Outline

Some Distinguishing Features of PDEs

- Interaction of scales.
  - e.g., often cannot let $\Delta x \longrightarrow 0$ unless $\Delta t \longrightarrow 0$ *fast enough.*
- Size of the systems.
  - Time-dependent: solve $A\mathbf{x} = \mathbf{b}$ $N_{step} \gg 1$ times.
  - Multiple space dimensions, $d > 1$: $A \in \mathbb{R}^{n \times n}$
    - $n = N^d$, $N :=$ number of points in *each* direction.
    - System bandwidth is $O(N^{d-1}) \gg 1$.
    - Systems are typically *sparse.*
    - Iterative solvers important, particularly for $d > 2$.

Partial Differential Equations

- *Partial differential equations* (PDEs) involve partial derivatives with respect to more than one independent variable
- Independent variables typically include one or more space dimensions and possibly time dimension as well
- More dimensions complicate problem formulation: we can have pure initial value problem, pure boundary value problem, or mixture of both
- Equation and boundary data may be defined over irregular domain

Partial Differential Equations, continued

- For simplicity, we will deal only with single PDEs (as opposed to systems of several PDEs) with only two independent variables, either
  - two space variables, denoted by x and y, or
  - one space variable denoted by x and one time variable denoted by t
- Partial derivatives with respect to independent variables are denoted by subscripts, for example
  - $u_t = \partial u / \partial t$
  - $u_{xy} = \partial^2 u / \partial x \partial y$

Classification of PDEs

- *Order* of PDE is order of highest-order partial derivative appearing in equation
- For example, advection equation is first order $u_t = -c\,u_x$
- Important second-order PDEs include
  - *Heat equation*: $u_t = u_{xx}$
  - *Wave equation*: $u_{tt} = u_{xx}$
  - *Laplace equation*: $u_{xx} + u_{yy} = 0$

Classification of PDEs, continued

- Second-order linear PDEs of general form

$$au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu + g = 0$$

  are classified by value of *discriminant* $b^2 - 4ac$

- $b^2 - 4ac > 0$: *hyperbolic* (e.g., wave equation)
- $b^2 - 4ac = 0$: *parabolic* (e.g., heat equation)
- $b^2 - 4ac < 0$: *elliptic* (e.g., Laplace equation)

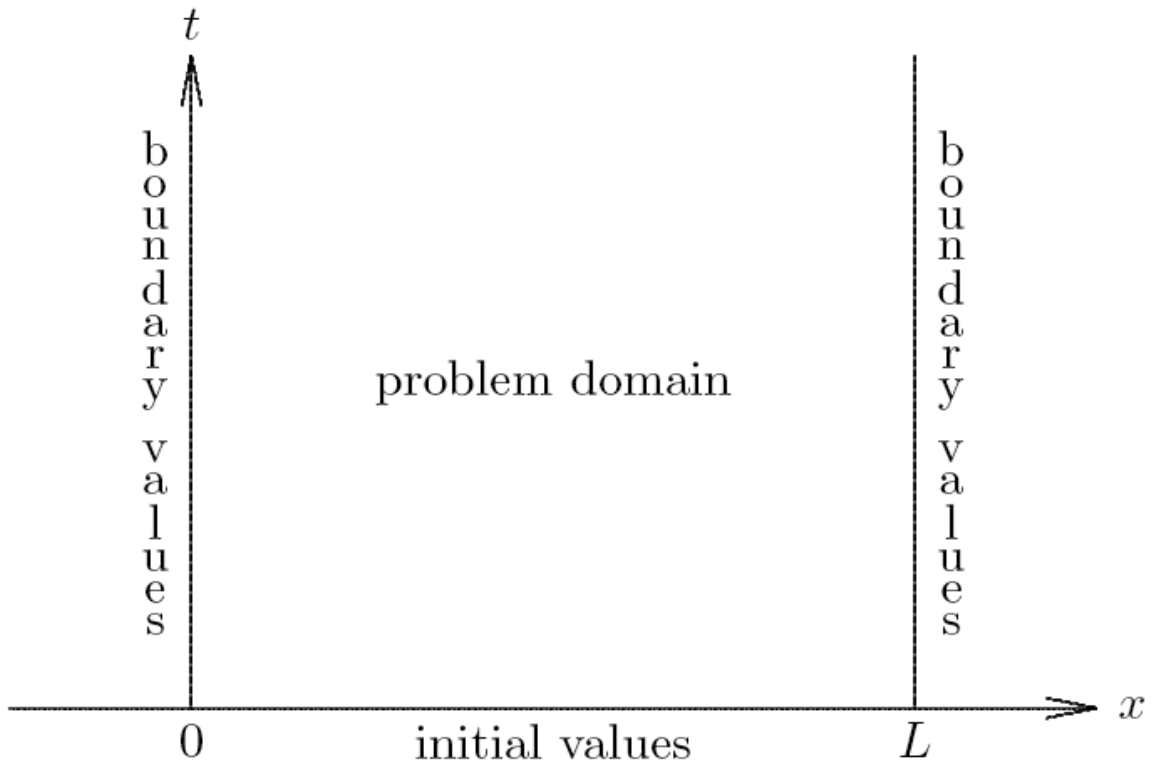
Classification of PDEs, continued

Classification of more general PDEs is not so clean and simple, but roughly speaking

- *Hyperbolic* PDEs describe time-dependent, conservative physical processes, such as convection, that *are not* evolving toward steady state
- *Parabolic* PDEs describe time-dependent, dissipative physical processes, such as diffusion, that *are* evolving toward steady state
- *Elliptic* PDEs describe processes that have already reached steady state, and hence are time-independent

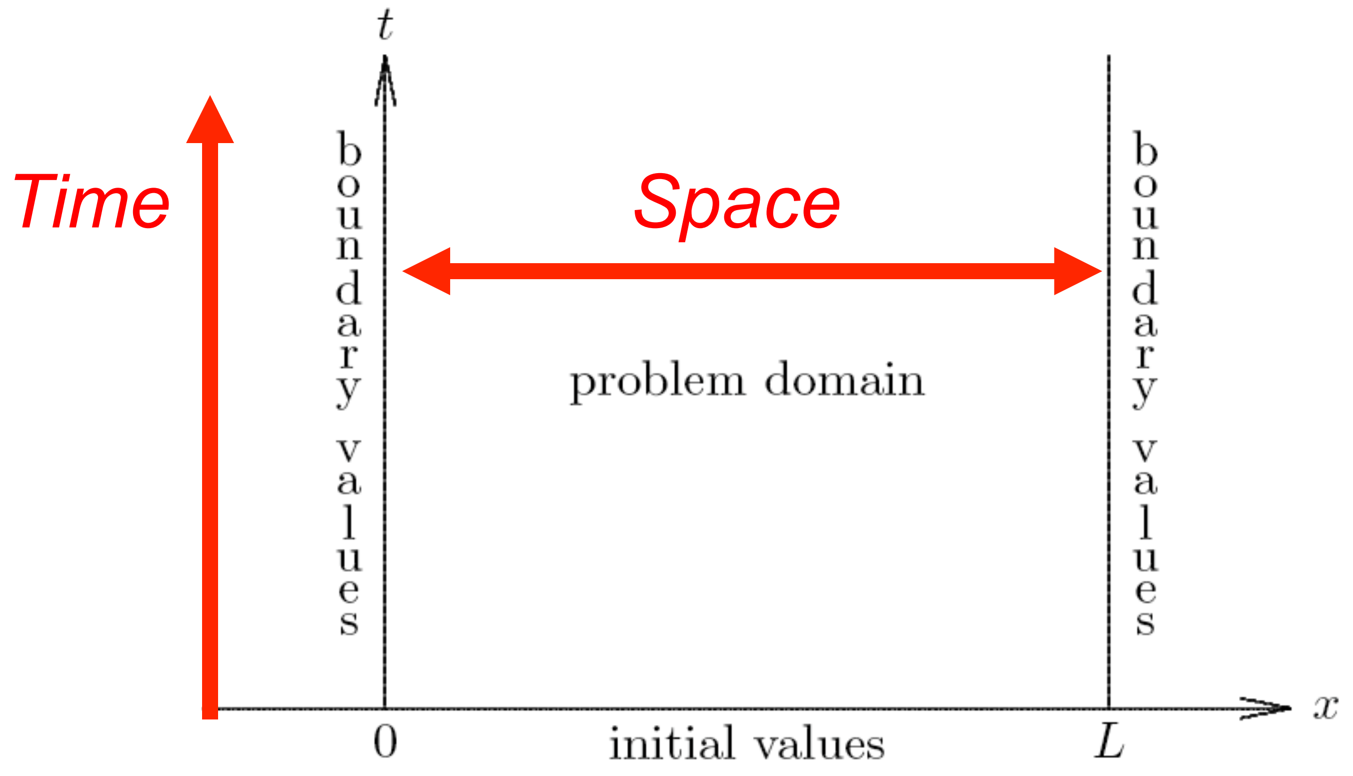
Time-Dependent Problems

- Time-dependent PDEs usually involve both initial values and boundary values



Time-Dependent Problems

- Time-dependent PDEs usually involve both initial values and boundary values



Example: Advection Equation

- *Advection equation*

$$u_t = -c\, u_x$$

  where c is nonzero constant

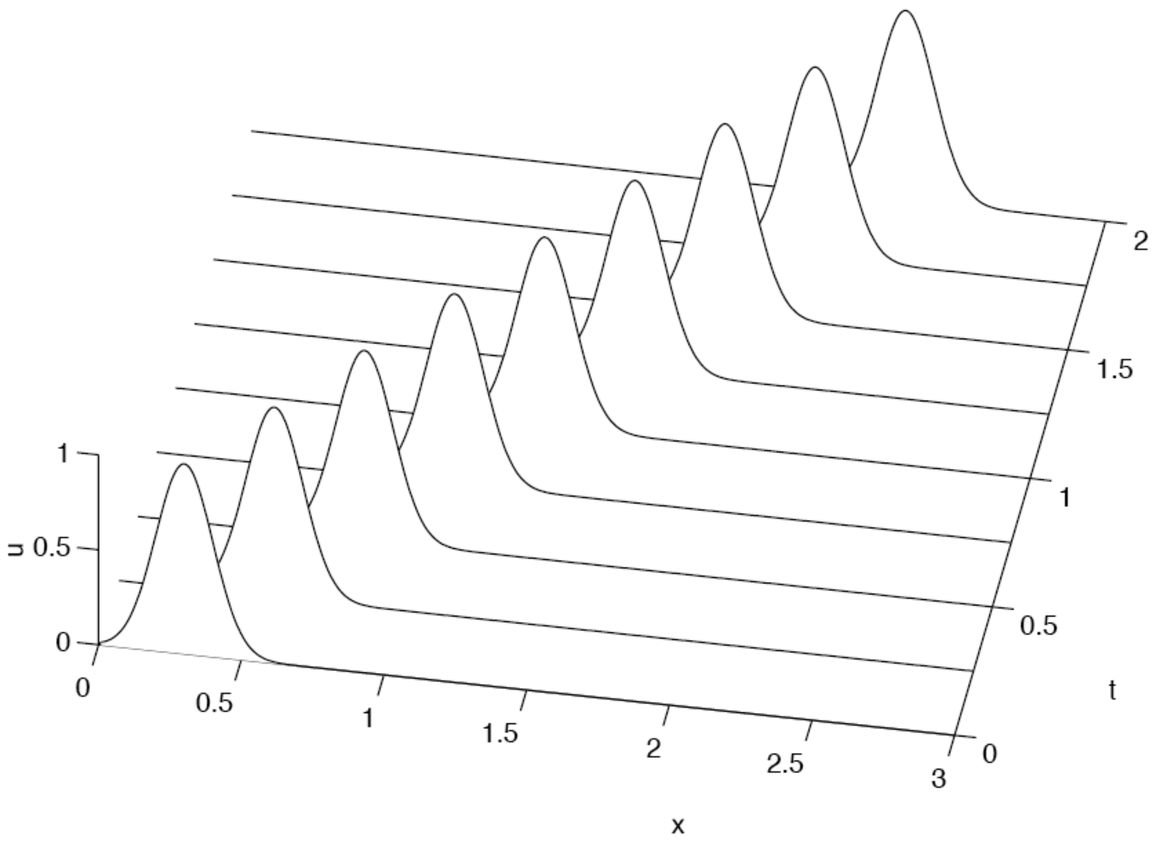
- Unique solution is determined by initial condition

$$u(0, x) = u_0(x), \qquad -\infty < x < \infty$$

  where u_0 is given function defined on $\mathbb{R}$

- We seek solution $u(t, x)$ for $t \geq 0$ and all $x \in \mathbb{R}$
- From chain rule, solution is given by $u(t, x) = u_0(x - c\,t)$
- Solution is initial function u_0 shifted by $c\,t$ to right if $c > 0$, or to left if $c < 0$

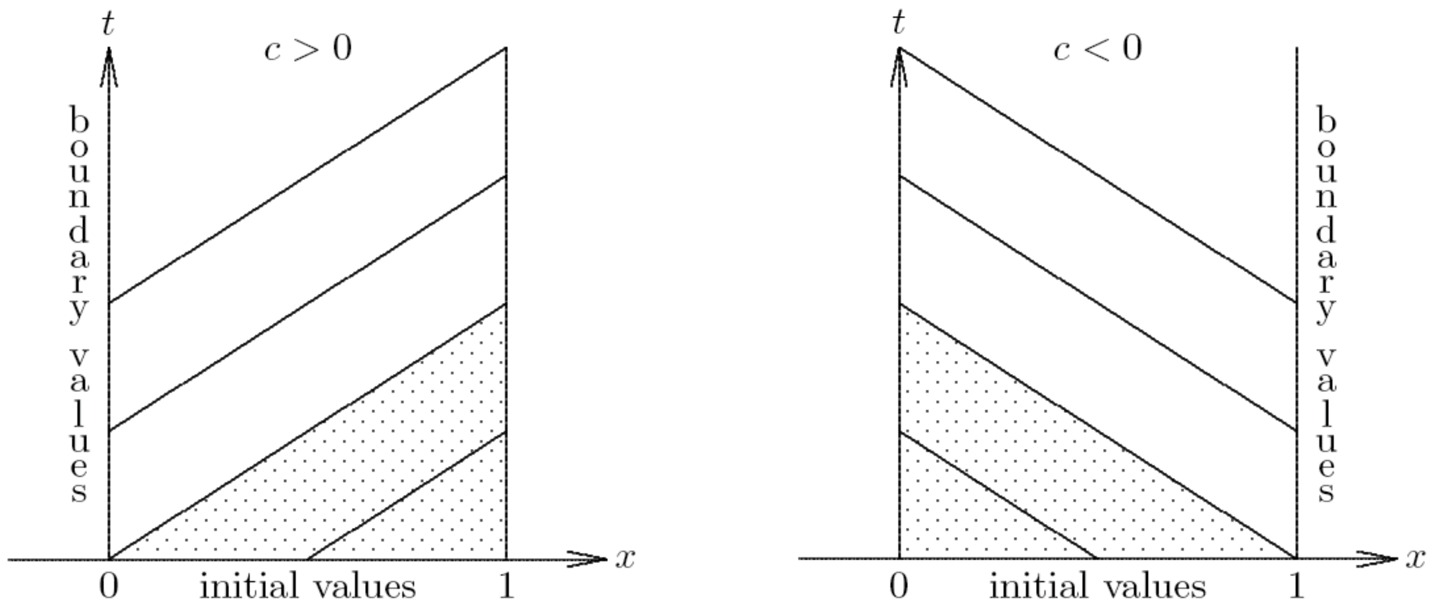
Example, continued



Typical solution of advection equation, with initial function "advected" (shifted) over time < interactive example >

Characteristics

- *Characteristics* for PDE are level curves of solution
- For advection equation $u_t = -c\, u_x$, characteristics are straight lines of slope $1/c$

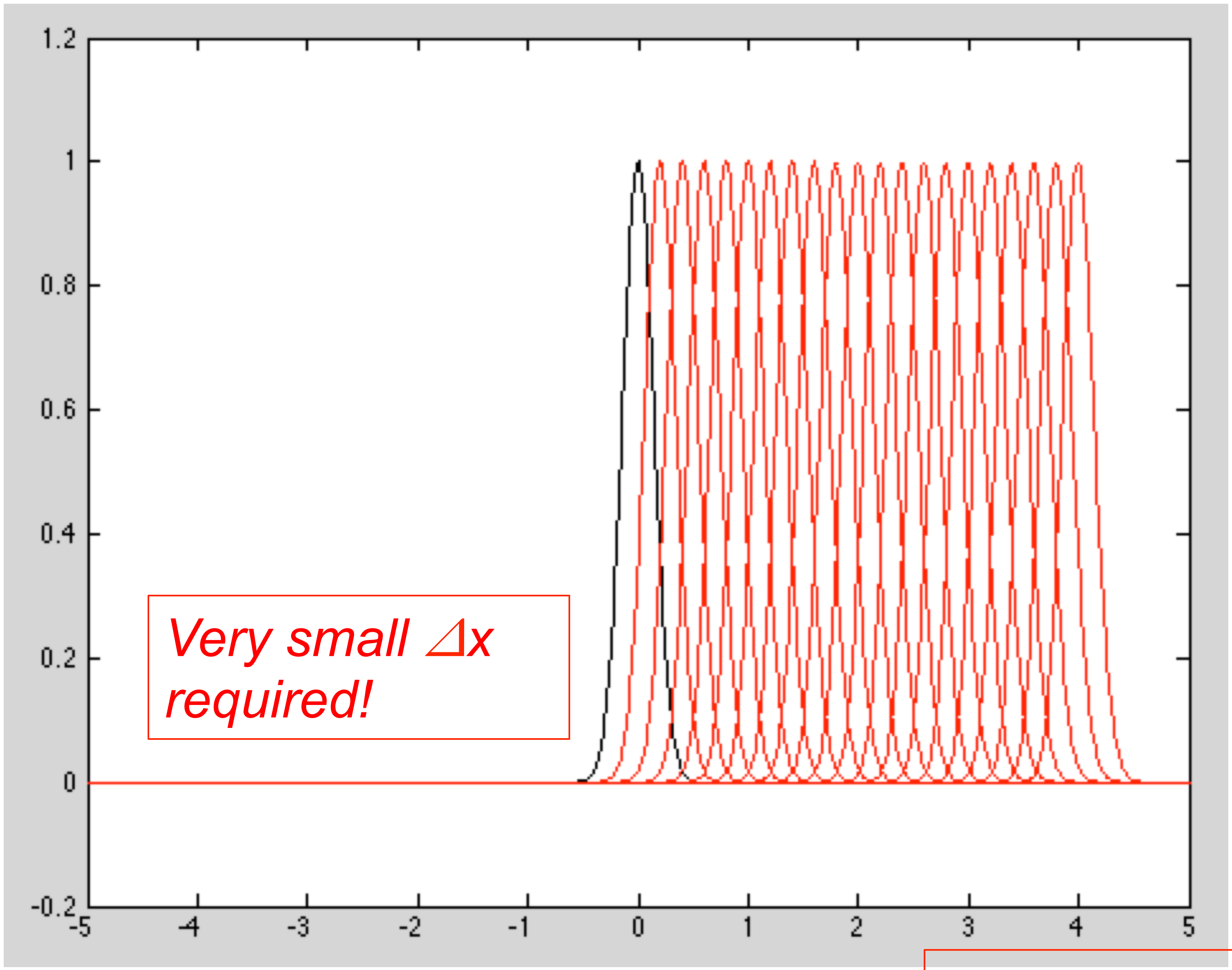


- Characteristics determine where boundary conditions can or must be imposed for problem to be well-posed

Matlab Demo: Convection

```
c=1; Tf = 4;  % Final time
x0=-5; xn=5;

dx = .01; x=x0:dx:xn; x=x'; n=length(x);

a = -1; b=0; c=1; e = ones(n,1);
C = spdiags([a*e b*e c*e],-1:1, n,n); C = C/(2*dx);

C(n,n)=-C(n,n-1); C(1,1)=C(1,2); % To drain energy at bdry

CFL = 0.50; dt  = CFL*dx/abs(c); nsteps = Tf/dt;

u=exp(-x.*x/.04); hold off; plot(x,u,'k-'); hold on;
f=0*u;f1=0*u;
io=floor(nsteps/20); kk=0; t=0;
for k=1:nsteps; t=t+dt;

    if k==1; c0=1; c1=0; c2=0;                      end;
    if k==2; c0=3/2; c1=-1/2; c2=0;                 end;
    if k==3; c0=23/12; c1=-16/12; c2=5/12;          end;

    f2=f1; f1=f; f= -C*u;

    rhs = c0*f + c1*f1 + c2*f2;
    u   = u+dt*rhs;

    if mod(k,io)==0; plot(x,u,'r-'); pause(.2); end;
end;
```

Matlab Demo: Convection



Very small Δx required!

conv_ab3_cd2.m

Time Stepping for Advection Equation: $\frac{\partial u}{\partial t} = -c\frac{\partial u}{\partial x}$

- Unlike the diffusion equation, which smears out the initial condition (with high wavenumber components decaying particularly fast), the advection equation simply moves things around, with no decay.
- This property is evidenced by the spatial operator having purely (or close to purely) imaginary eigenvalues.
- Preserving high-wavenumber content (in space) for all time makes this problem particularly challenging.
  - There is always some spatial discretization error.
  - Its effects accumulate over time (with no decay of the error).
  - For sufficiently large final time T any fixed grid (i.e., fixed n) simulation for general problems will eventually have too much error.
  - Long time-integrations, therefore, typically require relatively fine meshes and/or high-order spatial discretizations.

