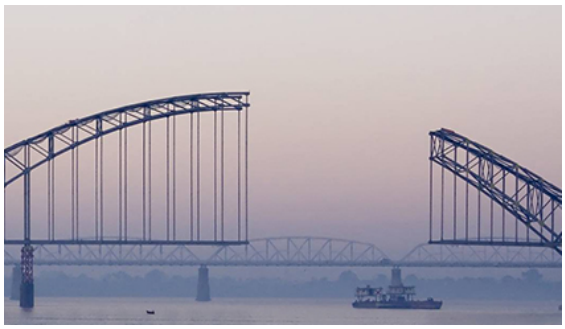
The steady 1D Poisson equation

MATH1091: ODE methods for a reaction diffusion equation

http://people.sc.fsu.edu/~jburkardt/classes/math1091_2020/poisson_1d_steady/poisson_1d_steady.pdf



Fill in the middle to connect the edges.

The Poisson Problem

Given a time-independent Poisson equation over an interval $[a, b]$, use finite differences to create a discrete model of the equation, set up the corresponding linear system, display the approximate solution and estimate its error.

1 A new kind of differential equation

Previously, we have used differential equations to simulate physical systems that evolve in time. An initial condition specified the starting value of some quantity, and the differential equation described how it changed over time. Such problems are known as *initial value problems* or **IVP's**.

Now we are going to look at differential equations that describe the shape or configuration of a quantity that varies over space, instead of time. To start with, our space will be a line segment, such as $a \leq x \leq b$, over which the quantity $u(x)$ is to be determined. The differential equation will involve the second derivative, and instead of an initial condition, we will have *boundary conditions* which describe the value of $u(x)$ or perhaps $u'(x)$ at the endpoints. Such a problem is known as a *boundary value problem* or **BVP**.

These models of physical systems arise because we are interested in predicting the response of a system to a force, which can often be described by a second order differential equation. The classic example of such a model is known as the *Poisson equation*. We will start with a simple case, over a 1D spatial domain, with no time variation. This example is the first step towards our goal of studying a more complicated system known as a *reaction diffusion equation*.

2 The Time-independent Poisson equation in 1D

Many laws in science and engineering can be expressed in terms of differential equations that are presumed to describe the variation in some quantity u over a spatial domain Ω. Usually, one or more additional equations are imposed along $\partial\Omega$, the boundary of the region.

Poisson's equation is one of the most useful ways of analyzing physical problems. Versions of this equation can be used to model heat, electric fields, gravity, and fluid pressure, in steady and time varying cases, and in 1, 2 or 3 spatial dimensions.

We will start by considering a physical system that does not vary in time, and extends over a finite one-dimensional interval $\Omega = [a, b]$. In that case, the Poisson equation can be written as

$$-u''(x) = f(x) \quad \text{for all } x \in \Omega$$

Where $f(x)$ is known as a *forcing function* or *source term*. The differential equation can be viewed as a model of how the quantity u responds to this force. To complete the specification of the problem, boundary conditions are imposed at the end points $x = a$ and $x = b$. Possible conditions include:

$$\begin{aligned} \textit{Dirichlet:} \quad & u(x) = \text{ value} \\ \textit{Neumann:} \quad & \frac{\partial u}{\partial x} = \text{ value} \\ \textit{Robin:} \quad & \alpha * u(x) + \beta * \frac{\partial u}{\partial x} = \text{ value} \quad \text{for some } \alpha, \beta \end{aligned}$$

There is no general technique to compute the exact mathematical solution of all Poisson equations. If we want to use computational techniques, we need a way to transform the mathematical problem. We can replace the interval by a sequence of points, the derivatives by finite difference approximations, and create a set of linear equations to be solved. The result will be the estimated value of $u(x)$ at a discrete set of points in the interval. To determine whether this method gives a good approximation, we can consider some special cases where the exact answer is known, and compare this to our computed approximations.

3 Discretizing the Poisson equation

For convenience, we will assume that there is a formula $g(x)$ for the exact solution of our problem. We designate the unknown solution as $u(x)$, so we are saying that secretly, we actually know a formula so that $u(x) = g(x)$. This formula $g(x)$ is strictly for experimental purposes. It makes it easier to describe boundary conditions, and to compare our computed solution to the exact solution. In real life problems, we won't have the luxury of knowing a formula for the exact solution!

