Numerical PDEs

- perhaps a larger subfield of numerical analysis than all the rest combined!

Unlike IVPS for ODEs where we came up with a general purpose method (i.e., VSVO based on blended linear multistep methods) method that will solve any problem in that class, for PDE’s we need to design a new method for every kind of problem (roughly speaking).

- Moreover, engineers, scientists and mathematicians come up with new kinds of problems all the time.

- Even more so than for ODEs, you want to look out for the ideas and principles.

- We already encountered these issues when studying ODEs:
  - Discretization
  - Local versus global accuracy
  - Consistency
  - Stability
  - Convergence
• New Issues, in addition to those mentioned above, for PDEs include:

- Usually $t$ is time, and $x, y, z$ are space coordinates. (For ODEs we rarely worried about the meaning of the variables.)

- The geometry of the underlying domain can present huge problems. (Examples: modeling an airplane or an internal combustion engine).

- mixture of initial and boundary conditions

- unknown or moving boundaries

- traveling waves (shocks)

- Solving large linear and nonlinear systems efficiently.

- reproducing qualitative behavior (like periodicity, dissipation, dispersion), in addition to plain accuracy (small global truncation error), for the actual DE as opposed to a simple test equation.

- The source of a PDE (and some ODE-BVPs) is often a minimization principle (like minimizing potential energy). Often one can solve the PDE by approaching the minimization principle directly.
The Method of Lines

- As mentioned above, for PDEs every problem is separate and special.

- However, the closest thing to a general purpose method is the **Method of Lines**

- Discretize in space and integrate in time! Consider the PDE

\[
    u_t = f(t, x, u, u_x, u_{xx}) \tag{1}
\]

where

\[
    u = u(t, x), \quad t \geq 0, \quad x \in [0, 1], \tag{2}
\]

\[
    u(0, x) = g(x) \tag{3}
\]

and

\[
    u(0, t) = \phi_0(t), \quad u(1, t) = \phi_1(t). \tag{4}
\]

- The problem (1)—(4) is a **second order one-dimensional initial-boundary value problem**. The equations (4) are boundary conditions, and the equation (3) is an initial condition.

- We could have more space variables, with little modification.

- We’ll be more specific about the function \( f \) later.
Figure 1. Method of Lines.

- Figure 1 shows the lines (in red). The vertical axis indicates time ($t$) and the horizontal axis indicates space ($x$). The black vertical lines form the boundary of the domain of $u$. (Time always almost always goes up in PDEs).

- we discretize in space:

$$x_n = nh, \quad n = 0, \ldots N, \quad h = \frac{1}{N}$$
and define functions
\[ u_n(t) \approx u(t, x_n) \]
by the IVP of ODEs:
\[ u'_n(t) = f \left( t, x_n, u_n, \frac{u_{n+1} - u_{n-1}}{2h}, \frac{u_{n+1} - 2u_n + u_{n-1}}{h^2} \right) \]
where
\[ n = 1, 2, \ldots, N - 1 \] (6)
and
\[ u_0(t) = \phi_0(t), \quad u_N(t) = \phi_1(t), \quad u_n(0) = g(x_n). \] (7)

- This approach is also called **semi-discretization**.
- The IVP (5)—(7) is a special tridiagonal system of \( N - 1 \) ODES and can in principle be solved by any ODE method, including a VSVO method like lsode.
- to gain more insight we specialize the function \( f \) and consider the **one-dimensional heat equation**
\[ u_t = u_{xx} \] (8)
which described the temperature in a rod.
- The ODEs (5) turn into
\[ u'_n(t) = \frac{1}{h^2} \left( u_{n+1}(t) - 2u_n(t) + u_{n-1}(t) \right). \]
• Suppose we solve this system with the
  1. Euler
  2. Backward Euler
  3. Trapezoidal
     Methods.

• We need a notation for the time step. The
  conventional choice is $k$ (since $h$ is taken for
  the space discretization parameter).

• We also use superscripts to denote the time
  level.

• Thus
  \[ U_n^m \approx u_n(t_m) \]
  where
  \[ t_m = mk, \quad n = 1, \ldots, N-1, \quad \text{and} \quad m = 0, 1, 2, \ldots. \]

• Moreover, we define the grid constant
  \[ r = \frac{k}{h^2} \]

• With every method we associate a stencil
  which indicates all points entering into the
  basic formula.
Euler’s Method

Figure 2. Stencil for Euler’s Method.

- In the ODE context Euler’s Method is

\[ y_{n+1} = y_n + hf_n. \]

Translated into our new context this becomes

\[
U_{n+1}^m = U_n^m + \frac{k}{h^2} \left( U_{n+1}^m - 2U_n^m + U_{n-1}^m \right)
= U_n^m + r \left( U_{n+1}^m - 2U_n^m + U_{n-1}^m \right)
\]

- Figure 2 shows the stencil for Euler’s Method.
- Euler’s Method is explicit.
Backward Euler Method

Figure 3. Stencil for the Backward Euler Method.

• In the ODE context the Backward Euler Method is
  \[ y_{n+1} = y_n + hf_{n+1}. \]
  Translated into our new context this becomes
  \[ U_{n+1}^m = U_n^m + r(U_{n+1}^{m+1} - 2U_n^{m+1} + U_{n-1}^{m+1}) \]

• Figure 3 shows the stencil for the Backward Euler Method.

• The Backward Euler Method is **implicit**. At every step we have to solve a tridiagonal linear system.
The Crank-Nicolson Method

Figure 4. Stencil for the Crank-Nicolson Method.

- The Trapezoidal Rule applied to the IVP (5)—(7) is usually called the **Crank-Nicolson** Method. Note that “Nicolson” is spelled without the letter “h”.

- In the ODE context the Trapezoidal Rule is given by

  \[ y_{n+1} = y_n + \frac{h}{2}(f_n + f_{n+1}). \]

  Translated into our new context this becomes

  \[ U_{n+1}^m = U_n^m + \frac{\tau}{2}(U_{n+1}^m - 2U_n^m + U_{n-1}^m + U_{n+1}^{m+1} - 2U_n^{m+1} + U_{n-1}^{m+1}) \]

- Figure 4 shows the stencil for the Crank-Nicolson Method.

- Just like the Backward Euler Method the Crank-Nicolson Method is **implicit** and requires the solution of a tridiagonal linear system at every time step.
• We can now ask the same questions as we did for ODEs, i.e.,
  – Local Truncation Error
  – Stability
  – Convergence

**Local Truncation Error**

• Let’s consider Euler’s Method.