Lecture 13

Nonlinear Systems - Newton's Method

An Example

The LORAN (LOng RAnge Navigation) system calculates the position of a boat at sea using signals from fixed transmitters. From the time differences of the incoming signals, the boat obtains differences of distances to the transmitters. This leads to two equations each representing hyperbolas defined by the differences of distance of two points (foci). An example of such equations from [2] are:

$$\begin{aligned} \frac{x^2}{186^2} - \frac{y^2}{300^2 - 186^2} &= 1 \quad \text{and} \\ \frac{(y-500)^2}{279^2} - \frac{(x-300)^2}{500^2 - 279^2} &= 1\,. \end{aligned} \tag{13.1}$$

Solving two quadratic equations with two unknowns, would require solving a 4 degree polynomial equation. We could do this by hand, but for a navigational system to work well, it must do the calculations automatically and numerically. We note that the Global Positioning System (GPS) works on similar principles and must do similar computations.

Vector Notation

In general, we can usually find solutions to a system of equations when the number of unknowns matches the number of equations. Thus, we wish to find solutions to systems that have the form:

$$\begin{aligned} f_1(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ f_3(x_1, x_2, x_3, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, x_3, \ldots, x_n) &= 0. \end{aligned} \tag{13.2}$$

For convenience we can think of $(x_1, x_2, x_3, \ldots, x_n)$ as a vector $\mathbf{x}$ and $(f_1, f_2, \ldots, f_n)$ as a vector-valued function $\mathbf{f}$. With this notation, we can write the system of equations (13.2) simply as:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0},$$

i.e. we wish to find a vector that makes the vector function equal to the zero vector.

As in Newton's method for one variable, we need to start with an initial guess $\mathbf{x}_0$. In theory, the more variables one has, the harder it is to find a good initial guess. In practice, this must be overcome by

using physically reasonable assumptions about the possible values of a solution, i.e. take advantage of engineering knowledge of the problem. Once $\mathbf{x}_0$ is chosen, let

$$\Delta\mathbf{x} = \mathbf{x}_1 - \mathbf{x}_0.$$

Linear Approximation for Vector Functions

In the single variable case, Newton's method was derived by considering the linear approximation of the function f at the initial guess $\mathbf{x}_0$. From Calculus, the following is the linear approximation of $\mathbf{f}$ at $\mathbf{x}_0$, for vectors and vector-valued functions:

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_0) + D\mathbf{f}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0).$$

Here $D\mathbf{f}(\mathbf{x}_0)$ is an $n \times n$ matrix whose entries are the various partial derivative of the components of $\mathbf{f}$. Specifically:

$$D\mathbf{f}(\mathbf{x}_0) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}_0) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}_0) & \frac{\partial f_1}{\partial x_3}(\mathbf{x}_0) & \dots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}_0) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}_0) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}_0) & \frac{\partial f_2}{\partial x_3}(\mathbf{x}_0) & \dots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}_0) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}_0) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}_0) & \frac{\partial f_n}{\partial x_3}(\mathbf{x}_0) & \dots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}_0) \end{pmatrix}. \tag{13.3}$$

Newton's Method

We wish to find $\mathbf{x}$ that makes $\mathbf{f}$ equal to the zero vectors, so let's choose $\mathbf{x}_1$ so that

$$\mathbf{f}(\mathbf{x}_0) + D\mathbf{f}(\mathbf{x}_0)(\mathbf{x}_1 - \mathbf{x}_0) = \mathbf{0}.$$

Since $D\mathbf{f}(\mathbf{x}_0)$ is a square matrix, we can solve this equation by

$$\mathbf{x}_1 = \mathbf{x}_0 - (D\mathbf{f}(\mathbf{x}_0))^{-1}\mathbf{f}(\mathbf{x}_0),$$

provided that the inverse exists. The formula is the vector equivalent of the Newton's method formula we learned before. However, in practice we never use the inverse of a matrix for computations, so we cannot use this formula directly. Rather, we can do the following. First solve the equation

$$D\mathbf{f}(\mathbf{x}_0)\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}_0). \tag{13.4}$$

