Numerical optimisation, generalities

Complementary material usable in CHEM-E7195, 2019-2020

Francesco Corona

Chemical and Metallurgical Engineering
School of Chemical Engineering

Numerical optimisation

Minimisation (maximisation)

$\rightsquigarrow$ Find a global or local minimum (maximum) of some objective function

Numerical optimisation (cont.)

Definition

Let $f : \mathcal{R}^n \to \mathcal{R}$ with $n \geq 1$ be a **cost** or an **objective function**

The **unconstrained optimisation** problem

$$\rightsquigarrow \quad \min_{\mathbf{x} \in \mathcal{R}^n} f(\mathbf{x}) \tag{1}$$

The **constrained optimisation** problem

$$\rightsquigarrow \quad \min_{\mathbf{x} \in \Omega \subset \mathcal{R}^n} f(\mathbf{x}) \tag{2}$$

$\Omega \subset \mathcal{R}^n$ is a closed subset determined by equality and inequality constraints

- Constraints are dictated by the nature of the problem to solve

Numerical optimisation (cont.)

Example

Find the optimal allocation of $i = 1, \ldots, n$ bounded resources x_i

$\rightsquigarrow$ Bounded resources means limited resources

The constraints express these limits in terms of inequalities

$$0 \leq x_i \leq C_i, \quad \text{with } C_i \text{ some given constants}$$

The set $\Omega = \{\mathbf{x} = (x_1, \ldots, x_n) : 0 \leq x_i \leq C_i, i = 1, \ldots, n\}$

- A subset of $\mathcal{R}^n$ determined by such constraints

Numerical optimisation (cont.)

For some problems, Ω is characterised by explicit conditions

$\rightsquigarrow$ **Equality constraints**

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

$\rightsquigarrow$ **Inequality constraints**

$$\mathbf{h}(\mathbf{x}) \leq \mathbf{0}$$

$\mathbf{h} : \mathcal{R}^n \to \mathcal{R}^m$ with $m \leq n$ indicates some function of $\mathbf{x}$

- $\mathbf{h}(\mathbf{x}) \leq \mathbf{0}$ corresponds to $h_i(x_1, x_2, \cdots, x_n) \leq 0$, for $i = 1, \ldots, m$

Numerical optimisation (cont.)

Definition

Let f be a continuous function and let Ω be a connected set

A constrained optimisation problem is a **non-linear programming** problem

Convex programming

$\rightsquigarrow$ f is a convex function and $\mathbf{h}$ has convex components

Linear programming

$\rightsquigarrow$ f and $\mathbf{h}$ are linear

Quadratic programming

$\rightsquigarrow$ f is quadratic and $\mathbf{h}$ is linear

Numerical optimisation (cont.)

Remark

Minimisation v maximisation

Computing the maximum of function f is equivalent to computing the minimum of function $g = -f$

$\rightsquigarrow$ We shall only consider minimisation algorithms

Definition

The minimum value of some given objective function is interesting

The point at which such minimum is achieved is more interesting

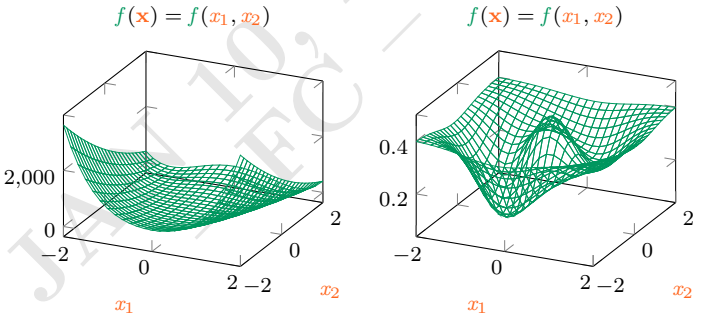
$\rightsquigarrow$ Such point is called **minimiser**

Numerical optimisation (cont.)

We consider the numerical solutions of optimisation problems

The ideal situation is a function with *unique global* minimiser

- There are often several (local) minimisers



Numerical optimisation (cont.)

We are interested in finding either a (good) local or the global minimiser

- This is the simple meaning of minimising an objective function

Definition

Point $\mathbf{x}^*$ is a **global minimiser** of f

$\rightsquigarrow$ if $f(\mathbf{x}^*) \leq f(\mathbf{x})$, $\forall \mathbf{x} \in \mathcal{R}^n$

Point $\mathbf{x}^*$ is a **local minimiser** of f

$\rightsquigarrow$ if there is a $B_r(\mathbf{x}^*) \subset \mathcal{R}^n$, a ball centred in $\mathbf{x}^*$ and radius $r > 0$, such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$, $\forall \mathbf{x} \in B_r(\mathbf{x}^*)$

Unconstrained optimisation

Unconstrained optimisation (cont.)

Definition

Let f be differentiable in $\mathcal{R}^n$ with first and second derivatives

Let the **gradient vector** of f at point $\mathbf{x} \in \mathcal{R}^n$ be denoted by the symbol

$$\boldsymbol{\nabla} f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n}\right]^T \tag{3}$$

Let the **Hessian matrix** of f at point $\mathbf{x} \in \mathcal{R}^n$ be denoted by the symbol

$$\mathbf{H}(\mathbf{x}) = (h_{ij})_{i,j=1}^n, \quad \text{with } h_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_j \partial x_i} \tag{4}$$

In general, it will be assumed that problem functions are smooth

- Continuous and continuously (Frétchet) differentiable, $\mathcal{C}^1$

Unconstrained optimisation (cont.)

For $f(\mathbf{x})$ at any point $\mathbf{x}$ there is a **vector of first derivatives**

$\rightsquigarrow$ **Gradient vector**

$$\begin{bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \\ \vdots \\ \partial f/\partial x_n \end{bmatrix}_{\mathbf{x}} = \boldsymbol{\nabla} f(\mathbf{x}) \tag{5}$$

$\boldsymbol{\nabla}$ is the gradient operator $\left[\partial/\partial x_1, \partial/\partial x_2, \cdots, \partial/\partial x_n\right]^T$

Unconstrained optimisation (cont.)

Let $f(\mathbf{x})$ be twice-differentiable, $\mathcal{C}^2$

There is a **matrix of second partial derivatives**

$\rightsquigarrow$ **Hessian matrix**

$$\left[\begin{array}{ccc} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{array}\right]_{\mathbf{x}} = \mathbf{H}(\mathbf{x}) = \boldsymbol{\nabla}^2 f(\mathbf{x}) \tag{6}$$

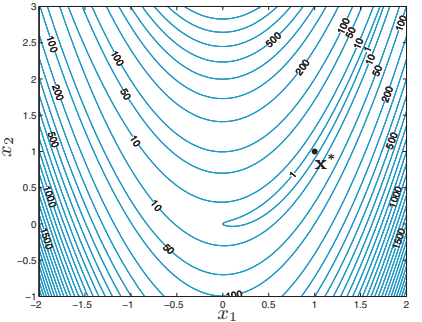
The (i, j)-th element of the Hessian matrix, $\partial^2 f / (\partial x_i \partial x_j)$

Unconstrained optimisation (cont.)

Example

Rosenbrock's function

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$



The global minimum is at $\mathbf{x}^* = (1, 1)$, and variation around $\mathbf{x}^*$ is low

Unconstrained optimisation (cont.)

A test-function for optimisation methods

$$f(\mathbf{x}) = f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

- $\partial f/\partial x_1 = -400x_1(x_2 - x_1^2) - 2(1 - x_1)$
- $\partial f/\partial x_2 = 200(x_2 - x_1^2)$

$$\boldsymbol{\nabla} f(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} \\ \dfrac{\partial f}{\partial x_2} \end{bmatrix}_{\mathbf{x}} = \begin{bmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix}_{\mathbf{x}} \tag{7}$$

Unconstrained optimisation (cont.)

$$f(\mathbf{x}) = f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

- $\partial f/\partial x_1 = -400x_1(x_2 - x_1^2) - 2(1 - x_1)$
- $\partial f/\partial x_2 = 200(x_2 - x_1^2)$

- $\partial^2 f/(\partial x_1 \partial x_1) = 1200x_1^2 - 400x_2 + 2$
- $\partial^2 f/(\partial x_1 \partial x_2) = -400x_1$
- $\partial^2 f/(\partial x_2 \partial x_1) = -400x_1$
- $\partial^2 f/(\partial x_2 \partial x_2) = 200$

$$\begin{aligned} \boldsymbol{\nabla}^2 f(\mathbf{x}) &= \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1 \partial x_1} & \dfrac{\partial^2 f}{\partial x_1 x_2} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2 x_2} \end{bmatrix}_{\mathbf{x}} \\ &= \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}_{\mathbf{x}} \end{aligned} \tag{8}$$

Unconstrained optimisation (cont.)

In general, ∇f and $\nabla^2 f$ will vary with $\mathbf{x}$

At $\mathbf{x}' = (0, 0)^T$

$$\begin{aligned}\nabla f(\mathbf{x}) &= \begin{bmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix}_{\mathbf{x}} \\ &= \begin{bmatrix} -2 \\ 0 \end{bmatrix}_{\mathbf{x}=(0,0)^T}\end{aligned}$$

$$\begin{aligned}\nabla^2 f(\mathbf{x}) &= \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}_{\mathbf{x}} \\ &= \begin{bmatrix} 2 & 0 \\ 0 & 200 \end{bmatrix}_{\mathbf{x}=(0,0)^T}\end{aligned}$$

Unconstrained optimisation (cont.)

The idea of a line is also important

Definition

We can define the **line** as a set of points

$$\mathbf{x}[= [\mathbf{x}(\alpha)] = \mathbf{x}' + \alpha\mathbf{d}, \quad \text{for all } \alpha$$

$\mathbf{x}'$ is some fixed point along the line

- It corresponds to $\alpha = 0$

$\mathbf{d}$ is the direction of the line

Example

Let the fixed point $\mathbf{x}'$ be the point $(2, 2)$, let the direction $\mathbf{d}$ be $(3, 1)$

$\rightsquigarrow$ Draw the line $\mathbf{x} = \mathbf{x}' + \alpha\mathbf{d}$, for all α

Unconstrained optimisation (cont.)

We can determine expressions for the derivatives of f along any line $\mathbf{x}(\alpha)$

$\rightsquigarrow$ Based on definitions of line, gradient vector and Hessian matrix

By the chain rule of derivation of scalar-valued function of a vector

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\{\,\cdot\,[\mathbf{x}(\alpha)]\} = \sum_i \frac{\mathrm{d}x_i(\alpha)}{\mathrm{d}\alpha}\frac{\partial}{\partial x_i}\{\,\cdot\,[\mathbf{x}(\alpha)]\} = \sum_i d_i \frac{\partial}{\partial x_i}\{\,\cdot\,[\mathbf{x}(\alpha)]\}$$
$$= \mathbf{d}^T\boldsymbol{\nabla}\{\,\cdot\,[\mathbf{x}(\alpha)]\}$$

The slope of $f\{=f[\mathbf{x}(\alpha)]\}$ along the line at any point $\mathbf{x}(\alpha)$

$$\frac{\mathrm{d}f}{\mathrm{d}\alpha} = \mathbf{d}^T\boldsymbol{\nabla}f = \boldsymbol{\nabla}f^T\mathbf{d}$$

This is the directional derivative of f with respect to $\mathbf{d}$

- ∇f is calculated at $\mathbf{x}(\alpha)$

Unconstrained optimisation (cont.)

The curvature of $f\{ = f[\mathbf{x}(\alpha)]\}$ along the line at any point $\mathbf{x}(\alpha)$

$$\frac{\mathrm{d}^2 f}{\mathrm{d}\alpha^2} = \frac{\mathrm{d}}{\mathrm{d}\alpha}\left(\frac{\mathrm{d}f}{\mathrm{d}\alpha}\right) = \mathbf{d}^T \boldsymbol{\nabla}(\boldsymbol{\nabla} f^T \mathbf{d}) = \mathbf{d}^T \boldsymbol{\nabla}^2 f \mathbf{d}$$

This is the second-order directional derivative of f

- $\boldsymbol{\nabla} f$ and $\boldsymbol{\nabla}^2 f$ are calculated at $\mathbf{x}(\alpha)$

Let $\mathbf{G} = \boldsymbol{\nabla}^2 f$, then $\mathbf{Gd}$ is a vector

$$(\mathbf{Gd})_i = \sum_j G_{ij}\, d_j$$

$\mathbf{d}^T\mathbf{Gd}$ is the scalar product of $\mathbf{d}$ and $\mathbf{Gd}$

Unconstrained optimisation (cont.)

Example

Rosenbrock's function

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

$$\boldsymbol{\nabla} f(\mathbf{x}) = \begin{bmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix}_{\mathbf{x}=\mathbf{0}}$$

$$\boldsymbol{\nabla}^2 f(\mathbf{x}) = \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}_{\mathbf{x}}$$

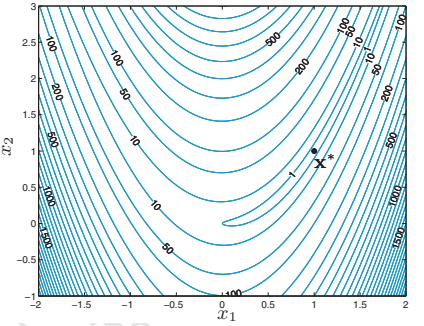
We consider point $\mathbf{x}' = (0, 0)^T$

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

$$\boldsymbol{\nabla} f(\mathbf{x}) = \begin{bmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix}_{\mathbf{x}}$$

$$\boldsymbol{\nabla}^2 f(\mathbf{x}) = \begin{bmatrix} 1200x_1^2 - 400x_2 + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}_{\mathbf{x}}$$

Unconstrained optimisation (cont.)



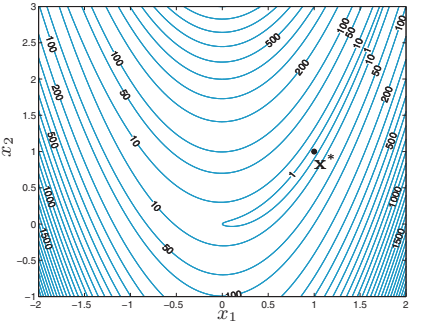
The slope at point $\mathbf{x}' = (0, 0)^T$, along the line with direction $\mathbf{d} = (1, 0)^T$

$$\mathbf{d}^T \nabla f(\mathbf{x}') = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} -2 \\ 0 \end{bmatrix} = -2$$

The slope at point $\mathbf{x}' = (0, 0)^T$, along the line with direction $\mathbf{d} = (0, 1)^T$

$$\mathbf{d}^T \nabla f(\mathbf{x}') = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 0 \end{bmatrix} = 1$$

Unconstrained optimisation (cont.)



The curvature at point $\mathbf{x}' = (0,0)^T$, along the line with direction $\mathbf{d} = (1,0)^T$

$$\mathbf{d}^T\mathbf{G}\mathbf{d} = \begin{bmatrix}1 & 0\end{bmatrix} \underbrace{\begin{bmatrix}2 & 0\\ 0 & 200\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}}_{\begin{bmatrix}2 & 0\end{bmatrix}^T} = 2$$

Unconstrained optimisation (cont.)

Definition

Let $f \in \mathcal{C}^2(\mathcal{R}^n)$, all first and second derivatives exist and are continuous

Then, $\mathbf{H}(\mathbf{x})$ is symmetric for every $\mathbf{x} \in \mathcal{R}^n$

Definition

A point $\mathbf{x}^*$ is called a **stationary** or **critical point** for f if $\boldsymbol{\nabla} f(\mathbf{x}^*) = \mathbf{0}$

A point such that $\boldsymbol{\nabla} f(\mathbf{x}^*) \neq \mathbf{0}$ is called a **regular point**

A function f over $\mathcal{R}^n$ does not necessarily admit a minimiser

- Should this point exist it is not necessarily unique

Unconstrained optimisation (cont.)

Example

- $f(\mathbf{x}) = x_1 + 3x_2$ is unbounded in $\mathcal{R}^2$
- $f(\mathbf{x}) = \sin(x_1)\sin(x_2)\cdots\sin(x_n)$ admits an infinite number of minimisers and maximisers in $\mathcal{R}^n$, they are both local and global

Unconstrained optimisation (cont.)

Definition

Function $f : \Omega \subseteq \mathcal{R}^n \to \mathcal{R}$ is **convex** in Ω if

$$\rightsquigarrow \quad f\big[\alpha\mathbf{x} + (1-\alpha)\mathbf{y}\big] \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y}), \quad \forall \mathbf{x}, \mathbf{y} \in \Omega \tag{9}$$

for all $\alpha \in [0, 1]$

Definition

Function f is **Lipschitz** in Ω if

$$\rightsquigarrow \quad ||f(\mathbf{x}) - f(\mathbf{y})|| \leq L||\mathbf{x} - \mathbf{y}||, \quad \forall \mathbf{x}, \mathbf{y} \in \Omega \tag{10}$$

for some constant $L > 0$

Unconstrained optimisation (cont.)

Proposition

Optimality conditions

Let $\mathbf{x}^* \in \mathcal{R}^n$ and $r > 0$ exists such that $f \in \mathcal{C}^1[B_r(\mathbf{x}^*)]$

- If $\mathbf{x}^*$ is a minimiser for f (local or global), then $\nabla f(\mathbf{x}^*) = \mathbf{0}$
  - Also, if $f \in \mathcal{C}^2[B_r(\mathbf{x}^*)]$, $\mathbf{H}(\mathbf{x}^*)$ is positive semidefinite (PSD)

Let $\mathbf{x}^* \in \mathcal{R}^n$ and $r > 0$ exists such that $f \in \mathcal{C}^2[B_r(\mathbf{x}^*)]$

- If $\boldsymbol{\nabla} f(\mathbf{x}^*) = \mathbf{0}$ and $\mathbf{H}(\mathbf{x}^*)$ is positive definite (PD) for all $\mathbf{x} \in B_r(\mathbf{x}^*)$, then $\mathbf{x}^*$ is a local minimiser of f
- If $f \in \mathcal{C}^1(\mathcal{R}^n)$ is convex in $\mathcal{R}^n$ and $\boldsymbol{\nabla} f(\mathbf{x}^*) = \mathbf{0}$, then $\mathbf{x}^*$ is a global minimiser for f

Unconstrained optimisation (cont.)

Definition

A symmetric real matrix $\mathbf{A} \in \mathcal{R}^{n \times n}$ is **positive definite** (**PD**) if

$$\forall \mathbf{x} \in \mathcal{R}^n \text{ with } \mathbf{x} \neq \mathbf{0}, \quad \rightsquigarrow \quad \mathbf{x}^T \mathbf{A} \mathbf{x} > 0$$

A symmetric real matrix $\mathbf{A} \in \mathcal{R}^{n \times n}$ is **positive semidefinite** (**PSD**) if

$$\forall \mathbf{x} \in \mathcal{R}^n \text{ with } \mathbf{x} \neq \mathbf{0}, \quad \rightsquigarrow \quad \mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$$

Unconstrained optimisation (cont.)

Most methods for numerical optimisation are of iterative type

- They can be classified into two main categories

Depending on whether they use derivatives of the objective

Derivative-free methods

- ⇝ They explore the local behaviour of a cost function
- ⇝ Direct comparison between function values

Methods using derivatives

- ⇝ They use local information of the objective

Unconstrained optimisation (cont.)

Methods based on derivatives are expected faster convergence

Remark

It can be shown that given $\overline{\mathbf{x}} \in \text{dom}(f)$, if $\nabla f(\overline{\mathbf{x}})$ exists and it is not null, then the largest increase of f from $\overline{\mathbf{x}}$ is along the gradient vector

Conversely, the largest decrease is along the opposite direction

Among them, the two most important classes of techniques

- ⇝ **Trust-region methods**
- ⇝ **Line-search methods**

The Newton method

Numerical optimisation

The Newton method

Let $f : \mathcal{R}^n \to \mathcal{R}$ with $n \geq 1$ be of class $\mathcal{C}^2(\mathcal{R}^n)$

We know how to compute its first and second order partial derivatives

We apply Newton's method to solve a system of nonlinear equation

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

The Newton method (cont.)

Remark

Newton's method

Consider the problem of finding the zero of some $f : [a, b] \subset \mathcal{R} \to \mathcal{R}$

$$\rightsquigarrow \quad \text{Find } \alpha \in [a, b] \text{ such that } f(\alpha) = 0$$

We know the equation of the tangent to function $f(x)$ at some point $x^{(k)}$

$$y(x) = f\left[x^{(k)}\right] + f'\left[x^{(k)}\right]\left[x - x^{(k)}\right]$$

We can solve for some point $x = x^{(k+1)}$, such that $y\left[x^{(k+1)}\right] = 0$

$$\rightsquigarrow \quad x^{(k+1)} = x^{(k)} - \frac{f\left[x^{(k)}\right]}{f'\left[x^{(k)}\right]}$$

All this, for $k = 0, 1, 2, \ldots$ and $f'\left[x^{(k)}\right] \neq 0$

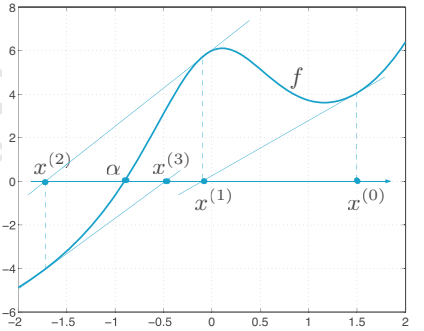
The Newton method (cont.)

$$y(x) = f\left[x^{(k)}\right] + f'\left[x^{(k)}\right]\left[x - x^{(k)}\right]$$

$$x^{(k+1)} = x^{(k)} - \frac{f\left[x^{(k)}\right]}{f'\left[x^{(k)}\right]}$$

Sequence $\{x^{(k)}\}$ is the **Newton method** for finding the zero of function f

$\rightsquigarrow$ The method reduces to locally substituting f with its tangent



The Newton method (cont.)

