Chapter 7

Minimisation and Maximisation of Functions

Outline of Section

- One–dimensional minimisation
- Multi-dimensional minimisation
- Monte-Carlo minimisation
- Using minimisation to solve matrix equations

We first focus on **iterative methods** for finding local or global minima (maxima) of a function of one variable $f(x)$ and of many independent variables $f(x_1, x_2, \ldots, x_N)$, written more succinctly as $f(\vec{x})$ where $\vec{x}$ is an N-dimensional vector. We also discuss Monte Carlo methods for function minimisation, borrowing the principles introduced in the previous chapter.

The function to be minimised is often called the "cost function" (and not just in economics), in which case the problem can be considered as minimising the "cost" of the system. It can also be called the "loss function". Both the position of a minimum $\vec{x}_*$ and the value of the function there $f(\vec{x}_*)$ may be of interest. The function can be non-linear. The methods covered will find minima; maxima can be found by applying these methods to $F(\vec{x}) = -f(\vec{x})$. Root-finding of non-linear functions $f(x)$ was introduced in section 2.3. In principle roots of a non-linear function involving many variables $f(\vec{x})$ can be found by finding minima $\vec{x}_*$ of $|f(\vec{x})|^2$ where $f(\vec{x}_*) = 0$. However, root-finding is a significant area of study in its own right and this s not a generally-appropriate method.

Interestingly, it turns out that solving a matrix equation can be formulated as a minimisation problem, as we shall see later. Some of the most efficient iterative matrix solvers are based on this approach and converge much faster than Jacobi, Gauss-Seidel and SOR.

We will deal with **unconstrained** minimisation here. ***Constrained*** optimisation—where further constraints are placed on the independent variables or the values of the cost function—are not addressed here. Many of the concepts introduced here are, however, applicable in extended form when performing constrained minimisation.

We will also limit ourselves to real valued, scalar functions $f(\vec{x}) \in \mathbb{R}$ and real valued variables; $x \in \mathbb{R}$ and $\vec{x} \in \mathbb{R}^N$. It is usually not difficult to generalise the methods here to these cases.

One setting where finding the minimum of a complicated non-linear function occurs is the optimisation of parameters in a model when fitting to data. Consider the observed data d_i^{obs} with errors σ_i in figure 7.1. The model we would like to fit is a linear one

$$d_i^{\text{model}} = a + \alpha\, X_i\,. \tag{7.1}$$

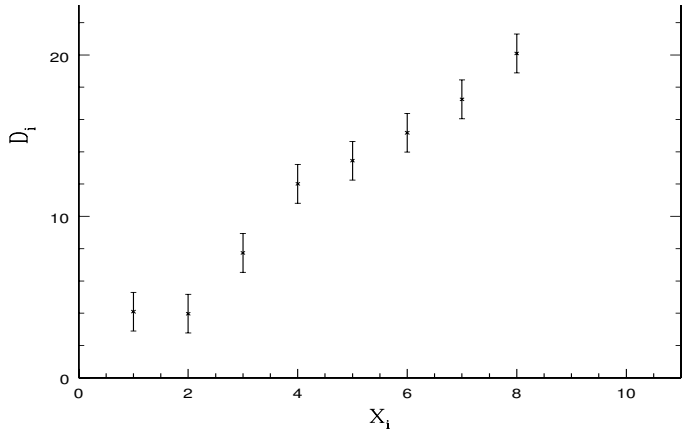

Figure 7.1: Some data with errors.

