

Circuit Optimisation



Unit: 3/EE/M Circuit Optimisation

Lecturer: James Grimbleby

URL: <http://www.elec.rdg.ac.uk/jbg.html>

email: j.b.grimbleby@reading.ac.uk

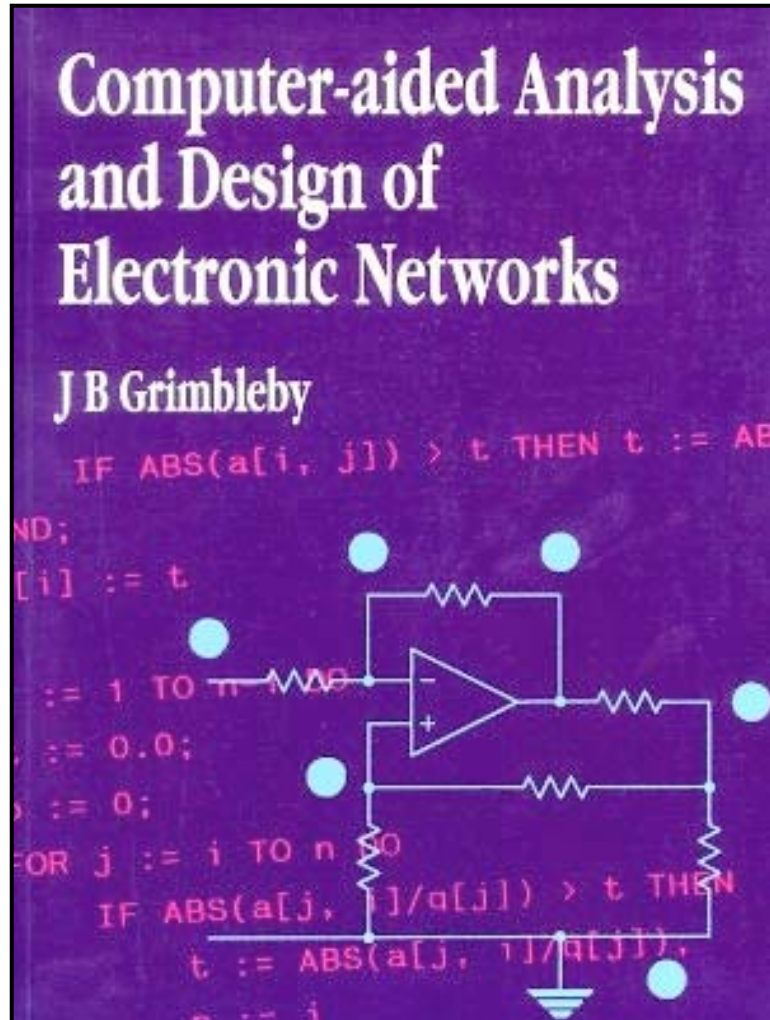
Number of Lectures: 5

Recommended text books:

Numerical Recipes in C, *W.H.Press et al.*, Cambridge University Press, ISBN 0-52143108-5

Computer-Aided Analysis and Design of Electronic Networks, *J.B.Grimbleby*, Pitman, ISBN 0-273-03148-1