Consider now a set of nonlinear equations

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\ldots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

We re-write the system in vector form

- Let $\mathbf{f} \equiv (f_1, \ldots, f_n)^T$
- Let $\mathbf{x} \equiv (x_1, \ldots, x_n)^T$

$$\rightsquigarrow \quad \mathbf{f}(\mathbf{x}) = \mathbf{0}$$

We want solve the system of nonlinear equation

$\rightsquigarrow$ We can extend Newton's method

Replace first derivative of function f with Jacobian $\mathbf{J_f}$ of function $\mathbf{f}$

$$\rightsquigarrow \quad (\mathbf{J_f})_{ij} \equiv \frac{\partial f_i}{\partial x_j}, \quad \text{with } i, j = 1, \ldots, n$$

The Newton method (cont.)

Consider the general system of nonlinear equations $\mathbf{f}(\mathbf{x}) = \mathbf{0}$

$$\begin{aligned} f_1(x_1, \ldots, x_j, \ldots, x_n) &= 0 \\ &\cdots \\ f_i(x_1, \ldots, x_j, \ldots, x_n) &= 0 \\ &\cdots \\ f_n(x_1, \ldots, x_j, \ldots, x_n) &= 0 \end{aligned}$$

The corresponding Jacobian matrix

$$\mathbf{J_f}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

The Newton method (cont.)

Given this notation, we define the multivariable Newton's method[1]

Pseudo-code

Let $\mathbf{x}^{(0)} \in \mathcal{R}^n$ be an initial solution

For $k = 0, 1, \ldots,$ until convergence

$$\text{Solve} \quad \mathbf{J_f}\left[\mathbf{x}^{(k)}\right]\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{f}\left[\mathbf{x}^{(k)}\right]$$
$$\text{Set} \quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)}$$

At each iteration, a linear system with matrix $\mathbf{J_f}\left[\mathbf{x}^{(k)}\right]$ must be solved

[1] $x^{(k+1)} = x^{(k)} - f\left[x^{(k)}\right]/f'\left[x^{(k)}\right]$, $\delta_x^{(k)} = x^{(k+1)} - x^{(k)}$, $\rightsquigarrow f'\left[x^{(k)}\right]\delta_x^{(k)} = -f\left[x^{(k)}\right]$

The Newton method (cont.)


```
function [x,res,iter] = sNWT(F_fun,J_fun,x_0,tol,imx)
% [ROOT,RES,ITER]=SNWT(F_FUN,J_FUN,X_0,TOL,IMX) Calculate vector ROOT,
% the zero of a nonlinear system defined in F_FUN with Jacobian J_FUN,
% from initial point X_0
%
% RES is residual in ROOT and ITER is number of iterations
% F_FUN e J_FUN are external functions (as M-files)

iter=0; x=x_0; err=1+tol;

while err >= tol & iter < imx
 J = J_fun(x);
 F = F_fun(x);
 deltax = -J\F;                        %(Matlab/Octave backslash operator)
 x = x + deltax;
 err = norm(deltax); iter = 1+iter;
end
res = norm(F_fun(x));

if(iter==imx & err > tol)
 disp('[Out by KMAX]');
else
 disp('[Out by TOL]'));
end
return
```

The Newton method (cont.)


```
function J = J_fun(x)
 J(1,1) = dF_1 / dx_1;                    % Add your own expression
 J(1,2) = dF_1 / dx_2;                    % Add your own expression
 ...

 J(2,1) = dF_2 / dx_1;                    % Add your own expression
 J(2,2) = dF_2 / dx_2;                    % Add your own expression
 ...

 J(N,1) = dF_N / dx_1;                    % Add your own expression
 J(N,2) = dF_N / dx_2;                    % Add your own expression
 ...

 return
```

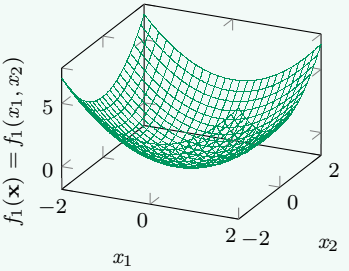
```
function F = F_fun(x)
 F(1,1) = F_1(x_1,x_2,...);               % Add your own expression
 F(2,1) = F_2(x_1,x_2,...);               % Add your own expression
 ...
 F(N,1) = F_N(x_1,x_2,...);               % Add your own expression

return
```

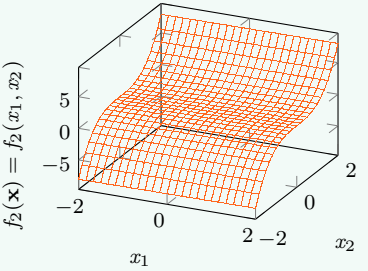
The Newton method (cont.)

Example

Consider the nonlinear system of equations

$$\begin{cases} f_1(x_1, x_2) = x_1^2 + x_2^2 = 1 \\ f_2(x_1, x_2) = \sin\left(\frac{\pi}{2} x_1\right) + x_2^3 = 0 \end{cases}$$





The system has two solutions

- $\approx (0.47, -0.88)$ and $\approx (-0.47, 0.88)$

The Newton method (cont.)

$$\begin{cases} f_1(x_1, x_2) = x_1^2 + x_2^2 = 1 \\ f_2(x_1, x_2) = \sin(\pi/2x_1) + x_2^3 = 0 \end{cases}$$

```
function F=F_fun(x)
 hpi = 0.5*pi;
 F(1,1) = x(1)^2 + x(2)^2 = 1;
 F(2,1) = sin(pih*x(1)) + x(2)^3 = 0;
return
```

```
function J=J_fun(x)
 hpi = 0.5*pi;
 J(1,1) = 2*x(1);              J(1,2) = 2*x(2);
 J(2,1) = hpi*cos(hpi*x(1));   J(2,2) = 3*x(2)^2;
return
```

Suppose we start the solution from point $\mathbf{x}^{(0)} = (1, 1)^T$

- Let $\varepsilon = 0.00001$ be the user-defined tolerance

```
x_0=[1;1]; % Initial solution
tol=1e-5;  % Tolerance
imx=20;    % Iteration

[x,res,iter] = sNWT(@F_fun,@J_fun,x_0,tol,imx);
```

■

The Newton method (cont.)

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

- The Jacobian $\mathbf{J_f}\left[\mathbf{x}^{(k)}\right]$ of the system is the Hessian matrix $\mathbf{H}(\mathbf{x})$ of f
- As computed at the generic iteration point $\mathbf{x}^{(k)}$

Pseudo-code

Given $\mathbf{x}^{(0)} \in \mathcal{R}^n$, for $k = 0, 1, \ldots,$ until convergence

$$\begin{aligned} \text{Solve} \quad & \underbrace{\mathbf{H}\left[\mathbf{x}^{(k)}\right]}_{\mathbf{J_f}\left[\mathbf{x}^{(k)}\right]} \boldsymbol{\delta}\mathbf{x}^{(k)} = -\underbrace{\boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]}_{\mathbf{f}\left[\mathbf{x}^{(k)}\right]} \\ \text{Set} \quad & \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)} \end{aligned} \tag{11}$$

We can define a suitable **stopping test/criterion**

$$\left|\left|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right|\right| \leq \varepsilon$$

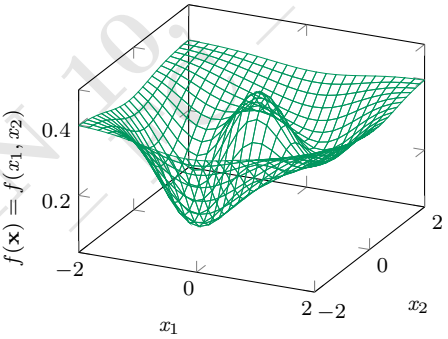
$\varepsilon > 0$ is an appropriate **tolerance value**

The Newton method (cont.)

Example

Consider the objective function $f(\mathbf{x})$,

$$f(\mathbf{x}) = 2/5 - 1/10(5x_1^2 + 5x_2^2 + 3x_1x_2 - x_1 - 2x_2)e^{\left[-\left(x_1^2+x_2^2\right)\right]}$$



We want to approximate the global minimum $\left[\mathbf{x}^* \approx (-0.63, -0.70)\right]$

The Newton method (cont.)

Netwon's method with a tolerance $\varepsilon = 10^{-5}$, from different initial points

Let $\mathbf{x}^{(0)} = (-0.9, -0.9)$

$\rightsquigarrow$ After 5 iterations the method converges to `x=[-0.63058;-0.70074]`

Let $\mathbf{x}^{(0)} = (-1.0, -1.0)$

$\rightsquigarrow$ After 400 iterations the stopping criterion is still not fulfilled

Let $\mathbf{x}^{(0)} = (+0.5, -0.5)$

$\rightsquigarrow$ After 5 iterations the method converges to the saddle point

- `x=[0.80659; -0.54010]`

Note that Newton's method may converge to any stationary point

- (A point that is not necessarily a minimiser)

The Newton method (cont.)

Remark

A necessary condition for convergence of Newton's method

- $\mathbf{x}^{(0)}$ should be sufficiently close to the minimiser $\mathbf{x}^*$

The **local convergence property** of the method

Remark

General convergence criterium for the Newton's method

If $f \in \mathcal{C}^2(\mathcal{R}^n)$ with stationary point $\mathbf{x}^*$

$\rightsquigarrow$ Positive definite Hessian $\mathbf{H}(\mathbf{x}^*)$

$\rightsquigarrow$ Lipschitz continuous components of $\mathbf{H}(\mathbf{x})$ in a neighbourhood of $\mathbf{x}^*$

Then, for $\mathbf{x}^{(0)}$ sufficiently close to $\mathbf{x}^*$, it converges (quadratically) to $\mathbf{x}^*$

The Newton method (cont.)

In spite of a simple implementation, the method is demanding for large n

- ⇝ It requires the analytic expression of the derivatives
- ⇝ The computation of both gradient and Hessian of f
- (Gradient and Hessian at each iteration)

Let alone that $\mathbf{x}^{(0)}$ has to be chosen near enough $\mathbf{x}^*$

Remark

A valid approach to design efficient and robust minimisation algorithms

- ⇝ Combine locally- with globally-convergent methods

Global convergence guarantees convergence to a stationary point

- (Not necessarily a global minimiser), for all $\mathbf{x}^{(0)} \in \mathcal{R}^n$

Line-search methods

Numerical optimisation

Line-search methods

Line-search or **descent methods** are iterative methods

We suppose that the objective function $f \in \mathcal{C}^2(\mathcal{R}^n)$ is bounded from below

For every step $k \geq 0$, let $\mathbf{x}^{(k+1)}$ be the next point of the minimising sequence

1. Point $\mathbf{x}^{(k+1)}$ is determined from current point $\mathbf{x}^k$ and a vector $\mathbf{d}^{(k)}$
2. Vector $\mathbf{d}^{(k)}$ itself depends on two terms
   - $\rightsquigarrow$ The gradient vector $\boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]$ of f
   - $\rightsquigarrow$ A **step-length** parameter $\alpha_k \in \mathcal{R}$

Line-search methods (cont.)

The algorithmic formulation of the general line-search/descent method

Pseudo-code

Let $\mathbf{x}^{(0)} \in \mathcal{R}^n$ be an initial minimiser

Find direction	$\mathbf{d}^{(k)} \in \mathcal{R}^n$
Compute step-length	$\alpha_k \in \mathcal{R}$
Set	$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$

Line-search methods (cont.)

Definition

Vector $\mathbf{d}^{(k)}$ must be a **descent direction**

A descent direction satisfies the following conditions

$$\begin{aligned} \mathbf{d}^{(k)} \nabla f\left[\mathbf{x}^{(k)}\right] < 0, \text{ if } \nabla f\left[\mathbf{x}^{(k)}\right] \neq \mathbf{0} \\ \mathbf{d}^{(k)} = \mathbf{0}, \text{ if } \nabla f\left[\mathbf{x}^{(k)}\right] = \mathbf{0} \end{aligned} \tag{12}$$

- $\rightsquigarrow$ $\nabla f\left[\mathbf{x}^{(k)}\right]$ gives the direction of max positive growth of f from $\mathbf{x}^{(k)}$
- $\rightsquigarrow$ $\mathbf{d}^{(k)} \nabla f\left[\mathbf{x}^{(k)}\right]$ is the directional derivative of f along $\mathbf{d}^{(k)}$

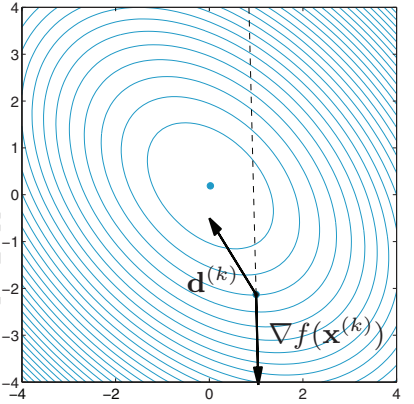
First condition ensures moves in a direction opposite to the gradient

- $\rightsquigarrow$ The iterates move towards a minimiser

Line-search methods (cont.)

Contour lines of function $f(\mathbf{x})$ and its gradient vector evaluated at $\mathbf{x}^{(k)}$

- $\mathbf{d}^{(k)}$ is a suitable descent direction



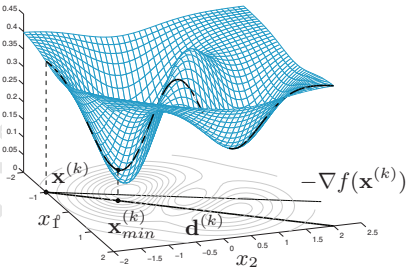
Optimal value $\alpha_k \in \mathcal{R}$ guarantees max variation of f along $\mathbf{d}^{(k)}$

- Once $\mathbf{d}^{(k)}$ is determined

Line-search methods (cont.)

α_k can be computed by solving a one-dimensional minimisation problem

- Minimise the restriction of $f(\mathbf{x})$ along $\mathbf{d}^{(k)}$
- $\mathbf{x}_{\min}^{(k)}$ is the minimiser along $\mathbf{d}^{(k)}$



The computation of α_k is quite involved (when f is not quadratic)

$\rightsquigarrow$ There are alternative techniques that approximate α_k well

Descent directions

Line-search methods

Descent directions

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \underbrace{\mathbf{d}^{(k)}}_{\text{descent direction}}$$

Newton's directions

$$\mathbf{d}^{(k)} = -\mathbf{H}^{-1}\left[\mathbf{x}^{(k)}\right]\boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right] \tag{13}$$

- Matrix $\mathbf{H}\left[\mathbf{x}^{(k)}\right]$ is the Hessian matrix at the k-th step
- Vector $\boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]$ is the gradient vector at the k-th step

Quasi-Newton directions

$$\mathbf{d}^{(k)} = -\mathbf{H}_k^{-1}\boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right] \tag{14}$$

- Matrix $\mathbf{H}_k$ is an approximation of the true Hessian $\mathbf{H}\left[\mathbf{x}^{(k)}\right]$
- It is used when second derivatives are heavy to compute

Descent directions (cont.)

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \underbrace{\mathbf{d}^{(k)}}_{\text{descent direction}}$$

Gradient directions

$$\mathbf{d}^{(k)} = -\boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right] \tag{15}$$

- These are quasi-Newton directions, with $\mathbf{H}_k = \mathbf{I}$, for all $k \geq 0$

Conjugate-gradient directions

$$\begin{aligned} \mathbf{d}^{(0)} &= -\boldsymbol{\nabla} f\left[\mathbf{x}^{(0)}\right] \\ \mathbf{d}^{(k+1)} &= -\boldsymbol{\nabla} f\left[\mathbf{x}^{(k+1)}\right] + \beta_k \mathbf{d}^{(k)}, \quad k \geq 0 \end{aligned} \tag{16}$$

- Coefficients β_k can be chosen according to different criteria

Descent directions (cont.)

For all $k \geq 0$, gradient directions are valid descent directions

$$\begin{aligned} \mathbf{d}^{(k)} \nabla f\left[\mathbf{x}^{(k)}\right] &< 0, \quad \text{if } \nabla f\left[\mathbf{x}^{(k)}\right] \neq \mathbf{0} \\ \mathbf{d}^{(k)} &= \mathbf{0}, \quad \text{if } \nabla f\left[\mathbf{x}^{(k)}\right] = \mathbf{0}, \end{aligned} \tag{17}$$

Conjugate-gradient directions are valid directions for some suitable β_k

Newton's and quasi-Newton's directions can also be valid directions

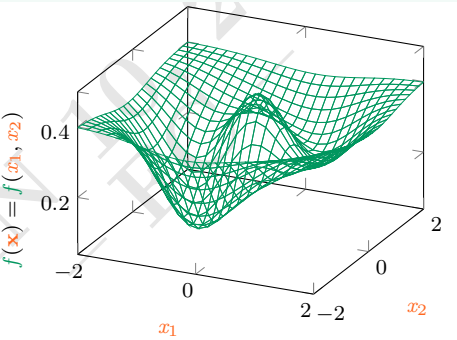
- $\mathbf{H}\left[\mathbf{x}^{(k)}\right]$ and $\mathbf{H}_k$ need be positive definite matrices

Descent directions (cont.)

Example

Consider the objective function $f(\mathbf{x})$

$$f(\mathbf{x}) = 2/5 - 1/10(5x_1^2 + 5x_2^2 + 3x_1x_2 - x_1 - 2x_2)e^{\left[-(x_1^2+x_2^2)\right]}$$



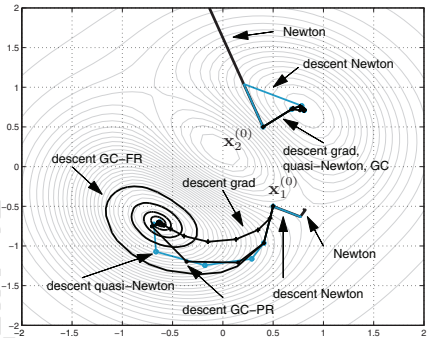
Two local minimisers, one local maximiser and two saddle points

We compare sequences $\{\mathbf{x}^{(k)}\}$ from Newton's and descent methods

- Various descent directions, from $\mathbf{x}_1^{(0)}$ and $\mathbf{x}_2^{(0)}$

Descent directions (cont.)

$\mathbf{x}_1^{(0)} = (0.5, -0.5)$



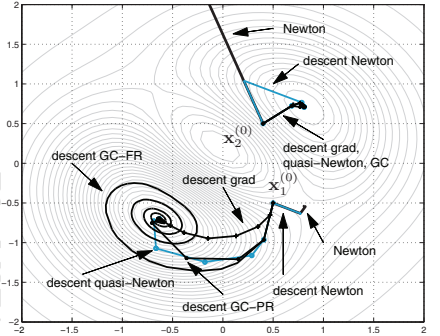
- Newton's method converges rapidly towards the saddle point
- Newton's directions take a first step identical to Newton's

⇝ Then collapse due to a non-positive definite matrix $\mathbf{H}_k$

- Others converge with different speeds into a local minimum
- Fastest convergence by quasi-Newton's directions

Descent directions (cont.)

$\mathbf{x}_2^{(0)} = (0.4, 0.5)$



- Newton's method diverges
- Newton's directions converge to a local minimum
- ⇝ Newton's method and directions share the same first direction
- All others also converge to the same local minimiser

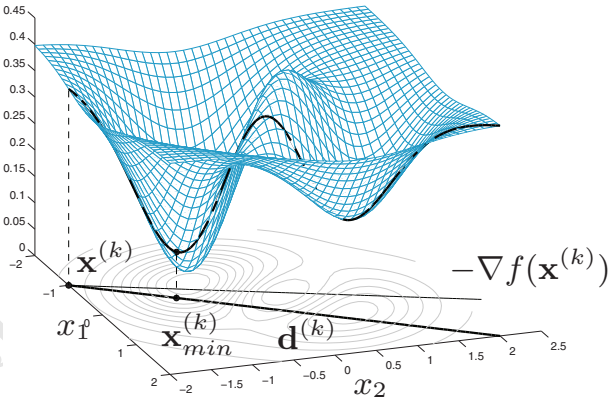
Step-length α_k

Line-search methods

Step-length α_k

We have let $\mathbf{d}^{(k)}$ be a descent direction, how to set the step-length α_k?

The new iterate $\mathbf{x}^{(k+1)}$ is (should be) the minimiser of f along $\mathbf{d}^{(k)}$



Step-length α_k (cont.)

The new iterate $\mathbf{x}^{(k+1)}$ should be the minimiser of f along $\mathbf{d}^{(k)}$

Choose α_k such that the minimisation is exact

$$\alpha_k = \arg\min_{\alpha \in \mathcal{R}} f\left[\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}\right]$$

or

$$f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] = \min_{\alpha \in \mathcal{R}} f\left[\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}\right] \tag{18}$$

Instead of f, we consider a second-order Taylor expansion of f around $\mathbf{x}^{(k)}$

$$f\left[\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}\right] = f\left[\mathbf{x}^{(k)}\right] + \alpha \mathbf{d}^{(k)} \nabla f\left[\mathbf{x}^{(k)}\right] + \frac{\alpha^2}{2} \mathbf{d}^{(k)^T} \mathbf{H}\left[\mathbf{x}^{(k)}\right] \mathbf{d}^{(k)} + o(||\alpha \mathbf{d}^{(k)}||^2) \tag{19}$$

Step-length α_k (cont.)