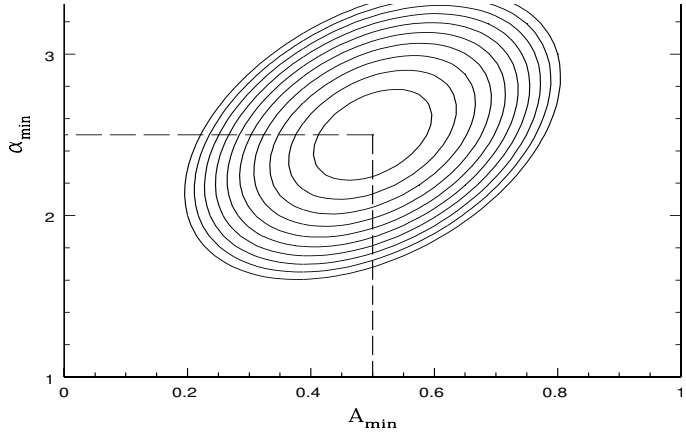

Figure 7.2: Contour plot of the χ^2 around the minimum. $a_{\min}$ and $\alpha_{\min}$ are the maximum likelihood values for the parameters in the model. The curvature around the minimum is related to the error in the parameters.

The normal procedure is to find the minimum χ^2 with respect to the parameters a and α

$$\chi^2(a,\alpha) = \sum_{i=1}^{N^{\text{data}}} \frac{\left(d_i^{\text{obs}} - d_i^{\text{model}}\right)^2}{\sigma_i^2}\,. \tag{7.2}$$

The assumption here is that the data are distributed as independent Gaussian random numbers and we are **maximising the likelihood**

$$L(a,\alpha) \propto e^{-\frac{1}{2}\chi^2(a,\alpha)}, \tag{7.3}$$

which is equivalent to finding the minimum in the χ^2. The result of the minimisation may look something like figure 7.2. In practice, the data that need to be fitted to and the models that are used are significantly more compliated than this example; cases with hundreds of free parameters (equivalently, degrees of freedom or dimensions) and numerous types of experimental data are common.

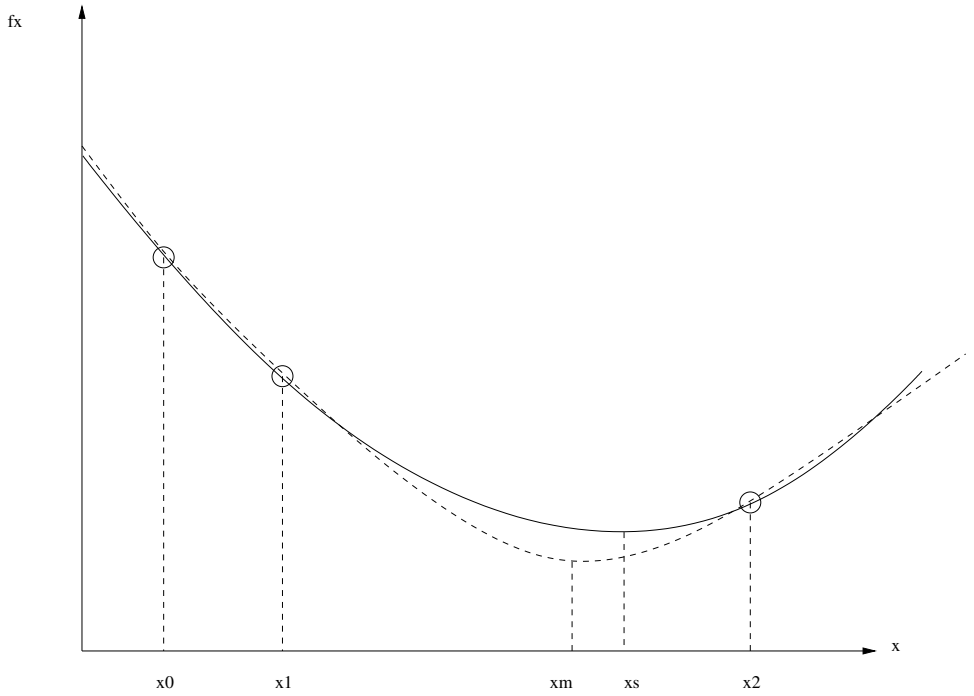

Figure 7.3: **The parabolic minimum search.** The function being minimised $f(x)$ is the dashed curve. A parabola (solid curve) is fit to the three points at x_0, x_1, x_2 and the minimum $x = x_3$ is found. Then the highest of the four points x_0, x_1, x_2, and x_3 is discarded and the method repeated. The minima of successive parabola approximations converge to the minimum of the function $(x_*, f(x_*))$.