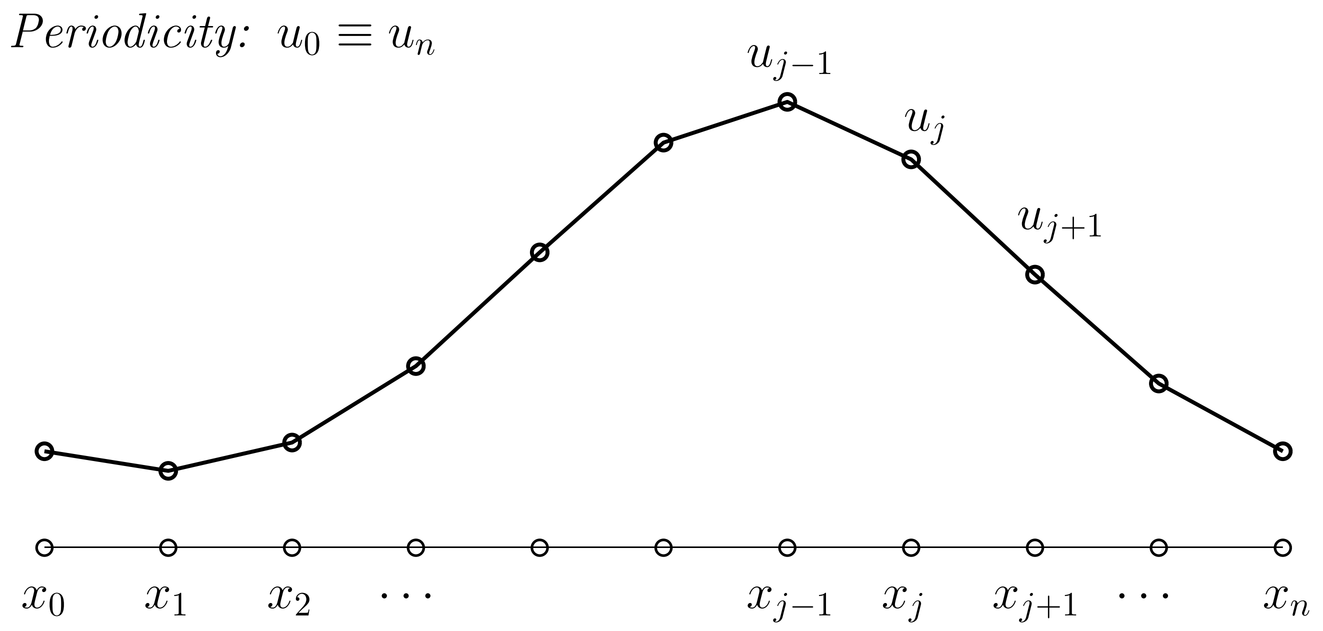
CFL, Eigenvalues, and Stability: Fourier Analysis

- Consider: $u_t = -cu_x, \quad u(0) = u(1)$ (periodic BCs)
- Centered difference formula in space:

$$\frac{du_j}{dt} = -\frac{c}{2\Delta x}(u_{j+1} - u_{j-1}) = C\,\underline{u}\big|_j$$

$$C = -\frac{1}{2\Delta x}\underbrace{\begin{bmatrix} 0 & 1 & & & -1 \\ -1 & 0 & 1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ 1 & & & -1 & 0 \end{bmatrix}}_{\textit{Periodic Matrix}}$$

Periodic Domain



- Allows us to run for long times without having to have a very long domain.
- Allows us to analyze the properties of our spatial discretization.

CFL, Eigenvalues, and Stability: Fourier Analysis

- Consider: $u_t = -cu_x, \quad u(0) = u(1)$ (periodic BCs)
- Centered difference formula in space:

$$\frac{du_j}{dt} = -\frac{c}{2\Delta x}\left(u_{j+1} - u_{j-1}\right) = C\,\underline{u}\big|_j$$

- Eigenvector: $u_j = e^{i2\pi k x_j}$.
- Eigenvalue:

$$\begin{aligned} C\,\underline{u}\big|_j &= -\frac{c}{2\Delta x}\left(e^{i2\pi k\Delta x} - e^{-i2\pi k\Delta x}\right) e^{i2\pi k x_j} \\ &= -\frac{2ic}{2\Delta x}\frac{\left(e^{i2\pi k\Delta x} - e^{-i2\pi k\Delta x}\right)}{2i} u_j \\ &= \lambda_k u_j \end{aligned}$$

$$\lambda_k = \frac{-ic}{\Delta x}\sin(2\pi k\Delta x)$$

CFL, Eigenvalues, and Stability: Fourier Analysis

- Eigenvalue:

$$
\begin{aligned}
C\,\underline{u}\big|_j &= -\frac{c}{2\Delta x}\left(e^{i2\pi k\Delta x} - e^{-i2\pi k\Delta x}\right) e^{i2\pi k x_j} \\
&= -\frac{2ic}{2\Delta x}\,\frac{\left(e^{i2\pi k\Delta x} - e^{-i2\pi k\Delta x}\right)}{2i}\,u_j \\
&= \lambda_k\, u_j
\end{aligned}
$$

$$
\lambda_k = \frac{-ic}{\Delta x}\sin(2\pi k\Delta x)
$$

- Eigenvalues are purely imaginary, max modulus is

$$
\max_k |\lambda_k| = \frac{|c|}{\Delta x}
$$

- For constant c and Δx, we define the CFL for the advection equation as

$$
\mathrm{CFL} = \frac{\Delta t|c|}{\Delta x}.
$$

Courant Number

Courant Number, Eigenvalues, and Stability: Fourier Analysis

- For constant c and Δx, we define the CFL for the advection equation as
$$\mathrm{CFL} = \frac{\Delta t |c|}{\Delta x}.$$
- CFL=1 would correspond to a timestep size where a particle moving at speed c would move one grid spacing in a single timestep.
- For centered finite differences in space, CFL=1 also corresponds $\lambda \Delta t = 1$.
- From our IVP stability analysis, we know that we need $|\lambda \Delta t| < .7236$ for AB3 and < 2.828 for RK4.
- This would correspond to CFL $< .7236$ and 2.828, respectively.

CFL, Eigenvalues, and Stability: Fourier Analysis

- MATLAB EXAMPLE: conv_ab3.m

Advection

- For advection, no decay in physical solution.
- Solution is *persistent*.
- Numerical method is either dispersive, dissipative, or both.
- If $C = -C^T$, discrete operator is skew-symmetric (imaginary eigenvalues) and numerical method has *no decay* (due to spatial error, at least).
- But it *will* be dispersive.
- We come back to dissipative shortly.

- Long time-integration $\longrightarrow$ accumulation of error.
- Second-order, $O(\Delta x^2)$, accuracy is *not* sufficient.
- Modulo boundary conditions (or with periodicity), we can easily extend our 2nd-order centered-difference formula to $O(\Delta x^4)$ through Richardson extrapolation.
- Let

  $$C_h \mathbf{u}\big|_j \;:=\; \frac{c}{2\Delta x}\left[u_{j+1} - u_{j-1}\right]$$

  and

  $$C_{2h} \mathbf{u}\big|_j \;:=\; \frac{c}{4\Delta x}\left[u_{j+2} - u_{j-2}\right]$$

  for $j = 1, \ldots, n$ (with wrap for periodic ends).

- Instead of

$$\frac{d\mathbf{u}}{dt} = -C_h\mathbf{u}$$

now use

$$\frac{d\mathbf{u}}{dt} = -\left[\frac{4}{3}C_h\mathbf{u} - \frac{1}{3}C_{2h}\mathbf{u}\right].$$

- For AB3, say,

$$\begin{aligned}\mathbf{u}^{k+1} &= \mathbf{u}^k + \Delta t\left(\frac{23}{12}\mathbf{f}^k - \frac{16}{12}\mathbf{f}^{k-1} + \frac{5}{12}\mathbf{f}^{k-2}\right)\\ \mathbf{f}^k &= -\left[\frac{4}{3}C_h\mathbf{u}^k - \frac{1}{3}C_{2h}\mathbf{u}^k\right].\end{aligned}$$

- Don't re-evaluate $\mathbf{f}^{k-1}$ or $\mathbf{f}^{k-2}$.
- Just re-use the previously computed values.

conv_ab3_cd4.m

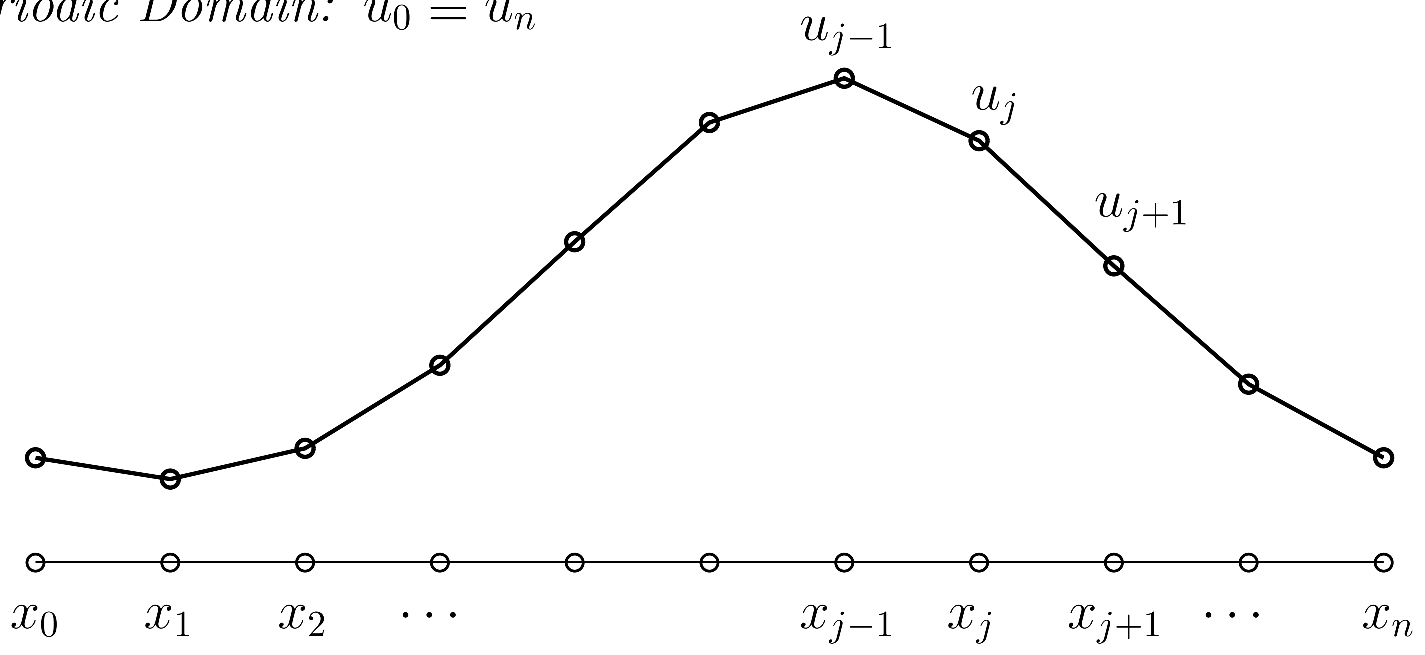
Numerical Dissipation

Numerical Dissipation

- So far, we've consider only central difference formulas.
- Upwind discretizations offer more stability, through the introduction of numerical dissipation.
- You must be very careful about the wind direction!

Alternative Discretizations for Advection

Periodic Domain: $u_0 \equiv u_n$



- First-order upwinding:

$$\begin{aligned}\frac{du_j}{dt} &= -\frac{c}{\Delta x}\left(u_j - u_{j-1}\right) \quad \text{if } c > 0,\\ \frac{du_j}{dt} &= -\frac{c}{\Delta x}\left(u_{j+1} - u_j\right) \quad \text{if } c < 0.\end{aligned}$$

- Questions:
  - What is the order of accuracy?
  - Do we preserve skew-symmetry?
  - Do we have stability?
  - Under which conditions?

- Consider $c > 0$. With some rearranging, we find:

$$\begin{aligned}
\frac{du_j}{dt} &= -\frac{c}{\Delta x}\left(u_j - u_{j-1}\right) \\
&= -\frac{c}{2\Delta x}\left(2u_j - 2u_{j-1}\right) \\
&= -\frac{c}{2\Delta x}\left(u_{j+1} - u_{j+1} + 2u_j - 2u_{j-1}\right) \\
&= -\frac{c}{2\Delta x}\left((u_{j+1} - u_{j-1}) + (-u_{j+1} + 2u_j - u_{j-1})\right) \\
&= -c\frac{u_{j+1} - u_{j-1}}{2\Delta x} + \frac{c\Delta x}{2}\frac{-u_{j+1} + 2u_j - u_{j-1}}{\Delta x^2} \\
&= -C\mathbf{u} - \nu_h A\mathbf{u}.
\end{aligned}$$

- Here, $\nu_h = \frac{c\Delta x}{2}$ is the *numerical diffusivity* and the term

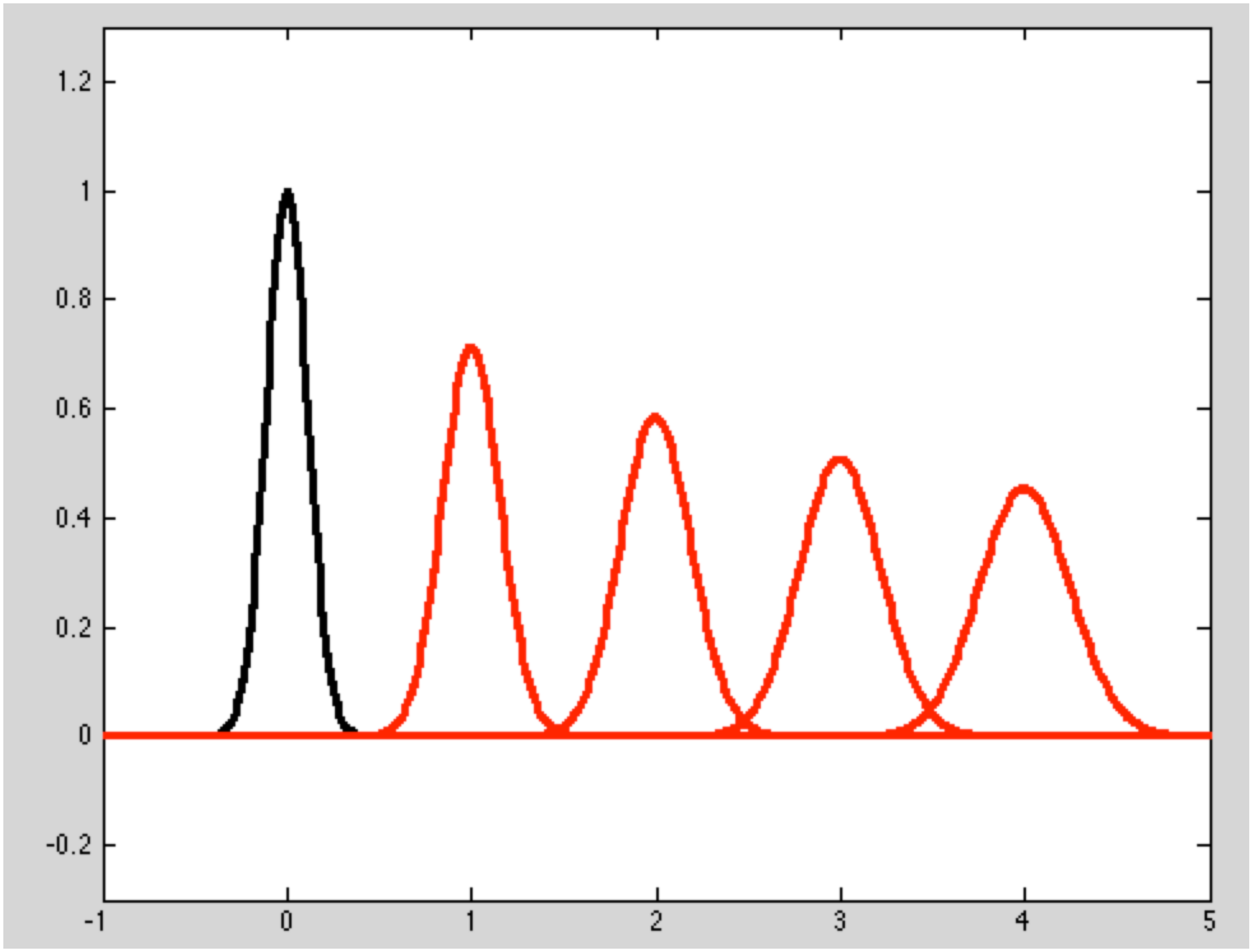
$$-\nu_h A\mathbf{u}$$

represents *numerical dissipation.*

- $\nu_h = \frac{c\Delta x}{2} \longrightarrow 0$ as $\Delta x \longrightarrow 0$ (but only linearly in Δx).