Consider the special case in which f is a quadratic function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{b} + c$$

- $\mathbf{A} \in \mathcal{R}^{n \times n}$ symmetric and positive definite
- $\mathbf{b} \in \mathcal{R}^n$
- $c \in \mathcal{R}$

The expansion is exact, the infinitesimal residual is null

$$f\left[\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)}\right] = f\left[\mathbf{x}^{(k)}\right] + \alpha\mathbf{d}^{(k)}\nabla f\left[\mathbf{x}^{(k)}\right] + \frac{\alpha^2}{2}\mathbf{d}^{(k)^T}\mathbf{H}\left[\mathbf{x}^{(k)}\right]\mathbf{d}^{(k)} + \cancel{o(\|\alpha\mathbf{d}^{(k)}\|^2)}$$

Step-length α_k (cont.)

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{b} + c$$

$$f\left[\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)}\right] = f\left[\mathbf{x}^{(k)}\right] + \alpha\mathbf{d}^{(k)}\nabla f\left[\mathbf{x}^{(k)}\right] + \frac{\alpha^2}{2}\mathbf{d}^{(k)\,T}\mathbf{H}\left[\mathbf{x}^{(k)}\right]\mathbf{d}^{(k)}$$

For every $k \geq 0$, we have

$$f\left[\mathbf{x}^{(k)}\right] = \frac{1}{2}\mathbf{x}^{(k)\,T}\mathbf{A}\mathbf{x}^{(k)} - \mathbf{x}^{(k)\,T}\mathbf{b} + c$$
$$\nabla f\left[\mathbf{x}^{(k)}\right] = \mathbf{A}\mathbf{x}^{(k)} - \mathbf{b} = -\mathbf{r}^{(k)}$$
$$\nabla^2 f\left[\mathbf{x}^{(k)}\right] = \mathbf{A} = \mathbf{H}\left[\mathbf{x}^{(k)}\right]$$

Differentiate $f\left[\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)}\right] = f\left[\mathbf{x}^{(k)}\right] + \alpha\mathbf{d}^{(k)}\nabla f\left[\mathbf{x}^{(k)}\right] + \frac{\alpha^2}{2}\mathbf{d}^{(k)\,T}\mathbf{H}\left[\mathbf{x}^{(k)}\right]\mathbf{d}^{(k)}$

- with respect to α

Set the derivative to be equal to zero to get $\min_{\alpha\in\mathcal{R}} f[\mathbf{x}^{(k)} + \alpha\mathbf{d}^{(k)}]$

$$\frac{\mathrm{d}}{\mathrm{d}\alpha_k} f\left[\mathbf{x}^{(k)} + \alpha_k\mathbf{x}^{(k)}\right] = -\mathbf{d}^{(k)\,T}\mathbf{r}^{(k)} + \alpha_k\mathbf{d}^{(k)}\mathbf{A}\mathbf{d}^{(k)} = 0$$

$$\rightsquigarrow \alpha_k = \frac{\mathbf{d}^{(k)\,T}\mathbf{r}^{(k)}}{\mathbf{d}^{(k)\,T}\mathbf{A}\mathbf{d}^{(k)}} \tag{20}$$

Step-length α_k (cont.)

Let $\mathbf{d}^{(k)}$ be gradient directions, that is $\mathbf{d}^{(k)} = -\nabla f(\mathbf{x}^{(k)}) = \mathbf{r}^{(k)}$

⇝ We get the gradient method for solving linear systems

Let $\mathbf{d}^{(k)}$ be conjugate-gradient directions, $\mathbf{d}^{(k+1)} = -\nabla f\left[\mathbf{x}^{(k+1)}\right] + \beta_k \mathbf{d}^{(k)}$

⇝ The conjugate-gradient method for linear systems

⇝ Set,

$$\beta_k = \frac{\left[\mathbf{A}\mathbf{d}^{(k)}\right]^T \mathbf{r}^{(k+1)}}{\mathbf{d}^{(k)\,T} \mathbf{A}\mathbf{d}^{(k)}} \tag{21}$$

Step-length α_k (cont.)

Consider the general case in which f is not a quadratic function of $\mathbf{x}$

The computation of the optimal α_k requires an iterative method

- $\rightsquigarrow$ Numerical solution of minimisation along $\mathbf{d}^{(k)}$
- $\rightsquigarrow$ Demanding and often not worth it
- $\rightsquigarrow$ Stick with an approximation of α_k

The question remains, how to pick a good approximated value of α_k?

Impose a condition to the new iterate $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$

$$\rightsquigarrow \quad f\left[\mathbf{x}^{(k+1)}\right] < f\left[\mathbf{x}^{(k)}\right] \tag{22}$$

Step-length α_k (cont.)

Example

A natural strategy for setting α_k values

1. Initially assign a large α_k
2. Then, reduce it iteratively

$\rightsquigarrow$ Until, $f\left[\mathbf{x}^{(k+1)}\right] < f\left[\mathbf{x}^{(k)}\right]$ is satisfied

The strategy does not guarantee a $\left\{\mathbf{x}^{(k)}\right\}$ that converges to $\mathbf{x}^*$

- Steps can be too long (go beyond the minimum)
- Steps can be too short (get infinitesimal)

Step-length α_k (cont.)

There exist alternative (better/reliable) criteria for $\alpha_k > 0$

$\rightsquigarrow$ **Wolfe's conditions**

Definition

Let α_k be the step-length determined according to some method

Conditions for α_k to be accepted

$$\begin{aligned} f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] &\leq f\left[\mathbf{x}^{(k)}\right] + \sigma \alpha_k {\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right] \\ {\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] &\geq \delta {\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right] \end{aligned} \tag{23}$$

The two additional parameters, constants σ and δ

- $0 < \sigma < \delta < 1$

$\mathbf{d}^{(k)} \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]$ is the directional derivative of f along direction $\mathbf{d}^{(k)}$

Step-length α_k (cont.)

$$f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] \leq f\left[\mathbf{x}^{(k)}\right] + \sigma \alpha_k {\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]$$
$${\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] \geq \delta {\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]$$

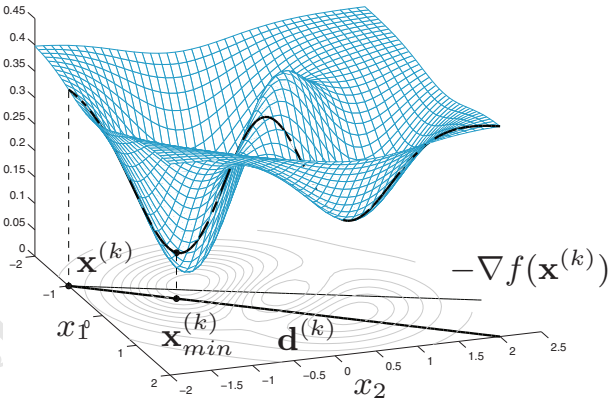
The first condition (**Armijo's rule**) inhibits too small variations of f

- With respect to step-length and directional derivative

Changes in f need be proportional to step-length and directional derivative

Step-length α_k (cont.)

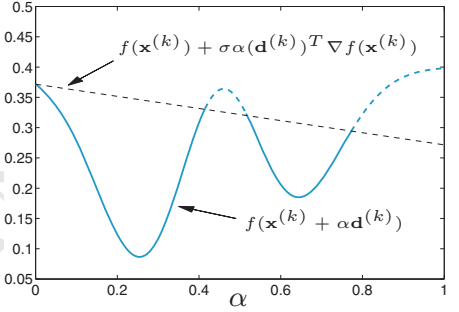
$$f(\mathbf{x}) = 2/5 - 1/10(5x_1^2 + 5x_2^2 + 3x_1x_2 - x_1 - 2x_2)e^{\left[-(x_1^2+x_2^2)\right]}$$



Step-length α_k (cont.)

The terms in the first of the two Wolfe's conditions, for $\sigma = 0.2$

$$f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] \leq f\left[\mathbf{x}^{(k)}\right] + \sigma \alpha_k {\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]$$



Condition is satisfied for α corresponding to the continuous line

Step-length α_k (cont.)

$$f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] \leq f\left[\mathbf{x}^{(k)}\right] + \sigma \alpha_k {\mathbf{d}^{(k)}}^T \nabla f\left[\mathbf{x}^{(k)}\right]$$
$${\mathbf{d}^{(k)}}^T \nabla f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] \geq \delta {\mathbf{d}^{(k)}}^T \nabla f\left[\mathbf{x}^{(k)}\right]$$

The second condition states that the directional derivative of f at the new point, $\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$, should be δ times larger than it was at point $\mathbf{x}^{(k)}$

- A new point $\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$ is a valid candidate if f at such point decreases less than it does at $\mathbf{x}^{(k)}$ (which is closer to a minimiser)

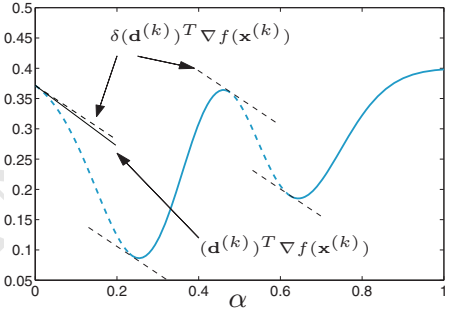
The second condition prevents steps whose length would be too small

- Happens where f has a largely negative directional derivative

Step-length α_k (cont.)

Lines with slope $\delta \mathbf{d}^{(k)^T} \nabla f\left[\mathbf{x}^{(k)}\right]$ in second condition, $\delta = 0.9$

$$\mathbf{d}^{(k)^T} \nabla f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] \geq \delta \mathbf{d}^{(k)^T} \nabla f\left[\mathbf{x}^{(k)}\right]$$



Condition is satisfied for α corresponding to the continuous line

Step-length α_k (cont.)

It can be seen that Wolfe's conditions are jointly satisfied

$$0.23 \leq \alpha \leq 0.41$$
$$0.62 \leq \alpha \leq 0.77$$

Values of $\alpha \in [0.62, 0.77]$ are far from the minimiser of f along $\mathbf{d}^{(k)}$

- Also α where the directional derivative is large are accepted

Step-length α_k (cont.)

Definition

Wolfe's strong conditions

$$\begin{aligned} f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] &\leq f\left[\mathbf{x}^{(k)}\right] + \sigma \alpha_k {\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right] \\ \|{\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right]\| &\leq -\delta {\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right] \end{aligned} \tag{24}$$

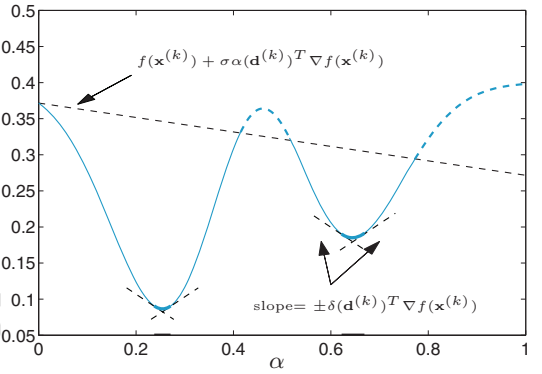
This conditions are more restrictive

- The first condition is unchanged
- The second one inhibits f from large variations about $\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$

Step-length α_k (cont.)

Acceptable α must belong to small intervals around the minimisers

$\rightsquigarrow$ Thick continuous arcs



- $\sigma = 0.2$
- $\delta = 0.9$

Step-length α_k (cont.)

Remark

Suppose that $f \in \mathcal{C}^2(\mathcal{R}^n)$ is bounded from below in $\{\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}, \alpha > 0\}$

- Let $\mathbf{d}^{(k)}$ be a descent direction at $\mathbf{x}^{(k)}$

It can be shown that, for all σ and δ such that $0 < \sigma < \delta < 1$, there exist non-empty intervals of α_k that satisfy Wolfe's weak and strong conditions

In practice[2], σ is usually chosen to be very small (e.g., $\sigma = 10^{-4}$)

Typical values for δ

- $\delta = 0.9$ for Newton, quasi-Newton and gradient directions
- $\delta = 0.1$ for conjugate-gradient directions

[2] J. Nocedal and S. Wrigth (2006): *Numerical optimization*.

Step-length α_k (cont.)

A strategy for step-lengths α_k that satisfy Wolfe's conditions

Backtracking

1. Start with $\alpha = 1$
2. Then reduce it by a given factor ρ ($\rho \in [0.1, 0.5)$)

$\rightsquigarrow$ Until, the first weak condition is satisfied

For $\mathbf{x}^{(k)}$ and a direction $\mathbf{d}^{(k)}$, for $\sigma \in (0, 1)$ and $\rho \in [0.1, 0.5)$

Pseudo-code

Set $\alpha = 1$
  while $f\left[\mathbf{x}^{(k)} + \alpha \mathbf{d}^{(k)}\right] > f\left[\mathbf{x}^{(k)}\right] + \sigma\alpha\mathbf{d}^{(k)}\nabla f\left[\mathbf{x}^{(k)}\right]$
    $\alpha = \rho\alpha$
  end
Set $\alpha_k = \alpha$

Second condition is never checked, as step-lengths are not small

Step-length α_k (cont.)


```
function [x,alpha_k]=bTrack(fun,x_k,g_k,d_k,varargin)
%BTRACK Backtracking with line search
% [X,ALPHA_K]=BTRACK(FUN,X_K,G_K,D_K)
%  Descent method: x_{k+1}=x_k+alpha_k*d_k
%  Backtracking: alpha_k, fixed sigma and rho
%  sigma = 1e-4, rho = 0.25
%
% [X,ALPHA_K]=BTRACK(FUN,X_K,G_K,D_K,SIGMA,RHO)
%  Sigma and rho can be inputed
%  sigma in (1e-4,0.1), rho in (0.1,0.5)
%
% FUN is the function handle of the objective function
% X_K is element x_k, G_K is the gradient, D_K is d_k

if nargin == 4
 sigma = 1.0e-4; rho = 1/4;
else
 sigma = varargin{1}; rho = varargin{2};
end

minAlpha = 1.0e-5;                                % Smallest steplength
alpha_k = 1.0; f_k = fun(x_k);

k = 0; x = x_k + alpha_k*d_k;
while fun(x) > f_k+sigma*alpha_k*g_k'*d_k & alpha_k > minAlpha
 alpha_k = alpha_k*rho;
 x = x_k + alpha_k*d_k; k = k+1;
end
```

Step-length α_k (cont.)

The descent method with various descent directions

- α_k is determined by backtracking

```
%DSCENT Descent method of minimisation
%[X,ERR,ITER]=DSCENT(FUN,GRAD_FUN,X_0,TOL,KMAX,TYP,HESS_FUN)
% Approximates the minimiser of FUN using descent directions
%  Newton (TYP=1), BFGS (TYP=2), GRADIENT (TYP=3), and the
%  CONJUGATE-GRADIENT method with
%   beta_k by Fletcher and Reeves (TYP=41)
%   beta_k by Polak and Ribiere   (TYP=42)
%   beta_k by Hestenes and Stiefel(TYP=43)
%
% Step length is calculated using backtracking (bTrack.m)
%
% FUN, GRAD_FUN and HESS_FUN (TYP=1 only) are function handles for the
%  objective, gradient and Hessian matrix
%  With TYP=2, HESS_FUN approximates the exact Hessian at X_0
%
% TOL is the stop check tolerance
% KMAX is the maximum number of iteration
```

```
function [x,err,iter]=dScent(fun,grad_fun,x_0,tol,kmax,typ,varargin)
if nargin>6; if typ==1; hess=varargin{1};
   elseif typ==2; H=varargin{1}; end; end

err=tol+1; k=0; xk=x0(:); gk=grad(xk); dk=-gk; eps2=sqrt(eps);

while err>tol & k<kmax
 if typ==1; H = hess_fun(xk); dk = -H\gk;                          % Newton
 elseif typ==2; dk = -H\gk;                                          % BFGS
 elseif typ==3; dk = -gk;                                         % Gradient
 end
 [xk1,alphak]=bTrack(fun,xk,gk,dk);
 gk1=grad_fun(xk1);
  if typ==2                                                  % BFGS update
   yk = gk1-gk; sk = xk1-xk; yks = yk'*sk;
   if yks > eps2*norm(sk)*norm(yk)
    Hs=H*sk; H=H+(yk*yk')/yks-(Hs*Hs')/(sk'*Hs);
   end
  elseif typ>=40                                              % CG upgrade
   if typ==41; betak=(gk1'*gk1)/(gk'*gk);                             % FR
   elseif typ==42; betak=(gk1'*(gk1-gk))/(gk'*gk);                    % PR
   elseif typ==43; betak=(gk1'*(gk1-gk))/(dk'*(gk1-gk));              % HS
   end
   dk = -gk1 + betak*dk;
  end
  xk = xk1; gk = gk1; k = 1 + k; xkt = xk1;
 for i=1:length(xk1); xkt(i) = max([abs(xk1(i)),1]); end
 err = norm((gk1.*xkt)/max([abs(fun(xk1)),1]),Inf);
end
x = xk; iter = k;
if (k==kmax & err>tol); disp('[KMAX]'); end
```

Descent method with Newton's directions

Line-search methods

Descent method with Newton's directions

Let us consider a descent method with Newton's directions

⇝ Newton directions

$$\mathbf{d}^{(k)} = -\mathbf{H}^{-1}\left[\mathbf{x}^{(k)}\right]\boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]$$

Let step-lengths α_k satisfy Wolfe's conditions

⇝ Wolfe step lengths α_k

$$f(\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}) \leq f\left[\mathbf{x}^{(k)}\right] + \sigma\alpha_k {\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]$$
$${\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] \geq \delta {\mathbf{d}^{(k)}}^T \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]$$

Let $f \in \mathcal{C}^2(\mathcal{R}^n)$ bounded from below

Pseudo-code

Find direction $\mathbf{d}^{(k)} \in \mathcal{R}^n$

Compute step $\alpha_k \in \mathcal{R}$

Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$

Descent method with Newton's directions (cont.)

Suppose that the Hessian $\mathbf{H}\left[\mathbf{x}^{(k)}\right]$ is symmetric, for all iterations $k \geq 0$

$\rightsquigarrow$ (from the assumption on f)

Suppose that $\mathbf{H}\left[\mathbf{x}^{(k)}\right]$ is also positive definite (no uphill moves)

Let $\mathbf{B}_k = \mathbf{H}\left[\mathbf{x}^{(k)}\right]$

Suppose that $\exists M > 0 : K(\mathbf{B}_k) = ||\mathbf{B}_k||_2 ||\mathbf{B}_k^{-1}||_2 \leq M$, for all $k \geq 0$

- $K(\mathbf{B}_k)$ is the (one) **spectral condition number** of $\mathbf{B}_k$
- Uniform upper bound on the condition number

Then, Newton's sequence $\left\{\mathbf{x}^{(k)}\right\}$ converges to a stationary point $\mathbf{x}^*$

$\rightsquigarrow$ By letting $\alpha_k = 1$ for $k \geq \overline{k}$, the converge is quadratic

Descent method with Newton's directions (cont.)

Definition

Let $\mathbf{A} \in \mathcal{R}^{n \times n}$ be a matrix

Consider the problem of finding a scalar λ (complex or real) and a non-null vector $\mathbf{x} \in \mathcal{C}^n$ such that the following equation is satisfied

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$$

Any λ that satisfy this equation is an **eigenvalue** of $\mathbf{A}$

- $\mathbf{x}$ is the corresponding **eigenvector**

The **spectral condition number** of $\mathbf{A}$ is the quantity $K(\mathbf{A})$

$$K(\mathbf{A}) = \frac{\lambda_{\max}}{\lambda_{\min}}$$

Descent method with Newton's directions (cont.)

Remark

If Hessians are positive definite, $\mathbf{x}^*$ cannot be a maximiser or saddle point

- ⇝ The stationary point must necessarily be a minimiser

It can happen that $\mathbf{H}\left[\mathbf{x}^{(k)}\right]$ is not positive definite for some point $\mathbf{x}^{(k)}$

- ⇝ Direction $\mathbf{d}^{(k)}$ may not be a valid descent direction
- ⇝ Wolfe's conditions might become meaningless

However, Hessians can transformed and forced to be positive definite

$$\mathbf{B}_k = \mathbf{H}\left[\mathbf{x}^{(k)}\right] + \mathbf{E}_k$$

- ⇝ $\mathbf{E}_k$ is some suitable matrix (it can either be diagonal or full)
- ⇝ $\mathbf{E}_k$ is such that $\mathbf{d}^{(k)} = -\mathbf{B}_k^{-1}\boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]$ is a descent direction

Descent method with quasi-Newton's directions

Line-search methods

Descent method with quasi-Newton

Let us consider a descent method based on quasi-Newton directions

- Quasi-Newton directions

$$\mathbf{d}^{(k)} = -\mathbf{H}_k^{-1} \nabla f\left[\mathbf{x}^{(k)}\right]$$

$\rightsquigarrow$ $\mathbf{H}_k$ approximates the true Hessian $\mathbf{H}\left[\mathbf{x}^{(k)}\right]$

Let step-lengths α_k satisfy Wolfe's conditions

$\rightsquigarrow$ Wolfe step lengths α_k

$$f(\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}) \leq f\left[\mathbf{x}^{(k)}\right] + \sigma \alpha_k {\mathbf{d}^{(k)}}^T \nabla f\left[\mathbf{x}^{(k)}\right]$$
$${\mathbf{d}^{(k)}}^T \nabla f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] \geq \delta {\mathbf{d}^{(k)}}^T \nabla f\left[\mathbf{x}^{(k)}\right]$$

As always, we also let $f \in \mathcal{C}^2(\mathcal{R}^n)$ bounded from below

Descent method with quasi-Newton's directions (cont.)