7.1 One–dimensional minimisation

Naively we might think that finding the minima of a function is trivial. All we need to do is differentiate the function and find the roots of the resulting equation. However there are many cases where the analytical method is not applicable, e.g., where the derivative in the function is discontinuous or the function has a regular (possibly infinite) set of minima, when all we are interested in is the single global minimum—which we cannot find easily using this method. For a function of many, many variables, it can be too computationally costly to evaluate $\partial f/\partial x_i$ for all variables or simply too laborious to implement them all in code!

In practice this is often the case when evaluating a function of some scattered data where the analytical dependence of the likelihood with respect to the parameters is not known *a priori* and must be evaluated numerically. In this case we have to choose between a method that searches for minima/maxima using just the function values or (more optimal) ones that also use approximations for the derivatives of the function.

Parabolic Method

Functions almost always approximate a parabola near a minimum (except for some pathological cases). A very simple method to find a local minimum of a function exploits this. The method works when the curvature of $f(x)$ is positive everywhere in the interval we are looking at. The parabolic method consists of selecting three points along the function $f(x)$, e.g., x_0, x_1, and x_2 where

$$f(x_0) = y_0, \quad f(x_1) = y_1, \quad \text{and} \quad f(x_2) = y_2 \tag{7.4}$$

and then fitting $P_2(x)$, the 2nd-order Lagrange polynomial, through the points (see section 4.1). The location of the minimum of the interpolating parabola $P_2(x)$ is found using equation (7.6) below; this value is x_3. This procedure is illustrated in fig. 7.3. We then keep the three lowest

points out of $f(x_0)$, $f(x_1)$, $f(x_2)$, and $f(x_3)$ and repeat the procedure. At each successive iteration the point x_3 will converge towards x_* where the minimum of the function $f(x)$ is located. The iterations can be stopped once the change in x_3 is less than a desired value.

The quadratic interpolating the 3 points is given by

$$P_2(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2, \tag{7.5}$$

[c.f. equation (4.3)]. We want to find the minimum of $P_2(x)$, which will be a better estimate of the minimum's position (than the middle point of our trio). Differentiating $P_2(x)$ gives

$$\frac{dP_2}{dx} = \frac{[(x-x_1)+(x-x_2)]\,(x_2-x_1)}{d}y_0 + \frac{[(x-x_0)+(x-x_2)]\,(x_0-x_2)}{d}y_1 + \frac{[(x-x_0)+(x-x_1)]\,(x_1-x_0)}{d}y_2,$$

where $d = (x_1-x_0)(x_2-x_0)(x_2-x_1)$. Then setting $dP_2/dx = 0$ and solving for x gives (after some algebra) a new estimate of the minimum x_3

$$\boxed{x_3 = \frac{1}{2}\frac{(x_2^2-x_1^2)y_0 + (x_0^2-x_2^2)y_1 + (x_1^2-x_0^2)y_2}{(x_2-x_1)y_0 + (x_0-x_2)y_1 + (x_1-x_0)y_2}.} \tag{7.6}$$

This method works because any minimum can be approximated by a parabola and the approximation becomes more and more accurate as you get closer to the minimum.

For the case where the curvature is not positive throughout the initial interval we can first evaluate the function at two points inside the interval. We then keep the interval which includes the lowest point and then use the parabolic method to find the minimum within the new interval.

7.2 Multi-Dimensional methods

Univariate method

For the case where the function is multi-dimensional, i.e., $f(\vec{x})$, we can extend the one-dimensional parabolic method. This can be done by searching for the minimum in each direction successively and iterating. However this is a very inefficient method which is not useful in most cases.

Univariate search is illustrated by the black arrows in figure 7.4. Contours are shown for a 2D function $f(x, y)$. The univariate search spirals into the minimum, converging slowly.

