Elliptic Problems / Multigrid
Summary of Hyperbolic PDEs

- We looked at a simple linear and a nonlinear scalar hyperbolic PDE
  - There is a speed associated with the change of the solution
  - Explicit methods cannot take a step larger than the time it takes for a solution to cross a single zone
    - For nonlinear equations, this speed changes on the grid—you need to find the most restrictive timestep in a zone
  - Upwinding (for linear advection) gives a stable method
    - For nonlinear PDEs, this idea is contained in the solution to the Riemann problem
- We'll continue to use my notes linked online
Elliptic Problems in Physics

- Gravitational and Electric potentials (Poisson equation)
  \[ \nabla^2 \Phi = 4\pi G \rho \quad \nabla^2 \Phi = -\frac{\rho}{\epsilon} \]
  (gravitational potential) (electric potential)

- Helmholtz equation:
  \[ (\alpha + \nabla \cdot \beta \nabla) \Phi = f \]
  - Often arises by discretizing or separating out time in a PDE
Elliptic Problems in Physics

- Sometimes we have a system with different PDE types.
- Fluid dynamics w/ self gravity
  \[
  \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U) = 0 \\
  \frac{\partial \rho U}{\partial t} + \nabla \cdot (\rho UU) + \nabla p = \rho \nabla \Phi \\
  \frac{\partial \rho E}{\partial t} + \nabla \cdot (\rho UE + Up) = \rho U \cdot \nabla \Phi
  \]

- Incompressible hydrodynamics
  \[
  U_t + U \cdot \nabla U + \nabla p = 0 \\
  \nabla \cdot U = 0
  \]
  (constraint on velocity)

  - Poisson equation enforces constraint
Elliptic PDEs and Boundary Conditions

- There is no time-dependence in these equations
- **Field responds instantly to the boundary conditions and source**
  - There is no propagation speed as in the hyperbolic / advection equations we studied
  - Treatment of boundary conditions becomes essential
Relaxation Methods

- We'll see that for a broad class of elliptic problems, relaxation methods (iterative) are easy to implement
  - Multigrid is a technique that we'll study to accelerate the convergence of relaxation methods
- There is an excellent book, *The Multigrid Tutorial*, that gives a great introduction to these methods and the math behind them
  - We'll follow this a bit, but focus on just some of the main results
  - We'll do things in terms of cell-centered finite-difference / finite-volume grids (the text focuses on finite-difference)
    - Differences come up in boundary conditions and transferring the problem through a hierarchy of grids (as we'll see shortly)
Grid Types

finite-difference

cell-centered finite-difference

finite-volume
Grid Types

● Major difference between grid types:
  – Does the data exist precisely on the boundary?
  – Since boundary conditions critically affect the solution, we need to keep the centering of the data in mind

● Cell-centered finite-difference vs. Finite-volume:
  – To second-order accuracy, we can treat the cell-averages as centered in the zone

\[ \langle f \rangle_i = f(x_i) + O(\Delta x^2) \]

  • Blackboard derivation...
  – Our methods will be second-order accurate for both of these grid types
Model Problem

1. Consider a one-dimensional Poisson equation:
   \[ \phi'' = f \]
   - This is a second-order ODE, so we need 2 boundary conditions
     - Dirichlet: \( \phi(a) = A \)
     - Neumann: \( \phi'(a) = C \)
   - In two or more dimensions, this would be a PDE

2. We already saw the shooting method for solving this (when we studied ODEs)
   - Dirichlet BCs at both end.
   - Solve by picking an initial guess for the derivative at starting boundary and use Newton's method to match the far BC
   - That does not translate to multi-dimensions easily
Model Problem

- To allow us to test things, let's pick something with an analytic solution

\[ \phi'' = \sin(x) \quad \phi(0) = 0, \; \phi(1) = 0 \]

- Solution:

\[ \phi(x) = -\sin(x) + x \sin(1) \]
Relaxation

- Recall from our lecture on derivatives that a second-order accurate difference for the second derivative is:

\[ \phi''_i = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \]

- True on F-D or F-V grids

- Our 1-d Poisson equation becomes:

\[ \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} = f_i \]

