Writeup for the banded solver.

Solve the Poisson equation on a $M \times N$ grid using the five point Laplacian. Use the LAPack banded solver.

Input is an array and a function object. The array is the value of f at the grid nodes including the boundary, the function object is for the boundary condition.

```
DTMatlabDataFile inputFile("Input.mat",DTFile::ReadOnly);
// Read in the input variables.
DTMesh2D f;
Read(inputFile,"f",f);
DTFunction2D g;
Read(inputFile,"g",g);
```

The `DTMesh2D` object is a wrapper around an array and grid. To get the step size from the mesh and the underlying data array use the following functions.

```
DTMesh2DGrid grid = f.Grid();
double h = grid.dx();
DTDoubleArray fData = f.DoubleData();
```

The `DTFunction2D` is a class that wraps a binary expression tree. The I/O methods break this function into a collection of arrays which allows you to read and write functions. For testing you can download a sample input and output file from the sakai web site. You can compare the output that you get with the output that I generated. Your solution will not be exactly the same, but the difference should be small.

To test your code on a simpler input, you can create your own `g` function. You can create an analytical functions based based on the following simple sample code. Note that the `pinfo()` function can be called directly from the debugger to display the function.

```
DTFunction2D x = DTFunction2D::x();
DTFunction2D y = DTFunction2D::y();

DTFunction2D g = x*x + y;
g.pinfo();
```

To evaluate the function `g` just use the standard function notation e.g. `g(3.1,0.4)`. For example after you computed the solution for the interior, you need to set the values on the boundary based on the `g` function, and can do that as follows for the right hand side of the domain.

```
xzero = grid.Origin().x;
x = xzero + (m-1)*h;
for (j=0;j<n;j++) {
    y = yzero + j*h;
    toReturn(m-1,j) = g(x,y);
}
```

Implementation suggestions You need to use the LAPack banded solver to solve the problem, but don't need to save any intermediate solution. One way to do this is to go through the following steps

1. Set up the problem so that you have a symmetric positive definite matrix that you need to solve.
2. Since it is symmetric create the banded structure needed to hold the matrix. Don't create the the full matrix. We will test your code with input that will cause the full matrix to require too much memory and be considerate of the test machine.
3. Create the right hand side using both the `f` array as well as the `g` function at the edge points.
4. To make things easier to implement allocate the right hand side as a two dimensional array but hand it into the LAPack function call as a list.
5. Call the LAPack factorization followed by the solver to compute the solution at the interior.
6. Allocate a return array that has the same size as the input `f` array. Copy the solution you found into the interior of that array, and evaluate the boundary values using the `g` function. This is primarily to make it easier to compute errors and display the result graphically.

Output If the computed array is called `u` then save the output using

```
DTMatlabDataFile outputFile("Output.mat",DTFile::NewReadWrite);
DTMesh2D uMesh(grid,u);
Write(outputFile,"u",uMesh);
```

This saves the grid information back with the scalar field. What is saved is very similar to what you did earlier to save an array. To see exactly what is saved in that file, use the `pinfo()` function for the output file, i.e.

```
outputFile.pinfo();
```

either in the code or in the debugger.