- This method is thus first-order, $O(\Delta x)$, accurate in space and *dissipative.*

conv_ab3_b.m demo



conv_ab3_b.m demo

- **Eigenvalues.**
- For our *periodic* boundary conditions, the eigenvectors are

$$u_j \;=\; e^{i2\pi k x_j} \qquad (i := \sqrt{-1}).$$

- With $\theta := 2\pi k \Delta x$, we have:

$$\begin{aligned}
C\mathbf{u} &= \frac{c}{2\Delta x} \cdot 2i \left[\frac{e^{i\theta} \;-\; e^{-i\theta}}{2i} \right] e^{i2\pi k x_j} \\
&= \frac{ic}{\Delta x} \sin(2\pi k \Delta x)\, e^{i2\pi k x_j}. \\
\nu_h A\mathbf{u} &= \frac{\nu_h}{\Delta x^2} \left[2 \;-\; 2\cos(2\pi k \Delta x) \right] e^{i2\pi k x_j} \\
\lambda(J) &= -\frac{ic}{\Delta x} \sin(2\pi k \Delta x) \;-\; \frac{\nu_h}{\Delta x^2} \left(2 \;-\; 2\cos(2\pi k \Delta x) \right).
\end{aligned}$$

- **Eigenvalues.**
- For our *periodic* boundary conditions, the eigenvectors are

$$u_j \;=\; e^{i2\pi k x_j} \qquad (i := \sqrt{-1}).$$

- With $\theta := 2\pi k \Delta x$, we have:

$$\begin{aligned}
C\mathbf{u} &= \frac{c}{2\Delta x} \cdot 2i \left[\frac{e^{i\theta} \;-\; e^{-i\theta}}{2i} \right] e^{i2\pi k x_j} \\
&= \frac{ic}{\Delta x} \sin(2\pi k \Delta x)\, e^{i2\pi k x_j}. \\
\nu_h A\mathbf{u} &= \frac{\nu_h}{\Delta x^2} \left[2 \;-\; 2\cos(2\pi k \Delta x) \right] e^{i2\pi k x_j} \\
\lambda(J) &= \underbrace{-\frac{ic}{\Delta x} \sin(2\pi k \Delta x)}_{\in\, \mathcal{I}m} \;-\; \underbrace{\frac{\nu_h}{\Delta x^2} \left(2 \;-\; 2\cos(2\pi k \Delta x) \right)}_{<\,0,\; \in\, \mathbb{R}}.
\end{aligned}$$

- Thus, the eigenvalues are complex and in the left (stable) half of the complex plane.

- **Q:** What happens if $c < 0$??

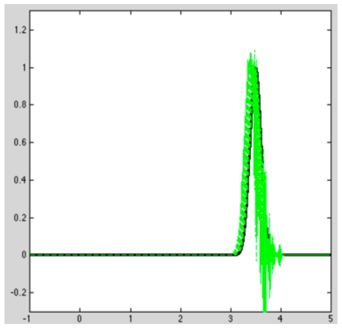
- Now, $\nu_h < 0$ and

$$\lambda(J) = \underbrace{-\frac{ic}{\Delta x}\sin(2\pi k \Delta x)}_{\in\,\mathcal{I}m} - \underbrace{\frac{\nu_h}{\Delta x^2}\left(2 - 2\cos(2\pi k \Delta x)\right).}_{>0,\ \in\,\mathbb{R}}$$

- Here, we will have very rapid instability.
- We must in this case use the one-sided derivative

$$\frac{du_j}{dt} = -\frac{c}{\Delta x}\left(u_{j+1} - u_j\right) \quad \text{if } c < 0. \qquad (1)$$

- Consider the logic of this statement.



conv_ab3_bb.m demo

- Suppose we use Euler forward, with $c = 1 > 0$ and $\Delta t = \Delta x$.
- Then, the update step is

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = J\mathbf{u}^n, \quad \text{or} \tag{2}$$

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = -c\frac{u_j^n - u_{j-1}^n}{\Delta x}, \tag{3}$$

  implying

$$u_j^{n+1} = u_j^n - \frac{c\Delta t}{\Delta x}\left(u_j^n - u_{j-1}^n\right). \tag{4}$$

- If our CFL = 1, then

$$u_j^{n+1} = u_{j-1}^n, \tag{5}$$

  which corresponds to a perfect shift of data from the left.

- Being in Illinois, we take our prediction of tomorrow's weather, u_j^{n+1}, from today's weather in Iowa, u_{j-1}^n.
- Not from Indiana (u_{j+1}^n).

Numerical Dispersion: Finite Differences

- Analytical advection, periodic boundary conditions:

$$u_t \;=\; -c\,u_x \;=\; \mathcal{L}u, \qquad\qquad u(0) \;=\; u(1),$$

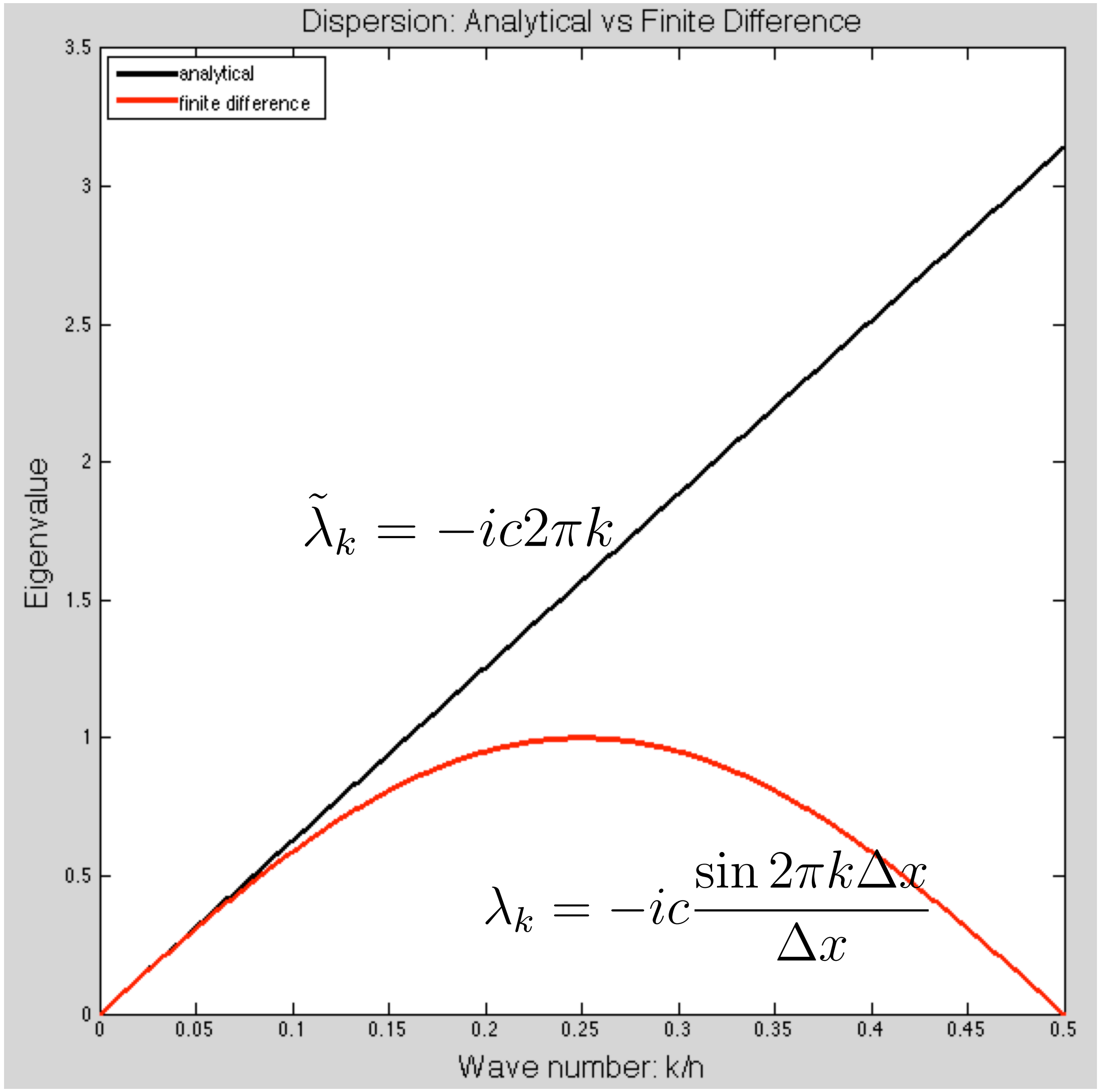
- Analytical eigenfunctions: $s_k(x) \;=\; e^{i2\pi kx}$.

- Analytical eigenvalues. Set $u(x, t = 0) \;=\; u_0 \;=\; s_k(x)$. Then,

$$\begin{aligned} \mathcal{L}u &= -c\,u_x \;=\; -ic2\pi k\,u \;=\; \tilde{\lambda}_k u \\ \tilde{\lambda}_k &= -ic2\pi k \end{aligned}$$

- Numerical eigenvalues:

$$\lambda_k \;=\; -ic\frac{\sin 2\pi k\Delta x}{\Delta x} \;\sim\; -ic2\pi k\left(1 - \frac{(2\pi k\Delta x)^2}{3!}\cdots\right).$$



$$\tilde{\lambda}_k = -ic2\pi k$$

$$\lambda_k = -ic\frac{\sin 2\pi k\Delta x}{\Delta x}$$

Advection Demos

- Numerical dispersion (spatial error dominating)
  - High wavenumber components have more error than low wavenumbers
  - Longer time-integrations yield larger error

- Instability if CFL too large
  - CFL < .72 (say, for AB3) for standard 2nd-order finite difference scheme
  - Smaller CFL required for 4th-order finite difference scheme

- First-order upwinding:
  - unstable if the wind direction ("c") changes without changing the stencil (e.g., $u_j - u_{j-1}$ to $u_{j+1} - u_j$).
  - Perfect shift if CFL == 1
  - Dissipative if CFL < 1
  - Explosive if CFL > 1

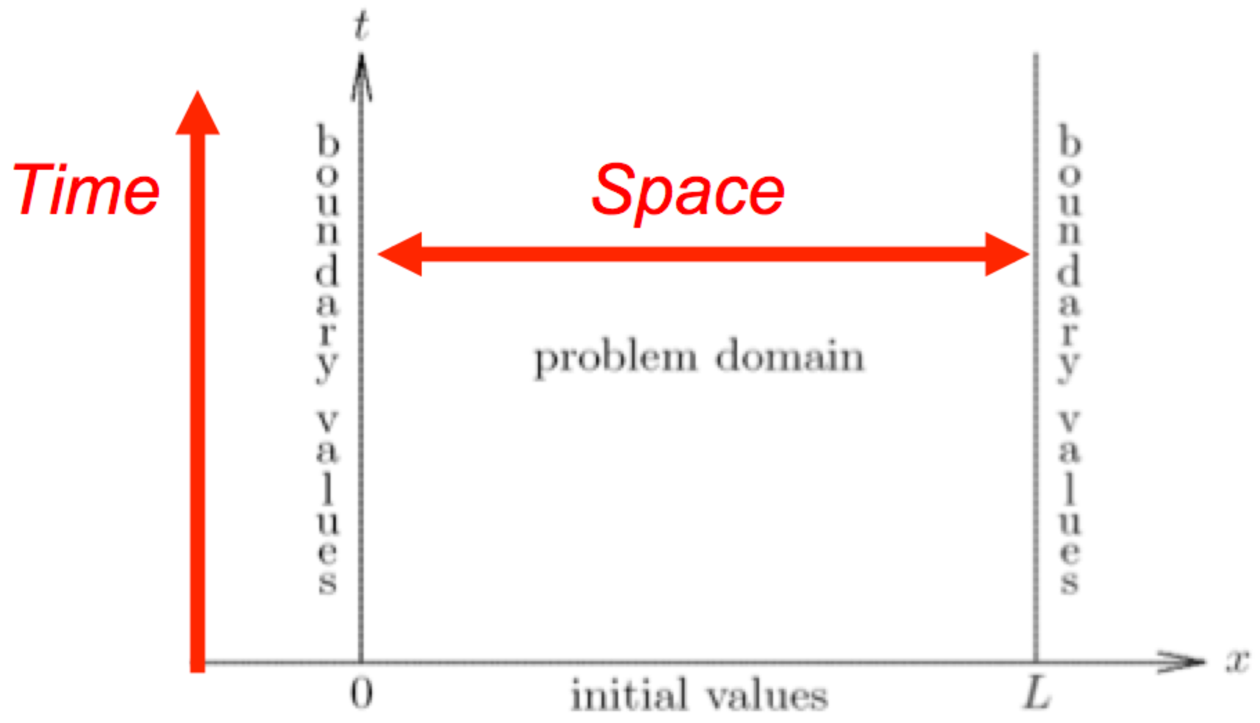
conv_sine.m

Time Dependent Problems

- We'll consider two examples: diffusion (heat equation) and advection.

heat equation: $$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2} \quad + \text{ BCs and IC}$$

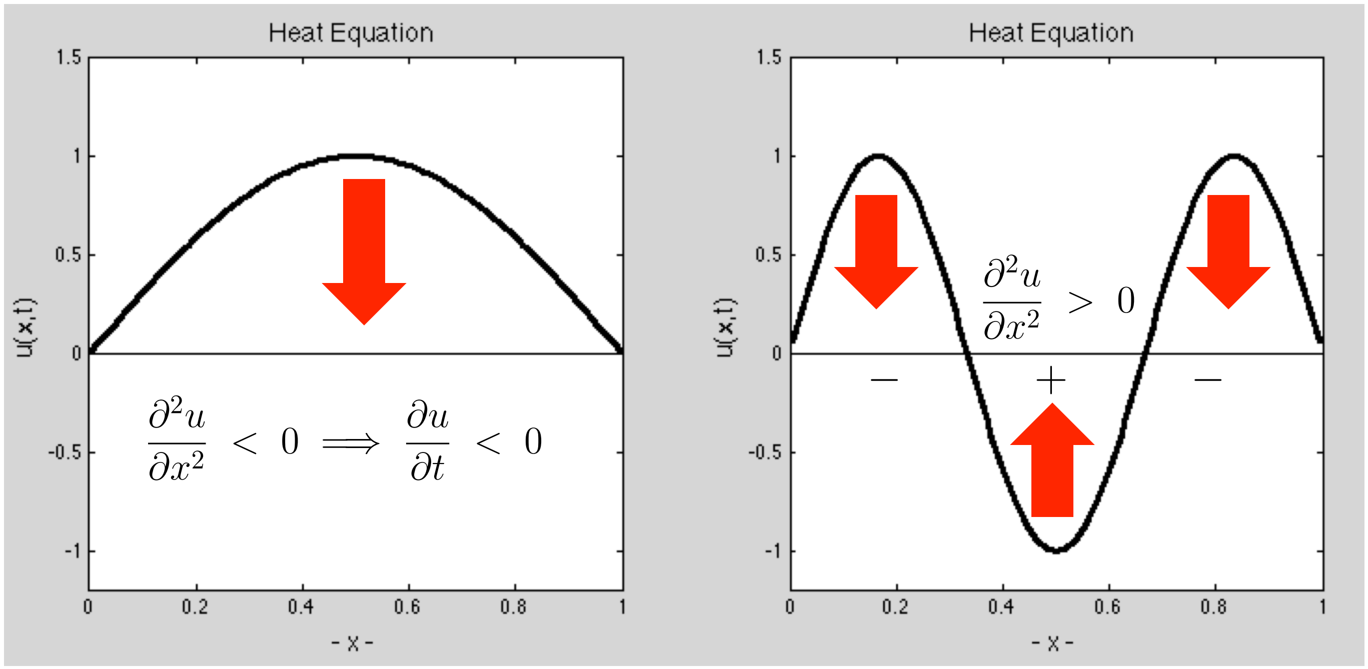
advection: $$\frac{\partial u}{\partial t} = -c \frac{\partial u}{\partial x} \quad + \text{ BCs and IC}$$



Heat Equation:

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2}, \qquad \nu > 0$$

- For the heat equation, the solution evolves in the direction of local curvature.
  - If the the solution is locally concave down, u decreases there.
  - If the the solution is concave up, u increases.