- Solve for a single zone:

\[ \phi_i = \frac{1}{2} \left( \phi_{i+1} + \phi_{i-1} - \Delta x^2 f_i \right) \]

- Set of coupled algebraic equations (think matrices)
Relaxation

• Instead of a direct matrix solve, we'll use an iterative method
  – We are just shy of being diagonally dominant, nevertheless, these methods converge

• Jacobi iteration
  – Pick initial guess: $\phi_i^{(0)}$
  – Improve the guess through relaxation:
    \[
    \phi_i^{(k+1)} = \frac{1}{2} (\phi_{i+1}^{(k)} + \phi_{i-1}^{(k)} - \Delta x^2 f_i)
    \]
  – Assess the error, if needed iterate...

• Gauss-Seidel iteration
  – Use new data as it becomes available:
    \[
    \phi_i^{(k+1)} \leftarrow \frac{1}{2} (\phi_{i+1}^{(k)} + \phi_{i-1}^{(k)} - \Delta x^2 f_i)
    \]
Relaxation

• We previously saw relaxation when we discussed linear algebra. There we wrote our system as:

\[ a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \]
\[ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \]
\[ \vdots \]
\[ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \]

• And iterated as:

\[
    x_1^{(k+1)} = - \frac{1}{a_{11}} \left( a_{12}x_2^{(k)} + a_{13}x_3^{(k)} + \ldots + a_{1n}x_n^{(k)} - b_1 \right)
\]
\[
    x_2^{(k+1)} = - \frac{1}{a_{22}} \left( a_{21}x_1^{(k)} + a_{23}x_3^{(k)} + \ldots + a_{2n}x_n^{(k)} - b_2 \right)
\]
\[ \vdots \]
\[
    x_n^{(k+1)} = - \frac{1}{a_{nn}} \left( a_{n1}x_1^{(k)} + a_{n2}x_2^{(k)} + \ldots + a_{n,n-1}x_{n-1}^{(k)} - b_n \right)
\]
Relaxation

- No need to write out the matrix with relaxation
- A popular version of Gauss-Seidel is called red-black Gauss-Seidel (think of a checkerboard)
  - First update the odd points—they only depend on the values of the even points
  - Then update the even points—they only depend on the values of the odd points
  - Decoupling between the odd and even is attractive for parallelization
Boundary Conditions: FV vs. FD

- **F-D grid:**
  - We have a point exactly on the boundary—iterate only over the interior points

- **F-V or cc F-D grid:**
  - Must interpolate to the boundary
Finite-Volume BCs

- Dirichlet: we need the value on the boundary itself to satisfy the boundary condition:
  \[ \phi(a) = A \]

- Use ghost cells to extend past the physical domain

- Interpolation to fill the ghost cell:
  - Naive guess: \( \phi_{lo-1} = A \)
  - Second-order accurate:
    \[ A = \frac{\phi_{lo} + \phi_{lo-1}}{2} \]
Finite-Volume BCs

- **Neumann**: we need the gradient, centered at the boundary, to match the given value

\[ \phi'(a) = C \]

- Use ghost cells to extend past the physical domain

- Centered difference at boundary:

\[ C = \frac{\phi_{10} - \phi_{10-1}}{\Delta x} \]

  - This is second-order accurate
Solvable Boundary Conditions

• Integrate our Poisson equation over the domain:

\[ \int_{\Omega} \nabla^2 \phi \, d\Omega = \int_{\partial\Omega} \nabla \phi \cdot \mathbf{n} \, dS = \int_{\Omega} f \, d\Omega \]

  – Consider homogeneous Neumann BCs on all sides, \( \nabla \phi \cdot \mathbf{n} = 0 \)
  
  • Our source must satisfy:

\[ \int_{\Omega} f \, d\Omega = 0 \]

  – Likewise, with periodic BCs all around, the flux in one end of the domain is the flux out the other end, so again, we require

\[ \int_{\Omega} f \, d\Omega = 0 \]

  – We will not converge if our source is not consistent with the boundary conditions
Solvable Boundary Conditions

• Another way to see this:
  – Consider a 1-d Laplace equation $\phi'' = 0$
  – Solution is just a line: $\phi = ax + b$
  – If you specify different inhomogeneous Neumann BCs on each end, then you are giving conflicting values for the slope—unsolvable!
Error and Norms

- There are many different norms that can be used to determine the error.

