ACM40290: Numerical Algorithms

Unconstrained Minimisation

Dr Barry Wardell
School of Mathematics and Statistics
University College Dublin

Unconstrained Minimisation

seek an $\mathbf{x}^*$ such that

$$g(\mathbf{x}^*) = \min_{\mathbf{x}\in\mathbb{R}^n} g(\mathbf{x}) \ .$$

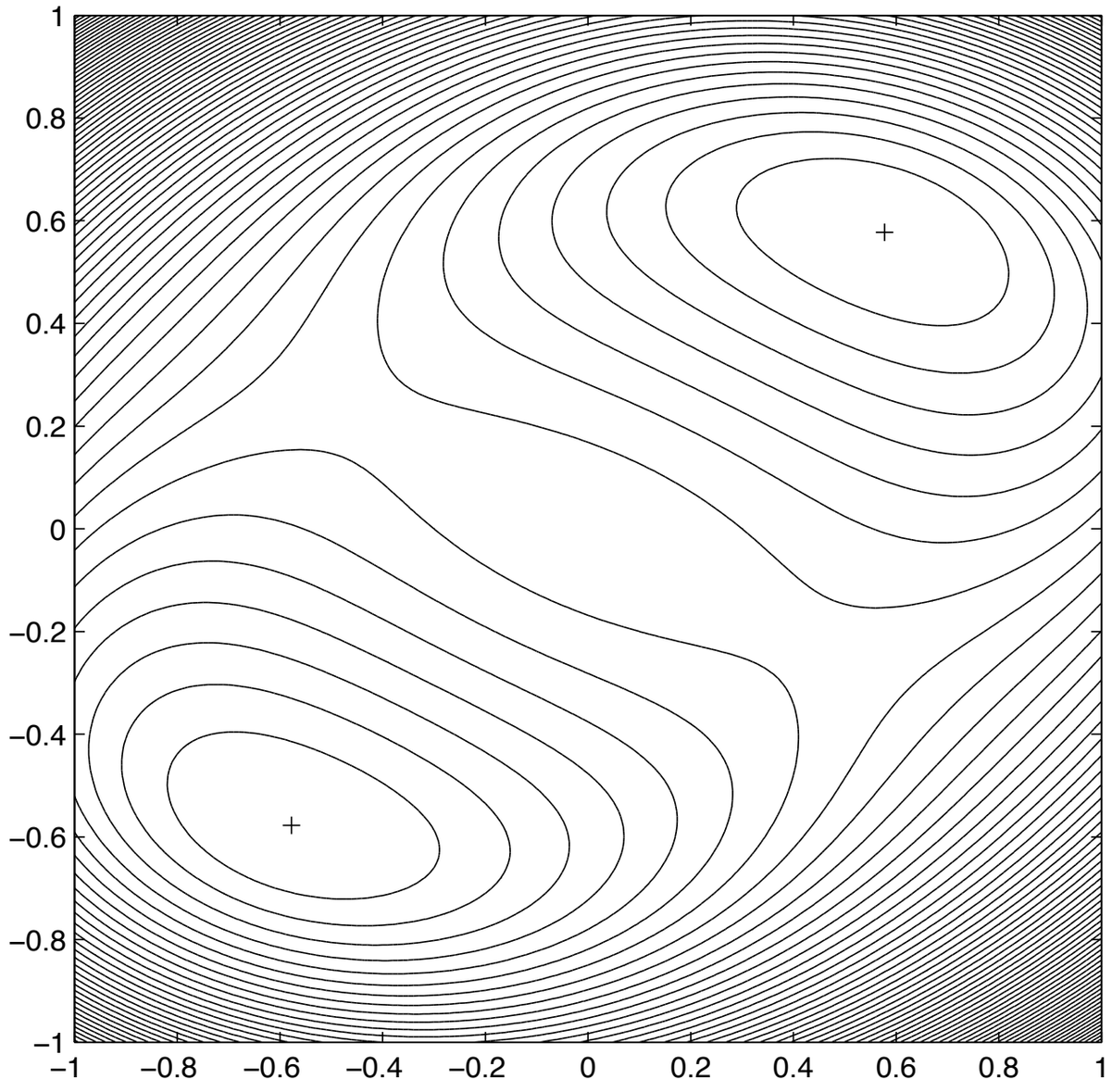
Most effective software only find local minima, i.e. they find solutions $\mathbf{x}^*$ to the equations

$$\nabla g(\mathbf{x}) = \begin{bmatrix} \partial g(\mathbf{x})/\partial x_1 \\ \partial g(\mathbf{x})/\partial x_2 \\ \vdots \\ \partial g(\mathbf{x})/\partial x_n \end{bmatrix} = 0$$

ensuring that $g(\mathbf{x}) \geq g(\mathbf{x}^*)$ for $\mathbf{x}$ "near" $\mathbf{x}^*$.