Gradient method

We can follow the steepest descent towards the minimum. This is achieved by calculating the local gradient and following its (negative) direction. The gradient of $f(\vec{x})$ at point $\vec{x}_n$ is given by the vector

$$\vec{d}(\vec{x}_n) = \vec{\nabla} f(\vec{x}_n) \qquad \text{where} \qquad \vec{\nabla} f = \begin{pmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \\ \vdots \\ \partial f/\partial x_N \end{pmatrix} \tag{7.7}$$

and is perpendicular to the local contour line. Note that the notation $\vec{\nabla} f(\vec{x}_n)$ is equivalent to $\vec{\nabla} f|_{\vec{x}_0}$, i.e., $\vec{\nabla} f$ evaluated at $\vec{x} = \vec{x}_0$. We can take a small step in the direction **opposite** to the gradient, i.e.,

$$\boxed{\vec{x}_{n+1} = \vec{x}_n - \alpha\, \vec{d}(\vec{x}_n),} \tag{7.8}$$

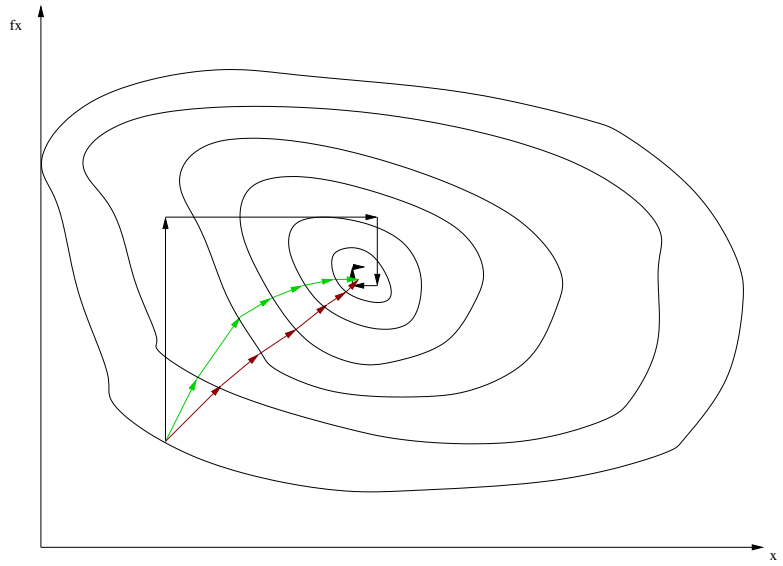

Figure 7.4: **Multi-dimensional minimisation** - for a function $f(x, y)$. Three methods are depicted: univariate method (black arrows), gradient method (green arrows) and Newton's method (red arrows). [Nb: this is not quite accurate; the green arrows are all supposed to be perpendicular to the contours!]

where $\alpha \ll 1$ and positive. If α is small enough then

$$f(\vec{x}_{n+1}) < f(\vec{x}_n).$$

We can iterate this to find the minimum. (See green arrows in fig. 7.4.)

The gradient $\vec{\nabla} f$ can be found analytically or using a finite difference approximation; e.g. via a forward-difference scheme (to be discussed in detail in a later chapter) applied in each variable x_i for $i = 1, \ldots, N$,

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_1, x_2, \ldots, x_i + \Delta, \ldots, x_N) - f(x_1, x_2, \ldots, x_i, \ldots, x_N)}{\Delta}. \tag{7.9}$$

Newton's method

A more efficient method to find the minimum is achieved by including the local curvature at each step. Consider starting at a location $\vec{x}_0$. The minimum is displaced $\vec{\delta}$ away, at position $\vec{x}_0 + \vec{\delta}$. How can we estimate $\vec{\delta}$?