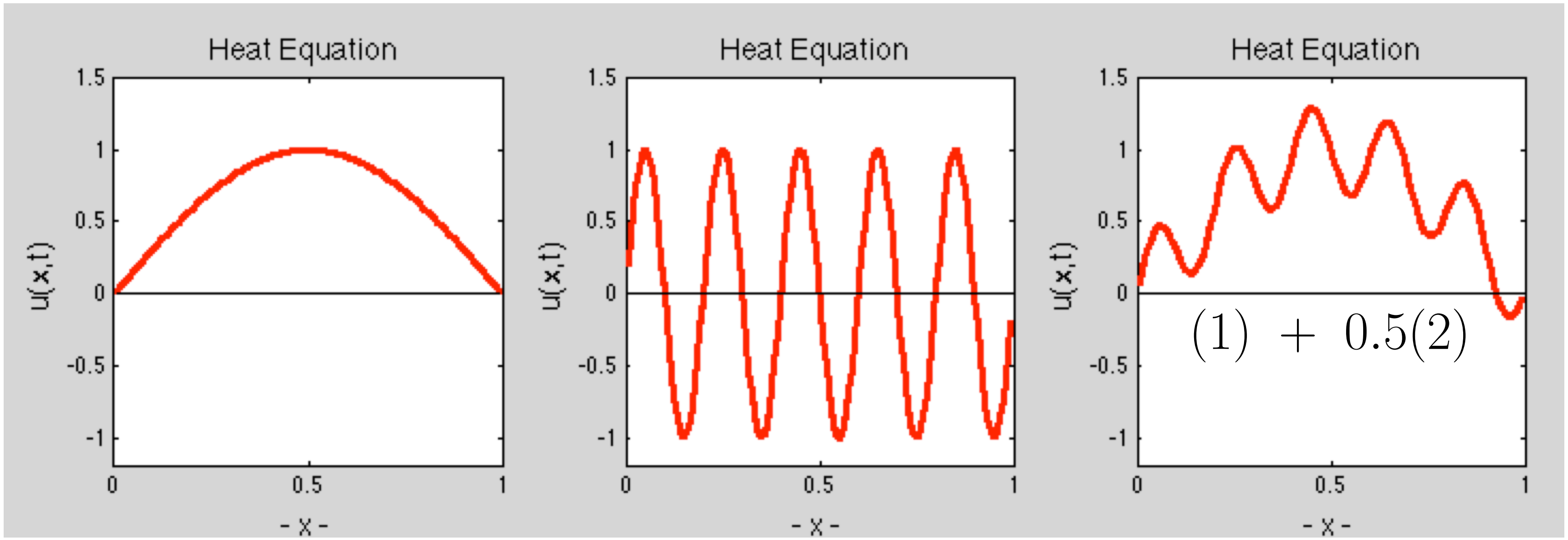


Example Solutions (eigenfunctions): $u_t = \nu u_{xx},\ u(0) = u(1) = 0$

$$\begin{aligned} u(x,t) &= \hat{u}(t)\sin \pi x \\ \frac{\partial u}{\partial t} &= \frac{d\hat{u}}{dt}\sin \pi x = -\nu\pi^2\hat{u}\sin \pi x \\ \frac{d\hat{u}}{dt} &= -\nu\pi^2\hat{u} \\ \hat{u} &= e^{-\nu\pi^2 t}\,\hat{u}(0) \end{aligned}$$

$$\begin{aligned} u(x,t) &= \hat{u}(t)\sin 10\pi x \\ \frac{\partial u}{\partial t} &= \frac{d\hat{u}}{dt}\sin \pi x = -\nu 100\pi^2\hat{u}\sin \pi x \\ \frac{d\hat{u}}{dt} &= -\nu 100\pi^2\hat{u} \\ \hat{u} &= e^{-\nu 100\pi^2 t}\,\hat{u}(0) \end{aligned}$$

$\longrightarrow$ *Very rapid decay.*



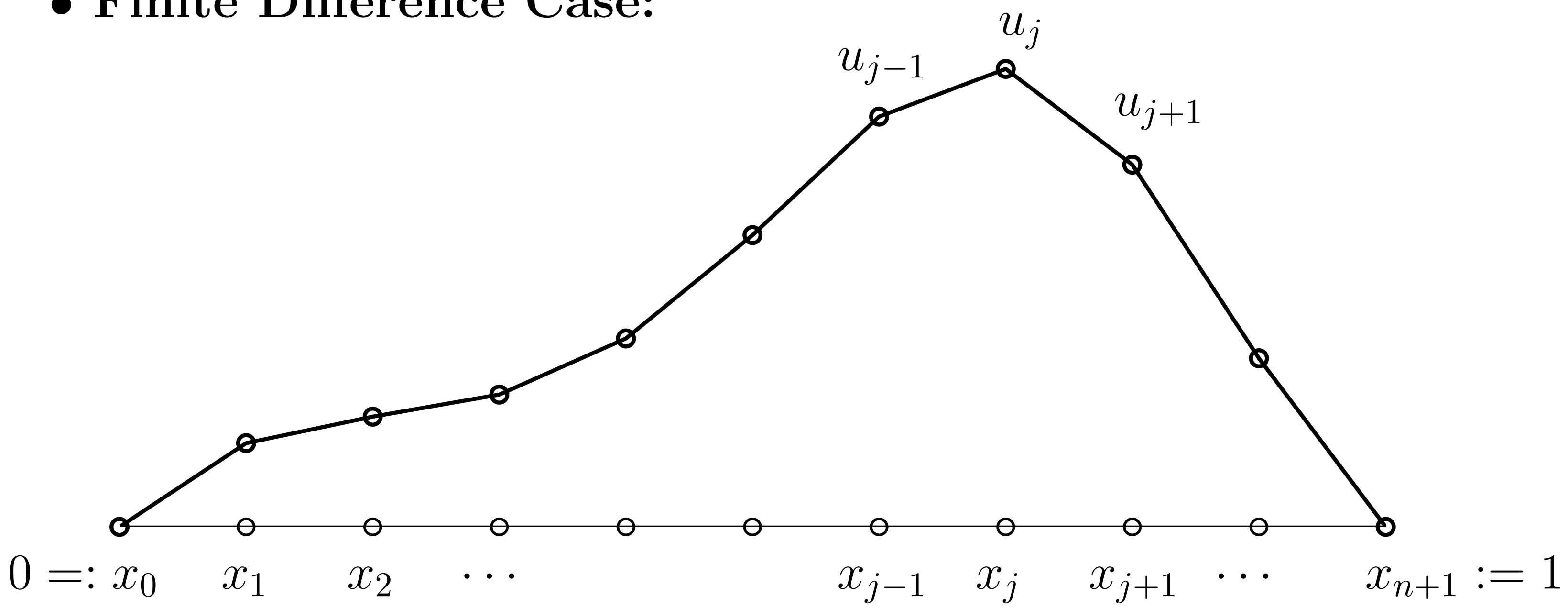
Solution of Partial Differential Equations

- **Unsteady Heat Equation:**

$$u_t \;=\; \nu\, u_{xx} \;+\; q(x,t), \qquad u(x=0,t)=u(x=L,t)=0, \qquad u(x,t=0)=u^0(x).$$

- Discretize in space:
  - Finite difference
  - Weighted residual technique (FEM, Galerkin + high-order polynomials, etc.)

- **Finite Difference Case:**



$$\frac{du_i}{dt} \;=\; -\nu\,(A\mathbf{u})_i \;+\; q_i,\;\; i = 1, \ldots, n$$

- In ODE form:

$$\frac{d\mathbf{u}}{dt} \;=\; -\nu A\mathbf{u} \;+\; \mathbf{q},\;\; \mathbf{u}(t=0) = u^0.$$

- Here, $\Delta x = 1/(n+1)$ and A is the SPD tridiagonal matrix

$$A = \frac{1}{\Delta x^2} \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix}.$$

- Eigenvalues:

$$\begin{aligned} \lambda(A) \;=\; \frac{2}{\Delta x^2}\,(1 \,-\, \cos(k\pi\Delta x)) \;&\in\; \big(\pi^2(1+O(\Delta x^2)), 4(n+1)^2\big) \\ &\in\; \left(\pi^2(1+O(\Delta x^2)), \frac{4}{\Delta x^2}\right). \end{aligned}$$

- Can view this semi-discrete form as a system of ODEs:

$$\frac{d\mathbf{u}}{dt} \;=\; \mathbf{f}(\mathbf{u}) \quad := \; -\nu A\mathbf{u} \;+\; \mathbf{q}(\mathbf{x},t).$$

- Jacobian $$\frac{df_i}{du_j} \;=\; -\nu\, a_{ij} \qquad J \;=\; -\nu\, A.$$

- Stability is determined by the eigenvalues of J *and* by the choice of timestepper.

- Some possible explicit timesteppers

$$\begin{aligned} \text{EF:} \quad \mathbf{u}^{k+1} &= \mathbf{u}^k \;+\; \Delta t\,\mathbf{f}^k \\ \text{AB3:} \quad \mathbf{u}^{k+1} &= \mathbf{u}^k \;+\; \Delta t\,\left(\frac{23}{12}\,\mathbf{f}^k \;-\; \frac{16}{12}\,\mathbf{f}^{k-1} \;+\; \frac{5}{12}\,\mathbf{f}^{k-2}\right) \end{aligned}$$

- **Stable,** as long as $\lambda(J)\Delta t$ in the stability region.

- **Stability:**

- $\lambda(J) = -\nu\lambda(A) \;=\; -\dfrac{2\nu}{\Delta x^2}\left(1 - \cos k\pi\Delta x\right).$

- Worst case is $$|\lambda(J)| \;\sim\; \left|\frac{4\nu}{\Delta x^2}\right|.$$

- For Euler forward (EF), require

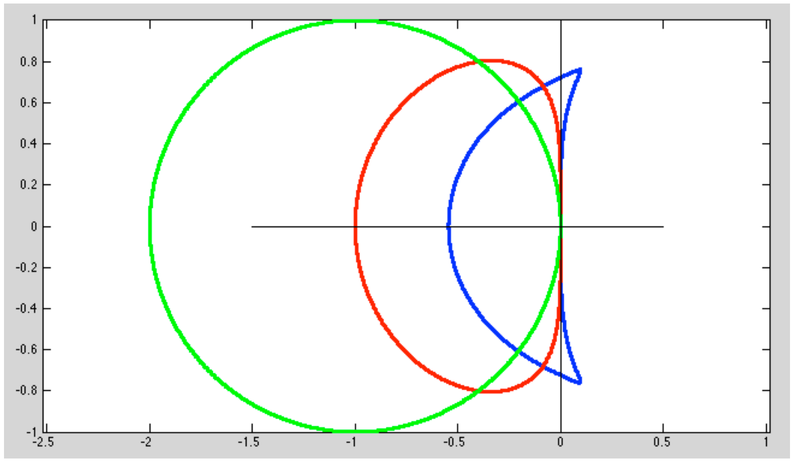
$$|\Delta t\lambda(J)| \;<\; 2$$

or

$$\Delta t \;<\; \frac{2\Delta x^2}{4\nu} \;=\; \frac{\Delta x^2}{2\nu},$$

which is a *very severe* timestep restriction.

Stability Regions, EF, AB2, AB3.



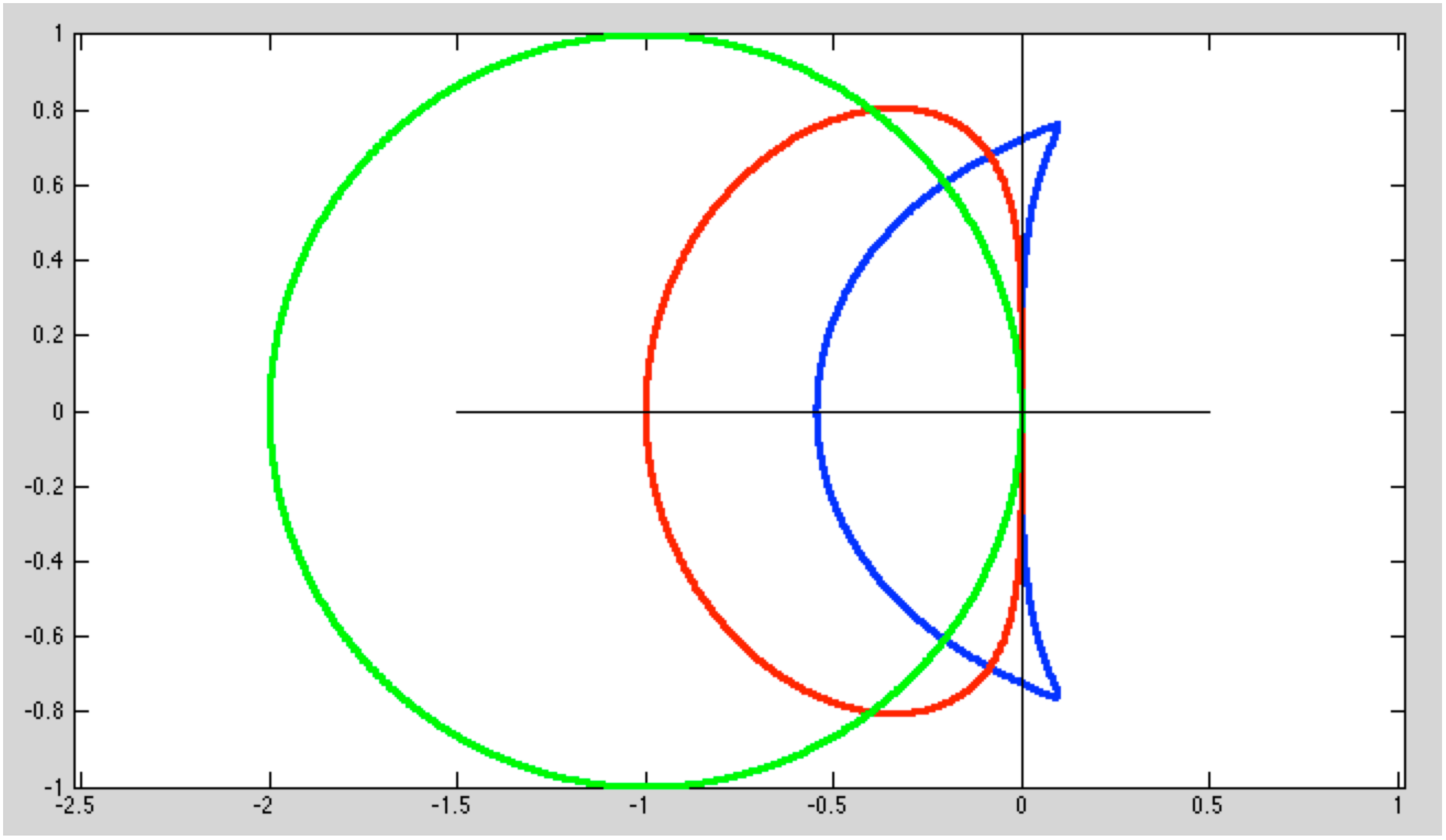
- **Question:**

  What is the maximum allowable timestep size for AB3 in this case?

- **Question:**

  What is the maximum allowable timestep size for AB3 in this case?

Stability Regions, EF, AB2, AB3.



- Severity of explicit timestep restriction:
  - Suppose $\nu = 1$ and you want error $\approx 10^{-6}$.
    $\longrightarrow \quad \Delta x \approx 10^{-3}$.
    $\longrightarrow \quad \Delta t \approx 10^{-6}$, just for stability.
- This is an example of a stiff system.
- High wavenumbers $(\lambda(A))$ are uninteresting but restrict the timestep size.
- For this reason, the heat equation is most often treated *implicitly.*

- **Possible Implicit Approaches:**

$$\frac{d\mathbf{u}}{dt} \;=\; \mathbf{f}(\mathbf{u}) \quad \left\{ \begin{array}{l} \textbf{EB} \\ \textbf{Trapezoid} \text{ (aka Crank-Nicolson)} \\ \textbf{BDF2} \text{ or } \textbf{BDF3} \end{array} \right.$$

- Examples:

$$\textbf{EB:} \qquad \mathbf{u}^{k+1} \;=\; \mathbf{u}^k \;+\; \Delta t \left[-\nu\, A\mathbf{u}^{k+1} \;+\; \mathbf{q}(\mathbf{x}, t^{k+1}) \right]$$

$$\textbf{CN:} \qquad \frac{\mathbf{u}^{k+1} \;-\; \mathbf{u}^k}{\Delta t} \;=\; +\frac{1}{2} \left(-\nu A\mathbf{u}^{k+1} \;+\; \mathbf{q}^{k+1} \;-\; \nu A\mathbf{u}^k \;+\; \mathbf{q}^k \right)$$

$$\textbf{BDF2:} \qquad \frac{3\mathbf{u}^{k+1} \;-\; 4\mathbf{u}^k \;+\; \mathbf{u}^{k-1}}{2\Delta t} \;=\; -\nu\, A\mathbf{u}^{k+1} \;+\; \mathbf{q}(\mathbf{x}, t^{k+1})$$

- EB Example:

$$\begin{aligned}
\mathbf{u}^{k+1} \;+\; \nu\Delta t A\mathbf{u}^{k+1} &= \mathbf{u}^{k} \;+\; \Delta t\mathbf{q}^{k+1} \\
\left[I \;+\; \nu\Delta t A\right]\mathbf{u}^{k+1} &= \mathbf{u}^{k} \;+\; \Delta t\mathbf{q}^{k+1} \\
H\mathbf{u}^{k+1} &= \mathbf{u}^{k} \;+\; \Delta t\mathbf{q}^{k+1}.
\end{aligned}$$

- Here, $H := [I + \nu\Delta t A]$ is SPD, tridiagonal, and strongly diagonally dominant. (In all number of space dimensions.)

- $H\mathbf{u} = \mathbf{f}$ is easier to solve than $A\mathbf{u} = \mathbf{f}$.

- Jacobi- (diagonal-) preconditioned conjugate gradient iteration is often the best choice of solver, particularly in higher space dimensions.

- Note that all the implicit solvers end up with the form $H\mathbf{u} = \mathbf{f}$ and generally have the *same* costs for the linear heat equation considered here.

- Note that CN (aka trapezoid method) is *not* L-stable and will have potential difficulties noted in our discussion of IVPs.

- **Discretization Based on Weighted Residual Technique in Space**
- Coming back to the heat equation (with BCs/ICs),

$$u_t \;=\; \nu\, u_{xx} \;+\; q(x,t),$$

- WRT - residual orthogonal to test functions

$$\int v(\nu\, u_{xx} \;+\; q(x,t) \;-\; u_t)\, dx \;=\; 0 \;\;\forall\, v\, X_0^N.$$

- If $\quad u \;=\; \displaystyle\sum_{j=1}^{n} u_j(t)\,\phi_j(x)\;$ and $\;v \;=\; \phi_i(x),\;$ then

$$\text{LHS:} \qquad \int v \frac{\partial u}{\partial t}\, dx \;=\; \left(\sum_{j=1}^{n} \phi_i \phi_j\, dx \right) u_j(t) \;=\; B \frac{d\mathbf{u}}{dt},$$

with the *mass matrix* B having entries

$$B_{ij} \;:=\; \int \phi_i(x)\,\phi_j(x)\, dx.$$

- On the right, we have

$$\begin{aligned}\text{RHS} \;&=\; \nu \int v \frac{\partial^2 u}{\partial x^2}\, dx \;+\; \int vq\, dx \\ &=\; -\nu \int \frac{\partial v}{\partial x}\frac{\partial u}{\partial x}\, dx \;+\; \int vq\, dx.\end{aligned}$$

- Setting $v = \phi_i$ and $u = \sum_j \phi_j u_j(t)$,

$$\begin{aligned}\text{RHS} \;&=\; -\nu \sum_{j=1}^{n} \left(\int \frac{d\phi_i}{dx}\frac{d\phi_i}{dx}\, dx \right) u_j(t) \;+\; \int \phi_i q\, dx \\ \\ &=\; -\nu A\mathbf{u} \;+\; \mathbf{b}, \qquad \left\{ \begin{array}{l} a_{ij} := \displaystyle\int \frac{d\phi_i}{dx}\frac{d\phi_i}{dx}\, dx \\ b_i \;:= \displaystyle\int \phi_i q\, dx \end{array} \right. .\end{aligned}$$

- In summary, the WRT formulation is, *Find* $u(x,t) \in X_0^N$ *such that,*

$$\int v \frac{\partial u}{\partial t}\, dx \;=\; -\nu \int \frac{\partial v}{\partial x}\frac{\partial u}{\partial x}\, dx \;+\; \int vq\, dx \quad \forall\, v \,\in\, X_0^N,$$

  which leads to the ODE

$$B\frac{d\mathbf{u}}{dt} \;=\; -\nu A\mathbf{u} \;+\; \mathbf{b}, \quad \text{plus initial condition } \mathbf{u}(t=0) = \mathbf{u}^0.$$

- In standard form,

$$\frac{d\mathbf{u}}{dt} \;=\; -\nu B^{-1}A\mathbf{u} \;+\; B^{-1}\mathbf{b},$$

- Stability is thus governed by $\lambda(J) = -\nu\lambda(B^{-1}A)$, *not* just $-\nu\lambda(A)$.
- Presence of B in front of $\frac{d\mathbf{u}}{dt}$ must not be ignored.