The problem of finding a **global minimum** x^* so that $g(x) \geq g(x^*)$ for **all** x is much harder and no good generally available algorithms exist to do it.

A typical function $g(x)$ is given below.



Two local minima

The Method of Steepest Descent

In the method of steepest descent, a series of steps is chosen, with each step taken in the direction $-\nabla g$. The iteration proceeds as follows

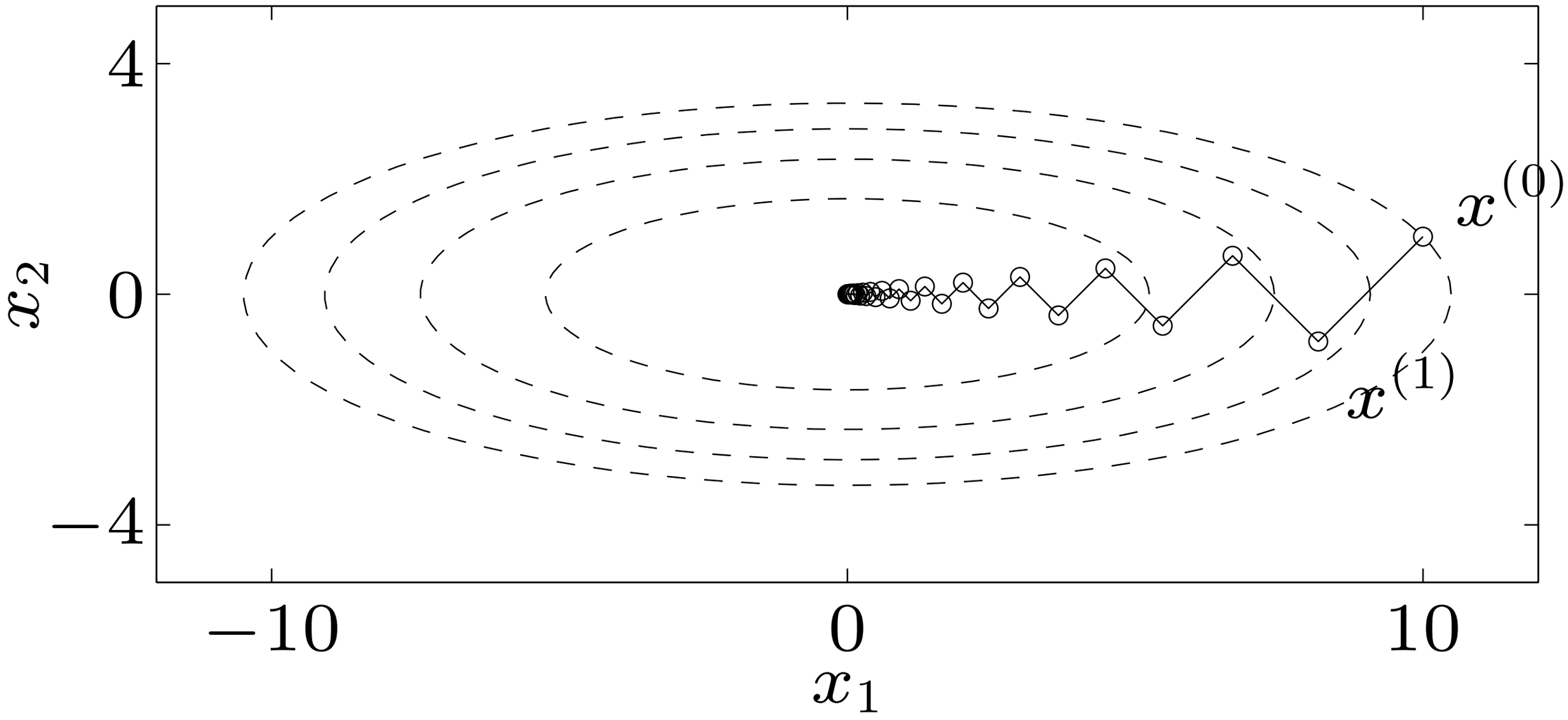
1. Start with a point $\mathbf{x}_0$,
2. Construct the vector $\mathbf{y} = \mathbf{x}_0 - t\nabla g(\mathbf{x}_0)$, for some $t > 0$,
3. Find the value of t which minimises $g(\mathbf{y})$
4. Set this value of $\mathbf{y}$ to be $\mathbf{x}_1$ and repeat from 2 until $g(\mathbf{y})$ cannot be reduced further.