We use the Taylor series for the function about $\vec{x}$. (Later, $\vec{x}$ will be set to $\vec{x}_0$.)

$$f(\vec{x} + \vec{\delta}) = f(\vec{x}) + [\vec{\nabla} f(\vec{x})]^T \cdot \vec{\delta} + \frac{1}{2} \vec{\delta}^T \cdot \mathbf{H}(\vec{x}) \cdot \vec{\delta} + \mathcal{O}(|\vec{\delta}|^3), \tag{7.10}$$

where $\mathbf{H}$ is the **Hessian** or **Curvature** matrix (an $N \times N$ matrix)

$$H_{ij}(\vec{x}) = \frac{\partial^2 f(\vec{x})}{\partial x_i \partial x_j}. \tag{7.11}$$

(c.f. equation 2.4 and the paragraph which precedes it in section 2.1.)

To express this in an alternative format; any function that is the sum of the terms x^2, y^2, $x \times y$, x, y with coefficients and a constant can be written:

$$f(x, y) = \frac{1}{2} \begin{pmatrix} x & y \end{pmatrix} \mathbf{H} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + C, \tag{7.12}$$

which is to say:

$$f(x,y) = \frac{1}{2}\left(\frac{\partial^2 f}{\partial x^2}x^2 + 2\frac{\partial f}{\partial x \partial y} + \frac{\partial^2 f}{\partial y^2}y^2\right) + ax + by + C.$$

Since by definition $\vec{x} + \vec{\delta}$ is the location of the minimum, we must have $\vec{\nabla} f(\vec{x} + \vec{\delta}) = \vec{0}$. To actually be a minimum (rather than a maximum or a saddle point) the Hessian needs to be **positive definite**, i.e.,

$$\vec{\delta}^{\,T} \cdot \mathbf{H} \cdot \vec{\delta} > 0 \qquad (\text{for } \textit{all} \quad \vec{\delta} \neq \vec{0}\,). \tag{7.13}$$

To determine $\vec{\delta}$ we can use the Taylor expansion (7.10) to first order in $\vec{\delta}$ and take its gradient yielding

$$\begin{aligned} \vec{\nabla} f(\vec{x} + \vec{\delta}) &= \vec{\nabla} f(\vec{x}) + \vec{\nabla}\left([\vec{\nabla} f(\vec{x})]^T \cdot \vec{\delta}\right) = \vec{0}, \\ &= \vec{\nabla} f(\vec{x}) + \mathbf{H}(\vec{x}) \cdot \vec{\delta} = \vec{0}. \end{aligned}$$

Since we start at $\vec{x} = \vec{x}_0$ we can solve the following matrix-equation for $\vec{\delta}$,

$$\boxed{\mathbf{H}(\vec{x}_0) \cdot \vec{\delta} = -\vec{\nabla} f(\vec{x}_0).} \tag{7.14}$$

The notation $\mathbf{H}(\vec{x}_0)$ means (7.11) evaluated at $\vec{x} = \vec{x}_0$. If $f(\vec{x})$ is exactly 'parabolic', i.e.,

$$f(\vec{x}) = \vec{x}^{\,T} \cdot \mathbf{A} \cdot \vec{x} + \vec{b}^{\,T} \cdot \vec{x} + c, \tag{7.15}$$

then there are no terms $\mathcal{O}(|\vec{\delta}|^3)$ in the Taylor expansion, and $\vec{x}_1 = \vec{x}_0 + \vec{\delta}$ is the exact position of the minimum.

If the function is not well approximated by a quadratic then the estimate of $\vec{\delta}$ obtained from solving (7.14) will not take us directly to the minimum so we iterate this until we get arbitrarily close to it, i.e., $\vec{x}_{n+1} = \vec{x}_n + \vec{\delta}$ which can be written as

$$\boxed{\vec{x}_{n+1} = \vec{x}_n - [\mathbf{H}(\vec{x}_n)]^{-1} \cdot \vec{\nabla} f(\vec{x}_n).} \tag{7.16}$$

This is illustrated by the red arrows in fig. 7.4. This method can be very efficient with **quadratic convergence**, $|\vec{\delta}|_{n+1} \sim |\vec{\delta}|_n^2$, however in problems with large dimensions the calculation of the inverse Hessian can become very slow. In addition calculating the Hessian using finite differences often yields a matrix which is not strictly positive definite.