- Choice of timestepper motivated by same concerns as for finite-differences:
  - $|\lambda(J)| \sim O(\Delta x^2)$
  - Implicit timestepping generally preferred
  - SPD systems
  - Jacobi (diagonal) preconditioned conjugate gradient iteration is generally the solver of choice.

Time Stepping for Diffusion Equation:

- Recall, with boundary conditions $u(0) = u(1) = 0$, the finite difference operator

$$A\mathbf{u} \;=\; -\frac{\nu}{h^2}\left[u_{j+1} \;-\; u_j \;-\; u_{j-1}\right]$$

with $h := 1/(n+1)$ has eigenvalues in the interval $[0, M]$ with

$$M \;=\; \max_k \;\lambda_k \;=\; \max_k \;\frac{2\nu}{h^2}\left[1 \;-\; \cos k\pi h\right] \;\sim\; \frac{4}{h^2}\,\nu$$

- Our ODE is $\mathbf{u}_t = -A\mathbf{u}$, so we are concerned with $-\lambda_k$.
- With Euler Forward, we require $|\lambda\Delta t| < 2$ for stability,
  - $\longrightarrow \;\Delta t < \frac{h^2}{2}\,\nu$
  - *no matter how smooth the initial condition.*
- This intrinsic *stiffness* motivates the use of implicit methods for the heat equation (BDF2 is a good one).
- *matlab example:* heat1d.m

heat1d_ef.m and heat1d_eb.m and heat1d_cn.m

Steady State Problems

- Heat equation evolves to a steady state:

$$u_t = \nu\, u_{xx} + q(x) \qquad [\ + \text{BCs and IC}\ ]$$

- After waiting long enough, $u(x,t{=}\infty)$ satisfies:

$$-\nu\, u_{xx} = q(x) \qquad [\ + \text{BCs}\ ]$$

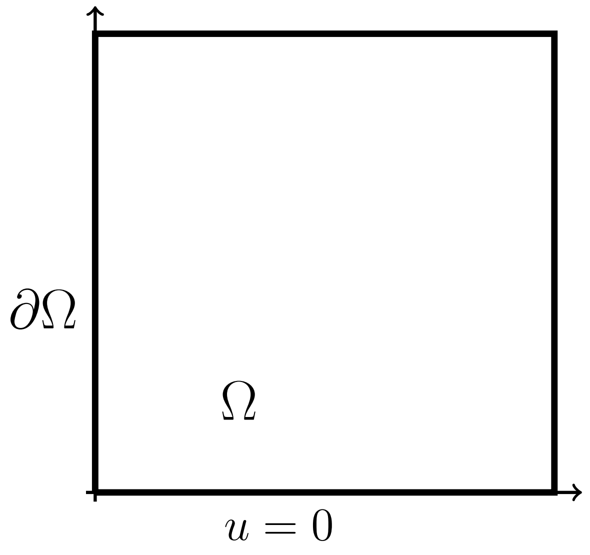
- In 2D, we have:

$$-\nu\,(u_{xx} + u_{yy}) = q(x,y) \qquad [\ + \text{BCs}\ ],$$

which can also be solved by time evolving

$$u_t = \nu\,(u_{xx} + u_{yy}) + q(x,y) \qquad [\ + \text{BCs and } \textit{arbitrary} \text{ IC}\ ]$$

Example: Poisson Equation in 2D



$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f(x,y) \text{ in } \Omega$$
$$u = 0 \text{ on } \partial\Omega$$

- Ex 1: If $f(x,y) = \sin \pi x \; \sin \pi y$,

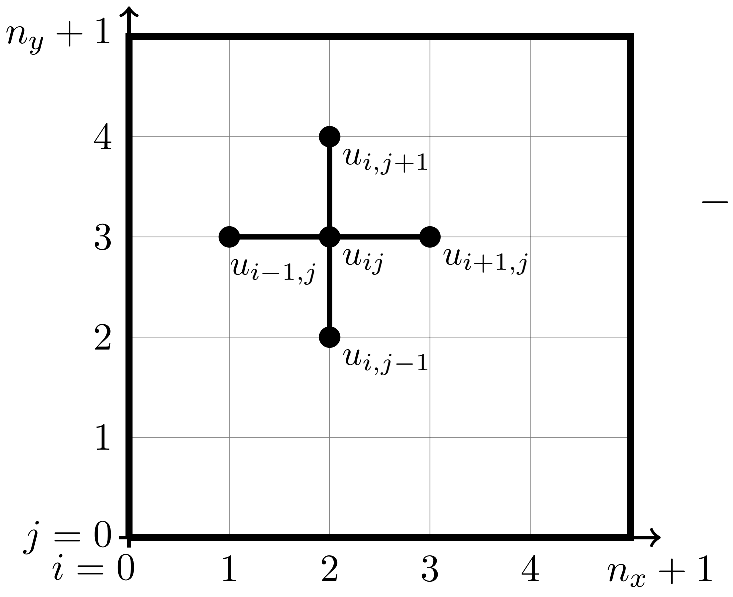
$$u(x,y) = \frac{1}{2\pi^2} \sin \pi x \; \sin \pi y$$

- Ex 2: If $f(x,y) = 1$,

$$u(x,y) = \sum_{k,l \text{ odd}}^{\infty,\infty} \frac{16}{\pi^4 kl(k^2 + l^2)} \sin k\pi x \; \sin l\pi y.$$

  - Q: How large must k and l be for “exact” solution to be correct to ϵ_M ?
  - Spectral collocation would yield $u = u_{\text{exact}} \pm \epsilon_M$ by $N \approx 15$.

Numerical Solution: Finite Differences



"5-point finite-difference stencil"

$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \approx - \left(\frac{u_{i+1,j} - 2u_{ij} - u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{ij} - u_{i,j-1}}{\Delta y^2}\right) = f_{ij}$$

$$i = 1 \ldots n_x$$
$$j = 1 \ldots n_y$$

- Here, the unknowns are $\mathbf{u} = [u_{11},\, u_{21},\, \ldots\, , u_{n_x,n_y}]^T$.
- This particular (so-called natural or lexicographical) ordering gives rise to a banded system matrix for $\mathbf{u}$.
- As in the 1D case, the error is $O(\Delta x^2) + O(\Delta y^2) = O(h^2)$ if we take $\Delta x = \Delta y =: h$.
- Assuming for simplicity that $N = n_x = n_y$, we have $n = N^2$ unknowns.

- For $i, j \in [1, \ldots, N]^2$, the governing finite difference equations are

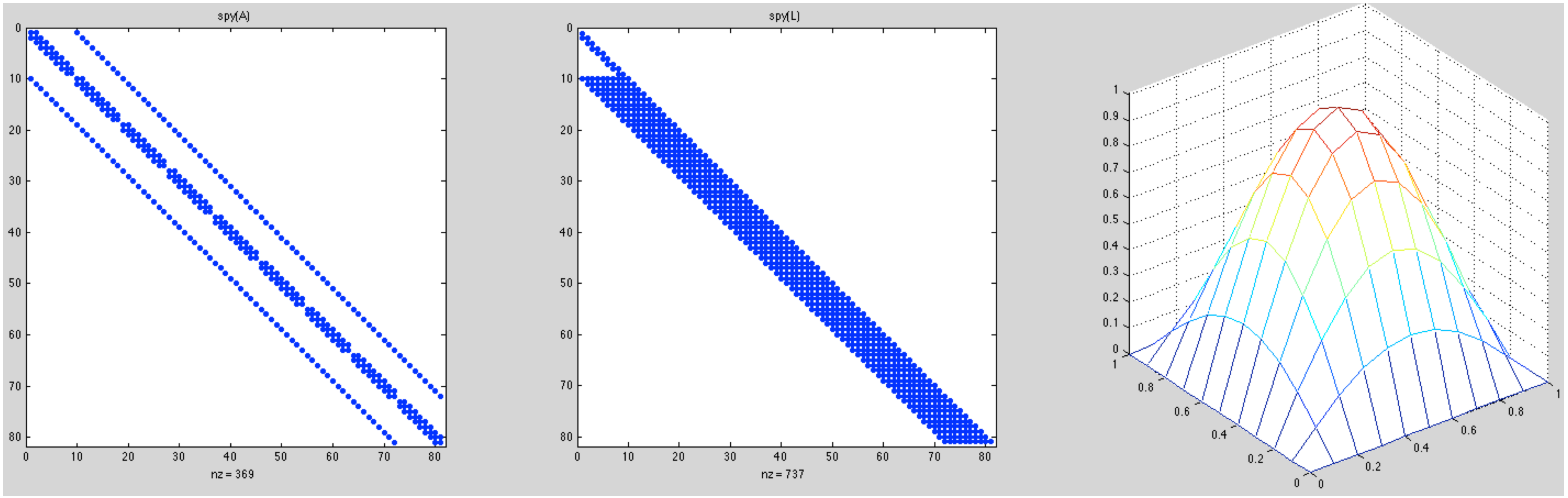
$$-\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}\right) \;=\; f_{ij}.$$

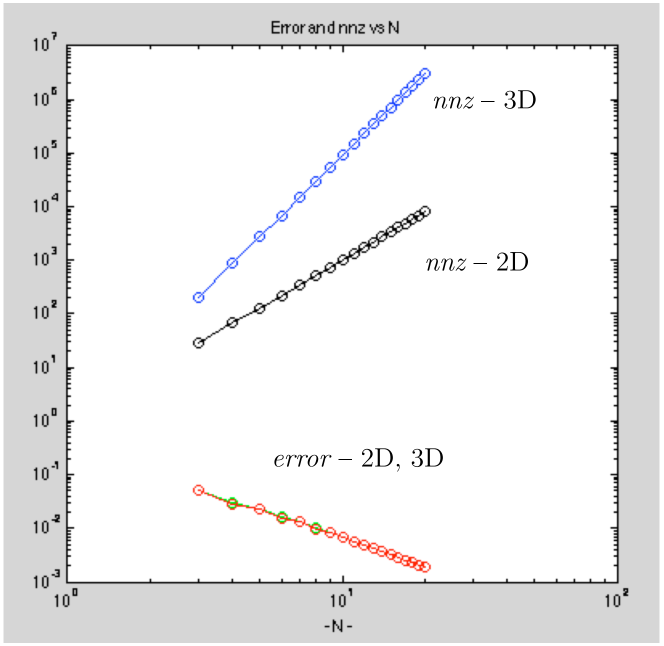
- Assuming a *lexicographical ordering* in which the i- (x-) index advances fastest, the system matrix has the form

$$\frac{1}{h^2}\underbrace{\left(\begin{array}{ccccc|ccccc|ccccc|ccccc}
4 & -1 & & & & -1 & & & & & & & & & & & & & & \\
-1 & 4 & -1 & & & & -1 & & & & & & & & & & & & & \\
 & -1 & \ddots & \ddots & & & & \ddots & & & & & & & & & & & & \\
 & & \ddots & \ddots & -1 & & & & \ddots & & & & & & & & & & & \\
 & & & -1 & 4 & & & & & -1 & & & & & & & & & & \\
\hline
-1 & & & & & 4 & -1 & & & & \ddots & & & & & & & & & \\
 & -1 & & & & -1 & 4 & -1 & & & & \ddots & & & & & & & & \\
 & & \ddots & & & & -1 & \ddots & \ddots & & & & \ddots & & & & & & & \\
 & & & \ddots & & & & \ddots & \ddots & -1 & & & & \ddots & & & & & & \\
 & & & & -1 & & & & -1 & 4 & & & & & \ddots & & & & & \\
\hline
 & & & & & \ddots & & & & & \ddots & & & & & -1 & & & & \\
 & & & & & & \ddots & & & & & \ddots & & & & & -1 & & & \\
 & & & & & & & \ddots & & & & & \ddots & & & & & \ddots & & \\
 & & & & & & & & \ddots & & & & & \ddots & & & & & \ddots & \\
 & & & & & & & & & \ddots & & & & & \ddots & & & & & -1 \\
\hline
 & & & & & & & & & & -1 & & & & & 4 & -1 & & & \\
 & & & & & & & & & & & -1 & & & & -1 & 4 & \ddots & & \\
 & & & & & & & & & & & & \ddots & & & & \ddots & \ddots & \ddots & \\
 & & & & & & & & & & & & & \ddots & & & & \ddots & \ddots & -1 \\
 & & & & & & & & & & & & & & -1 & & & & -1 & 4
\end{array}\right)}_{A}
\underbrace{\left(\begin{array}{c}
u_{11} \\ u_{21} \\ \vdots \\ \vdots \\ u_{N1} \\ \hline
u_{12} \\ u_{22} \\ \vdots \\ \vdots \\ u_{N2} \\ \hline
\vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \hline
u_{1N} \\ u_{2N} \\ \vdots \\ \vdots \\ u_{NN}
\end{array}\right)}_{\mathbf{u}}
=
\underbrace{\left(\begin{array}{c}
f_{11} \\ f_{21} \\ \vdots \\ \vdots \\ f_{N1} \\ \hline
f_{12} \\ f_{22} \\ \vdots \\ \vdots \\ f_{N2} \\ \hline
\vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \hline
f_{1N} \\ f_{2N} \\ \vdots \\ \vdots \\ f_{NN}
\end{array}\right)}_{\mathbf{f}}$$

- The system matrix A is
  - *sparse*, with 5 nonzeros per row (good)
  - and has a bandwith N (bad).
- The difficulty is that solving $A\mathbf{u} = \mathbf{f}$ using Gaussian elimination results in signifcant *fill*— each of the factors L and U have $N^3 = n^{3/2}$ nonzeros.
- Worse, for 3D problems with N^3 unknowns, $\mathbf{u} = [u_{111}, u_{211}, \ldots, u_{n_x,n_y,n_z}]^T$, A is
  - *sparse*, with 7 nonzeros per row (good)
  - and has a bandwith N^2 (awful).
- In 3D, LU decomposition yields $N^5{=}n^{5/3}$ nonzeros in L and U.
- The situation can be rescued in 2D with a reordering of the unknowns (e.g., via nested-dissection) to yield $O(n \log n)$ nonzeros in L and U.
- In 3D, nested-dissection yields $O(n^{3/2})$ nonzeros in the factors. Direct solution is not scalable for more than two space dimensions.
- The following Matlab examples illustrate the issue of fill:
  - fd_poisson_2d.m
  - fd_poisson_3d.m

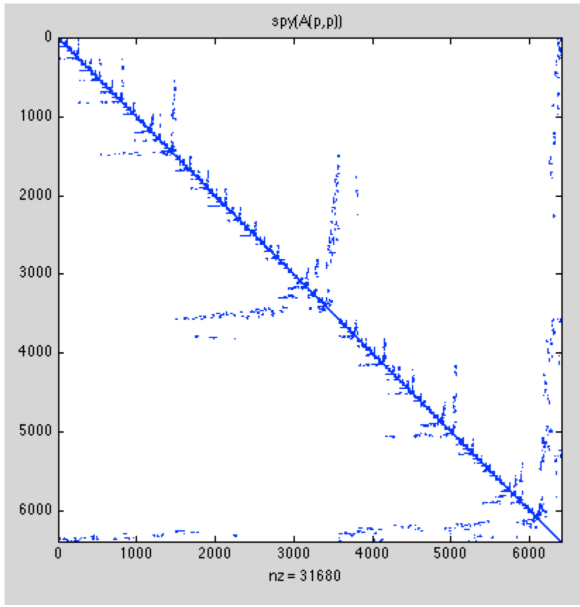
Matrix-Fill for 2D and 3D Poisson, Lexicographical Ordering

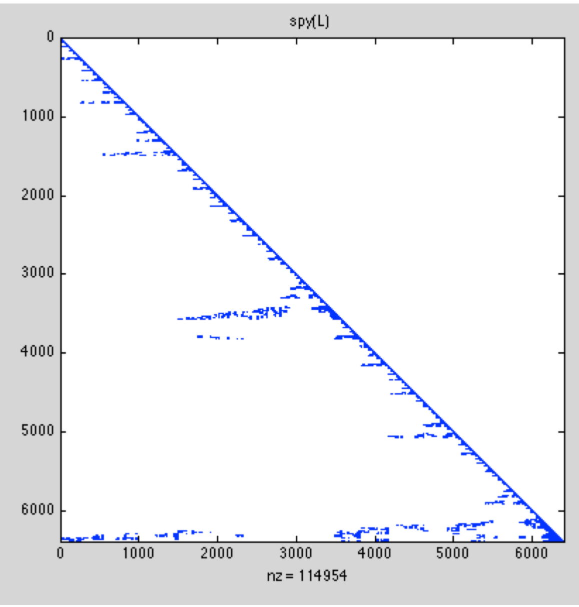


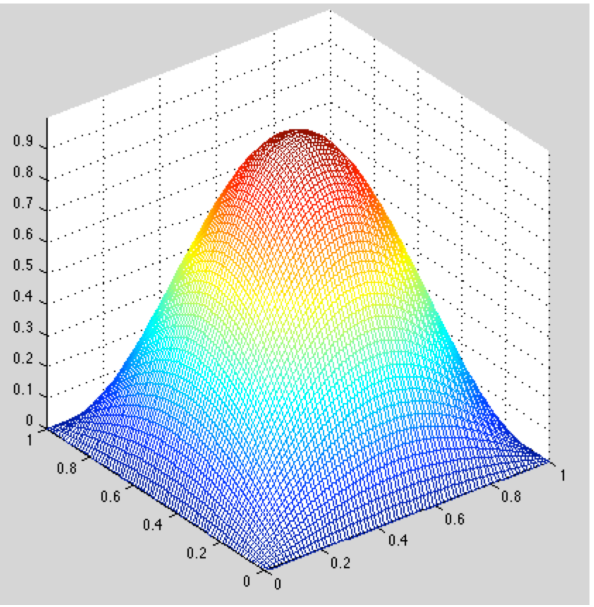


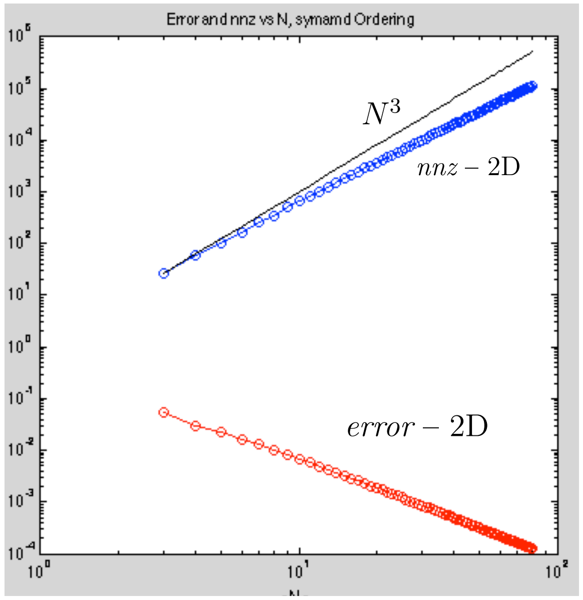
- As expected, the error scales like $h^2 \sim 1/N^2$ in both 2D and 3D.
- The resepctive storage costs (and work per rhs) are $\sim N^3$ and N^5.
- Alternative orderings are asymptotically better, but the constants tend to be large.