step 3 is a 1D minimisation problem

MATLAB fminbnd - 1D search

$$[x, fval] = fminbnd(f, a, b)$$

finds a local minimizer of f in interval a, b. x is the local minimiser and **fval is the** value of f at that point.


4
0
−4
−10
0
10
x_1
x_2
$x^{(0)}$
$x^{(1)}$

Advantages of steepest descent

Easy to use.

Disadvantages

- ∇q has to be calculated.
- slow. (search directions always orthogonal to each other

  - repeated searches in very different directions

Why are search directions orthogonal ?

$$x_{n+1} = x_n - t_n \nabla g(x_n)$$

where t_n satisfies

$$\frac{d}{dt}\left\{ g(x_n - t \nabla g(x_n)) \right\}\Big|_{t=t_n} = 0$$

Chain rule

$$-\nabla g(x_n - t_n \nabla g(x_n)) \cdot \nabla g(x_n) = 0$$

$$\therefore \quad \nabla g(x_{n+1}) \cdot \nabla g(x_n) = 0$$

$\Rightarrow$ zig-zag approach to minimum.

Newton's Method.

Set $\underline{f}(x) = \nabla q(\underline{x})$ then apply newton methods.

Need $$J = \nabla^2 q(\underline{x}) = \nabla(\nabla q)$$

$$= \frac{\partial^2 q}{\partial x_i \partial x_j}(\underline{x}) \qquad i,j = 1..n$$

Newtons method

$$\underline{x}_{n+1} = \underline{x}_n + \underline{d}_n \qquad \text{where}$$

$$\left[\nabla^2 q(\underline{x}_n)\right] \underline{d}_n = -\nabla q(\underline{x}_n)$$

Example.

$$g(\underline{x}) = \frac{1}{2} x_1^2 + \frac{9}{2} x_2^2$$

$$J = \begin{bmatrix} 1 & 0 \\ 0 & 9 \end{bmatrix}$$

$$\underline{\nabla} g = \begin{pmatrix} x_1 \\ 9x_2 \end{pmatrix}$$

Because function is quadratic Newton finds this in one iteration

$$\text{if } \underline{x}_1 = (9, 1)$$

$$J(\nabla q)\, \underline{d}_1 = -\nabla q$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 9 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = - \begin{bmatrix} 9 \\ 9 \end{bmatrix}, \quad \underline{d}_1 = (d_1, d_2).$$