- General p-norm:
  \[ \| e \|_p = \left( \Delta x \sum_{i=1}^{N} |e_i|^p \right)^{1/p} \]
  - We already saw the L2 norm.
  - Also interesting are the L1 norm:
    \[ \| e \|_1 = \Delta x \sum_{i=1}^{N} |e_i| \]
  - And the inf norm:
    \[ \| e \|_{\infty} = \max_i \{ |e_i| \} \]
Error and Norms

- The norm gives us a single number with which to measure if we are converged
  - The choice of norm should not matter—*if we converge, we should converge in all norms*
  - L2 falls somewhere between L1 and the inf-norm in magnitude
  - L1 and L2 are more “global”—all values contribute
Error and Norms

- We still need to define the error that we are taking the norm of
  - For our test problems, we can compare to the analytic solution, but that's not general
  - Only other measure: how well we satisfy the discrete equation—this is the **residual**

\[
r_i \equiv f_i - \frac{\phi^{(k)}_{i+1} - 2\phi^{(k)}_i + \phi^{(k)}_{i-1}}{\Delta x^2}
\]

- We use the **source norm** to provide a size to compare to. Stop when:
\[
\| r \| < \epsilon \| f \|
\]
Implementation

- We need to fill the ghost cells after each iteration to reflect the change in the solution
- Let's start by doing a fixed number of iterations
- Let's look at the code...
Convergence

- Residual error vs. true error using Gauss-Seidel (red-black)

As we increase the resolution, we need many more iterations to converge.

The true error stalls at a relatively high value—this is the truncation error of our method.

The residual error stalls only near machine epsilon—our solution satisfies our discrete equation “exactly.”

code: smooth.py (+multigrid.py, patch1d.py)
Convergence with Different Norms

Note that all 3 norms show the same trends...

code: smooth-norms.py (+multigrid.py, patch1d.py)
Effect of Boundary Conditions

- What if we didn't take into account the BCs properly?
  - i.e. Set the ghost cell value to the BC value instead of interpolate to the actual boundary

First-order accurate convergence with naive BCs

Look at the convergence!
Neumann BCs

- Let's change the code to solve:

\[ \phi'' = -4\pi^2 \cos(2\pi x) \quad \phi'(0) = 0 \quad \phi'(1) = 0 \]
Behavior of Different Modes

- Consider Laplace's equation:
  \[ \nabla^2 \phi = 0, \ \phi(0) = 0, \ \phi(1) = 0 \]
  - The solution is simply: \( \phi(x) = 0 \)
- Pick a single mode sine wave as an initial guess:
  \[ \phi^{(0)}(x) = \sin(2\pi mx) \]
  - The error after X iterations is simply \( \phi(x) \)
- Let's look at how different modes behave
Initial guess after 1, 10, 100, and 1000 smoothings for m = 1 and 128 zones
Not much progress...

code: smooth-modes.py
(+multigrid.py, patch1d.py)
M=8 Mode

Initial guess after 1, 10, 100, and 1000 smoothings for m = 8 and 128 zones
Here we see that after 100 smoothings, the error is mostly gone

code: smooth-modes.py
(+multigrid.py, patch1d.py)
M=16 Mode

Initial guess after 1, 10, 100, and 1000 smoothings for $m = 16$ and 128 zones. Now after 10 smoothings, the error is small.

code: smooth-modes.py
(+multigrid.py, patch1d.py)
Multiple Modes

Initial guess after 1, 10, 100, and 1000 smoothings for an initial guess consisting of $m=1, 8,$ and $16$ modes (equally weighted) and 128 zones. Notice that the highest wavelength errors disappear fastest.

code: smooth-modes.py
(+multigrid.py, patch1d.py)
Relaxation Observations

- Observe that the higher-frequency (shorter wavelength) errors smooth away fastest
  - Here we measure the wavelength in terms of number of zones

- Every zone is linked to every other zone
  - If an error has a wavelength of $N$ zones, then $N$ iterations are required to communicate across it

- Our PDE is linear
  - Each mode evolves independent of the others
Coarsening

- On a coarser grid, long wavelength errors appear to have a shorter wavelength
- Consider $m = 5$ mode on a 64, 32, and 16 zone grid
  - Notice that on a 64 zone mesh the error appears smooth, but on a 16 zone mesh, it is very oscillatory
Coarsening

- Since the error appears at higher frequency on the coarser grid, we expect that we will solve with fewer iterations.