Matrix-Fill for 2D Poisson, `symamd` Ordering









- We see for $N = 80$ ($n = 6400$) a $5\times$ reduction in number of nonzeros by reording with matlab's `symamd` function.
- The requirements for indirect addressing to access elements of the complacty-stored matrix further adds to overhead.
- Gains tend to be realized only for very large N and are even less beneficial in 3D.
- Despite this, it's still a reasonable idea to reorder in matlab because it's available and easy to use.

More on 2D Systems Matrices for Poisson Equation

$$\begin{aligned} -\nabla^2 u &= f(x,y), \qquad \text{plus BCs} && (10)\\ &= -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \\ &= -\left(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2}\right) + O(h^2), \end{aligned}$$

where we have substituted the finite difference approximations, assumed to be about the point $\mathbf{x}_{ij} := (x_i, y_j)$,

$$\begin{aligned} \frac{\delta^2 u}{\delta x^2} &:= \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} && (11)\\ \frac{\delta^2 u}{\delta y^2} &:= \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}, \end{aligned}$$

with the further assumption of uniform grid spacing, $\Delta x = \Delta y = h$. We'll also consider homogeneous Dirichlet boundary conditions, that is, $u(x,y)|_{\partial\Omega} \equiv 0$. The respective unknowns and data in this case are u_{ij} and f_{ij}, governed by the following system of equations

$$-\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2}\right) = f_{ij}, \qquad (12)$$

for $i,j \in [1,\ldots,N]^2$.

Assuming a *lexicographical ordering* in which the i- (x-) index advances fastest, the system takes on the following matrix structure for $\Delta x = \Delta y = h$.

$$
\frac{1}{h^2}
\underbrace{\left(\begin{array}{ccccc|ccccc|ccccc|ccccc}
4 & -1 & & & & -1 & & & & & & & & & & & & & & \\
-1 & 4 & -1 & & & & -1 & & & & & & & & & & & & & \\
 & -1 & \ddots & \ddots & & & & \ddots & & & & & & & & & & & & \\
 & & \ddots & \ddots & -1 & & & & \ddots & & & & & & & & & & & \\
 & & & -1 & 4 & & & & & -1 & & & & & & & & & & \\
\hline
-1 & & & & & 4 & -1 & & & & \ddots & & & & & & & & & \\
 & -1 & & & & -1 & 4 & -1 & & & & \ddots & & & & & & & & \\
 & & \ddots & & & & -1 & \ddots & \ddots & & & & \ddots & & & & & & & \\
 & & & \ddots & & & & \ddots & \ddots & -1 & & & & \ddots & & & & & & \\
 & & & & -1 & & & & -1 & 4 & & & & & \ddots & & & & & \\
\hline
 & & & & & \ddots & & & & & \ddots & & & & & -1 & & & & \\
 & & & & & & \ddots & & & & & \ddots & & & & & -1 & & & \\
 & & & & & & & \ddots & & & & & \ddots & & & & & \ddots & & \\
 & & & & & & & & \ddots & & & & & \ddots & & & & & \ddots & \\
 & & & & & & & & & \ddots & & & & & \ddots & & & & & -1 \\
\hline
 & & & & & & & & & & -1 & & & & & 4 & -1 & & & \\
 & & & & & & & & & & & -1 & & & & -1 & 4 & \ddots & & \\
 & & & & & & & & & & & & \ddots & & & & \ddots & \ddots & \ddots & \\
 & & & & & & & & & & & & & \ddots & & & & \ddots & \ddots & -1 \\
 & & & & & & & & & & & & & & -1 & & & & -1 & 4
\end{array}\right)}_{A_{2D}}
\underbrace{\left(\begin{array}{c}
u_{11} \\ u_{21} \\ \vdots \\ \vdots \\ u_{M1} \\ \hline u_{12} \\ u_{22} \\ \vdots \\ \vdots \\ u_{M2} \\ \hline \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \hline u_{1N} \\ u_{2N} \\ \vdots \\ \vdots \\ u_{MN}
\end{array}\right)}_{\underline{u}}
=
\underbrace{\left(\begin{array}{c}
f_{11} \\ f_{21} \\ \vdots \\ \vdots \\ f_{M1} \\ \hline f_{12} \\ f_{22} \\ \vdots \\ \vdots \\ f_{M2} \\ \hline \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \hline f_{1N} \\ f_{2N} \\ \vdots \\ \vdots \\ f_{MN}
\end{array}\right)}_{\underline{f}}
$$

Note that A_{2D} can be expressed as the sum of two systems, one associated with A_x coming from $\frac{\delta^2 u}{\delta x^2}$, and one associated with one associated with A_y coming from $\frac{\delta^2 u}{\delta y^2}$. Specifically, we can write

$$A_{2D} \quad = \quad (I_y \otimes A_x) \; + \; (A_y \otimes I_x), \tag{13}$$

where we have introduced the Kronecker (or *tensor*) product, $\otimes$. For two matrices A and B, their Kronecker product $C = A \otimes B$ is defined as the block matrix

$$C \quad := \quad \begin{pmatrix} a_{11}B & a_{12}B & \cdots & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & \cdots & a_{2n}B \\ \vdots & \vdots & & & \vdots \\ a_{m1}B & a_{m2}B & \cdots & \cdots & a_{mn}B \end{pmatrix}. \tag{14}$$

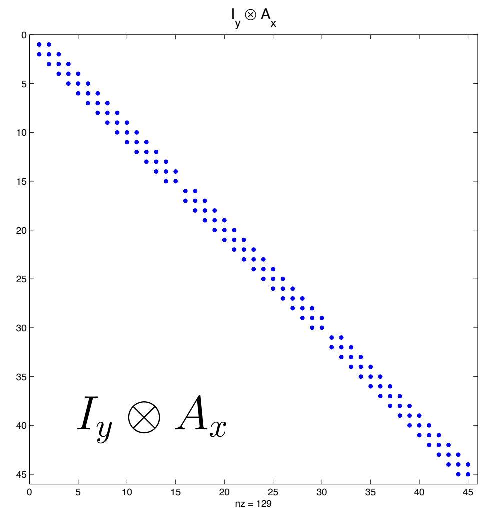
We will soon explore a few properties of this form, but for now simply note that it allows a clean expression of the discretized Poisson operator in 2D. Consider the following splitting of A_{2D}.

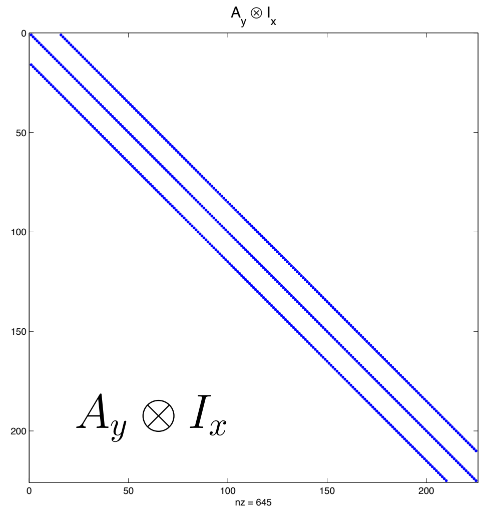
$$A_{2D} \quad = \quad \frac{1}{h^2} \left(\begin{array}{c|c|c|c} \begin{matrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{matrix} & & & \\ \hline & \begin{matrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{matrix} & & \\ \hline & & \begin{matrix} \ddots & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \ddots \end{matrix} & \\ \hline & & & \begin{matrix} 2 & -1 & & & \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{matrix} \end{array} \right) \qquad +$$

$$+\quad \frac{1}{h^2}\left(\begin{array}{c|c|c|c}
\begin{matrix}2 & & & & \\ & 2 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & 2\end{matrix} &
\begin{matrix}-1 & & & & \\ & -1 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & -1\end{matrix} & & \\ \hline
\begin{matrix}-1 & & & & \\ & -1 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & -1\end{matrix} &
\begin{matrix}2 & & & & \\ & 2 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & 2\end{matrix} &
\begin{matrix}\ddots & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \ddots\end{matrix} & \\ \hline
 &
\begin{matrix}\ddots & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \ddots\end{matrix} &
\begin{matrix}\ddots & & & & \\ & \ddots & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \ddots\end{matrix} &
\begin{matrix}-1 & & & & \\ & -1 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & -1\end{matrix} \\ \hline
 & &
\begin{matrix}-1 & & & & \\ & -1 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & -1\end{matrix} &
\begin{matrix}2 & & & & \\ & 2 & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & 2\end{matrix}
\end{array}\right)$$

$$
\begin{aligned}
A_{2D} &= \begin{pmatrix} A_x & & & \\ & A_x & & \\ & & \ddots & \\ & & & A_x \end{pmatrix} + \frac{1}{h^2} \begin{pmatrix} 2I_x & -I_x & & \\ -I_x & 2I_x & \ddots & \\ & \ddots & \ddots & -I_x \\ & & -I_x & 2I_x \end{pmatrix} \\
&= (I_y \otimes A_x) \; + \; (A_y \otimes I_x)
\end{aligned}
$$

```
close all; format compact;
% Kronecker Product Demo
%
%    NOTE:  It is important to use SPARSE matrices throughout.
%
%           Otherwise, your run times will be very long and
%           you will likely run out of memory!

Lx=2; Ly=1;
nx=15; ny=3;  % Number of _interior_ points

dx=Lx/(nx+1); dy=Ly/(ny+1);

%  USE help spdiags

e = ones(nx,1); Ax = spdiags([-e 2*e -e], -1:1, nx, nx)/(dx*dx);
e = ones(ny,1); Ay = spdiags([-e 2*e -e], -1:1, ny, ny)/(dy*dy);

Ix=speye(nx); Iy=speye(ny);

A = kron(Iy,Ax) + kron(Ay,Ix);   %%% FINITE DIFFERENCE STIFFNESS MATRIX

% A couple of demo cases without the 1/(dx*dx) scaling.

nd= 5;
e = ones(nd,1); Ad = spdiags([-e 2*e -e], -1:1, nd, nd);
T = kron(Iy,Ad); full(T)

nd= 15;
e = ones(nd,1); Ad = spdiags([-e 2*e -e], -1:1, nd, nd);
T = kron(Iy,Ad); spy(T)
title('I_y \otimes A_x','fontsize',16)
set(gcf,'PaperUnits','normalized');set(gcf,'PaperPosition',[0 0 1 1])
print -dpdf iyax.pdf

pause; figure
nd= 5;
e = ones(nd,1); Ad = spdiags([-e 2*e -e], -1:1, nd, nd);
T = kron(Ad,Ix); full(T)

nd= 15;
e = ones(nd,1); Ad = spdiags([-e 2*e -e], -1:1, nd, nd);
T = kron(Ad,Ix); spy(T)
title('A_y \otimes I_x','fontsize',16)
set(gcf,'PaperUnits','normalized');set(gcf,'PaperPosition',[0 0 1 1])
print -dpdf ayix.pdf
```





Note that our finite-difference stiffness matrix in matlab would be written as

```
A = kron(Iy,Ax) + kron(Ay,Ix)
```

where `Ax` and `Ay` are formed using the matlab `spdiags` command (`help spdiags`), and Iy and Ix are formed using `speye`.

It is important to use *sparse matrices* in matlab for these higher-dimensional (2D and 3D) problems or you will run out of memory and it will take *very long* to solve these problems.

This problem is known in scientific computing and *the curse of dimensionality*.

1.4 Poisson Equation in $\mathbb{R}^3$

We now extend the 1D and 2D concepts to the most important 3D case. The short story is that the 3D stiffness matrix takes the wonderfully symmetric form

$$\begin{aligned} A_{3D} &= (I_z \otimes A_{2D}) + (A_z \otimes I_{2D}) \qquad (15)\\ &= (I_z \otimes I_y \otimes A_x) + (I_z \otimes A_y \otimes I_x) + (A_z \otimes I_y \otimes I_x). \end{aligned}$$

and the discrete system is as before $A_{3D}\underline{u} = \underline{f}$. This of course is the form that arises for a finite difference discretization of $-\nabla^2 u = f$ in $\Omega = [0,1]^3$, $u = 0$ on $\partial\Omega$, or, more explicitly,

$$-\left(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} + \frac{\delta^2 u}{\delta z^2} \right) = f(x_i, y_j, z_k), \qquad (16)$$

with

$$\left.\frac{\delta^2 u}{\delta z^2}\right|_{ijk} := \frac{u_{ij,k+1} - 2u_{ijk} + u_{ij,k-1}}{\Delta z^2}, \qquad (17)$$

and equivalent expressions for $\frac{\delta^2 u}{\delta x^2}$ and $\frac{\delta^2 u}{\delta y^2}$.

Iterative Solvers

- The *curse of dimensionality* for $d > 2$ resulted in a move towards iterative (rather than direct-, LU-based) linear solvers once computers became fast enough to tackle 3D problems in the mid-80s.
- With iterative solvers, factorization

$$A\mathbf{u} = \mathbf{f} \implies \mathbf{u} = A^{-1}\mathbf{f} = U^{-1}L^{-1}\mathbf{f}$$

  is replaced by, say,

$$\mathbf{u}_{k+1} = \mathbf{u}_k + M^{-1}(\mathbf{f} - A\mathbf{u}_k),$$

  which only requires matrix-vector products.
- With $\mathbf{e}_k := \mathbf{u} - \mathbf{u}_k$, we have

$$\mathbf{e}_{k+1} = \left(I - M^{-1}A\right)\mathbf{e}_k, \qquad \text{(as we've seen before).}$$

- This is known as Richardson iteration.
- For the particular case $M = D = \text{diag}(A)$, it is Jacobi iteration.
- We can derive Jacobi iteration (and multigrid by looking at a *parabolic* PDE, known as the (unsteady) heat equation. (The Poisson equation is sometimes referred to as the steady-state heat equation.)

- The intrinsic advantage of iterative solvers is that there is no *fill* associated with matrix factorization.
- Often one does not even construct the matrix. Rather, we simply evaluate the residual $\mathbf{r}_k := \mathbf{f} - A\mathbf{u}_k$ and set $\mathbf{u}_{k+1} = \mathbf{u}_k + M^{-1}\mathbf{r}_k$.
- For a *sparse matrix* A, the operation count is $O(n)$ per iteration.
- Assuming the preconditioner cost is also sparse, the overall cost is $O(n\, k_{\max})$, where $k_{\max}$ is the number of iterations required to reach a desired tolerance.
- The choice of iteration (Richardson, conjugate gradient, GMRES) can greatly influence $k_{\max}$.
- Even more significant is the choice of M.
- Usually, one seeks an M such that the cost of solving $M\mathbf{z} = \mathbf{r}$ is $O(n)$ and that $k_{\max} = O(1)$. That is, the iteration count is bounded, independent of n.
- The overall algorithm is therefore $O(n)$, which is optimal.

Iterative Solvers - Linear Elliptic Problems

- PDEs give rise to large sparse linear systems of the form

$$A\mathbf{u} = \mathbf{f}.$$

Here, we'll take A to be the (SPD) matrix arising from finite differences applied to the Poisson equation

$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = f(x,y) \qquad x,y \in [0,1]^2, \qquad u = 0 \text{ on } \partial\Omega$$

$$-\left(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2}\right)_{ij} \approx \left. f\right|_{ij},$$

- Assuming uniform spacing in x and y we have

$$\frac{\delta^2 u}{\delta x^2} := \frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h^2} \quad \text{and} \quad \frac{\delta^2 u}{\delta y^2} := \frac{u_{i,j+1} - 2u_{ij} + u_{i,j-1}}{h^2}$$

- Our finite difference formula is thus,

$$\frac{1}{h^2}\left(u_{i+1,j}+u_{i-1,j}-4u_{ij}+u_{i,j+1}+u_{i,j-1}\right) \;=\; f_{ij}.$$

- Rearranging, we can solve for u_{ij}:

$$\begin{aligned}\frac{4}{h^2}u_{ij} &= f_{ij}+\frac{1}{h^2}\left(u_{i+1,j}+u_{i-1,j}+u_{i,j+1}+u_{i,j-1}\right)\\[2em] u_{ij} &= \frac{h^2}{4}f_{ij} \;+\; +\frac{1}{4}\left(u_{i+1,j}+u_{i-1,j}+u_{i,j+1}+u_{i,j-1}\right)\end{aligned}$$

- Jacobi iteration uses the preceding expression as a fixed-point iteration:

$$
\begin{aligned}
u_{ij}^{k+1} &= \frac{h^2}{4} f_{ij} \; + \; \frac{1}{4}\left(u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k\right) \\
&= \frac{h^2}{4} f_{ij} \; + \; \text{average of current neighbor values}
\end{aligned}
$$

- Note that this is analogous to

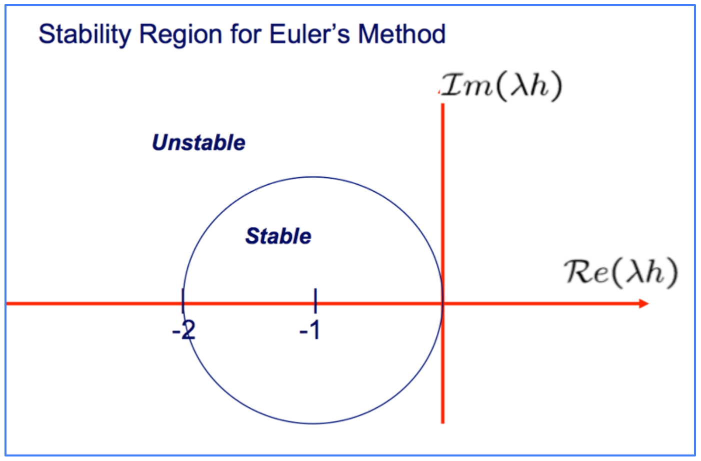
$$
u_{ij}^{k+1} \;=\; u_{ij}^k + \frac{h^2}{4}\left[f_{ij} + \frac{1}{h^2}\left(u_{i+1,j}^k + u_{i-1,j}^k - 4u_{ij}^k + u_{i,j+1}^k + u_{i,j-1}^k\right)\right]
$$

$$
\mathbf{u}_{k+1} \;=\; \mathbf{u}_k \; + \; \Delta t\left(\mathbf{f} - A\mathbf{u}_k\right), \quad \boxed{\Delta t := \frac{h^2}{4},}
$$

← FOR 2D Poisson!

which is Euler forward applied to

$$
\frac{d\mathbf{u}}{dt} \;=\; -A\mathbf{u} \; + \; \mathbf{f}.
$$


Stability Region for Euler's Method
$\mathcal{I}m(\lambda h)$
Unstable
Stable
$\mathcal{R}e(\lambda h)$
-2
-1

- We note that we have stability if $|\lambda \Delta t| < 2$

- Recall that the eigenvalues for the 1D diffusion operator are

$$\lambda_j \;=\; \frac{2}{h^2}\left(1-\cos j\pi\Delta x\right) < \frac{4}{h^2}$$

- In 2D, we pick up contributions from both $\frac{\delta^2 u}{\delta x^2}$ and $\frac{\delta^2 u}{\delta y^2}$, so

$$\max|\lambda| \;<\; \frac{8}{h^2}$$

  and we have stability since

$$\max|\lambda\Delta t| \;<\; \frac{8}{h^2}\frac{h^2}{4} = 2$$

- So, Jacobi iteration is equivalent to solving $A\mathbf{u} = \mathbf{f}$ by time marching $\frac{d\mathbf{u}}{dt} \;=\; -A\mathbf{u} + \mathbf{f}$ using EF with maximum allowable timestep size,

$$\Delta t \;=\; \frac{h^2}{4}.$$

MULTIGRID ITERATION

Jacobi Iteration in Matrix Form

- Our unsteady heat equation has the matrix form

$$\mathbf{u}_{k+1} \;=\; \mathbf{u}_k \,+\, \Delta t\,(\mathbf{f} - A\mathbf{u}_k)$$

- For variable diagonal entries, Richardson iteration is

$$\mathbf{u}_{k+1} \;=\; \mathbf{u}_k \,+\, \sigma M^{-1}\,(\mathbf{f} - A\mathbf{u}_k)$$

- If $\sigma = 1$ and $M = D^{-1}$ =diag(A) $[d_{ii} = 1/a_{ii},\, d_{ij} = 0,\, i \neq j]$, we have standard Jacobi iteration.
- If $\sigma < 1$ we have *damped Jacobi.*
- M is generally known as a smoother or a preconditioner, depending on context.