$$\underline{d}_1 = (-9, -1)$$

$$\underline{x}_2 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

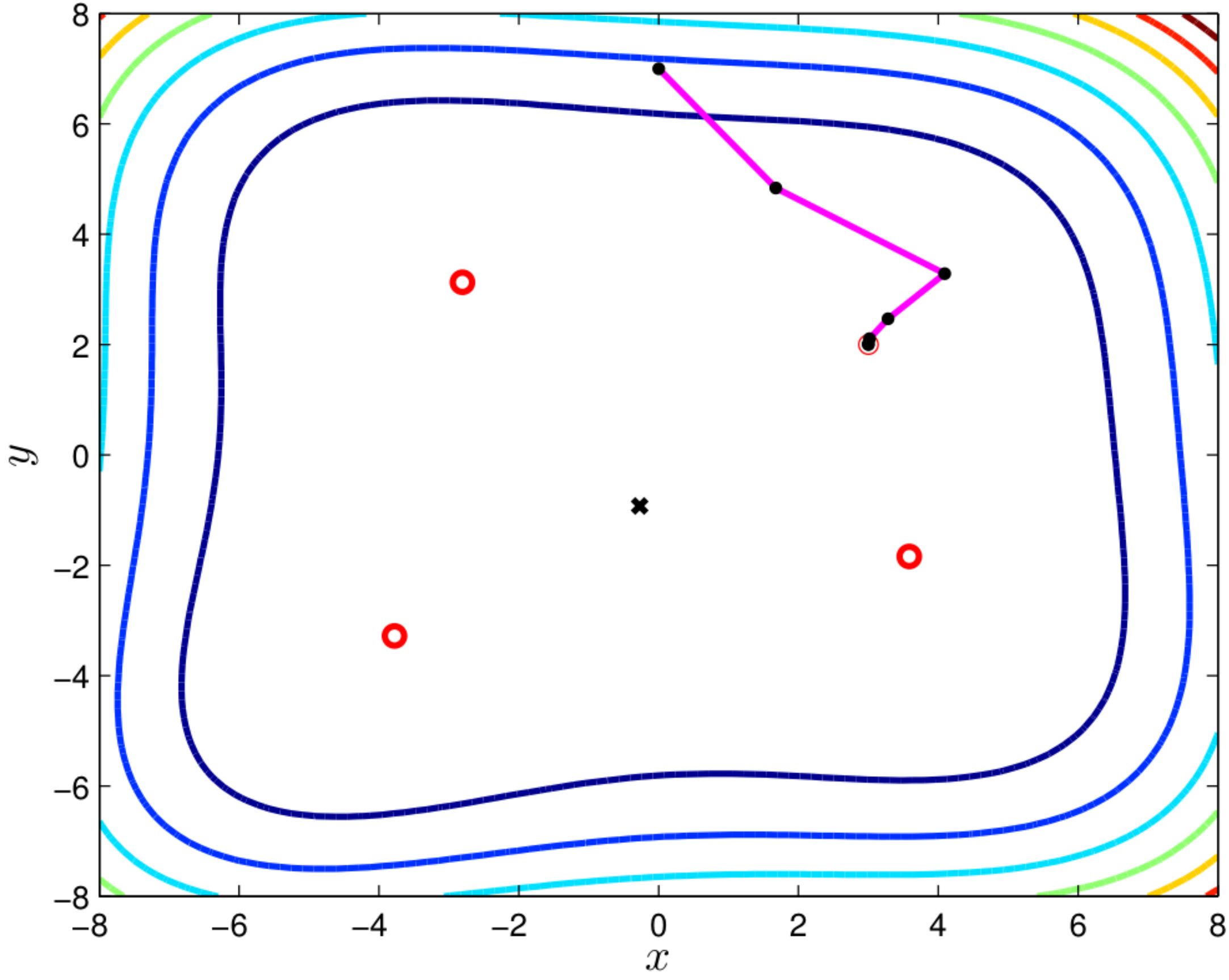
Matlab example: Newton's method for minimization of Himmelblau's function

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

Local maximum of 181.617 at $(-0.270845, -0.923039)$

Four local minima (function value at each minimum is 0) at

$$(3, 2),\ (-2.805, 3.131),\ (-3.779, -3.283),\ (3.584, -1.841)$$


8
6
4
2
0
−2
−4
−6
−8
−8
−6
−4
−2
0
2
4
6
8
y
x

Quasi-Newton Methods

Newton's method is effective for optimization, but it can be unreliable, expensive, and complicated

- Unreliable: Only converges when sufficiently close to a minimum
- Expensive: The Hessian H_f is dense in general, hence very expensive if n is large
- Complicated: Can be impractical or laborious to derive the Hessian

Hence there has been much interest in so-called quasi-Newton methods, which do not require the Hessian

Variant of Newton Method

Many methods approximate Newton method by iterates of the form $\underline{x}_{n+1} = \underline{x}_n + t_n J_n^{-1} \underline{d}_n$

Newton

$$\underline{x}_{n+1} = \underline{x}_n + (\nabla^2 g)^{-1}(-\nabla g)$$

t_n – step length

J_n^{-1} approximates $(\nabla^2 g)^{-1}$

$\underline{d}_n$ approximates $-\nabla g(x_n)$ — approximate search direction.

Can think of Steepest Descent in this way —

$$d_n \text{ — } -\nabla g(x_n)$$

$$J_n \text{ — } I$$

t_n — result of a line search.