Suppose that we are given a symmetric and positive definite matrix $\mathbf{H}_0$

⇝ How do we build matrices $\mathbf{H}_k$?

There exists a popular technique used for solving nonlinear systems

⇝ The recursive **Broyden's rank-one update**

Descent method with quasi-Newton's directions (cont.)

We know that matrices $\mathbf{H}_k$ are required to satisfy certain validity conditions

- They must satisfy the secant condition

$$\mathbf{H}_{k+1}\left[\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right] = \boldsymbol{\nabla} f\left[\mathbf{x}^{(k+1)}\right] - \boldsymbol{\nabla} f\left[\mathbf{x}^{(k)}\right]$$

- They must be symmetric, as $\mathbf{H}(\mathbf{x})$
- They must be positive definite, $\mathbf{d}^{(k)}$ are descent
- They must satisfy

$$\lim_{k\to\infty} \frac{\left|\left|\left[\mathbf{H}_k - \mathbf{H}(\mathbf{x}^*)\right]\mathbf{d}^{(k)}\right|\right|}{\left|\left|\mathbf{d}^{(k)}\right|\right|} = 0$$

  This ensures that $\mathbf{H}_k$ is a good approximation of $\mathbf{H}\left[\mathbf{x}^*\right]$ along the descent direction $\mathbf{d}^{(k)}$ and guarantees a super-linear rate of convergence

Descent method with quasi-Newton's directions (cont.)

Definition

A strategy by Broyden, Fletcher, Goldfarb and Shanno (**BFGS**)

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{y}^{(k)}\mathbf{y}^{(k)\,T}}{\mathbf{x}^{(k)\,T}\mathbf{s}^{(k)}} - \frac{\mathbf{H}_k\mathbf{s}^{(k)}\mathbf{s}^{(k)\,T}\mathbf{H}_k^T}{\mathbf{s}^{(k)\,T}\mathbf{H}_k\mathbf{s}^{(k)}} \tag{25}$$

- $\mathbf{s}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$
- $\mathbf{y}^k = \nabla f\left[\mathbf{x}^{(k+1)}\right] - \nabla f\left[\mathbf{x}^{(k)}\right]$

Matrices $\mathbf{H}_{k+1}$ are symmetric and positive definite under condition

$$\mathbf{y}^{(k)\,T}\mathbf{s}^{(s)} > 0$$

It is satisfied when step lengths α_k are either weak or strong Wolfe

Descent method with quasi-Newton's directions (cont.)

The BFGS is a descent method, with quasi-Newton $\mathbf{d}^{(k)}$s and Wolfe's α_ks

$$\rightsquigarrow \quad \mathbf{d}^{(k)} = -\mathbf{H}_k^{-1} \nabla f\left[\mathbf{x}^{(k)}\right]$$

$$\rightsquigarrow \quad f(\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}) \leq f\left[\mathbf{x}^{(k)}\right] + \sigma \alpha_k \mathbf{d}^{(k)\,T} \nabla f\left[\mathbf{x}^{(k)}\right]$$
$$\mathbf{d}^{(k)\,T} \nabla f\left[\mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}\right] \geq \delta \mathbf{d}^{(k)\,T} \nabla f\left[\mathbf{x}^{(k)}\right]$$

Pseudo-code

Let $\mathbf{x}^{(0)}$ be an initial solution

Find direction $\mathbf{d}^{(k)} \in \mathbb{R}^n$

Compute step length $\alpha_k \in \mathbb{R}$

Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$

Descent method with quasi-Newton's directions (cont.)

Pseudo-code

Let $\mathbf{x}^{(0)}$ be an initial solution

Let $\mathbf{H}_0 \in \mathcal{R}^{n \times n}$ be a suitable symmetric and positive definite matrix

$\rightsquigarrow$ $\mathbf{H}_0 \in \mathcal{R}^{n \times n}$ approximates $\mathbf{H}\left[\mathbf{x}^{(0)}\right]$

Solve $\mathbf{H}_k \mathbf{d}^{(k)} = -\nabla f\left[\mathbf{x}^{(k)}\right]$

Compute α_k that satisfies Wolfe's conditions

Set

$$\begin{aligned} \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)} \\ \mathbf{s}^{(k)} &= \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \\ \mathbf{y}^{(k)} &= \nabla f\left[\mathbf{x}^{(k+1)}\right] - \nabla f\left[\mathbf{x}^{(k)}\right] \end{aligned}$$

Compute $\mathbf{H}_{k+1} = \mathbf{H}_k + \dfrac{\mathbf{y}^{(k)} \mathbf{y}^{(k)^T}}{\mathbf{x}^{(k)^T} \mathbf{s}^{(k)}} - \dfrac{\mathbf{H}_k \mathbf{s}^{(k)} \mathbf{s}^{(k)^T} \mathbf{H}_k^T}{\mathbf{s}^{(k)^T} \mathbf{H}_k \mathbf{s}^{(k)}}$

Descent method with quasi-Newton's directions (cont.)

Remark

The cost of calculating $\mathbf{d}^{(k)}$ is $\mathcal{O}(n^3)$, at every iteration step $k \geq 0$

- Can be reduced to $\mathcal{O}(n^2)$ by using recursive QR on $\mathbf{H}_k$

Experimental evidence that setting $\mathbf{H}_0 = \mathbf{I}$ gives faster convergence to $\mathbf{x}^*$

Descent method with quasi-Newton's directions (cont.)

Example

Rosenbrock's function

$$f(\mathbf{x}) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

Let $\varepsilon = 10^{-6}$ be the tolerance

```
x_0 = [+1.2; -1.0];

fun = @(x) (1-x(1))^2 + 100*(x(2)-x(1)^2)^2;

options = optimset ('LargeScale','off');              % Switches to BFGS
[xstar,fval,exitflag,output] = fminunc(fun,x_0,options)
```

Convergence after 24 iterations and 93 function evaluations

Descent method with quasi-Newton's directions (cont.)

We did not input an expression for evaluating the gradient

- It was, silently, approximated
- (finite difference methods)

We can separately define and input the analytical gradient

```
1 x_0 = [+1.2; -1.0];
2
3 fun = @(x) (1-x(1))^2 + 100*(x(2)-x(1)^2)^2;
4 grad_fun = @(x)[-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1)); ...
5                 +200*(x(2)-x(1)^2)];
6
7 options = optimset('LargeScale','off','GradObj','on');
8 [xstar,fval,exitflag,output] = fminunc({fun,grad_fun},x_0,options)
```

Convergence after 25 iterations and 32 function evaluations

■

Descent method with quasi-Newton's directions (cont.)

Remark

In Octave, BFGS is implemented by the M-command `bfgsmin`

- M-command `fminunc` implements a different method
- (A trust-region method)

Descent method with gradient and conjugate-gradient directions

Line-search methods

Gradient and conjugate-gradient directions

Consider the general descent method

Pseudo-code

Find direction $\mathbf{d}^{(k)} \in \mathcal{R}^n$

Compute step $\alpha_k \in \mathcal{R}$

Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}^{(k)}$

Consider gradient (descent) directions

$$\mathbf{d}^{(k)} = -\boldsymbol{\nabla} f[\mathbf{x}^{(k)}]$$

If $f \in \mathcal{C}^2(\mathcal{R}^n)$ is bounded from below and step lengths α_k are Wolfe

$\rightsquigarrow$ This method converges (linearly) to a stationary point

Consider conjugate gradient directions

$$\mathbf{d}^{(0)} = -\boldsymbol{\nabla} f[\mathbf{x}^{(0)}]$$
$$\mathbf{d}^{(k+1)} = -\boldsymbol{\nabla} f[\mathbf{x}^{(k+1)}] - \beta_k \mathbf{d}^{(k)}, \quad k \geq 0$$

There are several options for setting β_k

Gradient or conjugate-gradient directions (cont.)

⇝ **Fletcher-Reeves**

$$\beta_k^{FR} = -\frac{||\nabla f\left[\mathbf{x}^{(k)}\right]||^2}{||\nabla f\left[\mathbf{x}^{(k-1)}\right]||^2} \tag{26}$$

⇝ **Polak-Ribière (-Polyak)**

$$\beta_k^{PR} = -\frac{\nabla f\left[\mathbf{x}^{(k)}\right]^T\left\{\nabla f\left[\mathbf{x}^{(k)}\right] - \nabla f\left[\mathbf{x}^{(k-1)}\right]\right\}}{||\nabla f\left[\mathbf{x}^{(k-1)}\right]||^2} \tag{27}$$

⇝ **Hestenes-Stiefel**

$$\beta_k^{HS} = -\frac{\nabla f\left[\mathbf{x}^{(k)}\right]^T\left\{\nabla f\left[\mathbf{x}^{(k)}\right]^T - \nabla f\left[\mathbf{x}^{(k-1)}\right]\right\}}{\mathbf{d}^{(k-1)^T}\left\{\nabla f\left[\mathbf{x}^{(k)}\right] - \nabla f\left[\mathbf{x}^{(k-1)}\right]\right\}} \tag{28}$$

Gradient and conjugate-gradient directions (cont.)

Remark

Suppose true the condition that f is quadratic and strictly convex

Then, all the aforementioned options are equivalent

$$\rightsquigarrow \quad \beta_k = \frac{[\mathbf{A}\mathbf{d}^{(k)}]^T \mathbf{r}^{(k+1)}}{\mathbf{d}^{(k)\,T} \mathbf{A}\mathbf{d}^{(k)}}$$

We used $\mathbf{A} = \mathbf{H}[\mathbf{x}^{(k)}]$ and $\mathbf{r}^{(k)} = -\nabla f[\mathbf{x}^{(k)}]$

Trust-region methods

Numerical optimisation

Trust-region methods

Line search methods are designed so to first set the descent direction $\mathbf{d}^{(k)}$

- Then, they determine the step-length α_k

These steps are recursively performed at each k-th step

Trust-region methods simultaneously set direction and step length

This is done by building a ball of radius δ_k centred at $\mathbf{x}^{(k)}$

- The ball is the **trust region**, at iteration k

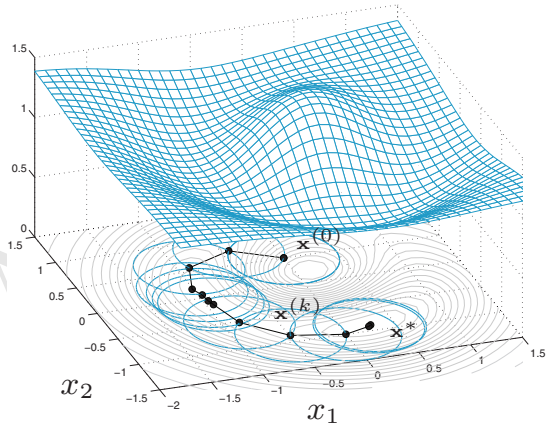
Within the ball, a quadratic approximation $\tilde{f}_k(\mathbf{x})$ of $f(\mathbf{x})$ is computed

- The new $\mathbf{x}^{(k+1)}$ is the minimiser of $\tilde{f}_k$ in the trust region

Trust-region methods (cont.)

Example

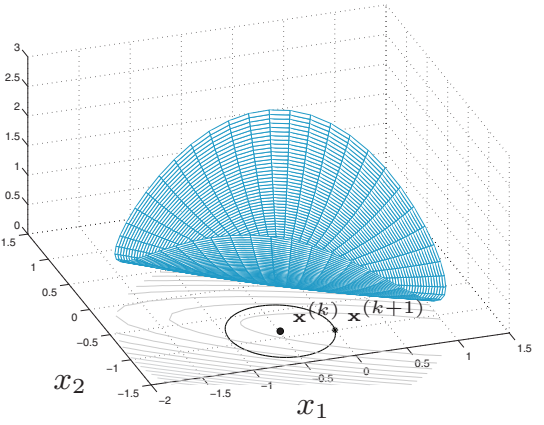
Convergence history for function f and quadratic approximation $\tilde{f}_k$ $(k = 8)$



Trust-region methods (cont.)

Example

Convergence history for function f and quadratic approximation $\tilde{f}_k$ $(k = 8)$



Trust-region methods (cont.)

To compute the quadratic approximation $\tilde{f}_k$, start with a trust radius $\delta_k > 0$

1. Determine a second-order Taylor expansion of function $f(\mathbf{x})$ at $\mathbf{x}^{(k)}$

$$\tilde{f}_k(\mathbf{s}) = f\left[\mathbf{x}^{(k)}\right] + \mathbf{s}\nabla f\left[\mathbf{x}^{(k)}\right] + \frac{1}{2}\mathbf{s}^T\mathbf{H}_k\mathbf{s}, \quad \forall \mathbf{s} \in \mathcal{R}^n \tag{29}$$

   $\mathbf{H}_k$ is either the Hessian of $f(\mathbf{x})$ at $\mathbf{x}^{(k)}$ or suitable approximation

2. We compute the solution $\mathbf{s}^{(k)}$ to be the minimiser of the function $\tilde{f}_k$

$$\mathbf{s}^{(k)} = \underset{\mathbf{s}\in\mathcal{R}^n : ||\mathbf{s}|| \leq \delta_k}{\arg\min} \tilde{f}_k(\mathbf{s}) \tag{30}$$

⇝ At this stage, we also compute the quantity

$$\rho_k = \frac{f\left[\mathbf{x}^{(k)} + \mathbf{s}^{(k)}\right] - f\left[\mathbf{x}^{(k)}\right]}{\tilde{f}_k\left[\mathbf{s}^{(k)}\right] - \tilde{f}_k(\mathbf{0})} \tag{31}$$

Trust-region methods (cont.)

A comparison between the variation of function f and the variation of $\tilde{f}_k$

$$\rho_k = \frac{f\left[\mathbf{x}^{(k)} + \mathbf{s}^{(k)}\right] - f\left[\mathbf{x}^{(k)}\right]}{\tilde{f}_k\left[\mathbf{s}^{(k)}\right] - \tilde{f}_k(\mathbf{0})}$$

$\rightsquigarrow$ From point $\mathbf{x}^{(k)}$ to point $\mathbf{x}^{(k)} + \mathbf{s}^{(k)}$

If ρ_k is approximately one, then approximation $\tilde{f}_k$ is considered to be good

Trust-region methods (cont.)

$$\rho_k = \frac{f\left[\mathbf{x}^{(k)} + \mathbf{s}^{(k)}\right] - f\left[\mathbf{x}^{(k)}\right]}{\tilde{f}_k\left[\mathbf{s}^{(k)}\right] - \tilde{f}_k(\mathbf{0})}$$

If ρ_k is approximately one, we accept $\mathbf{s}^{(k)}$ and move on to next iteration

- ⇝ We set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{s}^{(k)}$
  - (however, if the minimiser of $\tilde{f}_k$ lies on the boundary of the trust region, we enlarge the region before proceeding to next iteration)

If ρ_k is either negative or positive (and much smaller than one)

- ⇝ We reduce the ball's size and calculate a new $\mathbf{s}^{(k)}$

$$\mathbf{s}^{(k)} = \underset{\mathbf{s}\in\mathcal{R}^n : ||\mathbf{s}|| \leq \delta_k}{\arg\min}\ \tilde{f}_k(\mathbf{s})$$

If ρ_k is much larger than one, we accept $\mathbf{s}^{(k)}$ and keep the trust region

- ⇝ Then we move to the next iteration

Trust-region methods (cont.)

Remark

Consider the situation in which second derivatives of f are available

We could set $\mathbf{H}_k$ to be equal to the Hessian

Otherwise, $\mathbf{H}_k$ can be built recursively

- (Also a variant can be used, like BFGS, if not PD)

Trust-region methods (cont.)

Let $\mathbf{H}_k$ be symmetric positive definite and let $\left|\left|\mathbf{H}_k^{-1}\nabla f\left[\mathbf{x}^{(k)}\right]\right|\right| \leq \delta_k$

- Then, $\mathbf{s}^{(k)} = \mathbf{H}_k^{-1}\nabla f\left[\mathbf{x}^{(k)}\right]$ is a minimiser
- It falls within the trust region

Otherwise, the minimiser of $\tilde{f}_k$ lies outside the trust region

⇝ We must solve the minimisation of $\tilde{f}_k$

- Constrained to the δ_k-ball at $\mathbf{x}^{(k)}$

$$\min_{\mathbf{s}\in\mathcal{R}^n:||\mathbf{s}||=\delta_k} \tilde{f}_k(\mathbf{s}) \tag{32}$$

This is a constrained optimisation problem

⇝ We can use the Lagrange multipliers

⇝ We shall discuss them later on

Trust-region methods (cont.)

At each iteration k, we search for the minimiser of the Lagrangian function

$$\mathcal{L}(\mathbf{s}, \lambda) = \tilde{f}_k(\mathbf{s}) + 1/2\lambda(\mathbf{s}^T\mathbf{s} - \delta_k)$$

This is optimised with respect to both $\mathbf{s}$ and the regularisation term λ

We search for a vector $\mathbf{s}^{(k)}$ and a scalar $\lambda^{(k)} > 0$ satisfying the system

$$\begin{aligned} \left[\mathbf{H}_k + \lambda^{(k)}\mathbf{I}\right]\mathbf{s}^{(k)} &= -\nabla f\left[\mathbf{x}^{(k)}\right] \\ \left[\mathbf{H}_k + \lambda^{(k)}\mathbf{I}\right] &\text{ is PSD} \\ \left|\left|\mathbf{s}^{(k)}\right|\right| - \delta_k &= 0 \end{aligned} \tag{33}$$

Trust-region methods (cont.)

From $\left[\mathbf{H}_k + \lambda^{(k)}\mathbf{I}\right]\mathbf{s}^{(k)} = -\nabla f\left[\mathbf{x}^{(k)}\right]$, we compute $\mathbf{s}^{(k)} = \mathbf{s}^{(k)}\left[\lambda^{(k)}\right]$

We substitute it in $||\mathbf{s}^{(k)}|| - \delta_k = 0$

$$\rightsquigarrow \varphi\left[\lambda^{(k)}\right] = \frac{1}{||\mathbf{s}^{(k)}\left[\lambda^{(k)}\right]||} - \frac{1}{\delta_k} = 0$$

The non-linear equation in $\lambda^{(k)}$ is equivalent to system (33)

- It can be solved using Newton's method

Trust-region methods (cont.)

For some given initial λ_0, set $\mathbf{g}^{(k)} = \nabla f\left[\mathbf{x}^{(k)}\right]$

Pseudo-code

For $l = 0, 1, \ldots$ (typically, less than 5 iterations are needed)

Compute $\mathbf{s}_l^{(k)} = -\left[\mathbf{H}_k + \lambda_l^{(k)}\mathbf{I}\right]^{-1}\mathbf{g}^{(k)}$

Evaluate $\varphi\left[\lambda_l^{(k)}\right] = 1/\left|\left|\mathbf{s}_l^{(k)}\right|\right| - 1/\delta_k$

Evaluate $\varphi'\left[\lambda_l^{(k)}\right]$

Compute $\lambda_{l+1}^{(k)} = \lambda_l^{(k)} - \varphi\left[\lambda_l^{(k)}\right]/\varphi'\left[\lambda_l^{(k)}\right]$

Trust-region methods (cont.)

Vector $\mathbf{s}_l^{(k)}$ can be obtained by **Cholesky factorisation** of $\left[\mathbf{H}_k + \lambda_l^{(k)}\mathbf{I}\right]$

- Provided that matrix $\mathbf{B}^{(k)} = \mathbf{H}_k + \lambda_l^{(k)}\mathbf{I}$ is positive definite
- If $\mathbf{B}^{(k)}$ is symmetric (definition of $\mathbf{H}_k$), its eigenvalues are all real

Definition

Cholesky factorisation

Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be a symmetric and positive definite matrix

$$\mathbf{A} = \mathbf{R}^T\mathbf{R}$$

$\mathbf{R}$ is upper triangular with positive elements on the diagonal

---

Usually, a regularised matrix $\left[\mathbf{B}_l^{(k)} + \beta\mathbf{I}\right]$ is used instead of plain $\mathbf{B}^{(k)}$

- β should be larger than the largest negative eigenvalue of $\mathbf{B}^{(k)}$

Trust region methods (cont.)

