Goals. This Iterative Methods. Multigrid – Final Project
G Söderlind, 5 February, 2016.

Goals. This is the final computer project. Working in groups of two, you hand in a project report for evaluation. The grade is pass/fail. If the project report does not receive the pass grade, you will have to complement the report until the project has been solved in a satisfactory way and the report receives a pass grade. Project deadline 26 February, 2016. You may submit the report either on paper or by email as a pdf file (the latter is preferable). Be sure to put your name and personal number on the report, so that your grade can be registered. Choose a file name that identifies your report with yourself.

The project goal is to solve the wave equation in 2D, using a geometric multigrid method. For simplicity, we will not go into a full treatment of boundary conditions, but only make sure that we have a multigrid solver that can be used on each time step, and then animate the solution in 2D.

Specific goals include:

1. Solving the wave equation in 2D on the unit square
2. Using an implicit time stepping method, where the local solution of elliptic problems is done using your previous MG iterative solver
3. Testing the code and verifying that it works correctly
4. Visualizing the results

The wave equation
The final problem is to use the multigrid programs you have developed so far to solve the 2D wave equation

\[ u_{tt} = c^2 \cdot \Delta u, \]

where \( c \) is the wave propagation speed and \( \Delta u = u_{xx} + u_{yy} \). For simplicity we are going to use \( c = 1 \) and solve this equation on the unit square \([0, 1] \times [0, 1]\). The boundary conditions are \( u(x, y, t) = 0 \) on the entire boundary, and for all \( t \geq 0 \).

Physically, the solution to this problem can be thought of as corresponding to vibrations of a drumhead stretched (very loosely, due to \( c = 1 \)) over the unit square. We are going to use an implicit time stepping method to solve this problem. Although that is not necessary, it implies that we have to solve a linear system of equations on each step; this system is an elliptic Helmholtz equation, and it will be solved by the multigrid method on every time step.

Space discretization. The Laplace operator

\[ \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \]

is discretized (assuming \( \Delta x^2 = \Delta y^2 \)) using the standard five-point formula,

\[ \Delta u = u_{xx} + u_{yy} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i+1,j} + u_{i-1,j} - 4u_{i,j} + u_{i+1,j+1} + u_{i+1,j-1}}{\Delta x^2}, \]
where $u^n_{i,j}$ is the solution in the point $(i,j)$ at time step $n$. We suggest using a $127 \times 127$ grid of interior points on the finest to start with, but make this a parameter in the program so that you can try out different resolutions later, as well as adapt the problem size to the available RAM on your computer.

**Time discretization.** The wave equation is second order in time. The derivative $u_{tt}$ can be discretized directly, but it is usually better to rewrite the equation as two coupled first order equations,

\[
\begin{align*}
  u_t &= v \\
  v_t &= u_{xx} + u_{yy}.
\end{align*}
\]

This could be discretized either by an explicit time stepping method, or by an implicit one. In both cases, a second order method is advisable to match the second order space discretization. Further, as the wave equation does not dissipate energy (there is no damping of the wave amplitude), it is preferable to choose the time stepping method so that this behavior is numerically replicated.

**Implicit time stepping.** To overcome the CFL condition, an implicit method would have to be used. A second order implicit method that preserves amplitude and energy is the implicit midpoint rule (equivalent in the linear case to the trapezoidal rule). This scheme is

\[
\begin{align*}
  u^{n+1} &= u^n + \Delta t \cdot (v^n + v^{n+1})/2 \\
  v^{n+1} &= v^n + \Delta t \cdot (u_{xx}^{n} + u_{yy}^{n} + u_{xx}^{n+1} + u_{yy}^{n+1})/2.
\end{align*}
\]

Let $T$ denote the matrix corresponding to the 5pt spatial discretization of the 2D-Laplacian. (This linear operator could e.g. be implemented as a 2D convolution.) The scheme then becomes

\[
\begin{align*}
  u^{n+1} &= u^n + \Delta t \cdot (v^n + v^{n+1})/2 \\
  v^{n+1} &= v^n + \Delta t \cdot (T u^n + T u^{n+1})/2,
\end{align*}
\]

where the boldface notation is used to indicate that $\mathbf{u}$ and $\mathbf{v}$ are vectors (in your programs possibly represented by choosing matrices as your data representation). Rearranging terms, we have

\[
\begin{align*}
  u^{n+1} - \Delta t v^{n+1}/2 &= u^n + \Delta t v^n/2 \\
  -\Delta t T u^{n+1}/2 + v^{n+1} &= v^n + \Delta t T u^n/2.
\end{align*}
\]