Rate of Convergence for Jacobi Iteration

- Let $\mathbf{e}_k := \mathbf{u} - \mathbf{u}_k$.
- Since $A\mathbf{u} = \mathbf{f}$, we have

$$\begin{array}{rcl}
\mathbf{u}_{k+1} & = & \mathbf{u}_k \, + \, \Delta t \, (A\mathbf{u} - A\mathbf{u}_k) \\
-\mathbf{u} & = & -\mathbf{u} \\
- - - & - & - - - - - - - - - - - - \\
-\mathbf{e}_{k+1} & = & -\mathbf{e}_k \, - \, \sigma \Delta t A \mathbf{e}_k \\
-\mathbf{e}_{k+1} & = & -\left(I - \sigma \Delta t A\right) \mathbf{e}_k \\
\mathbf{e}_k & = & \left(I - \sigma \Delta t A\right)^k \mathbf{e}_0 \\
 & = & \left(I - \sigma \Delta t A\right)^k \mathbf{u} \qquad \text{if } \mathbf{u}_0 = 0.
\end{array}$$

- If $\sigma < 1$, then the high wavenumber error components will decay because $\lambda \Delta t$ will be well within the stability region for EF.

- The low-wavenumber components of the solution (and error) evolve like $e^{-\lambda\sigma\Delta tk}$, because these will be well-resolved in time by Euler forward.
- Thus, we can anticipate

$$||\mathbf{e}_k|| \;\approx\; ||\mathbf{u}||e^{-\lambda_{\min}\sigma\Delta tk}$$

  with $\lambda_{\min} \approx 2\pi^2$ (for 2D).

- If $\sigma \approx 1$, we have

$$||\mathbf{e}_k|| \;\approx\; ||\mathbf{u}||e^{-2\pi^2(h^2/4)k} \;\leq\; \text{tol}$$

- Example, find the number of iterations when tol$=10^{-12}$.

$$\begin{aligned} e^{-(\pi^2h^2/4)k} &\approx 10^{-12} \\ -(\pi^2h^2/4)k &\approx \ln 10^{-12} \;\approx\; 24 \;\; (27.6...) \\ k &\approx \frac{28\cdot 2}{\pi^2h^2} \;\approx\; 6N^2 \end{aligned}$$

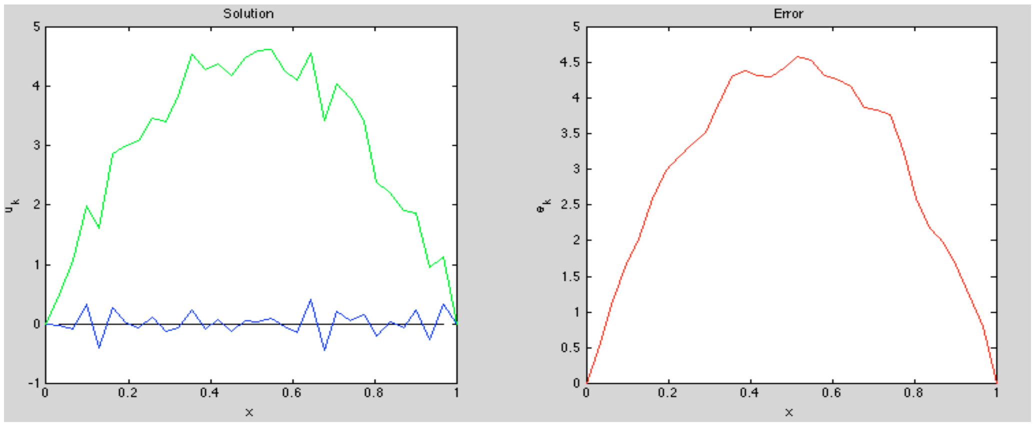
Here, N=number of points in each direction.

Recap

- Low-wavenumber components decay at a fixed rate: $e^{-\lambda_{\min}\Delta t k}$.
- Stability mandates $\Delta t < h^2/4 = 1/4(N+1)^{-2}$.
- Number of steps scales like N^2.
- Note, if $\sigma = 1$, then *highest* and *lowest* wavenumber components decay at *same* rate.
- If $\frac{1}{2} < \sigma < 1$, high wavenumber components of error decay very fast. We say that damped Jacobi iteration is a *smoother*.

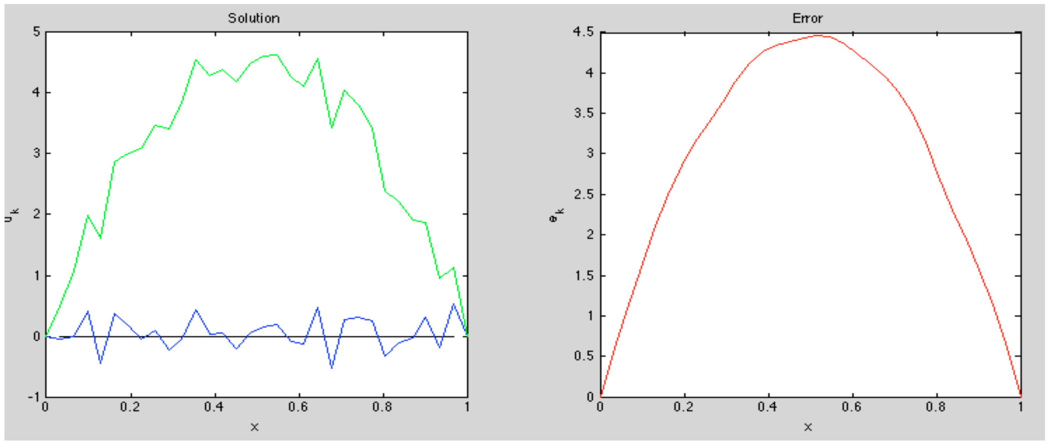
Example: 1D Jacobi Iteration

Solution after 1 iteration



Error after 1 iteration

Solution after 5 itierations



Error after 5 itierations

Observations:

- Error, $\mathbf{e}_k$ is smooth after just a few iterations:
  - Error components are $\approx \hat{u}_j e^{-j^2 k h^2 \pi^2/4} \sin k\pi x_j$, and components for $j > 1$ rapidly go to zero.
- Exact solution is $\mathbf{u} = \mathbf{u}_k + \mathbf{e}_k$ ($\mathbf{e}_k$ unknown, but smooth).
- Error satisfies, and can be computed from,

$$A\mathbf{e}_k = \mathbf{r}_k \qquad (:= \mathbf{f} - A\mathbf{u}_k = A\mathbf{u} - A\mathbf{u}_k = A\mathbf{e}_k)\,.$$

- These observations suggest that the *error* can be well approximated on a coarser grid and added back to $\mathbf{u}_k$ to improve the current guess.
- The two steps, *smooth* and *coarse-grid correction* are at the heart of one of the fastest iteration strategies, known as **multigrid**.

Multigrid:

- Solve $A\mathbf{e}_k = \mathbf{r}_k$ approximately on a coarse grid and set $\tilde{\mathbf{u}}_k = \mathbf{u}_k + \tilde{\mathbf{e}}_k$.
- Approximation strategy is similar to least squares. Let

$$\begin{aligned} \tilde{\mathbf{e}}_k &= V\mathbf{e}_c, \qquad \text{and} \\ AV\mathbf{e}_c &\approx \mathbf{r}, \end{aligned}$$

  where V is an $n \times n_c$ matrix with $n_c \approx n/2$.

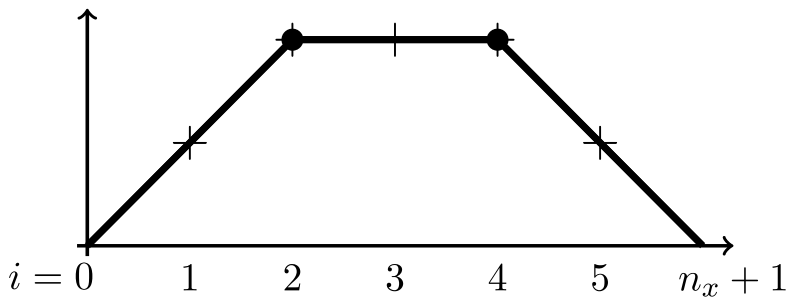
- Typically, columns of V interpolate coarse point values to their mid-points.
- Most common approach (for A SPD) is to require $\mathbf{e}_c$ to solve

$$\begin{aligned} & V^T \left[AV\mathbf{e}_c - \mathbf{r} \right] = 0 \\ \Longrightarrow\ & \tilde{\mathbf{e}}_k = V \left(V^T A V\right)^{-1} V^T \mathbf{r} = V \left(V^T A V\right)^{-1} V^T A\, \mathbf{e}_k. \end{aligned}$$

- For A SPD, $\tilde{\mathbf{e}}_k$ is the A-orthogonal projection of $\mathbf{e}_k$ onto $\mathcal{R}(V)$.

An example of V for $n = 5$ and n_c=2 is

$$V = \begin{bmatrix} \frac{1}{2} & \\ 1 & \\ \frac{1}{2} & \frac{1}{2} \\ & 1 \\ & \frac{1}{2} \end{bmatrix}$$



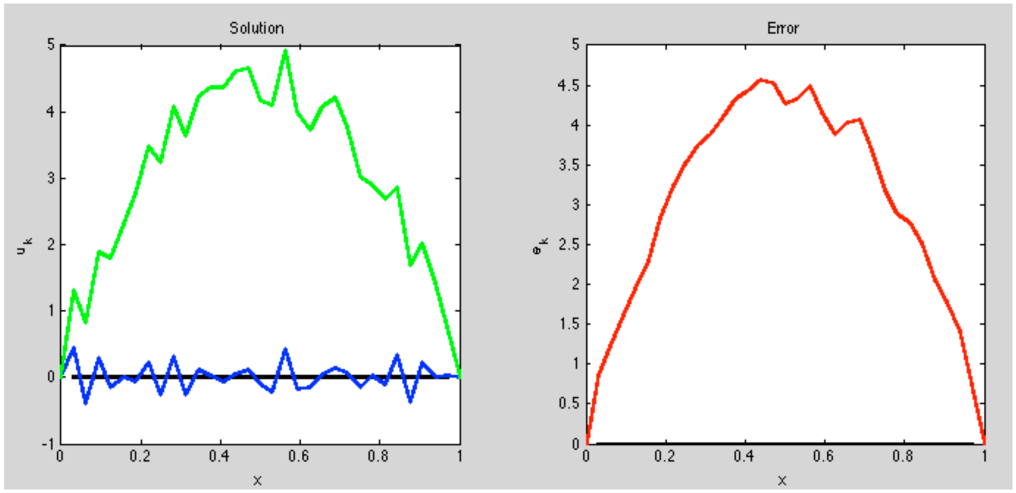
Coarse-to-fine interpolation

```
% Multigrid stuff % n must be odd!

nc = (n-1)/2; V=spalloc(n,nc,n*nc); i=1;
for j=1:nc;
   V(i,j)=1/2; V(i+1,j)=1; V(i+2,j)=1/2; i=i+2;
end;
Ac = V'*A*V;

% A Simple Two-Level MG iteration:

for k=1:5000

    r = f-A*u;                 % Smoothing step
    u = u + d*r;

    r  = f-A*u;                % Coarse-grid correction
    rc = V'*r;
    ec = V*( Ac \ rc );
    u = u+ec;

end;
```

poisson_mg.m demo

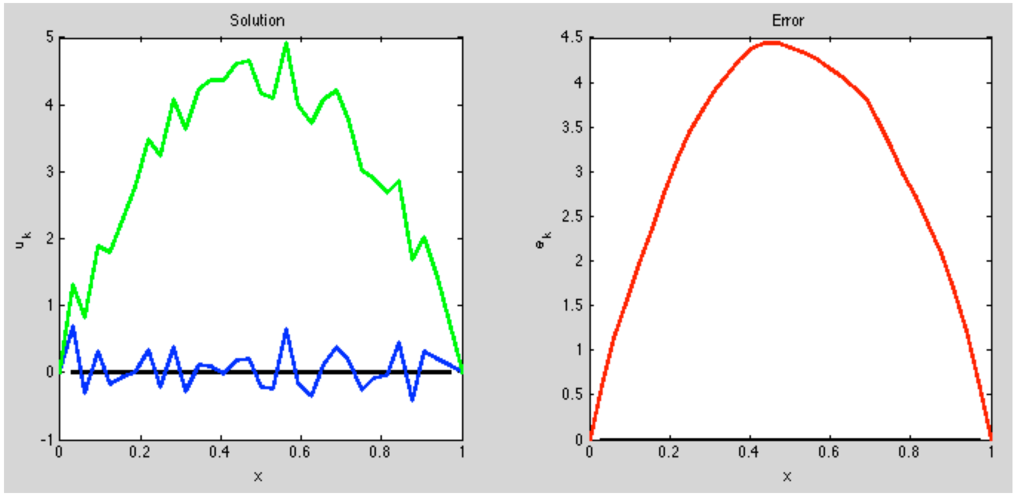
Example: Damped Jacobi (Richardson) Iteration

Solution after 1 iteration



Error after 1 iteration

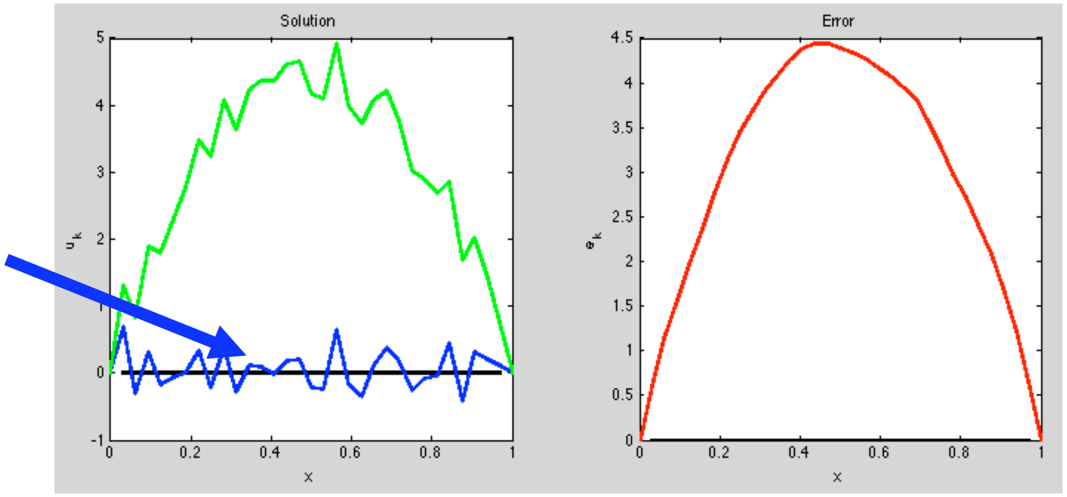
Solution after 5 itierations



Error after 5 itierations

Multigrid Summary – Main Ideas

Solution after 5 iterations



Error after 5 iterations

- Take a few damped-Jacobi steps (smoothing the *error*), to get $\mathbf{u}_k$.
- Approximate this *smooth error*, $\mathbf{e}_k := \mathbf{u} - \mathbf{u}_k$, on a coarser grid.
- Exact error satisfies

$$A\mathbf{e}_k = A\mathbf{u} - A\mathbf{u}_k = \mathbf{f} - A\mathbf{u} =: \mathbf{r}_k.$$

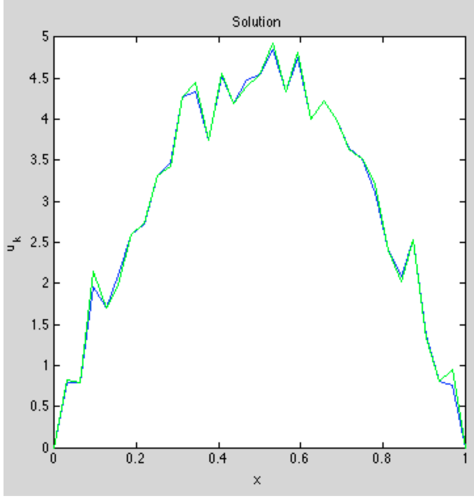
- Let $\mathbf{e}_f := V\mathbf{e}_c$ be the *interpolant* of $\mathbf{e}_c$, the coarse-grid approximation to $\mathbf{e}_k$.
- $\mathbf{e}_f$ is *closest element* in $\mathcal{R}(V)$ to $\mathbf{e}_k$ (in the A-norm), given by the **projection**:

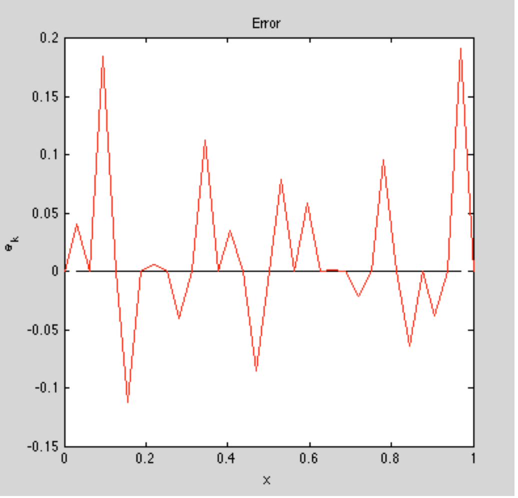
$$\mathbf{e}_f = V\left(V^T AV\right)^{-1} V^T A\mathbf{e}_k = V\left((A_c)^{-1}\right) V^T\mathbf{r}_k.$$

- **Update** $\mathbf{u}_k$ with the coarse-grid correction: $\boxed{\mathbf{u}_k \longleftarrow \mathbf{u}_k + \mathbf{e}_f.}$
- Smooth again and repeat.