Error for $m = 5$ mode with 64 zones (left) and 16 zones (right) after 1, 10, and 100 iterations.

code: smooth-coarsen.py
(+multigrid.py, patch1d.py)
However, if we coarsen a short wavelength error, it can appear to have a longer wavelength on the coarse grid—this is aliasing.

$m = 13$ mode on a 64, 32, and 16 zone grid
On To Multigrid

- Multigrid is a method to accelerate the convergence of relaxation
  - It eliminates the short wavelength errors on the original grid
  - Coarsens the problem and eliminates the formerly long wavelength errors on the new coarser grid
- Multigrid is very widely used in astrophysics to solve the Poisson equation for the gravitational potential
- Multigrid relies on a method to move the solution up and down a hierarchy of grids
- By coarsening the data, we accelerate the convergence as compared to straight relaxation
- We'll follow the notation from The Multigrid Tutorial text
Residual Equation

- For simplicity in notation, we will write the discrete Laplacian operator as $L$
  - Our discrete equation is then $L\phi = f$
  - We'll denote our current approximation to the true solution as $v$
  - The error is then $e = \phi - v$

- Our operator is linear, so

$$Le = L\phi - Lv = f - Lv = r$$

- Note that our error satisfies the same type of equation, with the residual as the source
  - We can relax on the error and use it to correct our current guess, $v$
Coarse Grid Correction

(Briggs et al., Ch. 3)

- We know:
  - Relaxation eliminates short wavelength errors efficiently
  - Long wavelength errors appear shorter on a coarse grid
- We can use multiple grids to exploit this behavior

Two-grid correction scheme

- Relax $L \phi = f$ on $h$ to obtain the approximation $v^h$
  - Compute the residual: $r = f - Lv^h$
    - Relax the residual equation: $L e = r$ on $2h$ to get an approximate error, $e^{2h}$
    - Correct the approximation on $h$ with the error from $2h$ via $v^h \leftarrow v^h + e^{2h}$
Implementation?

- We need a way to transfer the data back and forth between the coarse and fine grids
  - **Restriction**: take fine data and transfer it to the coarse grid
  - **Prolongation**: use coarse data to initialize the finer cells
- The restriction and prolongation operations will depend on the type of grid used
- We will only worry about jumps of 2x in resolution—this is the most common
Prolongation

- Moving data from the coarse mesh to the fine mesh
  - Finite-volume mesh: always 2 fine cells within a single coarse cell
  - Simple method: direct injection:
    \[ \phi_i^h = \phi_j^{2h} \quad \phi_{i+1}^h = \phi_j^{2h} \]
  - Better: linear reconstruction:
    \[ \phi(x) = m(x - x_j) + \phi_j^{2h} \]
    \[ m = \frac{\phi_{j+1}^{2h} - \phi_{j-1}^{2h}}{2\Delta x^{2h}} \]
    - Conservative. Integration over zones gives:
      \[ \phi_i^h = \phi_j^{2h} - \frac{1}{4} m \Delta x^{2h} \]
      \[ \phi_{i+1}^h = \phi_j^{2h} + \frac{1}{4} m \Delta x^{2h} \]
Restriction

- Moving data from fine mesh to the coarse mesh
  - Conservative quantities: average

\[ \phi_{2h}^j = \frac{1}{2} (\phi_i^h + \phi_{i+1}^h) \]
Aside: (Node-centered) Finite-Difference Grid

- On a F-D grid, for each pair of fine point, one of the fine points corresponds exactly to a coarse point
  - Simply copy that point
  - Average the coarse to initialize the other fine point
With the grid transfer operations defined, we can fill in more details:

Two-grid correction scheme

- Relax $L^h \phi^h = f^h$ on $h$ to obtain the approximation $v^h$
  - Compute the residual: $r^h = f^h - L v^h$
  - Restrict $r^h$ to $2h$ producing $r^{2h}$
    - Solve $L^{2h} e^{2h} = r^{2h}$ on $2h$
    - Prolong $e^{2h}$ to $h$ to produce $e^h$
  - Correct the approximation on $h$ with the error from $2h$ via $v^h \leftarrow v^h + e^h$
- Relax $L^h \phi^h = f^h$ with initial guess $v^h$
What Now?