For $\mathbf{g}^{(k)} = \nabla f\left[\mathbf{x}^{(k)}\right]$ and for some given δ_k

Pseudo-code

Solve $\mathbf{H}_k\mathbf{s} = -\mathbf{g}^{(k)}$ (It means that $\mathbf{s} = -\mathbf{H}_k^{-1}\mathbf{g}^{(k)}$)

If $\left|\left|\mathbf{s}\right|\right| \leq \delta_k$ and $\mathbf{H}_k$ is positive definite
  Set $\mathbf{s}^{(k)} = \mathbf{s}$
else
  Let β_1 be the negative eigenvalue of $\mathbf{H}_k$ with largest modulus
  Set $\lambda_0^{(k)} = 2\left|\beta_1\right|$
  For $l = 0, 1, \ldots$
    Compute $\mathbf{R} : \mathbf{R}^T\mathbf{R} = \mathbf{H}_k + \lambda_l^{(k)}\mathbf{I}$
    Solve $\mathbf{R}^T\mathbf{R}\mathbf{s} = \mathbf{g}^{(k)}$, $\mathbf{R}^T\mathbf{q} = \mathbf{s}$
    Update $\lambda_{l+1}^{(k)} = \lambda_l^{(k)} + \left(||\mathbf{s}||/||\mathbf{q}||\right)^2 \frac{||\mathbf{s}|| - \delta_k}{\delta_k}$
  Set $\mathbf{s}^{(k)} = \mathbf{s}$
endif

Trust-region methods (cont.)

A good choice of the radius δ_k is fundamental for achieving fast convergence

The criterion for accepting a solution $\mathbf{s}^{(k)}$ is based on a comparison

- The variation of f and that of its quadratic approximation $\tilde{f}_k$

Remark

As $\mathbf{x}^{(k)}$ moves to $\mathbf{x}^{(k)} + \mathbf{s}^{(k)}$, we compute

$$\rho_k = \frac{f\left[\mathbf{x}^{(k)} + \mathbf{s}^{(k)}\right] - f\left[\mathbf{x}^{(k)}\right]}{\tilde{f}_k\left[\mathbf{s}^{(k)}\right] - \tilde{f}_k(\mathbf{0})}$$

If $\rho_k \approx 1$

$\rightsquigarrow$ $\mathbf{s}^{(k)}$ is accepted, the ball is enlarged if the minimum is on the boundary

If $\rho_k \approx 0$ or $\rho_k < 0$

$\rightsquigarrow$ $\mathbf{s}^{(k)}$ is not accepted and the ball is diminished

Trust-region methods (cont.)

Let $\mathbf{x}^{(0)}$ be an initial solution

Let the initial radius of the ball be $\delta_0 \in (0, \hat{\delta})$ with maximum radius $\hat{\delta} > 0$

Let $\{\eta_1, \eta_2, \gamma_1, \gamma_2\}$ be the four real parameters for updating the ball

- $0 < \eta_1 < \eta_2 < 1$
- $0 < \gamma_1 < 1 < \gamma_2$

Let $0 \leq \mu \leq \eta_1$ be the real parameter for accepting a solution

...

Trust-region methods (cont.)

Then, for $k = 0, 1, \ldots$ and until convergence

Pseudo-code

Compute $f\left[\mathbf{x}^{(k)}\right]$, $\nabla f\left[\mathbf{x}^{(k)}\right]$ and $\mathbf{H}_k$

Solve $\min_{\mathbf{s}} \in \mathcal{R}^n : ||\mathbf{s}||_2 \leq \delta_k \tilde{f}_k(\mathbf{s})$

Compute ρ_k

If $\rho_k > \mu$
    Set $\mathbf{x}^{(x+1)} = \mathbf{x}^{(k)} + \mathbf{s}^{(k)}$
else
    Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$
endif

If $\rho_k < \eta_1$
    Set $\delta_{k+1} = \gamma_1 \delta_k$
elseif $\eta_1 \leq \rho_k \leq \eta_2$
    Set $\delta_{k+1} = \delta_k$
elseif $\rho_k > \eta_2$ and $||\mathbf{s}^{(k)}|| = \delta_k$
    Set $\delta_{k+1} = \min\{\gamma_2 \delta_k, \hat{\delta}\}$
endif

Trust-region methods (cont.)

Choice of parameters[3]

- $\rightsquigarrow \eta_1 = 1/4$
- $\rightsquigarrow \eta_2 = 3/4$
- $\rightsquigarrow \gamma_1 = 1/4$
- $\rightsquigarrow \gamma_2 = 8/4$

- By choosing $\mu = 0$, we accept any step yielding a decrease of f
- By choosing $\mu > 0$, we accept steps for which the variation of f is at least μ times the variation of its quadratic model $\tilde{f}_k$

[3] J. Nocedal and S. Wrigth (2006): *Numerical optimization*.

Trust-region methods (cont.)


```
%TREGION Trust region optimisation method
%[X,ERR,ITER]=TREGION(FUN,GRAD_FUN,X_0,DELTA_0, ...
% TOL,KMAX,TYP,HESS_FUN)
% Approximates the minimiser of FUN with gradient GRAD_FUN
%
% If TYP=1 Hessian is inputed as HESS_FUN
% If TYP NE 1 Hessian is rank-one approximated
%
% FUN and GRAD_FUN (and HESS_FUN) are function handles
% X_0 is the initial  point
% TOL is stop check tolerance
% DELTA_0 is initial radius of trust ball
% KMAX are maximum number of iterations
```

```
function [x,err,iter]= tRegion(fun,grad_fun,x_0,delta_0, ...
                               tol,kmax,typ,hess_fun)

delta = delta_0; err = 1 + tol; k = 0; mu = 0.1; delta_m = 5;
eta_1 = 0.25; eta_2 = 0.75; gamma_1 = 0.25; gamma_2 = 2.00;

xk = x_0(:); gk = grad_fun(xk); eps2 = sqrt(eps);
if typ==1; Hk=hess_fun(xk); else; Hk=eye(length(xk)); end

while err > tol & k < kmax
 [s]=trust_one(Hk,gk,delta);
 rho=(fun(xk+s)-fun(xk))/(s'*gk+1/2*s'*Hk*s);
 if rho > mu; xk1 = xk + s; else; xk1 = xk; end
 if rho < eta_1; delta = gamma_1*delta;
 elseif rho > eta_2 & abs(norm(s)-delta) < sqrt(eps)
  delta=min([gamma_2*delta,delta_m]);
 end
 gk1 = grad_fun(xk1);
 err = norm((gk1.*xk1)/max([abs(fun(xk1)),1]),Inf);
 if typ == 1; xk = xk1; gk = gk1; Hk = hess_fun(xk);  % Newton
 else                                                % quasi-Newton
  gk1 = grad(xk1); yk = gk1-gk; sk=xk1-xk; yks = yk'*sk;
  if yks > eps_2*norm(sk)*norm(yk)
   Hs = Hk*sk; Hk = Hk+(yk*yk')/yks-(Hs*Hs')/(sk'*Hs);
  end
  xk = xk1; gk = gk1;
 end
 k=k+1;
end

x = xk; iter = k;
if (k==kmax & err>tol); disp('Accuracy not met [KMAX]'); end
```

Trust-region methods (cont.)


```
function [s] = trust_one (Hk,gk,delta)
maxiter=5;

s = -Hk\gk; d = eigs(Hk,1,'sa'); % 1st smallest algebraic evalue

if norm(s) > delta | d<0
 lambda = abs(2*d); I = eye(size(Hk));
 for l=1:maxiter
  R = chol(lambda*I+Hk);
  s = -R\(R'\gk); q = R'\s;
  lambda = lambda+(s'*s)/(q'*q)*(norm(s)-delta)/delta;
  if lambda < -d
   lambda = abs(2*lambda);
  end
 end
end
```

Trust-region methods (cont.)

Example

Use the trust-region method to approximate the minimiser of function

$$f(x_1, x_2) = 7/5 + \frac{x_1 + 2x_2 + 2x_1x_2 - 5x_1^2 - 5x_2^2}{[5\exp(x_1^2 + x_2^2)]}$$

The function shows a local maximum, a saddle point and two local minima

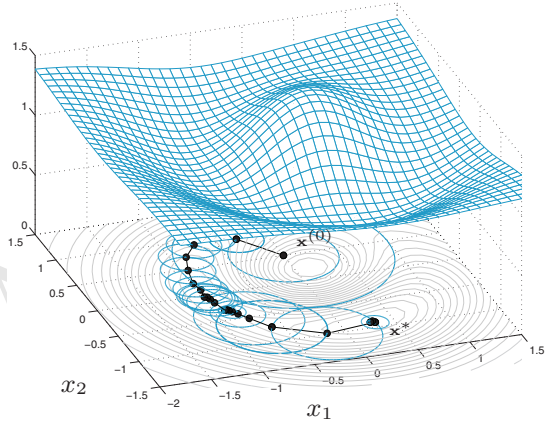
- The local minima are near $(-1.0, +0.2)$ and $(+0.3, -0.9)$
- The second minimum is the global one

```
fun = @(x) (x(1)+2*x(2)+2*x(1)*x(2)-5*x(1)^2-5*x(2)^2) / ...
           (5*exp(x(1)^2+x(2)^2)) + 7.5;

grad_fun = @(x) [(1 + 2*x(2)-10*x(1)-2*x(1)*(x(1)+2*x(2) + ...
                     2*x(1)*x(2)-5*x(1)^2-5*x(2)^2)) / ...
                     (5*exp(x(1)^2+x (2)^2));
                 (2 + 2*x(1)-10*x(2)-2*x(2)*(x(1)+2*x(2) + ...
                     2*x(1)*x(2)-5*x(1)^2-5*x(2)^2)) / ...
                     (5*exp(x(1)^2+x(2)^2))];

delta_0 = 0.5; x_0 = [0.0;0.5];
tol = 1e-5; kmax = 100; imax=5;
typ = 2;

[x,er,it]=tRegion(fun,grad_fun,x_0,delta_0,tol,kmax,typ,imax)
```

Trust-region methods (cont.)

Trust-region, approximated Hesse matrix

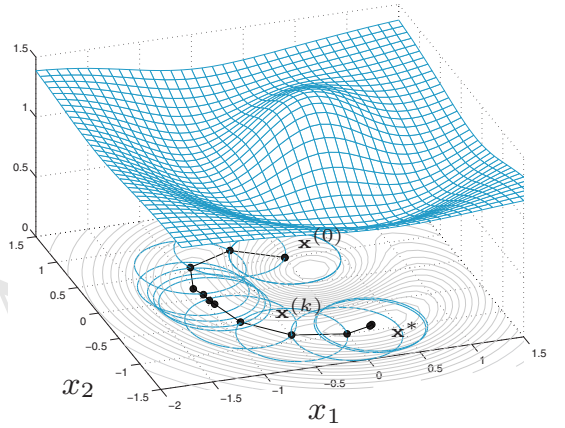
$\rightsquigarrow$ 24 iterations, $\mathbf{x}^* \approx (+0.28, -0.90)$



Trust-region methods (cont.)

Trust-region, exact Hessian

⇝ 12 iterations



Trust region methods (cont.)

Example

Rosenbrock's function

$$f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

```
1  fun = @(x) (1-x(1))^2+100*(x(2)-x(1)^2)^2;
2  grad_fun = @(x)[-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1)); ...
3                  200*(x(2)-x(1)^2)];
4
5  x_0=[+1.2;-1.0];
6
7  options = optimset ('LargeScale','on');                    % Trust-region
8  options = optimset ('GradObj','on');                          % Gradient
9
10 [x,fval,exitflag,output]=fminunc({fun,grad_fun},x_0,options)
```

Trust-region (Matlab)

⇝ 8 iterations, 9 function evaluations

■

Trust-region methods (cont.)

Remark

The M-command `fminunc` in Octave implements the trust region method

- With approximated Hessians $\mathbf{H}_k$, computed with BFGS method

$$\rightsquigarrow \quad \mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{y}^{(k)}\mathbf{y}^{(k)^T}}{\mathbf{x}^{(k)^T}\mathbf{s}^{(k)}} - \frac{\mathbf{H}_k\mathbf{s}^{(k)}\mathbf{s}^{(k)^T}\mathbf{H}_k^T}{\mathbf{s}^{(k)^T}\mathbf{H}_k\mathbf{s}^{(k)}}$$

The option `'LargeScale'` is not used

Non-linear least-squares

Numerical optimisation

Non-linear least-squares

The **least-squares method** is often used for approximating either functions $f(x)$ or sets of data $\{(x_k, y_k), k = 0, \dots, K\}$ by some function $\tilde{f}$

- Often $\tilde{f}$ depends linearly on a set of coefficients $\{a_j, j = 1, \dots, m\}$

Example

$$\tilde{f}(x|\{a_j\}_{j=0}^m) = a_0 + a_1 x + a_2 x^2 + \cdots + a_m x^m$$

The coefficients $\{a_j\}_{j=0}^m$ are unknown

They must be determined from data

$$\{(x_k, y_k), k = 0, \dots, K\}$$

$$\rightsquigarrow \min_{\{a_j, j=1,\dots,m\}} \sum_{k=0}^{K} \Big[y_k - \underbrace{\tilde{f}(x_k|\{a_j\})}_{a_0 + a_1 x_k + a_2 x_k^2 + \cdots + a_m x_k^m} \Big]^2$$

This problem is called a **least-squares** problem

The problem becomes nonlinear when $\tilde{f}$ non-linearly depends on $\{a_j\}$

Non-linear least-squares (cont.)

Definition

Let $\mathbf{R}(\mathbf{x}) = \big[r_1(\mathbf{x}), \dots, r_n(\mathbf{x})\big]^T$ with $r_i : \mathbb{R}^m \to \mathbb{R}$ be some smooth function

We want to find

$$\min_{\mathbf{x}\in\mathbb{R}^m} f(\mathbf{x}), \quad \text{with } f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{n} r_i^2(\mathbf{x}) = \frac{1}{2}\big|\big|\mathbf{R}(\mathbf{x})\big|\big|^2 \tag{34}$$

We assume that $n \geq m$

If functions $r_i(\mathbf{x})$ are non-linear, then function $f(\mathbf{x})$ may not be convex

$\rightsquigarrow$ Thus, $f(\mathbf{x})$ may have multiple stationary points

We can use Newton, descent directions and trust-region methods

Non-linear least-squares (cont.)

Consider the special form of f

We have assembled the components $r_i(\mathbf{x})$ into a residual vector

$$\mathbf{R}(\mathbf{x}) = \big[r_1(\mathbf{x}), \ldots, r_n(\mathbf{x})\big]^T$$

Because of this, we compactly rewrote the objective function

$$f(\mathbf{x}) = \frac{1}{2}\big|\big|\mathbf{R}(\mathbf{x})\big|\big|^2$$

The derivatives of $f(\mathbf{x})$ can be expressed in terms of the Jacobian of $\mathbf{R}$

$\rightsquigarrow$ Partial derivatives of $r_i(\mathbf{x})$ with respect to x_j

$$\mathbf{J}_{\mathbf{R}}(\mathbf{x}) = \Big[\frac{\partial r_i}{\partial x_j}\Big]_{\substack{j=1,\ldots,m \\ i=1,\ldots,m}} = \begin{bmatrix} \Big[\frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \cdots & \frac{\partial r_1}{\partial x_m}\Big] \\ \Big[\frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \cdots & \frac{\partial r_2}{\partial x_m}\Big] \\ \vdots & \vdots & \ddots & \vdots \\ \Big[\frac{\partial r_n}{\partial x_1} & \frac{\partial r_n}{\partial x_2} & \cdots & \frac{\partial r_n}{\partial x_m}\Big] \end{bmatrix} = \begin{bmatrix} \boldsymbol{\nabla} r_1(\mathbf{x})^T \\ \boldsymbol{\nabla} r_2(\mathbf{x})^T \\ \vdots \\ \boldsymbol{\nabla} r_n(\mathbf{x})^T \end{bmatrix}$$

Non-linear least-squares (cont.)

Gradient and Hessian of the cost function can be compactly written

$$\begin{aligned}\nabla f(\mathbf{x}) &= \sum_{i=1}^{n} r_i(\mathbf{x})\nabla r_i(\mathbf{x}) = \mathbf{J_R}(\mathbf{x})^T\mathbf{R}(\mathbf{x}) \\ \nabla^2 f(\mathbf{x}) &= \mathbf{J_R}(\mathbf{x})^T\mathbf{J_R}(\mathbf{x}) + \sum_{i=1}^{n} r_i(\mathbf{x})\nabla r_i(\mathbf{x}) = \mathbf{J_R}(\mathbf{x})^T\mathbf{J_R}(\mathbf{x}) + \mathbf{S}(\mathbf{x})\end{aligned} \tag{35}$$

$\rightsquigarrow$ The second derivatives of $\mathbf{R}$ cannot be calculated from the Jacobian

$$\mathbf{S}_{lj}(\mathbf{x}) = \sum_{i=1}^{n} \frac{\partial^2 r_i}{\partial x_l \partial x_j}(\mathbf{x}) r_i(\mathbf{x}), \text{ for } l, j = 1, \ldots, m$$

Non-linear least-squares (cont.)

Calculation of the Hesse matrix can be heavy when m and n are large

- This is mostly due to matrix $\mathbf{S}(\mathbf{x})$

In some cases, $\mathbf{S}(\mathbf{x})$ is less influent than $\mathbf{J_R}(\mathbf{x})^T\mathbf{J_R}(\mathbf{x})$

- ⇝ It could be approximated or neglected
- ⇝ It simplifies the construction of $\mathbf{H}(\mathbf{x})$

We discuss two methods devoted to handling such cases

Gauss-Newton method

Nonlinear least-squares

The Gauss-Newton method

The **Gauss-Newton method** is a variant of the Newton method

Given $\mathbf{x}^{(0)} \in \mathbb{R}^n$, for $k = 0, 1, \ldots$ until convergence

Pseudo-code

Solve $\mathbf{H}\big[\mathbf{x}^{(k)}\big]\boldsymbol{\delta}\mathbf{x}^{(k)} = -\boldsymbol{\nabla} f\big[\mathbf{x}^{(k)}\big]$

Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)}$

The Hessian $\mathbf{H}(\mathbf{x})$ is approximated by neglecting $\mathbf{S}(\mathbf{x})$

The Gauss-Newton method (cont.)

Given $\mathbf{x}^{(0)} \in \mathbb{R}^m$ and for $k = 0, 1, \ldots$ until the convergence

Pseudo-code

Solve $\left\{\mathbf{J_R}(\mathbf{x}^k)^T \mathbf{J_R}\left[\mathbf{x}^{(k)}\right]\right\}\boldsymbol{\delta}\mathbf{x}^{(k)} = -\mathbf{J_R}\left[\mathbf{x}^{(k)}\right]^T \mathbf{R}\left[\mathbf{x}^{(k)}\right]$

Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)}$

The system in the first equation may have infinitely many solutions

If $\mathbf{J_R}\left[\mathbf{x}^{(k)}\right]$ is not full rank

- ⇝ Stagnation
- ⇝ Non-convergence
- ⇝ Convergence to a non-stationary point

If $\mathbf{J_R}\left[\mathbf{x}^{(k)}\right]$ is full rank, the linear system has form $\mathbf{A}^T\mathbf{A}\mathbf{x}^* = \mathbf{A}^T\mathbf{b}$

- It can be solved by using QR or SVD factorisations of $\mathbf{J_R}(\mathbf{x})$

The Gauss-Newton method (cont.)


```
function [x,err,iter]=nllsGauNewtn(r,jr,x_0,tol,kmax,varargin)
%NLLSGAUNEW Nonlinear least-squares with Gauss-Newton method
% [X,ERR,ITER]=NLLSGAUNEW(R,JR,X_0,TOL,KMAX)
% R and JR: Function handles for objective R and its Jacobian
% X_0 is the initial solution
% TOL is the stop check tolerance
% KMAX is the max number of iterations

err = 1 + tol; k = 0;
xk = x_0(:);

rk = r(xk,varargin{:}); jrk = jr(xk,varargin{:});

while err > tol & k < kmax
 [Q,R] = qr(jrk,0); dk = -R\(Q'*rk);
 xk1 = xk + dk;
 rk1 = r(xk1,varargin{:});
 jrk1 = jr(xk1,varargin{:});

 k = 1 + k; err = norm(xk1 - xk);
 xk = xk1; rk = rk1; jrk = jrk1;
end

x = xk; iter = k;

if (k==kmax & err > tol)
 disp('nllsGauNewtn stopped w\o reaching accuracy [KMAX]');
end
```

The Gauss-Newton method (cont.)

Remark

Neglecting $\mathbf{S}(\mathbf{x}^{(k)})$ at step k amounts to approximating $\mathbf{R}(\mathbf{x})$

The first-order Taylor expansion of $\mathbf{R}(\mathbf{x})$ at $\mathbf{x}^*$

$$\tilde{\mathbf{R}}_k(\mathbf{x}) = \mathbf{R}\big[\mathbf{x}^{(k)}\big] + \mathbf{J}_{\mathbf{R}}\big[\mathbf{x}^{(k)}\big]\big[\mathbf{x} - \mathbf{x}^{(k)}\big] \tag{36}$$

Convergence of the method is not always guaranteed

- It depends on f and initial solution

The Gauss-Newton method (cont.)

Let $\mathbf{x}^*$ be a stationary point for $f(\mathbf{x})$

Let $\mathbf{J_R}(\mathbf{x})$ be full rank in a suitable neighbourhood of $\mathbf{x}^*$

Then,

If $\mathbf{S}(\mathbf{x}^*) = 0$ (if $\mathbf{R}(\mathbf{x})$ is linear or $\mathbf{R}(\mathbf{x}^*) = \mathbf{0}$)

1. The Gauss-Newton method is locally quadratically convergent
2. It coincides with the Newton's method

If $||\mathbf{S}(\mathbf{x}^*)||_2$ is small compared to the smallest positive eigenvalue of

$$\mathbf{J_R}(\mathbf{x}^*)^T\mathbf{J_R}(\mathbf{x}^*)$$

1. (e.g., when $\mathbf{R}(\mathbf{x})$ is mildly non-linear or its residual $\mathbf{R}(\mathbf{x}^*)$ is small)
2. Gauss-Newton converges linearly

If $||\mathbf{S}(\mathbf{x})||_2$ is large compared to the smallest positive eigenvalue of

$$\mathbf{J_R}(\mathbf{x}^*)^T\mathbf{J_R}(\mathbf{x}^*)$$

1. Gauss-Newton may not converge, even if $\mathbf{x}^{(0)}$ is very close to $\mathbf{x}^*$
2. (e.g., when $\mathbf{R}(\mathbf{x})$ is strongly non-linear or its residual $\mathbf{R}(\mathbf{x}^*)$ is large)

The Gauss-Newton method (cont.)

