Laplace’s Equation

Heat
- Energy (in the form of heat) flows from areas of high energy toward areas of lower energy
- Example: metal plate
- flow is orthogonal to isothermal contours
- rate of flow depends on thermal conductivity of the material

Steady State
- If the temperatures at the boundaries are constant, temperatures at interior points will eventually reach a steady state

Mathematical Model
- A set of equations that describe heat flow:
  \[ q_x = -k \frac{\partial T}{\partial x} \quad q_y = -k \frac{\partial T}{\partial y} \]
- \( q \) is a symbol for flux
- \( k \) is the thermal conductivity constant
- flow is related to the change in temperature with respect to position
Laplace’s Equation

- The second derivative describes the change in flux:
  \[
  \frac{\partial q_x}{\partial x} = \frac{\partial^2 T}{\partial x^2} \quad \frac{\partial q_y}{\partial y} = \frac{\partial^2 T}{\partial y^2}
  \]

- When the system is in a steady state the change in flux is zero:
  \[
  \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0
  \]

  - heat is flowing, but at the same rate everywhere on the surface
  - this second-order equation is Laplace’s equation

Boundary Value Problem

- In the previous lecture this model was introduced as an example of a boundary value problem
  - values of temperature are known at boundaries
  - goal is to compute temperatures at internal points

- The basic method will be the same as in the advection model
  - divide the domain into grid cells
  - derive a discrete form of the PDE model

- Big difference: we need a new algorithm
  - advection is an initial value problem
  - explicit methods worked in that model
  - here we need an implicit method

Small Example

- To illustrate the implicit algorithms, we’ll use a very simple example*
  - Goal: a method for computing \( T(i,j) \)
  - Temperature at grid point \((i,j)\)
  - Recall we use spatial coordinates, not array indices
  - \( T(i,j) \): temp at \( x = i, \ y = j \)

  ![Diagram](image)

* from Heath, Scientific Computing, 1997

Discretization

- A second-order difference approximation for Laplace’s equation:
  \[
  \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta x)^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta y)^2} = 0
  \]

  - begin with Taylor series
  - expand to include second derivative
  - this is a centered difference equation

  - Note changes in \( x \) (top line) and changes in \( y \) (bottom line)
Equation for \( T(i,j) \)

- Rearrange so \( T(i,j) \) is defined in terms of values at the other grid cells
  
  \[
  \frac{[T_{i+1,j} - 2T_{i,j} + T_{i-1,j}]}{(\Delta x)^2} + \frac{[T_{j+1} - 2T_{j,j} + T_{j-1,j}]}{(\Delta y)^2} = 0
  \]
  \[
  T_{i,j} = \frac{(\Delta y)^2(T_{i+1,j} + T_{i-1,j}) + (\Delta x)^2(T_{j+1} + T_{j,j+1})}{4}
  \]

- To make calculations easier, build a grid with \( \Delta x = \Delta y \)
- For the rest of the slides, assume \( \Delta x = 1 \)

\[
T_{i,j} = \frac{T_{i+1,j} + T_{i-1,j} + T_{j+1} + T_{j,j+1}}{4}
\]

Equations for Grid Points

\[
T_{i,j} = \frac{T_{i+1,j} + T_{i-1,j} + T_{j+1} + T_{j,j+1}}{4}
\]

\[
T_{1,1} = \frac{[0 + T_{2,1} + T_{1,2} + 0]}{4}
\]

\[
T_{1,2} = \frac{[0 + T_{2,2} + 1 + T_{1,1}]}{4}
\]

\[
T_{2,1} = \frac{[T_{1,1} + 0 + T_{2,2} + 0]}{4}
\]

\[
T_{2,2} = \frac{[T_{1,2} + 0 + 1 + T_{2,1}]}{4}
\]

Matrix Form

- Rearrange each equation so the variables are on the left side
  - write variables in the same order, and line up the columns
  - move constants to the right side

\[
T_{1,1} = \frac{[0 + T_{2,1} + T_{1,2} + 0]}{4} \quad 4T_{1,1} - T_{2,1} - T_{1,2} = 0
\]

\[
T_{1,2} = \frac{[0 + T_{2,2} + 1 + T_{1,1}]}{4} \quad -T_{1,1} + 4T_{2,1} - T_{2,2} = 0
\]

\[
T_{2,1} = \frac{[T_{1,1} + 0 + T_{2,2} + 0]}{4} \quad -T_{1,1} + 4T_{1,2} - T_{2,2} = 1
\]

\[
T_{2,2} = \frac{[T_{1,2} + 0 + 1 + T_{2,1}]}{4} \quad -T_{2,1} - T_{1,2} + 4T_{2,2} = 1
\]