Our domain Ω is the interior of the interval $[a, b]$. We assume that the Poisson equation holds in the interior, with right hand side function $f(x)$. We will assume Dirichlet conditions at the endpoints, and we will be allowed to use our secret exact formula to define these conditions:

$$\begin{aligned} u(a) &= g(a) \\ u(b) &= g(b) \end{aligned}$$

We begin by discretizing the geometry. That is, we will create a grid of n points, or *nodes*, with spacing $h = \frac{b-a}{n-1}$. Since we are using equal spacing, a typical discrete point x_i is:

$$x_i = a + (i - 1) * h \quad \text{for } 1 \leq i \leq n$$

and the whole set of nodes could be generated in MATLAB by

```
1   x = linspace ( a, b, n );
```

We will seek an approximate solution only at this discrete set of points. Thus, our discrete solution will be a vector of n values, not a function. We can use the mathematical notation u_i to indicate the value of our discrete solution at x_i, hoping that $u_i \approx u(x_i)$.

Now we discretize the differential equation and boundary conditions, replacing them by a set of n linear equations. Our first and last equations are associated with the boundary points x_1 and x_n:

$$\begin{aligned} u_1 &= g(x_1) \\ u_n &= g(x_n) \end{aligned}$$

Equations 2 through $n-1$ represent discretized versions of the differential equation. At node i, we approximate the second derivative u'' by the second difference approximation:

$$-u''(x_i) \approx \frac{-u(x_{i-1}) + 2u(x_i) - u(x_{i+1})}{h^2}$$

Our equations 2 through $n-1$ become:

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} = f(x_i) \quad \text{for } 1 < i < n$$

For each discrete point x_i, we have an explicit formula (at boundary points) or an implicit linear equation (at interior points), involving our unknown approximate values u_i. We can think of this combination of conditions on boundary and interior points as a linear system of the form:

$$A * u = rhs$$

It now remains to organize the process of defining the matrix A and right hand side rhs. Once this is done, the solution process is a relatively straightforward linear algebra task.

4 Exercise #1: Write a discretized Poisson equation

Consider the equation $-u'' = f(x)$ over the interval $-2 \leq x \leq 2$ with right hand side $f(x) = -6x$ and exact solution $g(x) = x^3$. The problem is discretized using 5 evenly spaced nodes x_i.

Write down the linear equations of this discretized model.

Check your work against *exercise1.txt*

5 Assembling the linear system

We have n equations to construct. We will associate each equation with a node, in a natural way, starting with the leftmost.

Equation 1 at node 1 is the first boundary condition:

$$u_1 = g(x_1)$$

Equations 2 through $n-1$, associated with the corresponding nodes, are each a discretized Poisson equation:

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} = f(x_i)$$

Equation n at node n is the final boundary condition:

$$u_n = g(x_n)$$

It is helpful to see the pattern that this system of equations forms when written in matrix form. If we suppose $n = 5$, then the equations have the form $A * u = rhs$:

```
    <-------------A------------->     <--rhs-->
    1       2       3       4       5

1   1       0       0       0       0         g(x1)
2  -1/h^2   2/h^2  -1/h^2   0       0         f(x2)
3   0      -1/h^2   2/h^2  -1/h^2   0         f(x3)
4   0       0      -1/h^2   2/h^2  -1/h^2     f(x4)
5   0       0       0       0       1         g(x5)
```

Now that we have a system of linear equations, we can solve for the values u_i in MATLAB by writing

```
u = A \ rhs
```

Listing 1: Solving a discretized Poisson equation.

6 Exercise #2: Solve a discretized Poisson equation

Consider again the equation $-u'' = f(x)$ over the interval $-2 \leq x \leq 2$ with $f(x) = -6x$ and $g(x) = x^3$. The problem is discretized using 5 evenly spaced nodes x_i. Previously, you wrote down the linear equations of the discretized model.

Write a MATLAB code that sets up and solves the linear equations for the discrete solution values u_i.

We want to compare our discrete solution to the exact solution stored in $g(x)$.

To plot u_i, define the 5 discrete nodes x_i using `linspace()`. To plot the exact solution, define 101 nodes $x2$ in $-2 \leq x \leq 2$, and define $u2 = g(x2)$.

Show (x_i, u_i) and $(x2, u2)$ together on one plot.

Check your work against *exercise2.m* and *exercise2.png*.

7 The MATLAB code `poisson_solve.m`

The following program accepts input from the user which defines the discretization of an interval, and the right hand side function $f(x)$ and exact solution $g(x)$, and returns vectors x and u containing the coordinate and solution values at the grid points.

