

Scientific Computing for Astro Grads Minicourse

Assignment #3

Due Fri, 7 Mar

Question 1 - Multigrid: One Dimensional Thermal Diffusion

As we discussed in class, multigrid is a very efficient and highly parallelizable way of approaching many elliptic type problems; the key insight is that smoothing works very slowly because it's very local; by working at coarser and coarser levels, one can efficiently diffuse away even large-scale errors.

In `coarsegrid.sci` is a Scilab program for solving the steady-state heat equation with a source term; that is,

$$\nabla^2 u(x) + \sin(4x) = 0 \quad (1)$$

with periodic boundary conditions, so that we know the true solution is $u(x) = \sin(4x)/16$. There are two implementations of our finite difference method; a simple red-black smoothing scheme which iterates an arbitrary number of times, and a 'coarse grid' scheme, which executes `nr` red-black smoothing steps, then moves to a coarser mesh and does $2nr$ steps there, then updates the fine mesh and does another `nr` smoothings.

1 (a): Comparison: Red-Black vs Coarse Grid

Plot the error versus the number of iterations for both the red-black scheme and the two-level, or 'coarse-grid' scheme, for example by doing

```
getf("multigrid.sci");
[trb,xrb,allurb,errorrb] = runredblack(127,100);
[tcg,xcg,allucg,errorcg] = runcoarsegrid(127,100,3);
clf; plot2d(trb,errorrb); plot2d(tcg,errorcg);
```

How many smoothing steps does it take for the error to be reduced by a factor of two for the red-black scheme? A factor of 4? How about for the two-level scheme?

1 (b): Work

The 'times' returned by the schemes are really just the number of smoothing steps – but this isn't a fair comparison of the work done by the two schemes, as it is a factor of two faster to execute the smoothing step on the coarse mesh than on the fine one. Modify the two-level routine to also return this information, and plot the error from the two methods as a function of 'work done' in some units.

1 (c): V-cycle Multigrid

The 'coarse grid' approach works so well that one might hope that you could use it again. Indeed, this is the basic idea behind multigrid; on the coarse grid, one *also* uses the coarse grid solver, so that one relaxes `nr` times, then coarsens and solves for the error, and updates with a correction. This goes as coarse as possible (eg, to only one zone) and then updates propagate to finer meshes. Implement the V-cycle, using the coarse-grid method as a template. Then compare the error behaviour between the single level, two-level, and V-cycle solvers as a function of work done.

This is now a proper multigrid solver. One can go a step further and implement so-called 'Full Multigrid', but this is an approach which is typically only necessary if one lacks a good initial guess; for many

applications (eg, time-dependant simulations) one typically has a pretty good initial guess for the solution, or lack one perhaps only at the initial timestep.

Question 2 - Pseudo-spectral method; Schrödinger Wave Eqn

Purely linear elliptic problems are too easy to solve in Fourier space – when one transforms over the derivatives become multiplications, and the problem is done. However, most interesting problems include nonlinearities which must be dealt with in configuration space rather than k -space, so one must alternate back and forth to deal with the two sorts of terms. Here we look at another possibility, where there is a term in the equation with an explicit spatial dependence, so that again one must flip back and forth.

In `spectral.sci` we have some Scilab code for evolving the Schrödinger wave equation with initial conditions of a Gaussian wave packet with some momentum. Run the code like so:

```
getf("spectral.sci");
[t, x, allphi] = runspectral(128, 50, 1., +0);
clf; contour(t, x, abs(allphi), 12); xtitle("", "Time", "Position", "");
```

and one can see the wave packet moving upwards and spreading outwards, as it should. This is the case of evolving the wave equation

$$\begin{aligned} H\phi(x, t) &= i\hbar \frac{\partial}{\partial t} \phi(x, t) \\ H &= T + V = T + 0 \\ T &= \frac{p^2}{2m} = -\frac{\hbar^2}{2m} \nabla^2 \end{aligned}$$

where we have set the background potential to zero. In this case one can timestep along using

$$\phi(x, t) = e^{-iHt/\hbar} \phi(x, 0)$$

except that the exponential of the operator is horrendous to deal with in configuration space; but in k -space, it's pretty easy to deal with:

$$\hat{\phi}(k, t) = e^{i\hbar k^2 \Delta t / (2m)} \hat{\phi}(k, 0)$$

so we implement an algorithm of running an FFT, timestepping, and inverting the FFT to get back into configuration space so we can plot our wavefunction.

But you notice that there is a square-wave potential well defined in `spectral.sci`. Good eye. Your job is to implement the configuration-space part of this procedure so that we can put in potential wells and watch the wave packet scatter through the potential. As with the Strang splitting we talked about in the last lectures, we will implement this as

$$\hat{\phi}(k, t) = e^{-iV(r)t/(2\hbar)} e^{i\hbar k^2 \Delta t / (2m)} e^{-iV(r)t/(2\hbar)} \hat{\phi}(k, 0)$$

where the $V(r)$ terms must be implemented in configuration space. This splitting, as is often the case with Strang splitting, is accurate to $O(\Delta t^2)$.

Implement the potential, and make contour plots with square potential wells of different amplitudes (say, -1.0 , -0.5 , -0.25 , and $+0.25$). Comment on the plots.