Lecture 5: Systems of nonlinear equations

COMP5930M Scientific Computation

Today

Notation

- Single equation in a single unknown

$$F(x) = 0$$

- n equations in n unknowns

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}$$

$\mathbf{x}$ is a vector $\{x_1, x_2, ..., x_n\}$ of n unknown values
$\mathbf{F}$ is a set $\{F_1(\mathbf{x}), F_2(\mathbf{x}), ...F_n(\mathbf{x})\}$ of n nonlinear equations

The nonlinear problem

Find the n-dimensional point $\{x_j^*\}$, $j = 1, 2, ...n$
such that the set of functions

$F_i(x_j^*) = 0$, $i = 1, 2, ...n$ simultaneously

Example: $n = 2$ system

Find (x_1^*, x_2^*) such that $F_1(x_1^*, x_2^*) = 0$ and $F_2(x_1^*, x_2^*) = 0$ simultaneously

$$
\begin{aligned}
F_1(x_1, x_2) &= 3\,x_1^2 + 4\,x_1 x_2 \\
F_2(x_1, x_2) &= x_1^2 + 2\,x_1 x_2^2
\end{aligned}
$$

Example: $n = 2$ system

$$
\begin{aligned}
\frac{\partial F_1}{\partial x_1} &= 6\,x_1 + 4\,x_2, \qquad \frac{\partial F_1}{\partial x_2} = 4\,x_1 \\
\frac{\partial F_2}{\partial x_1} &= 2x_1 + 2\,x_2^2, \qquad \frac{\partial F_2}{\partial x_2} = 4\,x_1 x_2
\end{aligned}
$$

Newton's Method

- Assume we have access to the set of functions $\mathbf{F}(\mathbf{x})$
- Assume we know an initial state $\mathbf{x}_0$

These first two are the minimum information necessary

- Assume we have access to all the partial derivatives of $\mathbf{F}$ with respect to $\mathbf{x}$

$$\frac{\partial F_i}{\partial x_j}, \quad i = 1, 2, ..., n, \quad j = 1, 2, ..., n$$

Derivation of Newton's method in the vectorial case

Given current iterate $\mathbf{x}_k$, find an increment $\delta \in \mathbb{R}^n$ s.t.

$$\mathbf{F}(\mathbf{x}_k + \delta) = 0. \tag{1}$$

If the function $\mathbf{F}$ is differentiable at $\mathbf{x}_k$, we can linearise it:

$$\mathbf{F}(\mathbf{x}_k + \delta) = \mathbf{F}(\mathbf{x}_k) + \frac{\partial \mathbf{F}}{\partial \mathbf{x}}(\mathbf{x}_k)\delta + o(|\delta|^2).$$

Derivation of Newton's method in the vectorial case

Given current iterate $\mathbf{x}_k$, find an increment $\delta \in \mathbb{R}^n$ s.t.

$$\mathbf{F}(\mathbf{x}_k + \delta) = 0. \tag{1}$$

If the function $\mathbf{F}$ is differentiable at $\mathbf{x}_k$, we can linearise it:

$$\mathbf{F}(\mathbf{x}_k + \delta) = \mathbf{F}(\mathbf{x}_k) + \frac{\partial \mathbf{F}}{\partial \mathbf{x}}(\mathbf{x}_k)\delta + o(|\delta|^2).$$

Dropping the second order terms, we can solve for δ from (1):

$$\mathbf{x}_{k+1} - \mathbf{x}_k = \delta = -\left[\frac{\partial \mathbf{F}}{\partial \mathbf{x}}(\mathbf{x}_k)\right]^{-1} \mathbf{F}(\mathbf{x}_k).$$

Derivation of Newton's method in the vectorial case

Given current iterate $\mathbf{x}_k$, find an increment $\delta \in \mathbb{R}^n$ s.t.

$$\mathbf{F}(\mathbf{x}_k + \delta) = 0. \tag{1}$$

If the function $\mathbf{F}$ is differentiable at $\mathbf{x}_k$, we can linearise it:

$$\mathbf{F}(\mathbf{x}_k + \delta) = \mathbf{F}(\mathbf{x}_k) + \frac{\partial \mathbf{F}}{\partial \mathbf{x}}(\mathbf{x}_k)\delta + o(|\delta|^2).$$