The DFP (Davidon, Fletcher, Powell) is another example.

Matlab: Function fminsearch

- Requires g and an initial guess $\underline{x}_0$.
- Finds local minimum using simplex method (for which there is very little theory.

↑
(Nelder-Meade simplex algorithm)

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) method is one of the most popular quasi-Newton methods:

1: choose initial guess x_0
2: choose B_0, initial Hessian guess, e.g. $B_0 = \mathrm{I}$
3: **for** $k = 0, 1, 2, \ldots$ **do**
4: solve $B_k s_k = -\nabla f(x_k)$
5: $x_{k+1} = x_k + s_k$
6: $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
7: $B_{k+1} = B_k + \Delta B_k$
8: **end for**

where

$$\Delta B_k \equiv \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$$

We won't go into details of the rationale behind the B_k updating scheme...

Basic idea is that B_k accumulates second derivative information on successive iterations, eventually approximates H_f well

BFGS (+ trust region) is implemented in Matlab's `fminunc` function, e.g.

```
x0 = [5;5];
options = optimset('GradObj','on');
[x,fval,exitflag,output] = ...
  fminunc(@himmelblau_function,x0,options);
```

where `himmelblau_function` is given by:

```
function [f,grad] = himmelblau_function(x)

f = (x(1,:).^2 + x(2,:) - 11).^2 + (x(1,:) + x(2,:).^2 - 7).^2;
grad = [4*x(1,:).*(x(1,:).^2+x(2,:)-11) + 2*(x(1,:)+x(2,:).^2-7)
        2*(x(1,:).^2+x(2,:)-11) + 4*x(2,:).*(x(1,:)+x(2,:).^2-7)]
```

`fminunc` with starting point $x_0 = [5, 5]^T$ finds the minimum $x^* = [3, 2]^T$

Applications

1. Minimisation of large systems

   Ex: Elliptic pdes (elasticity, electrostatics)

   solution minimises an energy function

2. Linear systems of form

$$A\underline{x} = \underline{b} \quad , \qquad \underline{x}^T A \underline{x} > 0 \quad \forall\, \underline{x} \in \mathbb{R}^n$$

symmetric, +ve definite

minimise $g(\underline{x}) = \frac{1}{2}\underline{x}^T A \underline{x} - \underline{b}^T \underline{x}$

$g'(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b$

If A symmetric $g'(x) = Ax - b$

Conjugate Gradient Algorithm.

3. Nonlinear Systems of form

$$\underline{f}(\underline{x}) = \underline{0}$$

Try to find a solution x^* by minimising the scalar valued function

$$g(\underline{x}) = \| \underline{f}(\underline{x}) \|_2^2 = \sum_i | f_i(\underline{x}) |^2$$

(more generally $g(\underline{x}) = \sum_i \alpha_i | f_i(\underline{x}) |^2$, $\alpha_i > 0$ - weights.

Careful: Need global minimum and we have an algorithm for a local minimum. Hopefully if initial guess is 'good' then we have found the global minimum. (Always check that the minmum located has $g = 0$).

Example:

$$f_1 = x - y$$

$$f_2 = x^2 + y^2 - 1$$

$\underline{f} = (f_1, f_2)$ $\qquad$ $\underline{f}(\underline{x}) = \underline{0} \Rightarrow$ $\qquad$ $\underline{x} = (x, y) = \pm\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$