Remark

Line-search can be used in combination with Gauss-Newton

- Replace $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\delta}\mathbf{x}^{(k)}$ with $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \boldsymbol{\delta}\mathbf{x}^{(k)}$
- Computation of step-lengths α_k is as per usual

If $\mathbf{J_R}(\mathbf{x}^{(k)})$ is full rank, matrix $\mathbf{J_R}(\mathbf{x}^{(k)})^T \mathbf{J_R}(\mathbf{x}^{(k)})$ is symmetric and PD

- $\boldsymbol{\delta}\mathbf{x}^{(k)}$ is a descent direction for $f(\mathbf{x})$

Under suitable assumptions on $f(\mathbf{x})$, we get the globally convergent method

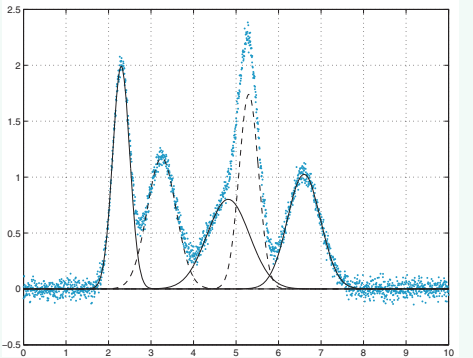
$\rightsquigarrow$ **Damped Gauss-Newton method**

The Gauss-Newton method (cont.)

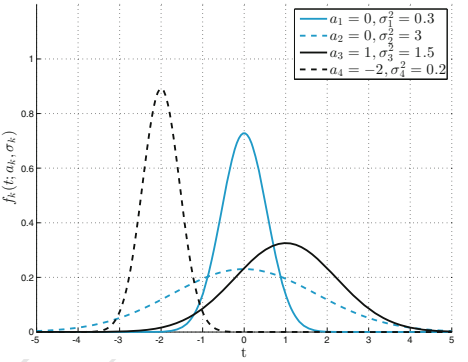
Example

The signal intensity is modelled as a sum of m Gaussian functions
Compress an audio signal to a set of parameters

$$f_k(t|a_k,\sigma_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\exp\Big[-\frac{(t-a_k)^2}{2\sigma_k^2}\Big], \quad t \in [t_0, t_F], k = 1, \ldots, m$$



The Gauss-Newton method (cont.)



Each peak or component is characterised by two coefficients

- The centre, a_k
- The (square of the) spread, σ_k^2

$$f(t|\mathbf{a}, \boldsymbol{\sigma}) = \sum_{k=1}^{m} f_k(t; a_k, \sigma_k)$$

- $\mathbf{a} = [a_1, \cdots, a_k]$
- $\boldsymbol{\sigma} = [\sigma_1, \cdots, \sigma_k]$

The Gauss-Newton method (cont.)

Find $\mathbf{a}$ and $\boldsymbol{\sigma}$ that minimise the residual sum of squares

$$\min_{\mathbf{a},\boldsymbol{\sigma}} \sum_{i=1}^{n} \Big[f(t_i|\mathbf{a},\boldsymbol{\sigma}) - y_i \Big]^2$$

From recorded audio intensities y_i at sampling times t_i

The Gauss-Newton method (cont.)

Generate $n = 2000$ time-intensity pairs $(t_i, y_i)_{i=1}^n$ with $t_i \in (0, 10)$

$\rightsquigarrow$ The sum of 5 Gaussian components

$$f_k(t|a_k, \sigma_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\Big[-\frac{(t-a_k)^2}{2\sigma_k^2}\Big]$$

- Plus some little random noise

```
1  a = [2.3, 3.2, 4.8, 5.3, 6.6]; m = length(a);
2  sigma = [0.2, 0.3, 0.5, 0.2, 0.4];
3
4  gComp = @(t,a,sigma) exp(-((t-a)/(sigma*sqrt(2))).^2)/ ...
5                       (sigma*sqrt(pi*2));
6
7  n = 2000; t = linspace(0,10,n)'; y = zeros(n,1);
8  for k=1:m
9   y = y + gComp(t,a(k),sigma(k));
10 end
11
12 y = y + 0.05*randn(n,1);                % Little additive noise
```

The Gauss-Newton method (cont.)

Solve the nonlinear least-squares problem of form

$$\min_{\mathbf{x}\in\mathbb{R}^m} \Phi(\mathbf{x}), \quad \text{with } \Phi(\mathbf{x}) = \frac{1}{2}\big|\big|\mathbf{R}(\mathbf{x})\big|\big|^2 = \frac{1}{2}\sum_{i=1}^{n} r_i^2(\mathbf{x})$$

$r_i(\mathbf{x}) = f(t_i|\mathbf{a},\boldsymbol{\sigma}) - y_i = \sum_{k=1}^{m} f_k(t_i|a_k,\sigma_k) - y_i$

We also have,

$$\frac{\partial r_i}{\partial a_k} = f_k(t_i|a_k,\sigma_k)\Big[\frac{t_i - a_k}{\sigma_k}\Big]$$

$$\frac{\partial r_i}{\partial \sigma_k} = f_k(t_i|a_k,\sigma_k)\Big[\frac{(t_i - a_k)^2}{\sigma_k^3} - \frac{1}{2\sigma_k}\Big]$$

The Gauss-Newton method (cont.)

Gauss-Newton

M-command `nllsGauNewtn` (22 iterations)

```
x_0 = [2.0,3.0,4.0,5.0,6.0,0.3,0.3,0.6,0.3,0.3];

tol = 3.0e-5;
kmax = 200;

[x,err,iter]=nllsGauNew(@gmR,@gmJR,x_0,tol,kmax,t,y)

x_a = x(1:m);
x_sigma = x(m+1:end);

h = 1./(x_sigma*sqrt(2*pi));
w = 2*x_sigma*sqrt(log(4));
```

The Gauss-Newton method (cont.)


```
function [R]=gmR(x,t,y)

x = x(:); m = round(0.5*length(x));
a = x(1:m); sigma = x(m+1: end );

gauFun = @(t,a,sigma) [exp(-((t-a)/(sigma*sqrt(2))).^2) ...
                       /(sigma*sqrt(pi*2))];

n = length(t); R = zeros(n,1);
for k = 1:m; R = R + gauFun(t,a(k),sigma(k)); end
R = R - y;
```

```
function [Jr]=gmJR(x,t,y)
x = x(:); m = round(0.5*length(x));
a = x(1:m); sigma = x(m+1: end);

gauFun = @(t,a,sigma) [exp(-((t-a)/(sigma*sqrt(2))).^2) ...
                       /(sigma*sqrt(pi*2))];

n = length(t); JR = zeros(n,2*m); fk = zeros(n,m);
for k = 1:m; fk(:,k) = gauFun(t,a(k),sigma(k)); end
for k = 1:m; JR(:,k) = (fk(:,k).*(t-a(k))/sigma(k)^2)'; end
for k = 1:m
 JR(:,k+m) = (fk(:,k).*((t-a(k)).^2/(k)^3-1/(2*sigma(k))))';
end
```

Levenberg-Marquardt

Nonlinear least-squares

Levenberg-Marquardt

Levenberg-Marquardt is a trust-region method

$$\min_{\mathbf{x}\in\mathbb{R}^m} f(\mathbf{x}), \quad \text{with } f(\mathbf{x}) = \frac{1}{2}\left|\left|\mathbf{R}(\mathbf{x})\right|\right|^2 = \frac{1}{2}\sum_{i=1}^{n} r_i^2(\mathbf{x})$$

We can use the general trust-region formulation

Levenberg-Marquardt (cont.)

Pseudo-code

Compute $f\left[\mathbf{x}^{(k)}\right]$, $\nabla f\left[\mathbf{x}^{(k)}\right]$ and $\mathbf{H}_k$

Solve $\min_{||\mathbf{s}||_2 \leq \delta_k} \tilde{f}_k(\mathbf{s})$

Compute ρ_k
If $\rho_k > \mu$
    Set $\mathbf{x}^{(x+1)} = \mathbf{x}^{(k)} + \mathbf{s}^{(k)}$
else
    Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$
endif

If $\rho_k < \eta_1$
    Set $\delta_{k+1} = \gamma_1 \delta_k$
elseif $\eta_1 \leq \rho_k \leq \eta_2$
    Set $\delta_{k+1} = \delta_k$
elseif $\rho_k > \eta_2$ and $||\mathbf{s}^{(k)}|| = \delta_k$
    Set $\delta_{k+1} = \min\{\gamma_2 \delta_k, \hat{\delta}\}$
endif

Levenberg-Marquardt (cont.)

At each step k, we solve

$$\min_{\mathbf{s}\in\mathbb{R}^n:||\mathbf{s}||\leq\delta_k} \tilde{f}_k(\mathbf{s}), \quad \text{with } \tilde{f}_k(\mathbf{s}) = \frac{1}{2}\Big|\Big|\mathbf{R}\big[\mathbf{x}^{(k)}\big] + \mathbf{J}_{\mathbf{R}}\big[\mathbf{x}^{(k)}\big]\mathbf{s}\Big|\Big|^2 \tag{37}$$

$\tilde{f}_k(\mathbf{x})$ is a quadratic approximation of $f(\mathbf{x})$ about $\mathbf{x}^{(k)}$

$\rightsquigarrow$ By approximating $\mathbf{R}(\mathbf{x})$ with its linear model

$$\tilde{\mathbf{R}}_k(\mathbf{x}) = \mathbf{R}\big[\mathbf{x}^{(k)}\big] + \mathbf{J}_{\mathbf{R}}\big[\mathbf{x}^{(k)}\big]\big[\mathbf{x} - \mathbf{x}^{(k)}\big]$$

Levenberg-Marquardt (cont.)

Often $\mathbf{J_R}(\mathbf{x})$ is not full rank, yet the method is well-posed

The method is suited for minimisation problems with strong non-linearities or large residuals $f(\mathbf{x}^*) = 1/2||\mathbf{R}(\mathbf{x}^*))||^2$ about the local minimiser $\mathbf{x}^*$

Remark

Hessian approximations are those of the Gauss-Newton method

The two methods share the same local convergence properties

Convergence rates when Levenberg-Marquardt iterations do converge

- Convergence rate is quadratic, if residual is small at local minimiser
- Convergence rate is linear, otherwise

Derivative-free methods

Unconstrained optimisation

Derivative-free methods

We describe two simple numerical methods

- **Minimisation of univariate real-valued functions**
- **Minimisation of multivariate real-valued functions**
- (along a single direction)

We then describe the **Nelder and Mead method**

- **Minimisation of functions of several variables**

Golden section and quadratic interpolation

Derivative-free methods

Golden section and quadratic interpolation

Let $f : (a, b) \to \mathbb{R}$ be a continuous function with unique minimiser

$$x^* \in (a, b)$$

Set $I_0 = (a, b)$, for $k \geq 0$ generate a sequence of intervals I_k

$$I_k = (a^{(k)}, b^{(k)})$$

The intervals I_k are of decreasing length and each contains x^*

Golden section and quadratic interpolation (cont.)

For any given k, the next interval I_{k+1} can be determined

1) Let $c^{(k)}, d^{(k)} \in I_k$ with $c^{(k)} < d^{(k)}$ be two points such that

$$\frac{b^{(k)} - a^{(k)}}{d^{(k)} - a^{(k)}} = \frac{d^{(k)} - a^{(k)}}{b^{(k)} - d^{(k)}} = \varphi \tag{38a}$$

$$\frac{b^{(k)} - a^{(k)}}{b^{(k)} - c^{(k)}} = \frac{b^{(k)} - c^{(k)}}{c^{(k)} - a^{(k)}} = \varphi \tag{38b}$$

Let φ be the **golden ratio** $\varphi = \dfrac{1+\sqrt{5}}{2} \simeq 1.628$

Golden section and quadratic interpolation (cont.)

2) Using Equation 38a and 38b, we find point $c^{(k)}$ and point $d^{(k)}$

$$c^{(k)} = a^{(k)} + \frac{1}{\varphi^2}(b^{(k)} - a^{(k)}) \tag{39a}$$

$$d^{(k)} = a^{(k)} + \frac{1}{\varphi}(b^{(k)} - a^{(k)}) \tag{39b}$$

They are symmetrically placed about the mid-point of I_k

$$\frac{a^{(k)} + b^{(k)}}{2} - c^{(k)} = d^{(k)} - \frac{a^{(k)} + b^{(k)}}{2} \tag{40}$$

Remark

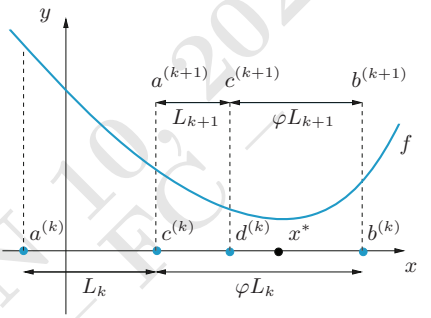
Replace $c^{(k)}$ and $d^{(k)}$ in Equation (40)

Divide by the common factor $\frac{b^{(k)} - a^{(k)}}{\varphi^2}$

Get the identity

$$\varphi^2 - \varphi - 1 = 0$$

Golden section and quadratic interpolation (cont.)



The generic iteration of the **golden-section method**

- φ is the golden ratio, while $L_k = c^{(k)} - a^{(k)}$

Golden section and quadratic interpolation (cont.)

Set $a^{(0)} = a$ and $b^{(0)} = b$, the golden section method formulates as

Pseudo-code

For $k = 0, 1, \ldots$ until convergence

Compute $c^{(k)}$ and $d^{(k)}$ through Equation (39)

If $f(c^{(k)}) \geq f(d^{(k)})$
  set $I_{k+1} = (a^{(k+1)}, b^{(k+1)}) = (c^{(k)}, b^{(k)})$
else
  set $I_{k+1} = (a^{(k+1)}, b^{(k+1)}) = (a^{(k)}, d^{(k)})$
endif

It follows that

- If $I_{k+1} = (c^{(k)}, b^{(k)})$, then $c^{(k+1)} = d^{(k)}$
- If $I_{k+1} = (a^{(k)}, d^{(k)})$, then $d^{(k+1)} = c^{(k)}$

Golden section and quadratic interpolation (cont.)

We need to set a **stopping criterion**

When the normalised size of the k-th interval is smaller than a tolerance ε

$$\frac{b^{(k+1)} - a^{(k+1)}}{|c^{(k+1)}| + |d^{(k+1)}|} < \varepsilon \tag{41}$$

The mid-point of the last interval I_{k+1} can be taken as solution

- This is an **approximation of the minimiser** $\mathbf{x}^*$

By using Equation (38a) and (38b), yields the expression

$$|b^{(k+1)} - a^{(k+1)}| = \frac{1}{\varphi}|b^{(k)} - a^{(k)}| = \cdots = \frac{1}{\varphi^{k+1}}|b^{(0)} - a^{(0)}| \tag{42}$$

The golden-section method converges linearly with rate

$$\varphi^{-1} \simeq 0.618$$

```
function [xmin,fmin,iter]=gSection(fun,a,b,tol,kmax,varargin)
%GSECTION finds the minimum of a function
% XMIN=GSECTION(FUN,A,B,TOL,KMAX) approximates a min point of
%  function FUN in [A,B] by using the golden section method
% If the search fails, an error message is returned
% FUN can be i) an inline function, ii) an anonymous function
%  or iii) a function defined in a M-file
% XMIN=GSECTION(FUN,A,B,TOL,KMAX,P1,P2,...) passes parameters
%  P1, P2,... to function FUN(X,P1,P2,...)
% [XMIN,FMIN,ITER]= GSECTION(FUN,...) returns the value of FUN
%  at XMIN and number of iterations ITER done to find XMIN

phi = (1+sqrt(5))/2;
iphi(1) = inv(phi); iphi(2) = inv(1+phi);
c = iphi(2)*(b-a) + a; d = iphi(1)*(b-a) + a;
err = 1+tol; k = 0;

while err > tol & k < kmax
 if(fun(c) >= fun(d))
  a = c; c = d; d = iphi(1)*(b-a) + a;
 else
  b = d; d = c; c = iphi(2)*(b-a) + a;
 end
 k = 1 + k; err = abs(b-a)/(abs(c)+abs(d));
end

xmin = 0.5*(a+b); fmin = fun(xmin); iter = k;
if (iter == kmax & err > tol)
 fprintf('The method stopped after reaching the maximum number
          of iterations, and without meeting the tolerance');
end
```

Golden section and quadratic interpolation (cont.)

- `fun` is either an anonymous or an inline function for function f
- `a` and `b` are endpoints of the search interval
- `tol` is the tolerance ε
- `kmax` is the maximum allowed number of iterations

- `xmin` contains the value of the minimiser
- `fmin` is the minimum value of f in (a, b)
- `iter` is the number of iterations carried out by the algorithm

Golden section and quadratic interpolation (cont.)

Example

Evolution of an isolated culture of 250 bacteria (Verhulst model)

$$f(t) = \frac{2500}{1 + 9e^{(-t/3)}}, \quad \text{for } t > 0$$

t denotes time (in days)

Find after how many days population growth rate is maximum

⇝ Where (when?) does function $g(t) = -f'(t)$ has its minimum

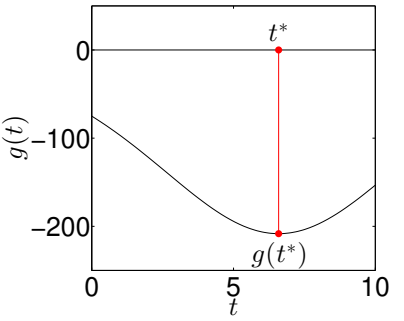
$$g(t) = -7500 \frac{\exp(t/3)}{\left[\exp(t/3) + 9\right]^2}$$

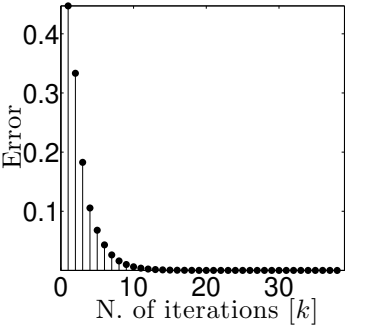
Golden section and quadratic interpolation (cont.)

Function $g(t)$ admits a global minimiser in $[6, 7]$

```
g = @(t) [-(7500*exp(t/3)) / (exp(t/3)+9)^2];

a = 0; b = 10;
tol = 1.0e-8; kmax = 100;

[tmin gmin,iter]= gSection(g,a,b,tol,kmax);
```

Golden section: 38 iterations, $t^* \approx 6.59$ and $g(t^*) \approx -208$





Golden section and quadratic interpolation (cont.)

The **quadratic interpolation method** is often used as alternative

- Let f be a continuous and convex function
- Let $x^{(0)}$, $x^{(1)}$ and $x^{(2)}$ be three distinct points

We build a sequence of points $x^{(k)}$ with $k \geq 3$ such that

- $x^{(k+1)}$ is the vertex (and thus the minimiser) of the parabola $p_2^{(k)}$
- $p_2^{(k)}$ interpolates f at (node points) $x^{(k)}$, $x^{(k-1)}$ and $x^{(k-2)}$

Golden section and quadratic interpolation (cont.)

Definition

For $k \geq 2$, the order-2 **Lagrange polynomial** at such nodes

$$\begin{aligned} p_2^{(k)}(x) = & f\left[x^{(k-2)}\right] + \\ & f\left[x^{(k-2)}, x^{(k-1)}\right]\left[x - x^{(k-2)}\right] + \\ & f\left[x^{(k-2)}, x^{(k-1)}, x^{(k)}\right]\left[x - x^{(k-2)}\right]\left[x - x^{(k-1)}\right] \end{aligned}$$

In the order-2 Lagrange polynomial $p_2^{(k)}$ for $k \geq 2$, consider the quantities

$$\begin{aligned} f\left[x_i, x_j\right] &= \frac{f(x_j) - f(x_i)}{x_j - x_i} \\ f\left[x_i, x_j, x_k\right] &= \frac{f\left[x_j, x_l\right] - f\left[x_i, x_j\right]}{x_l - x_i} \end{aligned} \tag{43}$$

The **Newton divided differences**

Golden section and quadratic interpolation (cont.)

Theorem

Consider $n+1$ distinct points

$$\left\{ \left[x_i, y_i(x_i) \right] \right\}_{n=0}^{n+1}$$

There exists only one polynomial $\Pi_n \in \mathbb{P}_n$ of order n or smaller that interpolates them

$$\Pi_n(x_i) = y_i, \quad \forall i = 0, \ldots, n$$

Π_n is said to be the **interpolating polynomial** of f, if $y_i = f(x_i)$

$\rightsquigarrow$ (for some continuous function f)

It is denoted by $\Pi_n f$

Golden section and quadratic interpolation (cont.)