It is available on the web page as *poisson_solve.m*:

```
function [ x, u ] = poisson_solve ( n, a, b, f, g )

  h = ( b - a ) / ( n - 1 );

  x = linspace ( a, b, n );

  A = zeros ( n, n );
  rhs = zeros ( n, 1 );

  i = 1;
  A(i,i) = 1.0;
  rhs(i) = g ( x(i) );

  for i = 2 : n - 1
    A(i,i-1) = - 1.0 / h^2;
    A(i,i)   =   2.0 / h^2;
    A(i,i+1) = - 1.0 / h^2;
    rhs(i)   = f(x(i));
  end

  i = n;
  A(i,i) = 1.0;
  rhs(i) = g ( x(i) );

  u = A \ rhs;

  return
end
```

Listing 2: poisson_solve.m

It is up to the user to set the input quantities that define a particular Poisson problem, and to plot the approximation after it is computed and returned.

8 Exercise #3: Use poisson_solve() for a specific problem

Let **"arnold"** be the name of the following Poisson problem:

$$-u''(x) = -(2x+5)\,e^x$$

over the interval $0 \leq x \leq 5.0$. Suppose we know that our exact solution is

$$g(x) = (2x+1)\,e^x$$

so that our Dirichlet conditions are

$$u(0.0) = g(0.0) = 1$$
$$u(5.0) = g(5.0) = 11\,e^5 \approx 1632.5$$

Write a code `arnold.m` which sets up the arnold problem, calls `poisson_solve()` for the solution, and plots the computed and exact solutions together.

Check your work against *exercise3.m* and *exercise3.png*.

9 Exercise #4: Plotting

How do we report the results of `poisson_solve()`? Depending on the size of the problem, and what we are studying, we might want to see

- a table of x_i and u_i;
- some numerical measurement of the approximation error;
- a plot of the computed solution, and the exact solution, if known;
- a table or plot of the approximation error for several different mesh spacings h;

These kinds of reports are known as *postprocessing* the data.

A plot of the approximate solution is always informative, especially if the exact solution is also known. Here is how you might have done plotting for your previous exercise:

```
1  [ x, u ] = poisson_solve ( n, a, b,  @f, @g );
2  x2 = linspace ( xmin, xmax, 101 );
3  u2 = g(x2);
4  plot ( x, u, x2, u2 );
```

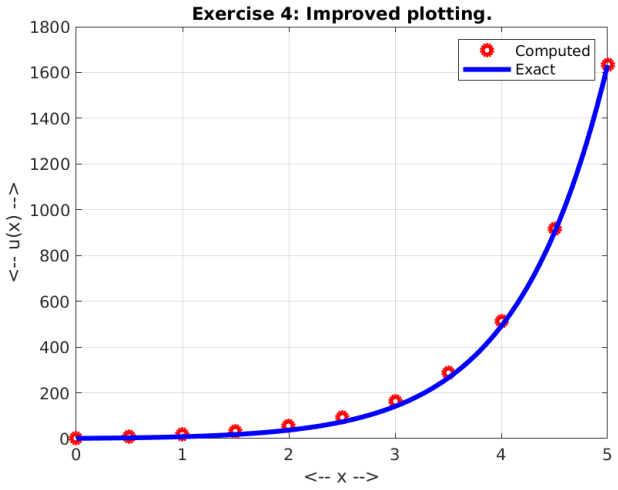
Listing 3: Plotting the approximation.

Although these commands do show the computed and exact solutions, the plot is pretty skimpy. A better plot might include:

- grid lines;
- labels for the x and y axes;
- a title;
- the use of markers, instead of a connected line, for the computed data;

- lines drawn with more than the default thickness;
- lines drawn with appropriate color;
- a legend that links each line to its line style;

Each of these improvements can be done within MATLAB.



What MATLAB commands made this plot?

Try to improve your plotting commands so that you reproduce a plot.

Compare your work to `exercise4.m`.

10 Measuring error

When you are trying to write a program to approximate the solution of a BVP, you can expect that each approximation has some error. What you are hoping is that, if you intensify the approximation process, the error will go down, and can be driven as close to zero as you can afford.

Thus, we need a way to summarize approximation error. This will help us to spot cases where our code is actually wrong. It will help us to verify whether the error decreases as we increase n (or decrease the spacing h). It will help us to estimate *how fast* the error decreases as we decrease h. And it will let us compare the usefulness of different solvers or approximation techniques.

For our purposes, the error vector is the set of n numbers $u_i - g(x_i)$. We want to summarize this into a single number. The MATLAB `norm()` function is not quite the right measurement, because it does not make a fair comparison between errors at different values of n. Instead, we will use the closely related, but correctly scaled, root-mean-square (RMS) error norm, which in MATLAB is evaluated by `err = rms(u-g(x))`.