Set $g(x, y) = (x-y)^2 + (x^2+y^2-1)^2$

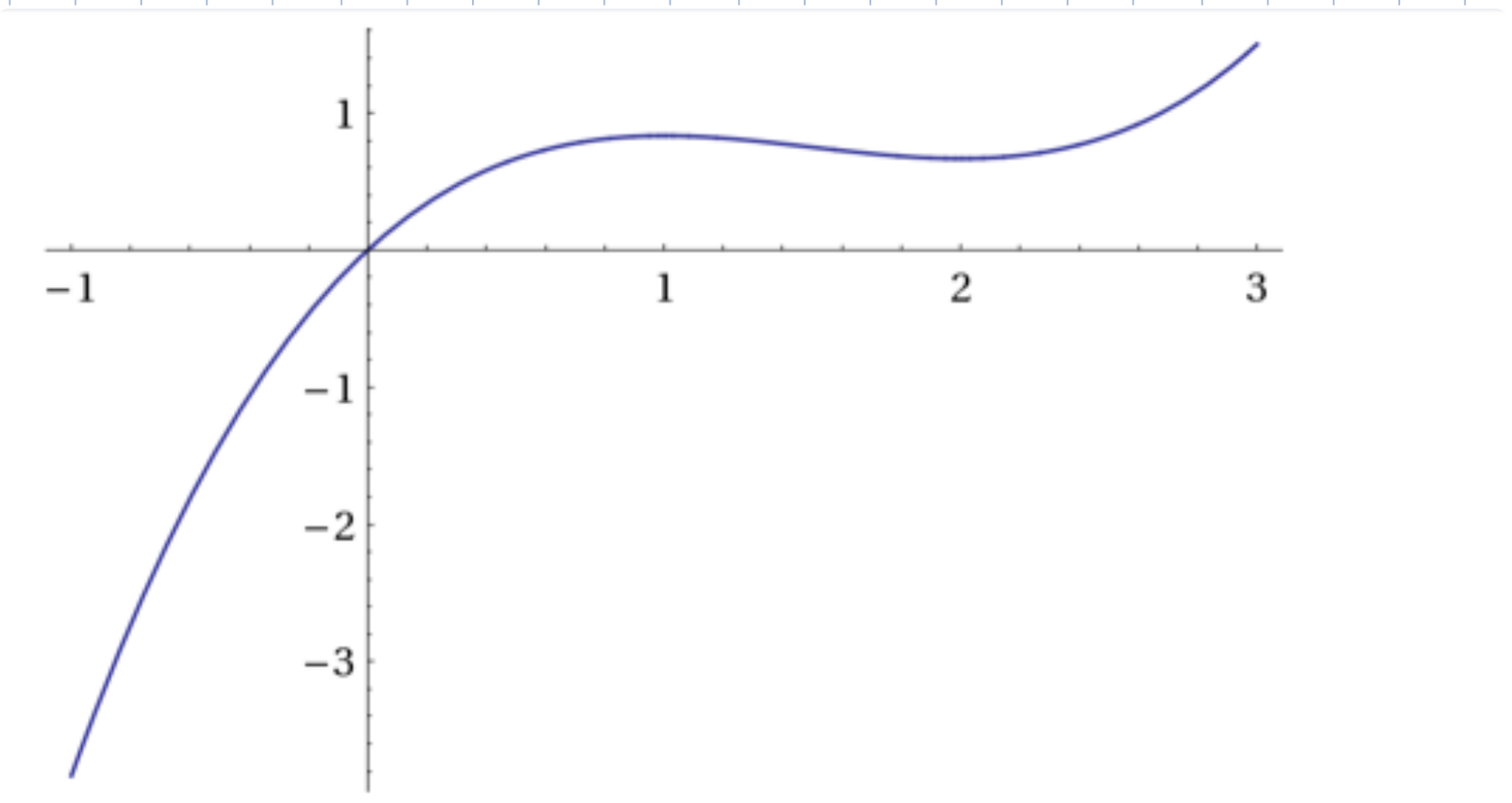
Matlab code:

```
function f = gmin(xx)

x = xx(1);
y = xx(2);
f= (x-y)^2+(x^2 +y^2 -1)^2;

end
```

```
>> [x,r] = fminsearch('gmin',[1,2])

x =

    0.7071    0.7071


r =

   5.4074e-09

>> [x,r] = fminsearch('gmin', [-1,2])

x =

   -0.7071   -0.7071


r =

   1.7369e-09

>>
```