Definition

Consider the components of the Lagrangian basis associated to nodes $\{x_i\}_{i=0}^n$

$$\varphi_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}, \quad i = 0, \ldots, n$$

They are polynomials such that $\{\varphi_i\}$ is the only basis of $\mathbb{P}_n$ satisfying

$$\varphi_i(x) \in \mathbb{P}_n, \varphi_i(x_j) = \delta_{ij} = \begin{cases} 1, \text{if } i = j \\ 0, \text{otherwise} \end{cases}$$

Golden section and quadratic interpolation (cont.)

Definition

The **Lagrange polynomial** is the interpolating polynomial $\Pi_n(x)$

$$\Pi_n(x) = \sum_{i=0}^{n} y_i \varphi_i(x)$$

It is expressed in Lagrange form, or wrt the Lagrange basis

$$\Pi_n(x_i) = \sum_{j=0}^{n} y_j \varphi_j(x_i) = \sum_{j=0}^{n} y_j \delta_{ij} = y_i, \quad i = 0, \ldots, n$$

Golden section and quadratic interpolation (cont.)

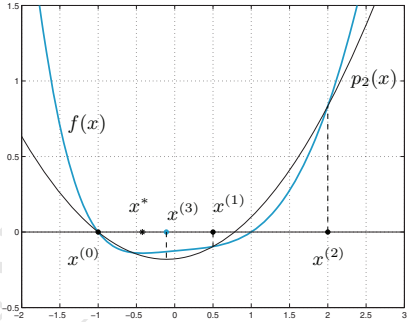
By solving the first-order equation $p_2'^{(k)}\left[x^{(k+1)}\right] = 0$, we get

$$x^{(k+1)} = \frac{1}{2}\left\{x^{(k-2)} + x^{(k-1)} - \frac{f\left[x^{(k-2)}, x^{(k-1)}\right]}{f\left[x^{(k-2)}, x^{(k-1)}, x^{(k)}\right]}\right\} \tag{44}$$

Next point in the sequence, by setting to zero the derivative of $p_2^{(k)}(x)$

We iterate until $\left|x^{(k+1)} - x^k\right| < \varepsilon$, for some tolerance $\varepsilon > 0$

Golden section and quadratic interpolation (cont.)



The first step of the quadratic interpolation method

Golden section and quadratic interpolation (cont.)

Example

$$g(t) = -7500 \frac{\exp(t/3)}{\left[\exp(t/3) + 9\right]^2}$$

`fminbnd` combines golden section and parabolic interpolation

```
g = @(t) [-(7500*exp(t/3))/(exp(t/3)+9)^2];

a = 0.0; b = 10.0;
tol = 1.0e-8; kmax = 100;

optionsQ = optimset('TolX', 1.0e-8)
[tminQ,gminQ,exitflagQ,outputQ] = fminbnd(g,a,b,optionsQ);
```

Golden section and quadratic interpolation (cont.)

Quadratic interpolation

8 iterations, $t^* \approx 6.59$ and $f(t^*) \approx -208$

- `optimset` sets the tolerance value in structure `optionsQ`
- `qminQ` contains the evaluation of f at the minimiser `tminQ`
- `exitflagQ` indicates the termination state
- `outputQ` has number of iterations and function evaluations

■

Golden section and quadratic interpolation (cont.)

The golden section and the quadratic interpolation method

- They are genuinely one-dimensional techniques
- They can be used to solve multidimensional optimisation problems
- They need be restricted to search along one dimensional directions

Nelder and Mead

Derivative-free methods

Nelder and Mead

Let $n > 1$ and $f : \mathbb{R}^n \to \mathbb{R}$ be a continuous function

Definition

The **n-simplex** with $n+1$ vertices $\mathbf{x}_i \in \mathbb{R}^n$ for $i = 0, \ldots, n$

$$S = \left\{ \mathbf{y} \in \mathbb{R}^n : \mathbf{y} = \sum_{i=0}^{n} \lambda_i \mathbf{x}_i, \text{with } \lambda_i \geq 0 : \sum_{i=0}^{n} \lambda_i = 1 \right\} \tag{45}$$

Intrinsic assumption: Linearly independent vectors $\{(\mathbf{x}_i - \mathbf{x}_0)\}_{i=1}^{n}$

S is a segment in $\mathbb{R}$, it is a triangle in $\mathbb{R}^2$ and a tetrahedron in $\mathbb{R}^3$

Nelder and Mead (cont.)

The **Nelder and Mead method** is a derivative-free minimisation method

- It generates a sequence of simplices $\{S^{(k)}\}_{k\geq 0}$ in $\mathbb{R}^n$

The simplices either run after or circumscribe the minimiser $\mathbf{x}^* \in \mathbb{R}^n$ of f

The method uses simple operations

1. Evaluations of f at the simplices' vertices
2. Geometrical transformations (reflections, expansions, contractions)

Nelder and Mead (cont.)

- At the k-th iteration, the 'worst' vertex of simplex $S^{(k)}$ is identified

$$\mathbf{x}_M^{(k)}, \text{ such that } f\left[\mathbf{x}_M^{(k)}\right] = \max_{0 \leq i \leq n} f\left[\mathbf{x}_i^{(k)}\right]$$

- $\mathbf{x}_M^{(k)}$ is substituted with a new point at which f takes a smaller value
- The new point is got by reflecting/expanding/contracting the simplex along the line joining $\mathbf{x}_M^{(k)}$ and the centroid of the other vertices

$$\mathbf{x}_c^{(k)} = \frac{1}{n} \sum_{\substack{i=0 \\ i \neq M}}^{n} \mathbf{x}_i^{(k)}$$

Nelder and Mead (cont.)

How to generate the initial simplex $S^{(0)}$

We take a point $\tilde{\mathbf{x}} \in \mathbb{R}^n$ and a positive real number η

Then, we set

$$\mathbf{x}_i^{(0)} = \tilde{\mathbf{x}} + \eta \mathbf{e}_i, \text{ with } i = 1, \ldots, n$$

$\{\mathbf{e}_i\}$ are the vectors of the standard basis in $\mathbb{R}^n$

Nelder and Mead (cont.)

While $k \geq 0$ and until convergence, select the 'worst' vertex of $S^{(k)}$

$$\mathbf{x}_M^{(k)} = \max_{0 \leq i \leq n} f\left[\mathbf{x}_i^{(k)}\right] \tag{46}$$

Then, replace it by a new point to form the new simplex $S^{(k+1)}$

Nelder and Mead (cont.)

The new point is chosen by firstly selecting

$$\begin{aligned} \mathbf{x}_m^{(k)} &= \min_{0 \leq i \leq n} f\left[\mathbf{x}_i^{(k)}\right] \\ \mathbf{x}_\mu^{(k)} &= \max_a f\left[\mathbf{x}_i^{(k)}\right] \end{aligned} \tag{47}$$

and secondly by defining the **centroid** point

$$\overline{\mathbf{x}}^{(k)} = \frac{1}{n} \sum_{\substack{i=0 \\ i \neq M}}^{n} \mathbf{x}_i^{(k)} \tag{48}$$

This is the centroid of hyperplane $H^{(k)}$ passing through vertices $\{\mathbf{x}_i\}_{\substack{i=0 \\ i \neq M}}^{n}$

Nelder and Mead (cont.)

Thirdly, compute reflection $\mathbf{x}_\alpha^{(k)}$ of $\mathbf{x}_M^{(k)}$ with respect to hyperplane $H^{(k)}$

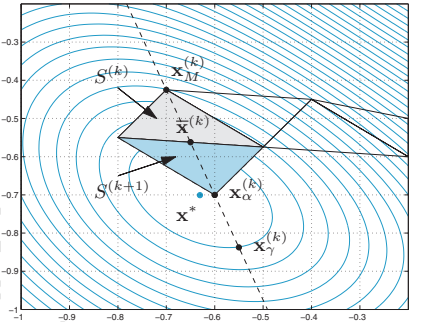
$$\mathbf{x}_\alpha^{(k)} = (1-\alpha)\overline{\mathbf{x}}^{(k)} + \alpha \mathbf{x}_M^{(\alpha)} \tag{49}$$

The **reflection coefficient** $\alpha < 0$ is typically set to be -1

Point $\mathbf{x}_\alpha^{(k)}$ lies on the straight line joining points $\overline{\mathbf{x}}^{(k)}$ and $\mathbf{x}_M^{(k)}$

- It is on the side of $\overline{\mathbf{x}}^{(k)}$, far from $\mathbf{x}_M^{(k)}$

Nelder and Mead (cont.)



$n = 2$, the centroid is midpoint of edge of $S^{(k)}$ opposite to $\mathbf{x}_M^{(k)}$

Nelder and Mead (cont.)

We fourthly compare $f\left[\mathbf{x}_\alpha^{(k)}\right]$ with f at the other vertices of the simplex

- Before accepting $\mathbf{x}_\alpha^{(k)}$ as the new vertex

We also try to move $\mathbf{x}_\alpha^{(k)}$ on the straight line joining $\overline{\mathbf{x}}^{(k)}$ and $\mathbf{x}_M^{(k)}$

To set the new simplex $S^{(k+1)}$

(1) If $f\left[\mathbf{x}_\alpha^{(k)}\right] < f\left[\mathbf{x}_m^{(k)}\right]$ (reflection produced a minimum), then

$$\mathbf{x}_\gamma^{(k)} = (1-\gamma)\overline{\mathbf{x}}^{(k)} + \gamma \mathbf{x}_M^{(k)}, \quad \text{with } \gamma < -1^4 \tag{50}$$

- Then, if $f\left[\mathbf{x}_\gamma^{(k)}\right] < f\left[\mathbf{x}_m^{(k)}\right]$, replace $\mathbf{x}_M$ by $\mathbf{x}_\gamma^{(k)}$
- Otherwise, $\mathbf{x}_M^{(k)}$ is replaced by $\mathbf{x}_\alpha^{(k)}$

We then proceed by incrementing k by one

[4] Typically, $\gamma = -2$

Nelder and Mead (cont.)

(2) If $f\left[\mathbf{x}_m^{(k)}\right] \leq f\left[\mathbf{x}_\alpha^{(k)}\right] < f\left[\mathbf{x}_\mu^{(k)}\right]$, then $\mathbf{x}_M^{(k)}$ is replaced by $\mathbf{x}_\alpha^{(k)}$

k is incremented by one

(3) If $f\left[\mathbf{x}_\mu^{(k)}\right] \leq f\left[\mathbf{x}_\alpha^{(k)}\right] < f\left[\mathbf{x}_M^{(k)}\right]$, we compute

$$\mathbf{x}_\beta^{(k)} = (1-\beta)\overline{\mathbf{x}}^{(k)} + \beta\mathbf{x}_\alpha^{(k)}, \quad \text{with } \beta > 0^5 \tag{51}$$

- Then, if $f\left[\mathbf{x}_\beta^{(k)}\right] > f\left[\mathbf{x}_M^{(k)}\right]$ define the vertices $S^{(k+1)}$

$$\mathbf{x}_i^{(k+1)} = \frac{1}{2}\left[\overline{\mathbf{x}}^{(k)} + \mathbf{x}_m^{(k)}\right] \tag{52}$$

- Otherwise $\mathbf{x}_M^{(k)}$ is replaced by $\mathbf{x}_\beta$

Then, we increment k

[5] Typically, $\beta = 1/2$

Nelder and Mead (cont.)

(4) If $f\left[\mathbf{x}_{\alpha}^{(k)}\right] > f\left[\mathbf{x}_{M}^{(k)}\right]$, we compute

$$\mathbf{x}_{\beta} = (1-\beta)\overline{\mathbf{x}}^{(k)} + \beta\mathbf{x}_{M}^{(k)}, \quad \text{with } \beta > 0 \tag{53}$$

- If $f\left[\mathbf{x}_{\beta}^{(k)}\right] > f\left[\mathbf{x}_{M}^{(k)}\right]$ define the vertices of $S^{(k+1)}$

$$\mathbf{x}_{i}^{(k+1)} = \frac{1}{2}[\overline{\mathbf{x}}^{(k)} + \mathbf{x}_{m}^{(k)}]$$

- Otherwise we replace $\mathbf{x}_{M}^{(k)}$ with $\mathbf{x}_{\beta}^{(k)}$

Then, we increment k

Nelder and Mead (cont.)

When the stopping criterion $\max_{i=0,\ldots,n} ||\mathbf{x}_i^{(k)} - \mathbf{x}_m^{(k)}||_\infty < \varepsilon$ is met

$\rightsquigarrow$ $\mathbf{x}_m^{(k)}$ is retained as approximation of the minimiser

Convergence is guaranteed in very special cases only

Stagnation may occur, algorithm needs to be restarted

- The algorithm is nevertheless quite robust
- It is efficient for small dimensional problems
- Convergence rate depends on initial simplex

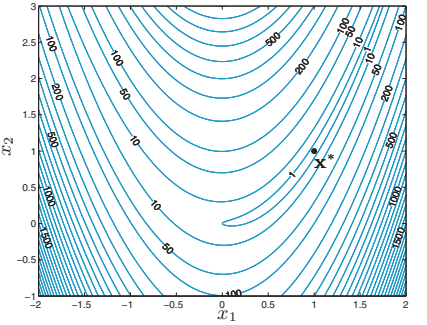
Nelder and Mead (cont.)

Example

The Rosenbrock function

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

The global minimum is at $\mathbf{x}^* = (1, 1)$, and variation around $\mathbf{x}^*$ is low



Nelder and Mead (cont.)

The simplex method

The M-command is `fminsearch`

```
x_0 = [-1.2,+1.0];

fun = @(x) (1-x(1))^2 + 100*(x(2)-x(1)^2)^2;

xstar = fminsearch(fun,x_0)

xstar =
1.000022021783570  1.000042219751772
```

To obtain additional information on the minimum value of f

```

[xstar,fval,exitflag,output] = fminsearch(fun,x_0)
```

Unconstrained optimisation

Constrained optimisation

We discuss two strategies for solving constrained minimisation problems

The penalty method

- Problems with both equality and inequality constraints

The augmented Lagrangian method

- Problems with equality constraints only

The two methods allow the solution of relatively simple problems

- Basic tools for more robust and complex algorithms

Constrained optimisation (cont.)

Definition

Let $f : \mathcal{R}^n \to \mathcal{R}$ with $n \geq 1$ be a **cost** or **objective function**

The **constrained optimisation** problem

$$\min_{\mathbf{x} \in \Omega \subset \mathcal{R}^n} f(\mathbf{x}) \tag{54}$$

Ω is a closed subset determined by equality or inequality constraints

Given functions $h_i : \mathcal{R}^n \to \mathcal{R}$, for $i = 1, \ldots, p$

$$\rightsquigarrow \quad \Omega = \left\{ \mathbf{x} \in \mathcal{R}^n : h_i(\mathbf{x}) = 0, \text{ for } i = 1, \ldots, p \right\} \tag{55}$$

Given functions $g_j : \mathcal{R}^n \to \mathcal{R}$, for $j = 1, \ldots, g$

$$\rightsquigarrow \quad \Omega = \left\{ \mathbf{x} \in \mathcal{R}^n : g_j(\mathbf{x}) \geq 0, \text{ for } j = 1, \ldots, q \right\} \tag{56}$$

p and q are natural numbers

Constrained optimisation (cont.)

More generally,

$$\min_{\mathbf{x}\in\Omega\subset\mathcal{R}^n} f(\mathbf{x}) \tag{57}$$

Ω a closed subset determined by both equality and inequality constraints

$$\Omega = \left\{ \mathbf{x} \in \mathcal{R}^n : h_i(\mathbf{x}) = 0 \text{ for } \underbrace{i \in \mathcal{I}_h}_{i=1,\ldots,p} \text{ and } g_j(\mathbf{x}) \geq 0 \text{ for } \underbrace{j \in \mathcal{I}_g}_{j=1,\ldots,q} \right\}$$

Constrained optimisation (cont.)

Definition

Let $f : \mathcal{R}^n \to \mathcal{R}$ with $n \geq 1$ be a **cost** or **objective function**

The general **constrained optimisation** problem

$$\begin{aligned} &\min_{\mathbf{x} \in \mathcal{R}^n} f(\mathbf{x}) \\ &\text{subjected to} \\ &h_i(\mathbf{x}) = 0, \quad \text{for all } i \in \mathcal{I}_h \\ &g_j(\mathbf{x}) \geq 0, \quad \text{for all } j \in \mathcal{I}_g \end{aligned} \tag{58}$$

The two sets $\mathcal{I}_h = \{1, 2, \ldots, p\}$ and $\mathcal{I}_g = \{1, 2, \ldots, q\}$

- $\rightsquigarrow$ In Equation (56), we used $\mathcal{I}_h = \emptyset$
- $\rightsquigarrow$ In Equation (55), we used $\mathcal{I}_g = \emptyset$

Constrained optimisation (cont.)

Suppose that $f \in \mathcal{C}^1(\mathcal{R}^n)$ and that h_i and g_j are class $\mathcal{C}^1(\mathcal{R}^n)$, for all i, j

Points $\mathbf{x} \in \Omega \subset \mathcal{R}$ that satisfy all the constraints are **feasible points**

$\rightsquigarrow$ The closed subset Ω is the set of all feasible points

---

Consider a point $\mathbf{x}^* \in \Omega \subset \mathcal{R}^n$,

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega \tag{59}$$

Point $\mathbf{x}^*$ is said to be a **global minimiser** for the problem

Consider a point $\mathbf{x}^* \in \Omega \subset \mathcal{R}^n$,

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in B_r(\mathbf{x}^*) \cap \Omega \tag{60}$$

Point $\mathbf{x}^*$ is said to be a **local minimiser** for the problem

$\rightsquigarrow$ $B_r(\mathbf{x}^*) \in \mathcal{R}^n$ is a ball centred in $\mathbf{x}^*$, radius $r > 0$

Constrained optimisation (cont.)

A constraint is said to be **active** at $\mathbf{x} \in \Omega$ if it is satisfied with equality

- Active constraints at $\mathbf{x}$ are the $h_i(\mathbf{x}) = 0$ and the $g_j(\mathbf{x}) = 0$

Let Ω be a non-empty, bounded and closed set in $\mathcal{R}^n$

Weierstrass guarantees existence of a maximum and a minimum for f in Ω

$\rightsquigarrow$ The general constrained optimisation problem admits a solution

Constrained optimisation (cont.)

Example

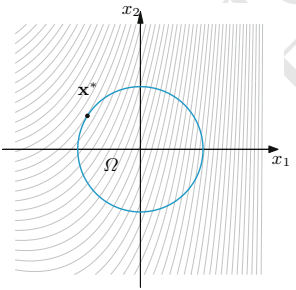
Consider the minimisation of function $f(\mathbf{x})$ under equality constraint $h_1(\mathbf{x})$

Let

$$f(\mathbf{x}) = 3/5x_1^2 + 1/2x_1x_2 - x_2 + 3x_1$$

Let

$$h_1(\mathbf{x}) = x_1^2 + x_2^2 - 1 = 0$$



Global minimiser $\mathbf{x}^*$ constrained to Ω

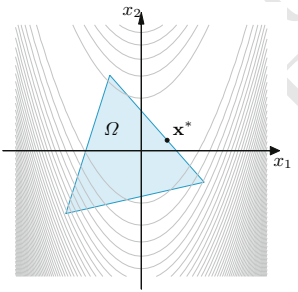
- Contour lines of the cost $f(\mathbf{x})$
- Admissibility set $\Omega \in \mathcal{R}^2$

Constrained optimisation (cont.)

Example

Minimise $f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$, under inequality constraints

$$g_1(\mathbf{x}) = -34x_1 - 30x_2 + 19 \geq 0$$
$$g_2(\mathbf{x}) = +10x_1 - 05x_2 + 11 \geq 0$$
$$g_3(\mathbf{x}) = +03x_1 + 22x_2 + 08 \geq 0$$



Global minimiser $\mathbf{x}^*$ constrained to Ω

- Contour lines of the cost $f(\mathbf{x})$
- Admissibility set $\Omega \in \mathcal{R}^2$

Constrained optimisation (cont.)

Definition

Strongly convexity

The condition for function $f : \Omega \subseteq \mathcal{R}^n \to \mathcal{R}$ to be **strongly convex** in Ω

f is strongly convex if $\exists \rho > 0$ such that $\forall(\mathbf{x}, \mathbf{y}) \in \Omega$ and $\forall \alpha \in [0, 1]$

$$\underbrace{f\left[\alpha\mathbf{x} + (1-\alpha)\mathbf{y}\right] \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y})}_{\text{Convexity}} - \alpha(1-\alpha)\rho||\mathbf{x} - \mathbf{y}||^2 \tag{61}$$

Strong convexity reduces to the usual convexity when $\rho = 0$

Constrained optimisation (cont.)

Proposition

Optimality conditions

Let $\Omega \subset \mathcal{R}^n$ be a convex set and let $\mathbf{x}^* \in \Omega$ be such that $f \in \mathcal{C}^1\big[B_r(\mathbf{x}^*)\big]$

Suppose that $\mathbf{x}^*$ is a local minimiser for constrained minimisation,

$$\nabla f(\mathbf{x}^*)^\top (\mathbf{x} - \mathbf{x}^*) \geq 0, \quad \forall \mathbf{x} \in \Omega \tag{62}$$

If f is convex in Ω and (62) is satisfied, then $\mathbf{x}^*$ is a global minimiser

Suppose that we require Ω to be closed and f to be strongly convex

$\rightsquigarrow$ It can be shown that the minimiser $\mathbf{x}^*$ is also unique

Constrained optimisation (cont.)