Example (adapted from Gerald and Wheatley p. 424) – Consider the 2D parabolic function

$$f(x,y) = x^2 + 2y^2 + xy + 3x = (2x + y + 3)(x + 4y).$$

with the minimum at $\vec{x_*} = (x_*, y_*) = (-12/7, 3/7)$ from equating the derivatives to zero. If we start at the origin, the gradient and Hessian are given by

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} 2x + y + 3 \\ x + 4y \end{pmatrix} \quad , \qquad \mathbf{H} = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 4 \end{pmatrix} \quad .$$

It is easy to find the inverse of the Hessian matrix:

$$\mathbf{H}^{-1} = \begin{pmatrix} \frac{4}{7} & \frac{-1}{7} \\ \frac{-1}{7} & \frac{2}{7} \end{pmatrix} \tag{7.17}$$

.

Hence starting at $\vec{x} = \vec{x}_0 = (x_0, y_0)$, equation (7.14) yields

$$\begin{aligned}
\vec{\delta} &= -\mathbf{H}^{-1}\nabla f \\
&= -\begin{pmatrix} \frac{4}{7} & \frac{-1}{7} \\ \frac{-1}{7} & \frac{2}{7} \end{pmatrix} \cdot \begin{pmatrix} 2x_0 + y_0 + 3 \\ x_0 + 4y_0 \end{pmatrix} \\
&= -\frac{1}{7}\left(\begin{pmatrix} 8x_0 + 4y_0 + 12 - x_0 - 4y_0 \\ -2x_0 - y_0 - 3 + 2x_0 + 8y_0 \end{pmatrix}\right) \\
&= \begin{pmatrix} -x_0 - \frac{12}{7} \\ -y_0 + \frac{3}{7} \end{pmatrix} \\
&= \vec{x_*} - \vec{x}_0
\end{aligned}$$

so that $\vec{\delta}$ takes us directly to the minimum (i.e. $\vec{x}_0 + \vec{\delta} = \vec{x_*}$) in this case of a parabolic function.

Aside – What we have done in obtaining equation (7.14) is completely analogous to what would be done for finding the extremum of a simple quadratic $f(x) = ax^2 + bx + c$ if we know its gradient and curvature at a point x_0. The turning point $f' = 2ax + b = 0$ occurs at $x = -b/2a \equiv x_*$. Also $f'' = 2a$. The exact Taylor expansion is, $f(x_0 + h) = f(x_0) + hf'|_{x_0} + \frac{1}{2}h^2 f''|_{x_0} = (ax_0^2 + bx_0 + c) + h(2ax_0 + b) + \frac{1}{2}h^2(2a)$. The gradient of the Taylor expansion is

$$f'|_{x_0+h} = f'|_{x_0} + h\, f''|_{x_0} = (2x_0 + b) + h(2a) + h^2(0).$$

From this we get the value of h to take us from x_0 to the turning point ($f'|_{x_0+h} = 0$);

$$h = -f'|_{x_0}/f''|_{x_0} = -x_0 - b/2a.$$

This can be re-expressed as $h = x_* - x_0$, demonstrating that getting h from the gradient of the Taylor expansion does indeed work for a parabola. For the multi-dimensional case equation (7.14) corresponds to the expression for h above.

Quasi-Newton Method

A disadvantage of Newton's method is that we need to calculate both first and second derivatives of the multi-dimensional function, and the inverse of the curvature matrix which takes $\mathcal{O}(N^3)$.

The Quasi-Newton method gets around this by approximating the inverse Hessian using the local gradient. The basic iteration step is

$$\boxed{\vec{x}_{n+1} = \vec{x}_n - \alpha\, \mathbf{G}_n \cdot \vec{\nabla} f(\vec{x}_n),} \tag{7.18}$$

where $\mathbf{G}_n$ is an approximation of $\mathbf{H}^{-1}(\vec{x}_n)$ and $\alpha \ll 1$.