Dropping the second order terms, we can solve for δ from (1):

$$\mathbf{x}_{k+1} - \mathbf{x}_k = \delta = -\left[\frac{\partial \mathbf{F}}{\partial \mathbf{x}}(\mathbf{x}_k)\right]^{-1} \mathbf{F}(\mathbf{x}_k).$$

The algorithm

Generate a sequence of approximations $\mathbf{x}_1$, $\mathbf{x}_2$, ... using

$$\mathbf{x}_{k+1} \;=\; \mathbf{x}_k - \left(\frac{\partial \mathbf{F}}{\partial \mathbf{x}}(\mathbf{x}_k)\right)^{-1} \mathbf{F}(\mathbf{x}_k)$$

starting from an initial guess $\mathbf{x}_0$.

The Jacobian Matrix

The $n \times n$ matrix of partial derivatives is called the Jacobian, $\mathbf{J}$, where $J_{ij} = \frac{\partial F_i}{\partial x_j}$

Newton's Method is more compactly written as

$$\mathbf{x}_{k+1} \; = \; \mathbf{x}_k - \mathbf{J}(\mathbf{x}_k)^{-1}\mathbf{F}(\mathbf{x}_k)$$

This implies the solution of an $n \times n$ linear system at every step

This is a costly part of the algorithm

Rewrite as 2-step algorithm

$$\begin{aligned}\mathbf{J}(\mathbf{x}_k)\delta &= -\mathbf{F}(\mathbf{x}_k) \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \delta\end{aligned}$$

Notes:

- Step 2 is trivial in this basic form
- Step 1 is a linear algebra problem: system of $n \times n$ linear equations with J a known $n \times n$ matrix at each iteration

Computational cost

	Function calls	Algorithmic
Evaluate $\mathbf{J}$	?	n^2
Solve $\mathbf{J}\delta = -\mathbf{F}$	1	n^3
Update $\mathbf{x}_{k+1}$	0	n

Evaluating the Jacobian in practice

- Analytical Jacobian may be expensive to evaluate
- Numerical approximation of Jacobian requires $2n^2$ evaluations
- Quasi-Newton methods rely on approximation of Jacobian that is updated at each step

Evaluating the Jacobian numerically

- We require n^2 values to fill the Jacobian matrix
- Numerical approximation term-by-term

$$\frac{\partial F_i}{\partial x_j} \approx \frac{F_i(x_1, \ldots, x_j + \delta_j, \ldots, x_n) - F_i(x_1, \ldots, x_n)}{\delta_j}$$

- Efficient implementation
  - Can simultaneously perturb all the F_i with respect to x_j
  - Equivalent to n Matlab function calls
    – where a function call evaluates all the F_i

Jacobian for the $n = 2$ system

```
% Vectorised version of the function F(x1,x2)
F=@(x1,x2)([3*x1.^2 + 4.*x1.*x2;
           x1.^2 + 2*x1.*x2.^2]);

% Optimal choice of perturbation parameter h
h  = 10 * sqrt(eps);

% Call F(x1,x2) once and store
Fx = F(x1,x2);

% Numerical Jacobian based on difference approximation
dFnum = [ (F(x1+h,x2) - Fx) / h ...
          (F(x1,x2+h) - Fx) / h ];
```

Computational cost

	Function calls	Algorithmic
Evaluate $\mathbf{J}$	n	n^2
Solve $\mathbf{J}\delta = -\mathbf{F}$	1	n^3
Update $\mathbf{x}_{k+1}$	0	n

Problems?

- More difficult to make the algorithm robust overall
  - There is no bisection method in higher dimensions
- For this reason damped Newton-like methods are preferred
  - Take steps in the direction $\frac{\delta}{||\delta||}$ but control the step size to avoid divergence when the Jacobian $\partial \mathbf{F}/\partial \mathbf{x} \approx \mathbf{0}$.
  - In some circumstances this allows global convergence of Newton-type algorithms

Next time...

Tutorial

- Solving nonlinear systems with MATLAB
- Problems with convergence of non-damped Newton's method

Lecture

- Line-search algorithms for Newton's method
- Computational algorithms for systems