There are many algorithms for solving constrained minimisation problems

Many search for the stationary points of the **Lagrangian function**

⇝ The **KKT** or **Karush-Kuhn-Tucker points**

Definition

$$\min_{\mathbf{x}\in\Omega} f(\mathbf{x})$$

The **Lagrangian function** associated with the constrained minimisation

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) - \sum_{i\in\mathcal{I}_h} \lambda_i h_i(\mathbf{x}) - \sum_{j\in\mathcal{I}_g} \mu_j g_j(\mathbf{x}) \tag{63}$$

$\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are **Lagrangian multipliers**

⇝ $\boldsymbol{\lambda} = (\lambda_i)$, for $i \in \mathcal{I}_h$

⇝ $\boldsymbol{\mu} = (\mu_i)$, for $j \in \mathcal{I}_g$

They are (weights) associated with the equality and inequality constraints

Constrained optimisation (cont.)

Definition

Karush-Kuhn-Tucker conditions

A point $\mathbf{x}^*$ is said to be a KKT point for $\mathcal{L}$ if there exist $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that the triplet $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ satisfies the Karush-Kuhn-Tucker conditions

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \nabla f(\mathbf{x}^*) - \sum_{i \in \mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*) - \sum_{j \in \mathcal{I}_g} \mu_j^* \nabla g_j(\mathbf{x}^*) = \mathbf{0}$$

$$\begin{aligned} h_i(\mathbf{x}^*) &= 0, \quad \forall i \in \mathcal{I}_h \\ g_i(\mathbf{x}^*) &\geq 0, \quad \forall j \in \mathcal{I}_g \\ \mu_j^* &\geq 0, \quad \forall j \in \mathcal{I}_g \\ \mu_j^* g_j(\mathbf{x}^*) &= 0, \quad \forall j \in \mathcal{I}_g \end{aligned}$$

Constrained optimisation (cont.)

Let $\mathbf{x}$ be some given point

Consider $\nabla h_i(\mathbf{x})$ and $\nabla g_j(\mathbf{x})$ associated with active constraints in $\mathbf{x}$

- Suppose that these gradients are linearly independent

Linear independence (**constraint**) **qualification** (**LI**(**C**)**Q**) in $\mathbf{x}$

Constrained optimisation (cont.)

Theorem

First-order KKT conditions

Let $\mathbf{x}^*$ be a local minimum for the constrained problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x})$$

subjected to

$$h_i(\mathbf{x}) = 0, \forall i \in \mathcal{I}_h$$
$$g_j(\mathbf{x}) \geq 0, \forall j \in \mathcal{I}_g$$

Let functions f, h_i and g_j be $\mathcal{C}^1(\Omega)$ and let the constraints be LIQ in $\mathbf{x}^*$

There exist $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^*$ such that $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ is a KKT point

Constrained optimisation (cont.)

In the absence of inequality constraints, the Lagrangian takes the form

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i \nabla h_i(\mathbf{x}^*)$$

The KKT conditions are known as **Lagrange** (**necessary**) **conditions**

$$\begin{aligned} \nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \nabla f(\mathbf{x}^*) - \sum_{i \in \mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*) = \mathbf{0} \\ h_i(\mathbf{x}^*) = 0, \forall i \in \mathcal{I}_h \end{aligned} \tag{64}$$

Constrained optimisation (cont.)

Remark

Sufficient conditions for a KKT point $\mathbf{x}$ to be a minimiser of f in Ω

$\rightsquigarrow$ Knowledge about the Hessian of the Lagrangian is required

Alternatively, we need strict convexity hypothesis on f and the constraints

In general, it is possible to reformulate a constrained optimisation problem

- As an unconstrained optimisation problem

The idea is to replace the original problem by a sequence of subproblems in which the constraints are represented by terms added to the objective

$\rightsquigarrow$ **(Quadratic) Penalty function**

$\rightsquigarrow$ **Augmented Lagrangian**

The penalty method

Constrained optimisation

The penalty method

Consider solving the general constrained optimisation problem

$$\min_{\mathbf{x}\in\mathcal{R}^n} f(\mathbf{x})$$
$$\text{subjected to}$$
$$h_i(\mathbf{x}) = 0, \quad \forall i \in \mathcal{I}_h$$
$$g_j(\mathbf{x}) \geq 0, \quad \forall j \in \mathcal{I}_g$$

We reformulate it as an unconstrained optimisation problem

Definition

The modified **penalty function**, for a fixed **penalty parameter** $\alpha > 0$

$$\mathcal{P}_\alpha(\mathbf{x}) = f(\mathbf{x}) + \frac{\alpha}{2}\sum_{i\in\mathcal{I}_h} {h_i}^2(\mathbf{x}) + \frac{\alpha}{2}\sum_{j\in\mathcal{I}_g}\Big[\max\{-g_j(\mathbf{x}), 0\}\Big]^2 \tag{65}$$

The method adds a multiple of the square of the violation of each constraint

- Terms are zero when $\mathbf{x}$ does not violate the constrain

The penalty method (cont.)

$$\mathcal{P}_\alpha(\mathbf{x}) = f(\mathbf{x}) + \frac{\alpha}{2}\sum_{i\in\mathcal{I}_h} {h_i}^2(\mathbf{x}) + \frac{\alpha}{2}\sum_{j\in\mathcal{I}_g}\Big[\max\{-g_j(\mathbf{x}), 0\}\Big]^2$$

By making the coefficients larger, we penalise violations more severely

- This forces the minimiser closer to the feasible region

Consider the situation in which the constraints are not satisfied at $\mathbf{x}$

- The sums quantify how far point $\mathbf{x}$ is from the feasibility set Ω
- A large α heavily penalises such a violation

If $\mathbf{x}^*$ is a solution to the constrained problem, $\mathbf{x}^*$ is a minimiser of $\mathcal{P}$

Conversely, under some regularity hypothesis for f, h_i and g_i,

$$\lim_{\alpha\to\infty} \mathbf{x}^*(\alpha) = \mathbf{x}^*,$$

$\mathbf{x}^*(\alpha)$ denotes the minimiser of $\mathcal{P}_\alpha(\mathbf{x})$

As $\alpha >> 1$, $\mathbf{x}^*(\alpha)$ is a good approximation of $\mathbf{x}^*$

The penalty method (cont.)

Example

Consider the minimisation of function $f(\mathbf{x})$ under equality constraint $h_1(\mathbf{x})$

Let

$$f(\mathbf{x}) = x_1 + x_2$$

Let

$$h_1(\mathbf{x}) = x_1^2 + x_2^2 - 2 = 0$$

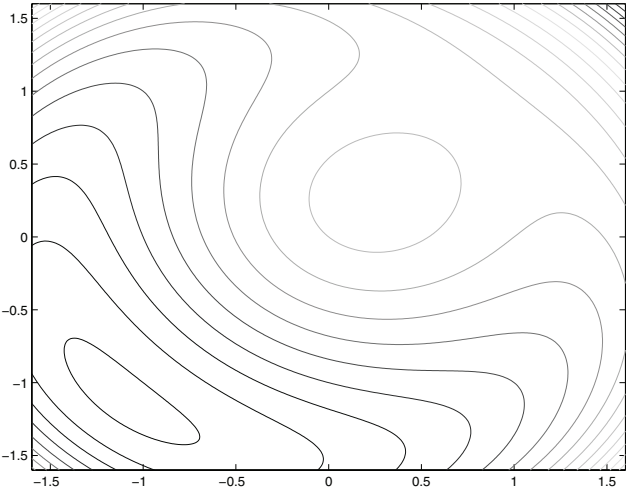
Consider the quadratic penalty function

$$\mathcal{P}_\alpha(\mathbf{x}) = (x_1 + x_2) + \frac{\alpha}{2}(x_1^2 + x_2^2 - 2)^2$$

The minimiser is $(-1, -1)'$

The penalty method (cont.)

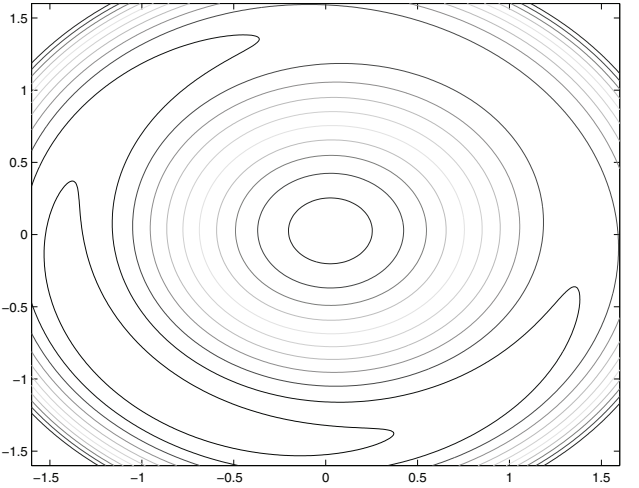
The plot of the contour of the penalty function for $\alpha = 1$



There is a local minimiser near $(0.3, 0.3)'$

The penalty method (cont.)

The plot of the contour of the penalty function for $\alpha = 10$



Points outside the feasible region suffer a much greater penalty

The penalty method (cont.)

Not advised (instability) to minimise $\mathcal{P}_\alpha(\mathbf{x})$ directly for large values of α

Rather, consider an increasing and unbounded sequence of parameters $\{\alpha_k\}$

- For each α_k, calculate an approximation $\mathbf{x}^{(k)}$ of the solution $\mathbf{x}^*(\alpha_k)$ to the unconstrained optimisation problem $\min_{\mathbf{x}\in\mathcal{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$

$$\mathbf{x}^{(k)} = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$$

- At step k, α_{k+1} is a chosen as a function of α_k (say, $\alpha_{k+1} = \delta\alpha_k$, for $\delta \in [1.5, 2]$) and $\mathbf{x}^{(k)}$ is used to initialise the minimisation at step $k+1$

In the first iterations there is no reason to believe that the solution to $\min_{\mathbf{x}\in\mathcal{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$ should resemble the correct solution to the original problem

- This supports the idea of searching for an inexact solution to $\min_{\mathbf{x}\in\mathcal{R}^n} \mathcal{P}_{\alpha_k}(\mathbf{x})$ that differs from the exact one, $\mathbf{x}^{(k)}$, a small ε_k

The penalty method (cont.)


```
1  % PENALTY Constrained optimisation with penalty function
2  % [X,ERR,K]=PFUNCTION(F,GRAD_F,H,GRAD_H,G,GRAD_G,X_0,TOL,...
3  %                     KMAX,KMAXD,TYP)
4  %  Approximate a minimiser of the cost function F
5  %  under constraints H=0 and G>=0
6  %
7  % X0 is initial point, TOL is tolerance for stop check
8  % KMAX is the maximum number of iterations
9  % GRAD_F, GRAD_H, and GRAD_G are the gradients of F, H, and G
10 % H and G, GRAD_H and GRAD_G can be initialised to []
11 %
12 % For TYP=0 solution by FMINSEARCH M-function
13 %
14 % For TYP>0 solution by a DESCENT METHOD
15 %  KMAXD is maximum number of iterations
16 %  TYP is the choice of descent directions
17 %  TYP=1 and TYP=2 need the Hessian (or an approx. at k=0)
18 %  [X,ERR,K]=PFUNCTION(F,GRAD_F,H,GRAD_H,G,GRAD_G,X_0,TOL,...
19 %                      KMAX,KMAXD,TYP,HESS_FUN)
20 %  For TYP=1 HESS_FUN is the function handle associated
21 %  For TYP=2 HESS_FUN is a suitable approx. of Hessian at k=0
```

The penalty method (cont.)


```
function [x,err,k]=pFunction(f,grad_f,h,grad_h,g,grad_g,...
                             x_0,tol,kmax,kmaxd,typ,varargin)

xk=x_0(:); mu_0=1.0;

if typ==1; hess=varargin{1};
 elseif typ==2; hess=varargin{1};
 else; hess=[]; end
if ~isempty(h), [nh,mh]=size(h(xk)); end
if ~isempty(g), [ng,mg]=size(g(xk)); end

err=1+tol; k=0; muk=mu_0; muk2=muk/2; told=0.1;

while err>tol && k<kmax
 if typ==0
  options=optimset('TolX',tol);
  [x,err,kd]=fminsearch(@P,xk,options); err=norm(x-xk);
 else
  [x,err,kd]=dScent(@P,@grad_P,xk,told,kmaxd,typ,hess);
  err=norm(grad_P(x));
 end

 if kd<kmaxd; muk=10*muk; muk2=0.5*muk;
 else muk=1.5*muk; muk2=0.5*muk; end

 k=1+k; xk=x; told=max([tol,0.10*told]);
end
```

The penalty method (cont.)


```
function y=P(x) % This function is nested inside pFunction

y=fun(x);
if ~isempty(h); y=y+muk2*sum((h(x)).^2); end
if ~isempty(g); G=g(x);
 for j=1:ng
  y=y+muk2*max([-G(j),0])^2;
 end
end
```

```
function y=grad_P(x) % This function is nested in pFunction

y=grad_fun(x);
if ~isempty(h), y=y+muk*grad_h(x)*h(x); end
if ~isempty(g), G=g(x); Gg=grad_g(x);
 for j=1:ng
  if G(j)<0
   y=y+muk*Gg(:,j)*G(j);
  end
 end
end
```

The augmented Lagrangian

Constrained optimisation

The augmented Lagrangian

Consider a minimisation problem with equality constraints only ($\mathcal{I}_g = \emptyset$)

$$\min_{\mathbf{x} \in \mathcal{R}^n} f(\mathbf{x})$$

subjected to

$$h_i(\mathbf{x}) = 0, \forall i \in \mathcal{I}_h$$

Definition

Define the **augmented Lagrangian** objective function

$$\mathcal{L}_A(\mathbf{x}, \boldsymbol{\lambda}, \alpha) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x}) + \alpha/2 \sum_{i \in \mathcal{I}_h} {h_i}^2(\mathbf{x}) \tag{66}$$

$\alpha > 0$ is a suitable coefficient

The augmented Lagrangian (cont.)

$$\mathcal{L}_A(\mathbf{x}, \boldsymbol{\lambda}, \alpha) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x}) + \alpha/2 \sum_{i \in \mathcal{I}_h} {h_i}^2(\mathbf{x})$$

Constrained optimisation using the augmented Laplacian is iterative

α_0 and $\boldsymbol{\lambda}^{(0)}$ are set arbitrarily, then build a sequence of parameters $\alpha_k \to \infty$

$\alpha_k \to \infty$ is st $\{(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)})\}$ converges to a KKT point for the Lagrangian

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_{i \in \mathcal{I}_h} \lambda_i h_i(\mathbf{x})$$

At the k-th iteration, for a given α_k and for a given $\boldsymbol{\lambda}^{(k)}$, we compute

$$\mathbf{x}^{(k)} = \arg\min_{\mathbf{x} \in \mathcal{R}^n} \mathcal{L}_A\left[\mathbf{x}, \boldsymbol{\lambda}^{(k)}, \alpha_k\right] \tag{67}$$

The augmented Lagrangian (cont.)

We get multipliers $\boldsymbol{\lambda}^{(k+1)}$ from the gradient of the augmented Lagrangian

- We set it to be equal to zero

$$\nabla_{\mathbf{x}} \mathcal{L}_A \left[\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \alpha_k\right] = \nabla f\left[\mathbf{x}^{(k)}\right] - \sum_{i \in \mathcal{I}_h} \left\{ \lambda_i^{(k)} - \alpha_k h_i\left[\mathbf{x}^{(k)}\right] \right\} \nabla h_i\left[\mathbf{x}^{(k)}\right]$$

By comparison with optimality condition

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \nabla f(\mathbf{x}^*) - \sum_{i \in \mathcal{I}_h} \lambda_i^* \nabla h_i(\mathbf{x}^*) = \mathbf{0}$$

$$h_i(\mathbf{x}^*) = 0, \quad \forall i \in \mathcal{I}_h$$

We identify $\lambda_i^{(k)}$ as $\lambda_i^{(k)} - \alpha_k h_i\left[\mathbf{x}^{(k)}\right] \simeq \lambda_i^*$

We thus define,

$$\lambda_i^{(k+1)} = \lambda_i^{(k)} - \alpha_k h_i\left[\mathbf{x}^{(k)}\right] \tag{68}$$

We get $\mathbf{x}^{(k+1)}$ by solving with k replaced by $k+1$

$$\mathbf{x}^{(k)} = \arg\min_{\mathbf{x} \in \mathcal{R}^n} \mathcal{L}_A\left[\mathbf{x}, \boldsymbol{\lambda}^{(k)}, \alpha_k\right]$$

The augmented Lagrangian (cont.)

Given α_0 (typically, $\alpha_0 = 1$), given ε_0 (typically $\varepsilon_0 = 1/10$), given $\overline{\varepsilon} > 0$, given $\mathbf{x}_0^{(0)} \in \mathbb{R}^n$ and given $\boldsymbol{\lambda}_0^{(0)} \in \mathbb{R}^p$, for $k = 0, 1, \ldots$ until convergence

Pseudo-code

Compute an approximated solution

$$\mathbf{x}^{(k)} = \arg\min_{\mathbf{x} \in \mathbb{R}^n} \mathcal{L}_A\left[\mathbf{x}, \boldsymbol{\lambda}^{(k)}, \alpha_k\right]$$

(Using the initial point $\mathbf{x}_0^{(0)}$ and tolerance ε_k)

If $\left|\left|\nabla_{\mathbf{x}} \mathcal{L}_A\left[\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \alpha_k\right]\right|\right| \leq \overline{\varepsilon}$
  Set $\mathbf{x}^* = \mathbf{x}^{(k)}$ (convergence)
else
  Compute $\lambda_i^{(k+1)} = \lambda_i^{(k)} - \mu_k h_i\left[\mathbf{x}^{(k)}\right]$
  Choose $\alpha_{k+1} > \alpha_k$
  Choose $\varepsilon_{k+1} < \varepsilon_k$
  Set $\mathbf{x}_0^{(k+1)} = \mathbf{x}^{(k)}$
Endif

The augmented Lagrangian (cont.)

The implementation of the algorithm

```
% ALGRNG Constrained optimisation with augmented Lagrangian
% [X,ERR,K]=ALGRNG(F,GRAD_F,H,GRAD_H,X_0,LAMBDA_0,...
%                  TOL,KMAX,KMAXD,TYP)
%  Approximate a minimiser of the cost function F
%  under equality constraints H=0
%
% X_0 is initial point, TOL is tolerance for stop check
% KMAX is the maximum number of iterations
% GRAD_F and GRAD_H are the gradients of F and H
%
% For TYP=0 solution by FMINSEARCH M-function
% FOR TYP>0 solution by a DESCENT METHOD
%  KMAXD is maximum number of iterations
%  TYP is the choice of descent directions
%  TYP=1 and TYP=2 need the Hessian (or an approx. at k=0)
```

The augmented Lagrangian (cont.)


```
function [x,err,k]=aLgrng(f,grad_f,h,grad_h,x_0,lambda_0,...
                          tol,kmax,kmaxd,typ,varargin)

mu_0=1.0;

if typ==1; hess=varargin{1};
 elseif typ==2; hess=varargin{1};
 else; hess=[]; end

err=1+tol+1; k=0; xk=x_0(:); lambdak=lambda_0(:);

if ~isempty(h); [nh,mh]=size(h(xk)); end

muk=mu_0; muk2=muk/2; told=0.1;

while err>tol && k<kmax
 if typ==0
  options=optimset ('TolX',told);
  [x,err,kd]=fminsearch(@L,xk,options); err=norm(x-xk);
 else
  [x,err,kd]=descent(@L,@grad_L,xk,told,kmaxd,typ,hess);
  err=norm(grad_L(x));
 end

 lambdak=lambdak-muk*h(x);
 if kd<kmaxd; muk=10*muk; muk2=0.5*muk;
 else muk=1.5*muk; muk2=0.5*muk; end

 k=1+k; xk=x; told=max([tol,0.10*told]);
```

The augmented Lagrangian (cont.)

```
function y=L(x) % This function is nested inside aLgrng

y=fun(x);
if ~isempty(h)
 y=y-sum(lambdak'*h(x))+muk2*sum((h(x)).^2);
end
```

```
function y=grad_L(x) % This function is nested inside aLgrng

y=grad_fun(x);
if ~isempty(h)
 y=y+grad_h(x)*(muk*h(x)-lambdak);
end
```

`lambda_0` contains the initial vector $\boldsymbol{\lambda}^{(0)}$ of Lagrange multipliers

The augmented Lagrangian (cont.)

Example

```
fun = @(x) 0.6*x(1).^2 + 0.5*x(2).*x(1) - x(2) + 3*x(1);
grad_fun = @(x) [1.2*x(1) + 0.5*x(2) + 3; 0.5*x(1) - 1];

h = @(x) x(1).^2 + x(2).^2 - 1;
grad_h = @(x) [2*x(1); 2*x(2)];

x_0 = [1.2,0.2]; tol = 1e-5; kmax = 500; kmaxd = 100;
p=1; % The number of equality constraints
lambda_0 = rand(p,1); typ=2; hess=eye(2);

[xmin,err,k] = aLagrange(fun,grad_fun,h,grad_h,x_0,...
                         lambda_0,tol,kmax,kmax,typ,hess)
```

Stopping criterion: A tolerance set 10^{-5}

The unconstrained minimisation by quasi-Newton descent directions

- (with `typ=2` and `hess=eye(2)`)