For the first iteration $\mathbf{G}_0$ is set to the identity matrix $\mathbf{I}$, which makes this iteration the same as a gradient search. To update $\mathbf{G}_n \to \mathbf{G}_{n+1}$ we use the updates in $\vec{x}$ and $\vec{\nabla} f$;

$$\vec{\delta}_n = \vec{x}_{n+1} - \vec{x}_n \qquad , \qquad \vec{\gamma}_n = \vec{\nabla} f(\vec{x}_{n+1}) - \vec{\nabla} f(\vec{x}_n).$$

Then comparing the above expression for the gradient update $\vec{\gamma}_n$ with the gradient of the Taylor expansion of $f(\vec{x})$, to linear order,

$$\vec{\nabla} f(\vec{x}_{n+1}) \approx \vec{\nabla} f(\vec{x}_n) + \vec{\nabla}\left(\vec{\nabla} f(\vec{x}_n) \cdot \vec{\delta}_n\right)$$

we see that to linear order

$$\mathbf{H}_n^{-1} \cdot \vec{\gamma}_n = \vec{\delta}_n.$$

(This is equivalent to equation (7.14) and the preceding paragraph when $\nabla f(\vec{x}_0 + \vec{\delta}) \neq 0$.) The trick is then to update $\mathbf{G}$ to satisfy

$$\mathbf{G}_n \cdot \vec{\gamma}_n = \vec{\delta}_n \,. \tag{7.19}$$

so that $\mathbf{G}_n$ mimics the inverse Hessian. A number of methods exist for this update, each with different efficiency and convergence characteristics. One of the most common is the DFP (Davidon–Fletcher–Power) algorithm where the update is

$$\mathbf{G}_{n+1} = \mathbf{G}_n + \frac{(\vec{\delta}_n \otimes \vec{\delta}_n)}{\vec{\gamma}_n \cdot \vec{\delta}_n} - \frac{\mathbf{G}_n \cdot (\vec{\delta}_n \otimes \vec{\delta}_n) \cdot \mathbf{G}_n}{\vec{\gamma}_n \cdot \mathbf{G}_n \cdot \vec{\gamma}_n}, \tag{7.20}$$

where $(\vec{u} \otimes \vec{v})_{ij} \equiv u_i v_j$ is the outer product of $\vec{u}$ and $\vec{v}$.

The advantage of this scheme is that it only involves (at worst) matrix multiplications, i.e., $\mathcal{O}(N^k)$ operations at each iteration and the resulting (approximated) Hessian is positive definite by construction. For full matrices (such as $\mathbf{G}_n$ and $(\vec{\delta}_n \otimes \vec{\delta}_n)$ here), naively one would think that the index k is 3. However it can be shown that matrix multiplication can be done with $k < 3$, in particular the Coppersmith & Winograd (1990) method scales with $k = 2.376$ and the commonly used `BLAS` routine scales with $k = 2.807$. Although these may seem close to $k = 3$ they make a big difference in computation time!

Global minimum search

These methods do not allow for the fact that the function may have multiple minima (local minima) in the interval we are interested in. There is no fool-proof method to find the global minimum (lowest minimum) of a function and we often need to restart algorithms from different starting points to check we have not found a local minimum.

7.3 Monte Carlo Minimisation – Simulated Annealing

Simulated annealing and related methods for optimisation introduce some randomness in the search for the minimum. They are particularly useful in cases where;

- The degrees of freedom in the system are so many that a direct search for an optimal configuration is too time consuming.
- Many local minima exist in which direct search methods can get stuck instead of finding the global minimum.

The simulated annealing approach is motivated by thermodynamics where random fluctuations can temporarily move a system to a less optimal (i.e. higher energy) configuration. The analogy is quantified in the use of the **Boltzmann Probability Distribution**

$$P(E)\, dE \sim \exp\left(-\frac{E}{k_{_B} T}\right) dE, \tag{7.21}$$

where "energy" E encodes a **cost function** and "temperature" T is a variable which determines the probability of changes in the energy of the system. In particular, even for low temperatures, it allows for the system to move to higher energies with very low probability. This is what can allow the system to get 'unstuck' from local minima and continue the search for a global minimum.