The most common way to use this tool is to solve the same problem repeatedly, using a mesh that is twice as fine each time. Remember the peculiar releationship between n and h: $h = \frac{1}{n-1}$. Thus, to get a nice spacing h, we typically add 1 to n. A spacing of $h = \frac{1}{10}$ corresponds to $n = 11$. Halving the spacing to $h = \frac{1}{20}$ means not-quite-doubling n to 21. Keep this minor detail in mind when doing such an investigation.

11 Exercise #5: How does mesh size h affect error norm?

In the `exercise4()` program, you specified a value for n. Now we want to write a new program that calls a version of `exercise4()` for several different values of n. In particular, for the sequence of values $n = 11, 21, 41, 81, 161, 321, 641$, we want to compute the RMS error err, and make some kind of report of these values.

How do we create a program `exercise5.m` that does this?

Step 1: Modify `exercise4()` so that it accepts the value of n as input, and returns a quantity `err` as output:

```
1 function err = exercise4 ( n )
```

Listing 4: New calling sequence

Step 2: Remove the plotting statements from `exercise4()`. Insert a line that computes the RMS error:

```
1   err = rms ( u' - g(x) );
```

Listing 5: Compute the RMS error norm.

Note the apostrophe that transposes `u` above! That's so that both `u'` and `g(x)` are row vectors, so that we can combine them. Step 3: Create the file *exercise5.m*, and have it call `exercise4(n)` repeatedly, supplying the value of n.

```
1 for n = [ 11, 21, 41, 81, 161, 321, 641]
2   err = exercise4 ( n );
3 end
```

Listing 6: exercise5 requests many error norms

Step 4: we want to plot our error norm data. We can do this in *exercise5.m* by creating an empty vector, and then appending each result

```
1 eplot = [];
2 for n = [ 11, 21, 41, 81, 161, 321, 641]
3   err = exercise4 ( n );
4   eplot = [ eplot, err ];
5 end
```

Listing 7: exercise5 saves the RMS error norm in a vector.

Step 5: Once all the errors have been computed, `exercise5()` can make a log-log plot.

```
1 plot ( log ( n - 1 ), log ( err ) );
```

Listing 8: exercise5 makes a log-log plot

Follow these steps to create your program, and generate the error plot.

Compare with the files *exercise5.m* and *exercise5.png*.

12 A more complicated equation

We have seen a very simple version of the Poisson equation. In some cases, the problem being studied will involve not just the second derivative of u, but also contributions from the solution u or its first derivative u'. Such contributions can usually be easily included in the formulation.

Consider a problem, defined over $0 \leq x \leq \frac{1}{2}$, for which the exact solution is $g(x) = 2xe^x$, with right hand side function $f(x) = -4e^x$. This BVP involves not just the second derivative u'' but the solution value u as well:

$$\begin{aligned} -u'' + u &= f(x) \\ u(0) &= g(0) \\ u(0.5) &= g(0.5) \end{aligned}$$

Writing the discretized equation at node i, we have

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} + u_i = f(x_i)$$

This means that the left hand side of the linear equation can be broken into three parts:

$$\frac{-u_{i-1}}{h^2} \qquad + \frac{2u_i}{h^2} + u_i \qquad + \frac{-u_{i+1}}{h^2}$$

so that the nonzero entries in row i of the matrix A are now

```
    <-------------A------------->     <--rhs-->
    i-1           i      i+1

i  -1/h^2   2/h^2 + 1  -1/h^2         f(x_i)
```

In other words, for equations 2 through $n-1$, we have to add an extra term of 1 to the $A(i,i)$ entry.

13 Homework: Solve the equation

1. Create *poisson2_solve.m* as a copy of *poisson_solve.m*.
2. Modify *poisson2_solve.m* by adding 1 to the diagonal matrix elements of rows 2 through $n-1$.
3. Create *hw1.m*, which might start as a copy of *exercise3.m*.
4. Set $n = 21$;
5. Revise the `g()` and `f()` functions.
6. Revise the location of the boundary points.
7. In `hw1`, call `poisson2_solve()` instead of `poisson2_solve()`..
8. Plot the computed solution u and exact solution $g(x)$ together.
9. Run `hw1()` and save the plot as a PNG file, `hw1.png`.

Send your plot *hw1.png* to me at **jvb25@pitt.edu**. I would like to see your work by Friday, May 8.