Example: Two-Level Multigrid

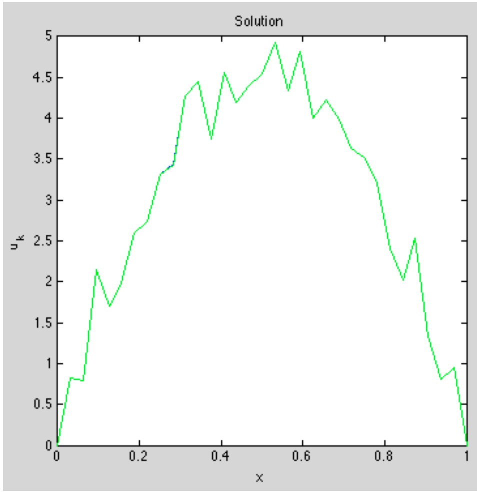
Solution after 1 iteration

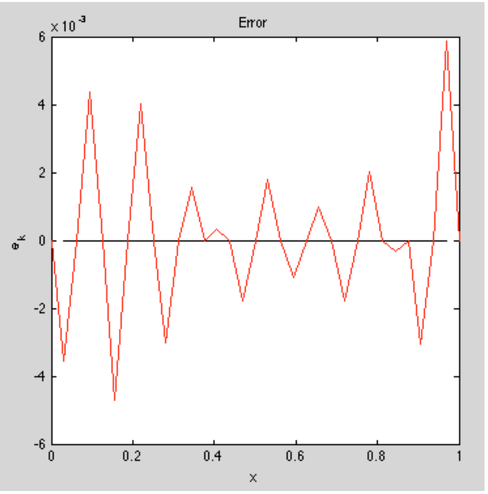




Error after 1 iteration

Solution after 5 itierations

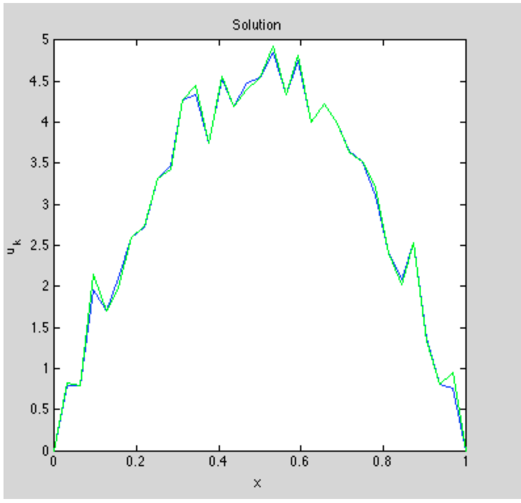




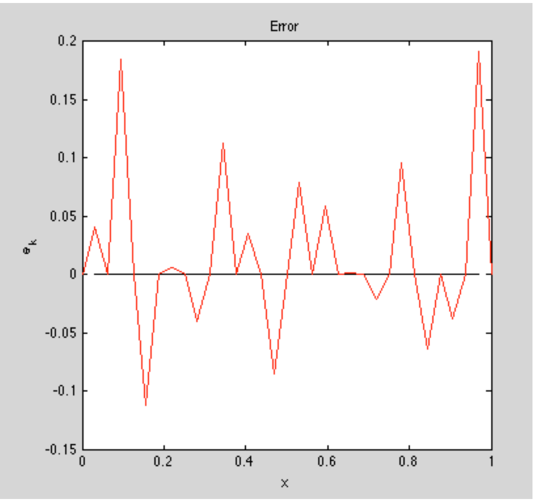
Error after 5 itierations

Example: Two-Level Multigrid

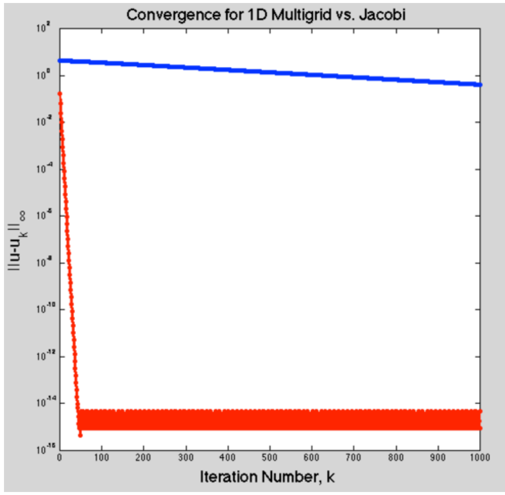
Solution after 1 iteration



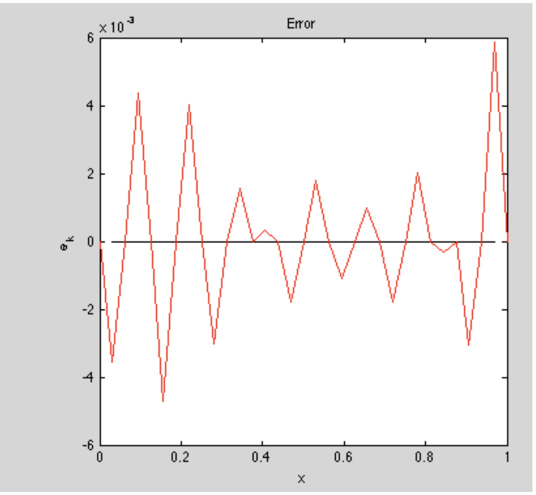
Error after 1 iteration



Iteration History



Error after 5 itierations



Multigrid Comments

- Smoothing can be improved using under-relaxation (σ = 2/3 is optimal for 1D case).
  - Basically – want more of the high-end error spectrum to be damped.
- We've described in the previous slides ***two-level multigrid***.
- System in A_c is less expensive to solve, but is typically best solved by repeating the smooth/coarse-grid correct pair on yet another level down.
- Can recur until $n_c \sim 1$, at which point system is easy to solve.
- Typical MG complexity is O(n) or O(n log n), with very good constants in higher space dimensions ($N_c = N/2 \rightarrow n_c = n/8$ in 3D).
- For high aspect-ratio cells, variable coefficients, etc., smoothing and coarsening strategies require more care, so this continues to be an active research area.

A Two-Level Multigrid Algorithm:

Solve $A\mathbf{x} = \mathbf{r}_0$, starting with $\mathbf{x}_0 = \mathbf{0}$.

```
for $j = 1 : n_{smooth}$
    $\mathbf{s}_j = \sigma S^{-1} \mathbf{r}_{j-1}$
    $\mathbf{x}_j = \mathbf{x}_{j-1} + \mathbf{s}_j$
    $\mathbf{r}_j = \mathbf{b} - A\mathbf{x}_j = \mathbf{r}_{j-1} - A\mathbf{s}_j$
end
$\tilde{\mathbf{r}}_c = V^T \mathbf{r}_j$
$\tilde{\mathbf{e}}_c = A_c^{-1} \tilde{\mathbf{r}}_c,$          $A_c := V^T A V$
$\mathbf{e}_f = V\tilde{\mathbf{e}}_c = V A_c^{-1} V^T \mathbf{r}_j, = V A_c^{-1} V^T A \mathbf{e}_j$

$\mathbf{x}_c = \mathbf{x}_j + \mathbf{e}_f$          (coarse-grid correction)

Return to smoothing step or return $\mathbf{x}_c$.
```

- Produces a polynomial in AS^{-1} and $P_V := V A_c^{-1} V^T$.
- Does *not* produce the best-fit (i.e., *projection*) in $\{\mathbf{s}_j\} \bigcup \{\mathbf{e}_f\}$.
- Best fit can be realized by coupling multigrid with GMRES or conjugate gradient iteration.
- This is known as *preconditioned* GMRES or CG or, multigrid-preconditioned GMRES or CG.
- These are state-of-the-art methods as they are fast and robust.

Recall GMRES/CG Projection for $A\mathbf{x} = \mathbf{b}$:

Solve $A\mathbf{x} = \mathbf{b}$ starting with $\mathbf{x}_0 = \mathbf{0}$ and $\mathbf{r}_0 := \mathbf{b}$.

```
for $k = 1 : k_{\max}$
    $\mathbf{p}_k = \mathbf{r}_{k-1} - \sum_{j=1}^{k-1} \beta_j \mathbf{p}_j$   such that $\mathbf{p}_k^T A \mathbf{p}_j = 0$
    $\mathbf{w}_k = A\mathbf{p}_k$
    $\alpha_k = \frac{\mathbf{p}_k^T A \mathbf{e}_k}{\mathbf{p}_k^T A \mathbf{p}_k} = \frac{\mathbf{p}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{w}_k}$
    $\mathbf{x}_k = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
    $\mathbf{r}_k = \mathbf{r}_k - \alpha_k \mathbf{w}_k$
end
```

- CG: Produces best-fit in $K_k(A, \mathbf{b})$ in the $|| \cdot ||_A$ norm, assuming A is SPD.
- CG: Short-term recurrence in computation of $\mathbf{p}_k$ ($\beta_j = 0$ if $j < k - 1$).
- GMRES: If A not SPD, used $\alpha_k = (\mathbf{w}_k^T \mathbf{r}_k)/(\mathbf{w}_k^T \mathbf{w}_k)$ to get best fit in $A^T A$ norm, and insist that $(\mathbf{w}_k^T \mathbf{w}_j) = 0$.
- GMRES: No short term recurrence.
- (Both algorithms have more standard formulations, which should be used.)

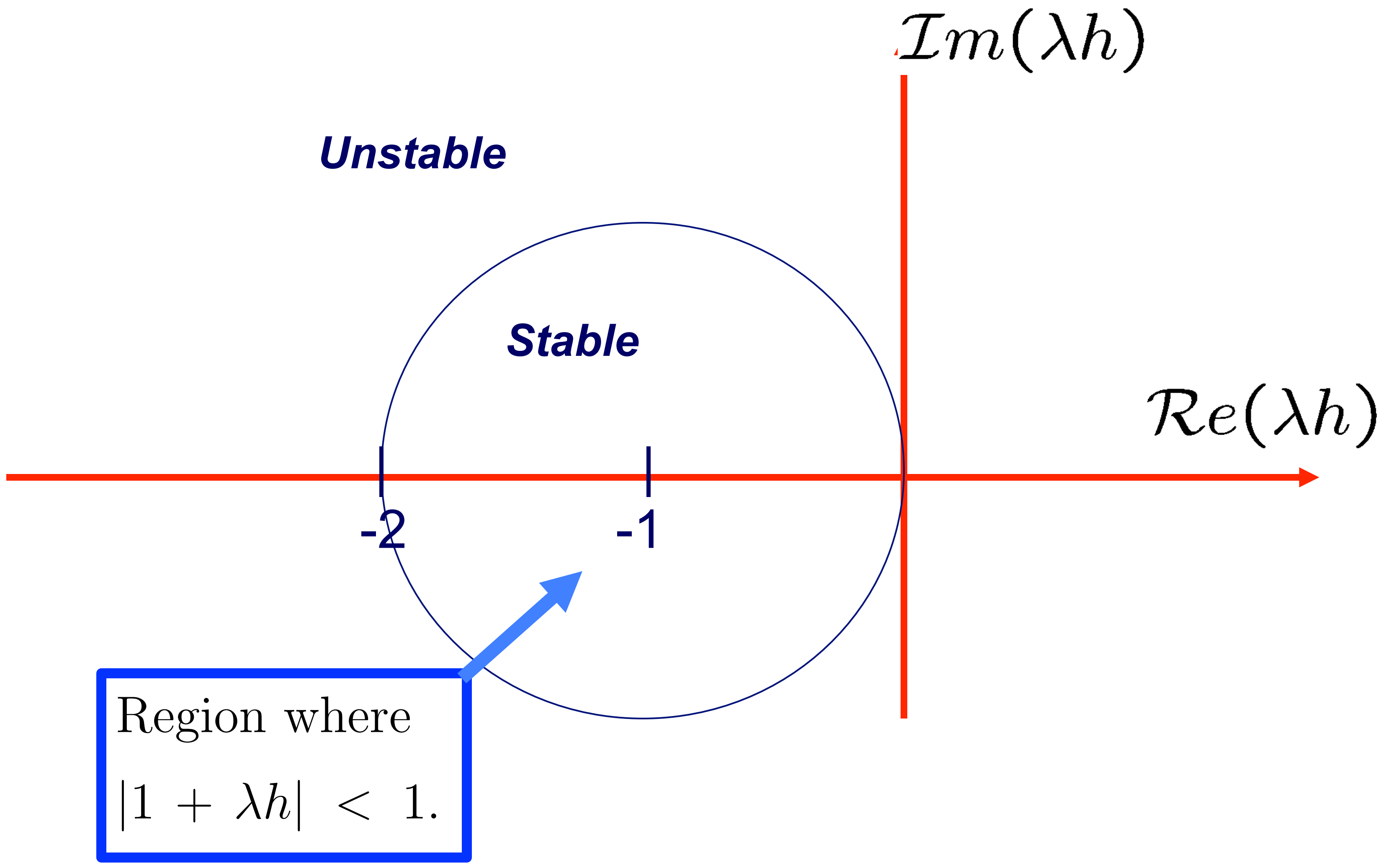
Preconditioned GMRES/CG Projection for $A\mathbf{x} = \mathbf{b}$:

Solve $A\mathbf{x} = \mathbf{b}$ starting with $\mathbf{x}_0 = \mathbf{0}$ and $\mathbf{r}_0 := \mathbf{b}$.

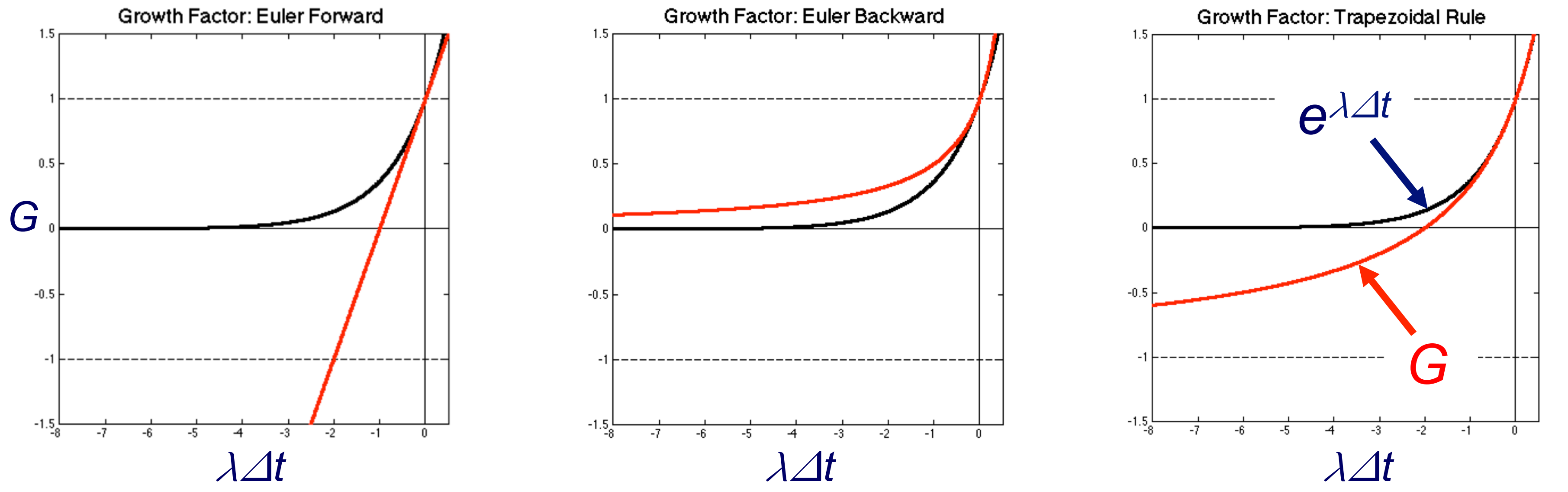
```
for $k = 1 : k_{\max}$
    Solve $M\mathbf{z}_k = \mathbf{r}_{k-1}$          **Precondtioning step.**
    $\mathbf{p}_k = \mathbf{r}_k - \sum_{j=1}^{k-1} \beta_j \mathbf{p}_j$   such that $\mathbf{p}_k^T A \mathbf{p}_j = 0$
    $\mathbf{w}_k = A\mathbf{p}_k$
    $\alpha_k = \frac{\mathbf{p}_k^T A \mathbf{e}_k}{\mathbf{p}_k^T A \mathbf{p}_k} = \frac{\mathbf{p}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{w}_k}$
    $\mathbf{x}_k = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
    $\mathbf{r}_k = \mathbf{r}_k - \alpha_k \mathbf{w}_k$
end
```

- Produces best-fit in $K_k(M^{-1}A, M^{-1}\mathbf{b})$.
- For CG, M must be SPD to get short-term recurrence in $\mathbf{p}_k$ step.
- To solve the $M\mathbf{z} = \mathbf{r}$ step, can use, for example, multigrid.

Stability Region for Euler's Method


$\mathcal{I}m(\lambda h)$
Unstable
Stable
$\mathcal{R}e(\lambda h)$
-2
-1
Region where $|1 + \lambda h| < 1.$

Growth Factors for Real λ



- Each growth factor approximates $e^{\lambda\Delta t}$ for $\lambda\Delta t \rightarrow 0$
- For EF, |G| is not bounded by 1
- For Trapezoidal Rule, local (small$\lambda\Delta$t) approximation is $O(\lambda\Delta t^2)$, but |G| $\rightarrow$ -1 as $\lambda\Delta t \rightarrow -\infty$. [Trapezoid method is not ***L-stable.***]
- BDF2 will give 2nd-order accuracy, stability, and $|G|\rightarrow 0$ as $\lambda\Delta t \rightarrow -\infty$.