Since $D\mathbf{f}(\mathbf{x}_0)$ is a known matrix and $-\mathbf{f}(\mathbf{x}_0)$ is a known vector, this equation is just a system of linear equations, which can be solved efficiently and accurately. Once we have the solution vector $\Delta\mathbf{x}$, we can obtain our improved estimate $\mathbf{x}_1$ by:

$$\mathbf{x}_1 = \mathbf{x}_0 + \Delta\mathbf{x}.$$

For subsequent steps, we have the following process:

- Solve $D\mathbf{f}(\mathbf{x}_i)\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}_i)$ for $\Delta\mathbf{x}$.
- Let $\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta\mathbf{x}$

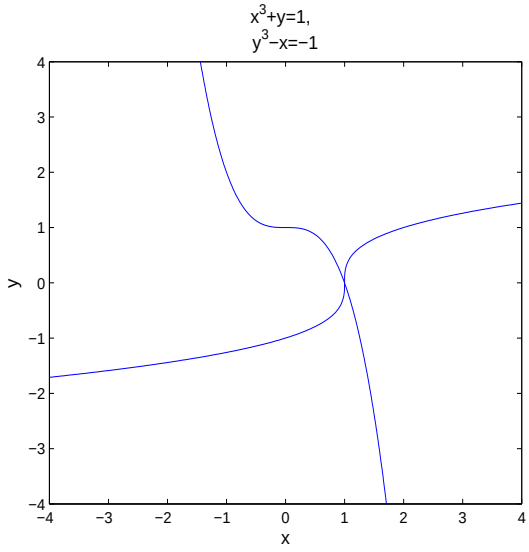

Figure 13.1: Graphs of the equations $x^3 + y = 1$ and $y^3 - x = -1$. There is one and only one intersection; at $(x, y) = (1, 0)$.

An Experiment

We will solve the following set of equations:

$$\begin{aligned} x^3 + y &= 1 \\ y^3 - x &= -1. \end{aligned} \tag{13.5}$$

You can easily check that $(x, y) = (1, 0)$ is a solution of this system. By graphing both of the equations you can also see that $(1, 0)$ is the only solution (Figure 13.1).

We can put these equations into vector form by letting $x_1 = x$, $x_2 = y$ and

$$\begin{aligned} f_1(x_1, x_2) &= x_1^3 + x_2 - 1 \\ f_2(x_1, x_2) &= x_2^3 - x_1 + 1. \end{aligned} \tag{13.6}$$

or

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} x_1^3 + x_2 - 1 \\ x_2^3 - x_1 + 1 \end{pmatrix}. \tag{13.7}$$

Now that we have the equation in the vector form, write the following script program:

```
format long
f = inline('[ x(1)^3+x(2)-1 ; x(2)^3-x(1)+1 ]');
x = [.5;.5]
x = fsolve(f,x)
```

Save this program as `myfsolve.m` and run it. You will see that the internal MATLAB solving command `fsolve` approximates the solution, but only to about 7 decimal places. While that would be close enough for most applications, one would expect that we could do better on such a simple problem.

Next we will implement Newton's method for this problem. Modify your `myfsolve` program to:

```
% mymultnewton
format long
n=8  % set some number of iterations, may need adjusting
f = inline('[x(1)^3+x(2)-1 ; x(2)^3-x(1)+1]'); % the vector function
Df = inline('[3*x(1)^2,  1 ; -1,  3*x(2)^2]'); % the matrix of partial derivatives
x = [.5;.5]             % starting guess
for i = 1:n
      Dx = -Df(x)\f(x); % solve for increment
      x = x + Dx        % add on to get new guess
      f(x)              % see if f(x) is really zero
end
```

Save and run this program (as `mymultnewton`) and you will see that it finds the root exactly (to machine precision) in only 6 iterations. Why is this simple program able to do better than MATLAB's built-in program?

Exercises

13.1 (a) Put the LORAN equations (13.1) into the function form (13.2).

(b) Construct the matrix of partial derivatives $D\mathbf{f}$ in (13.3).

(c) Adapt the `mymultnewton` program to find a solution for these equations. By trying different starting vectors, find at least three different solutions. (There are actually four solutions.) Think of at least one way that the navigational system could determine which solution is correct.