From these equations, we obtain

\[
\begin{align*}
  \left( I - \frac{\Delta t^2}{4} T \right) v^{n+1} &= \Delta t T u^n + \left( I + \frac{\Delta t^2}{4} T \right) v^n, \tag{2}
\end{align*}
\]

which can be solved for $v^{n+1}$. By letting

\[
\begin{align*}
  f(u, v) := \Delta t T u + \left( I + \frac{\Delta t^2}{4} T \right) v, \tag{3}
\end{align*}
\]

the entire computational scheme, derived by combining equations (1) and (2) becomes

\[
\begin{align*}
  &\text{Solve } \left( I - \frac{\Delta t^2}{4} T \right) v^{n+1} = f(u^n, v^n) \\
  &\text{Compute } u^{n+1} := u^n + \Delta t \cdot (v^n + v^{n+1})/2.
\end{align*}
\]
This advances the solution one step of size \( \Delta t \), from \((u^n, v^n)\) to \((u^{n+1}, v^{n+1})\). Thus we see that the entire time-stepping process is just a loop, where in each step we need to solve the special Helmholtz equation

\[ v - \gamma \Delta v = f \]

for various right-hand sides \( f \), and \( \gamma = \Delta t^2/4 \), by calling your multigrid solver. Note that no matter what time step \( \Delta t > 0 \) you choose, the system always remains elliptic (positive definite), so you will not encounter any problems solving the equations using the multigrid iteration. Work out how to change your existing solver to account for these modifications. Copy the pertinent files so that you don’t mess up your old solver.

In each time step, compute the right-hand side \( f \), and solve the Helmholtz type equation using your multigrid solver, to update the solution \( u^n \) and \( v^n \). Note that \( u \) represents “position” and \( v \) represents “velocity,” as \( \dot{u} = v \).

**Initial conditions.** We shall need initial conditions for both \( u \) and \( v \). While testing the code, it is a good idea to use the well-known eigenfunctions \( u(x, y) = \sin k \pi x \sin \ell \pi y \) as data. For the “position” \( u^0 \) use a suitable eigenfunction, to indicate that the surface shape at the initial time. For the “velocity” \( v^0 \), use a zero matrix. This implies that the initial values, as well as the solution, consist of two matrices on the computational grid.

Once your code runs properly, try out some more interesting initial conditions, e.g. in the form of Gaussian peaks \( u^0 = ae^{-b((x-x_0)^2+(y-y_0)^2)} \), \( v^0 = 0 \). You can use your \( X \) and \( Y \) matrices from the mesh to evaluate this function.

**Solving the wave equation.** Now you are ready to solve the wave equation. In order to get a smooth and nice-looking animation of the solution some Matlab tricks are needed. You may use the following template (copy-paste from the PDF). Insert code at the TODO-comments:

```matlab
function waves

% TODO : Set up a mesh of the square [0,1]x[0,1] using meshgrid.
%        Store the mesh in the matrices X and Y.
%
% TODO : Set up u0, v0, and initialize your multigrid
%        solver.
%
% Close all open figure windows.
close all

% Create a new figure window. This window will be located at the
% lower left corner of the screen and will be 800x600 pixels.
% Change here if the window doesn’t fit on the screen.
figure('Name','Waves','Position',[0 0 800 600]);

% Here we draw the initial conditions, the handle h is used to
% directly modify the plot properties. If you are curious about
% what can be changed, try "get(h)" and "set(h)".
% h = surf(X,Y,u0);
```
% We need this to control the color-coding.
set(h,'CDataMapping','direct');

% The main for-loop, it will go on for a while. You may abort at any time by pressing CTRL-C.
for main = 1:10000

    % TODO: Solve for the next time step using your multigrid solver. Store the next time step in the matrix u.

    % Controls the colors of the drawn mesh
    set(h,'CData',32*u+32)

    % Change the height data to be that of the new time step
    set(h,'ZData',u);

    % The axis normally wants to follow the data. Force it not to.
    axis([0 1 0 1 -1 1]);

    % Draw mesh
drawnow
end

**Slow animation?** Run the simulation. Is the animation slow? Remember that when you solved the Poisson or Helmholtz equation you had no reasonable initial guess, and because of this many V-cycles may have been needed to reduce the error. Alternatively, you had to use a special starting process to generate an initial guess. In this case you can however instead exploit the time dependency of the problem. Thus we do have a good initial guess, by using the solution *from the previous time step*. Experiment with the number of V-cycles you now need to produce a nice animation.