u(i)-uold(i)))
  enddo
  ! check for convergence:
  if (dumax .lt. tol) exit
  uold = u ! for next iteration
enddo
```

**Note:** we must use old value at \( i - 1 \) for Jacobi. Otherwise we get the Gauss-Seidel method.

Jacobi with OpenMP

```fortran
parallel do (fine grain)
```

```fortran
uold = u ! starting values before updating
do iter=1,maxiter
  dumax = 0.d0
  !$omp parallel do reduction(max : dumax)
  do i=1,n
    u(i) = 0.5d0*(uold(i-1) + uold(i+1) + dx**2*f(i))
    dumax = max(dumax, abs(u(i)-uold(i)))
  enddo
  ! check for convergence:
  if (dumax .lt. tol) exit
  !$omp parallel do
  do i=1,n
    uold(i) = u(i) ! for next iteration
  enddo
enddo
```

**Note:** Forking threads twice each iteration.

Jacobi with OpenMP – coarse grain

**General Approach:**
- Fork threads only once at start of program.
- Each thread is responsible for some portion of the arrays, from \( i=i_{\text{start}} \) to \( i=i_{\text{end}} \).
- Each iteration, must copy \( u \) to \( uold \), update \( u \), check for convergence.
- Convergence check requires coordination between threads to get global \( dumax \).
- Print out final result after leaving parallel block.

See code in the repository or the notes:
$CLASSHG/codes/openmp/jacobi2_omp.f90
Jacobi with MPI

Each process is responsible for some portion of the arrays, from \( i=\text{istart} \) to \( i=\text{iend} \).

**No shared memory:** each process only has part of array.

**Updating formula:**
\[
u(i) = 0.5d0 \times (u_{\text{old}}(i-1) + u_{\text{old}}(i+1) + dx^2f(i))
\]

Need to exchange values at boundaries:
- Updating at \( i=\text{istart} \): requires \( u_{\text{old}}(\text{istart}-1) \)
- Updating at \( i=\text{iend} \): requires \( u_{\text{old}}(\text{istart}+1) \)

Example with \( n = 9 \) interior points (plus boundaries):

- Process 0 has \( \text{istart} = 1, \text{iend} = 5 \)
- Process 1 has \( \text{istart} = 6, \text{iend} = 9 \)

![Process intervals](chart.png)

Other issues:
- Convergence check requires coordination between processes to get global \( \text{dumax} \).
  Use `MPI_ALLREDUCE` so all process check same value.
- Part of final result must be printed by each process (into common file `heatsoln.txt`), in proper order.

See code in the repository or the notes:
`$CLASSHG/codes/mpi/jacobi2_mpi.f90`

Notes:

**Note:** Non-blocking `mpi_isend` is used.

Different tags (1 and 2) for left-going, right-going messages.
Jacobi with MPI — Receiving from neighbors

Note: \( u_{\text{old}}(\text{istart}) \) from \( \text{me}+1 \) goes into \( u_{\text{old}}(\text{iend}+1) \):
\[ \text{uold(iend) from me-1 goes into uold(istart-1)}: \]

\[
\begin{align*}
\text{do iter = 1, maxiter} & \\
& \text{! mpi_send’s from previous slide} \\
& \text{if (me < ntasks-1) then} \\
& \quad \text{! Receive right endpoint value} \\
& \quad \text{call mpi_recv(uold(iend+1), 1, MPI_DOUBLE_PRECISION, \&me + 1, 1, MPI_COMM_WORLD, mpistatus, ierr)} \\
& \quad \text{end if} \\
& \text{if (me > 0) then} \\
& \quad \text{! Receive left endpoint value} \\
& \quad \text{call mpi_recv(uold(istart-1), 1, MPI_DOUBLE_PRECISION, \&me - 1, 2, MPI_COMM_WORLD, mpistatus, ierr)} \\
& \quad \text{end if} \\
& \text{! Apply Jacobi iteration on my section of array} \\
& \text{do i = istart, iend} \\
& \quad \text{u(i) = 0.5d0*(uold(i-1) + uold(i+1) + dx**2*f(i))} \\
& \quad \text{dumax_task = max(dumax_task, abs(u(i) - uold(i)))} \\
& \end{align*}
\]

Jacobi with MPI — Writing solution in order

Want to write table of values \( x(i), u(i) \) in heatsoln.txt.

Need them to be in proper order, so Process 0 must write to this file first, then Process 1, etc.

Approach:

Each process \( \text{me} \) waits for a message from \( \text{me-1} \) indicating that it has finished writing its part. (Contents not important.)

Each process must open the file (without clobbering values already there), write to it, then close the file.

Assumes all processes share a file system!

On cluster or supercomputer, need to either: send all results to single process for writing, or write distributed files that may need to be combined later (some visualization tools handle distributed data!)