Matrix Form (cont’d)

- Now the problem can be written in the form of a matrix-vector product

\[
\begin{pmatrix}
4 & -1 & -1 & 0 \\
-1 & 4 & 0 & -1 \\
-1 & 0 & 4 & -1 \\
0 & -1 & -1 & 4
\end{pmatrix}
\begin{pmatrix}
T_{1,1} \\
T_{2,1} \\
T_{1,2} \\
T_{2,2}
\end{pmatrix}
= \begin{pmatrix}
0 \\
0 \\
1 \\
1
\end{pmatrix}
\]
Solution

- Use Gaussian elimination ("row reduction") to find values of the variables that satisfy the equation

\[
\begin{bmatrix}
T_{1,1} & 0.125 \\
T_{2,1} & 0.125 \\
T_{1,2} & 0.375 \\
T_{2,2} & 0.375
\end{bmatrix}
\]


Time Complexity

- Gaussian elimination provides an exact value for each grid point, but it is expensive
- With an \( N \times N \) grid:
  - number of variables to solve for: \( v = N^2 \)
  - time complexity: \( O(v^2) \)
- Each row will have at most four non-zero values
- there are efficient algorithms for sparse matrices

Iterative Methods

- Another way to solve the system of equations is to use an iterative method
- Our goal is to solve \( A \cdot x = b \)
  - \( A \) is the coefficient matrix, normalized so the diagonal is all 1’s
  - \( x \) is the vector of variables
  - \( b \) is the vector of constants

\[
A = \begin{bmatrix}
1 & -0.25 & -0.25 & 0 \\
-0.25 & 1 & 0 & -0.25 \\
-0.25 & 0 & 1 & -0.25 \\
0 & -0.25 & -0.25 & 1
\end{bmatrix}
\]

Iterative Methods (cont’d)

- In an iterative method, start with an initial estimate of the solution
  - initial set of values for the variables: \( x^{(0)} \)
- Define two new matrices \( M \) and \( N \), so that

\[
M \cdot x^{(i+1)} = N \cdot x^{(i)} + b
\]

- Solve this equation for \( x^{(i+1)} \)
- With the right choice of \( M \) and \( N \) the new vector will be closer to the actual solution \( x^{(i)} \)
- Eventually the sequence will converge to the actual solution

\[
|x^{(i+1)} - x^{(0)}| < \varepsilon
\]
Jacobi Iteration

- In Jacobi’s method, split the coefficient matrix $A$ into three parts:

\[
M \cdot x^{(i+1)} = N \cdot x^{(i)} + b
\]

\[
M = D
datastructure
N = U + L
\]

- The iterative step is simple, because $D = I$

\[
Dx^{(i+1)} = (D - A)x^{(i)} + b
\]

Well, Duh

- What this new equation says is that the value at a grid cell is the average of the values at the neighboring cells

\[
T_{i,j}^{(n+1)} = \left[ T_{i-1,j}^{(n)} + T_{i+1,j}^{(n)} + T_{i,j-1}^{(n)} + T_{i,j+1}^{(n)} \right] / 4
\]

- BUT:
  - this calculation is not a simulation of heat flow
    - there is no time dimension
    - we don’t have a thermal conductivity to represent the rate of flow
  - this equation is part of an iterative solution of a system of linear equations

Other Iterative Methods

- Why Jacobi’s equation works, and the sorts of systems for which it eventually converges, is a topic for another class

- Efficiency of iterative methods depends on the number of iterations

- Other methods, requiring fewer iterations:
  - Gauss-Seidel
  - Selective Over-Relaxation (SOR)
  - See books on linear algebra or PDEs for more information (and for pointers to many other iterative methods)

- All use different forms of matrices $M, N$, but have the same basic structure
Space Complexity

There are two big advantages of iterative methods:

- they use far less space
- $O(n^2)$ for an $n \times n$ grid
- compared to $O(n^4)$ for Gaussian elimination or other exact solution

It is possible to get an approximate solution after a few iterations

- in some situations an exact answer isn’t required

Outline

Algorithm:

- use two matrices, $T^1$ and $T^2$
- put initial guesses (i.e. $x^{(0)}$) in $T^1$
- repeat until $|T^1 - T^2| < \varepsilon$
  - on odd steps, use $T^1$ to update $T^2$
  - on even steps, use $T^1$ to update $T^2$

To Be Continued...

Next time:

- efficient implementation in C++
- array allocation
- OpenMP for parallel implementations