- We still have a step that says “solve” on the coarse grid.
  - We know relaxation will eventually stall as it deals with the long wavelength errors
- Solution: apply the two-grid technique recursively
  - Keep doing it until you get to a really small grid (like 2 zones)
  - Exactly solve the problem on the coarsest grid (relaxation may work just fine there)
V-Cycles

- Simplest hierarchy: \textit{V-cycle}
- At each level, do a few (~3) smoothing iterations to eliminate the short wavelength errors there
  - Coarsen and solve the error equation on the coarse grid
- Once you reach the coarsest grid, solve exactly
- On the upward part, transfer the error, correct, and smooth a few times before passing it up to the next higher level

\[ \text{Solve exactly} \]
Consider the hierarchy of grids on the right

- Each has a single ghost cell
- Coarsest grid has 2 zones in the interior—minimum needed to enforce BCs

You are not restricted to grids that are a power of two, but it will affect the size of your coarsest grid

Sometimes more elaborate methods (like conjugate gradient) are used for the bottom solve
Stopping Criteria

- We can continue to iterate (perform V-cycle after V-cycle) until we converge.

- Typical convergence criteria is:
  \[ \| r \| < \epsilon \| f \| \]
  - The source norm provides a scale to measure against.
  - If the source is zero, then we stop when
    \[ \| r \| < \epsilon \]
Testing it Out

- You should be able to test the different components individually
  - Multigrid simply accelerates the convergence of relaxation (smoothing). If your smoothing routine doesn't converge as 2\textsuperscript{nd} order, then the MG solve won't either
  - Prolongation and restriction can be tested by initializing dummy data and examining the results of the operations
  - Print! At each point in the cycle, print out the norm of the error (before and after smoothing)
- Let's look at the code...
Performance

- Our model problem with 256 zones:

Notice that each V-cycle reduces the residual by about an order-of-magnitude.

That's a good rule-of-thumb.
MG Convergence with Resolution

- A recurring theme—always check the convergence of your routines.

Note: this should look identical to what you get from smoothing/relaxation alone. MG does not reduce the error, it just accelerates the convergence.
There are many other arrangements of grid hierarchy.

Full multigrid (shown below) starts on the coarsest grid and works up to the finest

- You can continue to do V-cycles afterwards to further reduce the error
Inhomogeneous BCs

- Near the left boundary, our discretized equation appears as:

\[
\frac{\phi_{lo-1} - 2\phi_{lo} + \phi_{lo+1}}{\Delta x^2} = f_{lo}
\]

- Inhomogeneous BCs would give the condition:

\[
\phi_{lo-1} = 2\phi_{l} - \phi_{lo}
\]

- Substituting this in:

\[
\frac{2\phi_{l} - \phi_{lo} - 2\phi_{lo} + \phi_{lo+1}}{\Delta x^2} = f_{lo}
\]

- Rewriting:

\[
\frac{(-\phi_{lo}) - 2\phi_{lo} + \phi_{lo+1}}{\Delta x^2} = f_{lo} - \frac{2\phi_{l}}{\Delta x^2}
\]
Extending to multi-dimensions is easy. Now our discretized equation is:

\[
\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j}
\]

- A lot of times, we make the unit cells square
- The prolongation and restriction routines extend to multi-dimensions in a straightforward manner.
multigrid solution of $u_{xx} + u_{yy} = -2[(1-6x^2)y^2(1-y^2) + (1-6y^2)x^2(1-x^2)]$
Consider a constant-coefficient Helmholtz equation:

\[(\alpha - \beta \nabla^2) \phi = f\]

- This can be discretized as

\[
\alpha \phi_{i,j} - \beta \left( \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} \right) = f_{i,j}
\]