In practice the method involves the use of a **Metropolis Algorithm** where at each step in the iteration the configuration of the system is changed randomly. The "energy" of the system before (E_1) and after (E_2) the change are calculated to obtain the change in the system's energy: $\Delta E = E_2 - E_1$. The step is **accepted** with a probability p_{acc} given by the simple algorithm

$$p_{\text{acc}} = \begin{cases} 1 & \text{if} \quad \Delta E \leq 0 \\ \exp(-\Delta E / k_{_B} T) & \text{if} \quad \Delta E > 0 \end{cases} \tag{7.22}$$

In functional minimisation the "energy" is simply the value of the function $E = f(\vec{x})$ and the "temperature" T is slowly lowered to 0, hence the analogy with annealing of metals.

7.4 Using minimisation to solve matrix equations

The value of $\vec{x}$ that minimises the **quadratic form**

$$\boxed{f(\vec{x}) = \frac{1}{2}\vec{x}^T \cdot \mathbf{A} \cdot \vec{x} - \vec{b}^T \cdot \vec{x},} \tag{7.23}$$

happens to be the solution to

$$\mathbf{A} \cdot \vec{x} = \vec{b}$$

for a **symmetric, positive-definite** $N \times N$ matrix $\mathbf{A}$. This is because the gradient of this quadratic form is

$$\vec{\nabla} f(\vec{x}) = \mathbf{A} \cdot \vec{x} - \vec{b}. \tag{7.24}$$

At the minimum $\vec{x} = \vec{x}_*$,

$$\vec{\nabla} f(\vec{x_*}) = 0 \qquad \text{hence} \qquad \mathbf{A} \cdot \vec{x}_* = \vec{b}.$$

(Note the similarity of this quadratic form with equation (7.15) seen earlier.) So to solve the matrix equation, one of the multi-dimensional iterative methods seen in section 7.2 can be used with the quadratic form above.

The best iterative methods will (in the absence of round-off error) arrive at the exact solution in $N_{\text{iter}} = N$ iterations or less. This is better than Jacobi where convergence is as slow as $N_{iter} \sim \mathcal{O}(N^2)$ for some problems, and better than G-S & SOR. Convergence can be further accelerated by using a technique called ***pre-conditioning***. This effectively pushes the contours of $f(\vec{x})$ towards being spherical (in N dimensions), rather than very elongated ellipsoids (i.e. imagine fig. 7.2, but more elongated), so that a descent along the local gradient comes close to the global minimum.

To show that $\vec{\nabla} f = \mathbf{A} \cdot \vec{x} - \vec{b}$, consider the k-th component of the gradient of (7.23);

$$\begin{aligned}
(\vec{\nabla} f)_k &= \frac{\partial}{\partial x_k}\left[\frac{1}{2}\sum_i x_i \left(\sum_j A_{ij}x_j\right) - \sum_i b_i x_i\right] \\
&= \frac{1}{2}\sum_i \left[\delta_{ik}\left(\sum_j A_{ij}x_j\right) + x_i\left(\sum_j A_{ij}\delta_{jk}\right)\right] - b_k \\
&= \frac{1}{2}\left[\sum_j A_{kj}x_j + \sum_i x_i A_{ik}\right] - b_k \\
&= \frac{1}{2}\left[\mathbf{A}\cdot\vec{x}\right]_k + \frac{1}{2}\left[\vec{x}^T\cdot\mathbf{A}\right]_k - b_k \\
&= \frac{1}{2}\left[\mathbf{A}\cdot\vec{x}\right]_k + \frac{1}{2}\left[\mathbf{A}^T\cdot\vec{x}\right]_k - b_k \\
&= \left[\mathbf{A}\cdot\vec{x}\right]_k - b_k
\end{aligned}$$

where the last line makes use of the fact that $\mathbf{A}$ is symmetric. We have also used $\partial x_i/\partial x_k = \delta_{ik}$ where δ_{ik} is the Kronecker delta function.