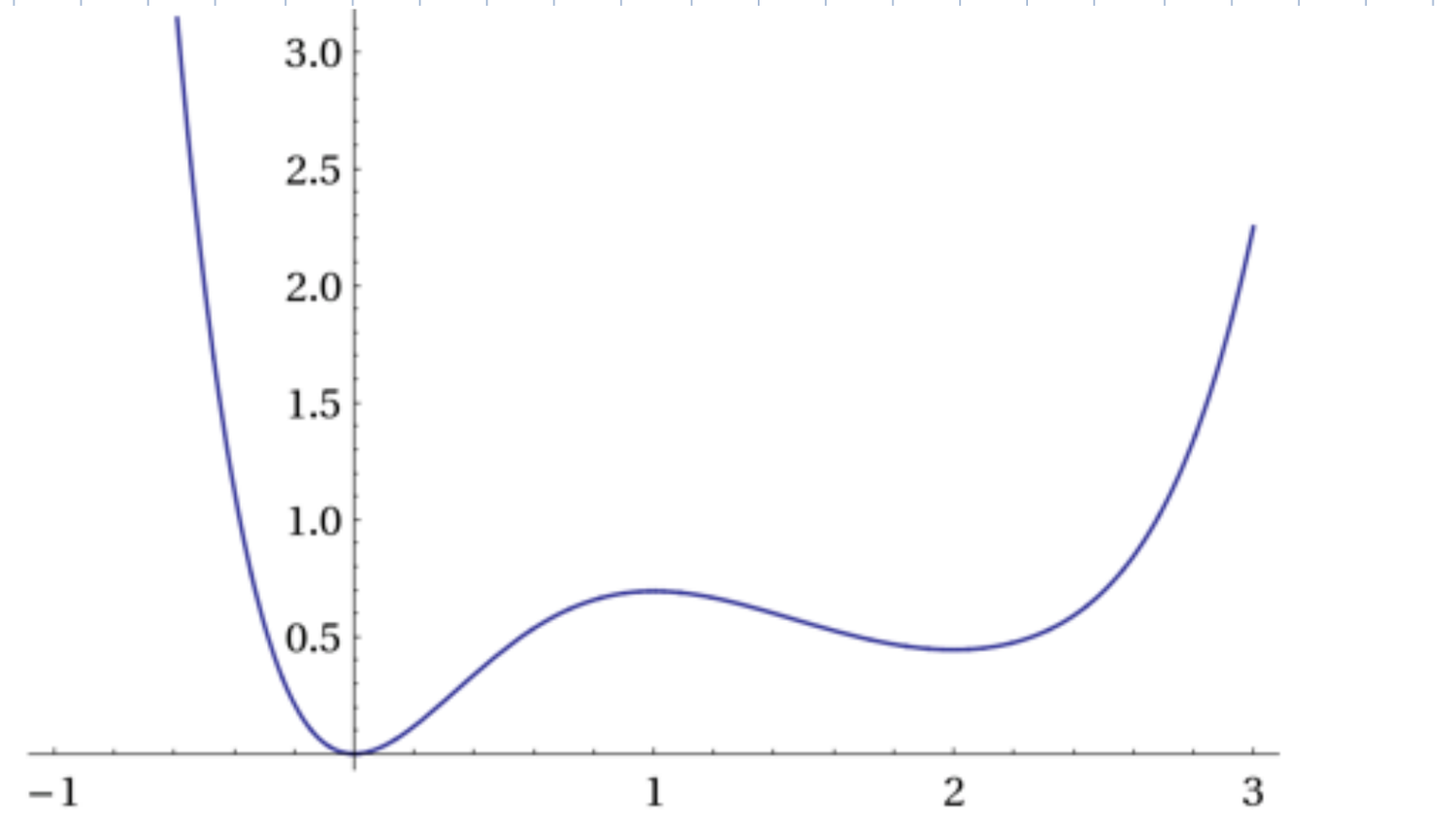
A problem with using this method for finding the root of $f(x)$ is that $g(x)$ may have several local minima

Ex: $f(x) = \frac{x^3}{3} - \frac{3}{2}x^2 + 2x$



$f'(x) = 0 \Rightarrow x = 0$

The function $g(x) = f^2(x)$ has a local minimum at $x = 0$, $x = 2$



Matlab uses function fminbnd (golden section search and parabolic interpolation) to find a local minimum of a real valued function of a real variable.

```
>> [z,r] = fminbnd('(x^3/3 -(3/2)*x^2+2*x)^2',-1,10)

z =

    2.0000


r =

    0.4444
```

The interval supplied contains both minima — and 2 is chosen.

In higher dimensions the problem is worse and fminsearch can easily be fooled into finding a 'false' local minimum.

Many excellent and sophisticated packages for both unconstrained and constrained optimisation. For example LANCELOT code.

Finding a global minimum of a general function g(x) is a very hard task. Only recently have effective algorithms been developed. These algorithms include simulated annealing and genetic algorithms.