- The only changes we need to make are:
  - Our relaxation/smoothing routine needs to be modified:

\[
\phi_{i,j} \leftarrow \left( f_{i,j} + \frac{\beta}{\Delta x^2} \phi_{i+1,j} + \frac{\beta}{\Delta x^2} \phi_{i-1,j} + \frac{\beta}{\Delta y^2} \phi_{i,j+1} + \frac{\beta}{\Delta y^2} \phi_{i,j-1} \right) / \left( \alpha + \frac{2\beta}{\Delta x^2} + \frac{2\beta}{\Delta y^2} \right)
\]
  - Our residual equation needs to be changed

- Note that \(\alpha = 0, \beta = -1\) gives the Poisson equation
Fourier Transforms

- We can also use FFTs directly for solving (some) PDEs.
- Consider the 1-d Poisson equation:

\[
\frac{d^2 \phi_n}{dx^2} = f_n
\]

- Express things in terms of the transforms

\[
\phi(x) = \int \Phi(k)e^{2\pi ikx} \, dk
\]

\[
f(x) = \int F(k)e^{2\pi ikx} \, dk
\]
Fourier Transforms

- Easy to differentiate:
  \[
  \frac{d^2 \phi(x)}{dx^2} = -4\pi^2 k^2 \int \Phi(k) e^{2\pi i k x} dk
  \]

- Then:
  \[
  -4\pi^2 k^2 \Phi(k) = F(k)
  \]
  - Easy to solve:
    \[
    \Phi(k) = -\frac{F(k)}{4\pi^2 k^2}
    \]

- Solve algebraically in Fourier space and then transform back
  - Only works for certain boundary conditions
FFTs assume periodic data, so we need a periodic problem.

- On [0,1], take:

\[ \phi(x, y) = \sin^2(2\pi x) \cos(4\pi y) + \sin(4\pi x) \cos^2(2\pi y) \]

- Then:

\[ \nabla^2 \phi(x, y) = 8\pi^2 \cos(4\pi y) [\cos(4\pi x) - \sin(4\pi x)] \\
-16\pi^2 \left[ \sin(4\pi x) \cos^2(2\pi y) + \sin^2(2\pi x) \cos(4\pi y) \right] \]
FFT Solution of Poisson Eq.

- First discretize in space, and then take the FFT of the discrete equation
  - If we use the 2nd order Laplacian, then we know that the result should converge as second order
  - Assume that $\Delta x = \Delta y$
    
    $\phi_{m+1,n} + \phi_{m-1,n} + \phi_{m,n+1} + \phi_{m,n-1} - 4\phi_{m,n} = \Delta x^2 f_{m,n}$
  - Substitute in the inverse transforms:

$$
\phi_{m,n} = \frac{1}{M N} \sum_{k_x=0}^{M-1} \sum_{k_y=0}^{N-1} \Phi_{k_x,k_y} e^{2\pi i m k_x / M } e^{2\pi i n k_y / N }
$$

$$
\phi_{m,n} = \frac{1}{M N} \sum_{k_x=0}^{M-1} \sum_{k_y=0}^{N-1} F_{k_x,k_y} e^{2\pi i m k_x / M } e^{2\pi i n k_y / N }
$$
FFT Solution of Poisson Eq.

- Group together like wavenumbers in the sums, cancel common factors:

\[
\Phi_{k_x,k_y} = \frac{\Delta x^2 F_{k_x,k_y}}{2 \left[ \cos \left( \frac{2\pi k_x}{M} \right) + \cos \left( \frac{2\pi k_y}{N} \right) - 2 \right]}
\]

- Compute this in Fourier space and then transform back

- Some caveats:
  - Some FFT routines return a \( k=0 \) wavenumber (the DC offset). You can run into a divide-by-zero problem. Leave this alone
  - The wavenumbers, \( k \), need to be physical (in the sense that the largest \( k \) should be \( N \), so it is fully one wavelength in the cosine)
FFT Solution of Poisson Eq.

- Solution (64x64 zones)
Poisson FFT Convergence

- Second-order convergence seen:
FFT Limitations (for Poisson Eqs.)

- General FFT works only for periodic BCs
  - Sine series for Dirichlet and Cosine series for Neumann, but mixed BCs can be a problem
- Non-constant coefficient equations are no longer simple

\[
\nabla \cdot \frac{1}{\rho(x)} \nabla \phi(x) = S(x)
\]

(This equation arises in some methods for variable-density incompressible flow)