Circuit Optimisation



Computer-Aided Analysis
and Design of Electronic
Networks

J.B. Grimbleby

Pitman

ISBN 0-273-03148-1

Circuit Optimisation Syllabus



Introduction, types of optimisation, objective function

Single-variable optimisation, bracketing, golden-section search

Numerical optimisation, non-gradient methods, simplex, direction-set methods

Gradient methods, steepest-descent, quasi-Newton methods

Constrained variables

Genetic optimisation, fitness function, selective breeding, mutation

Circuit Optimisation

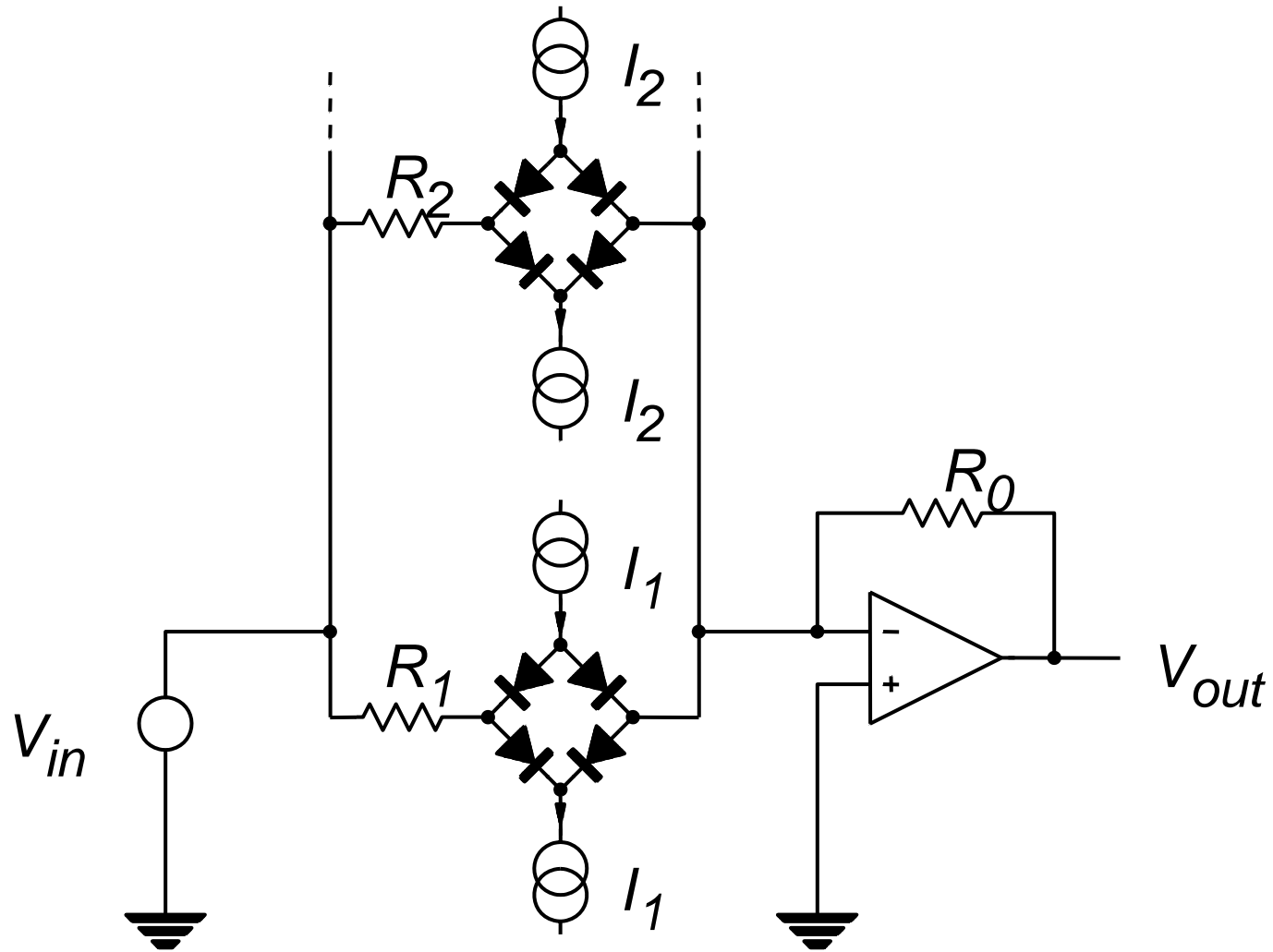
Optimisation is the process of selecting the parameters (component values) of a system in order to obtain the best performance

Optimisation is used in circuit design where formal design methods are not available.

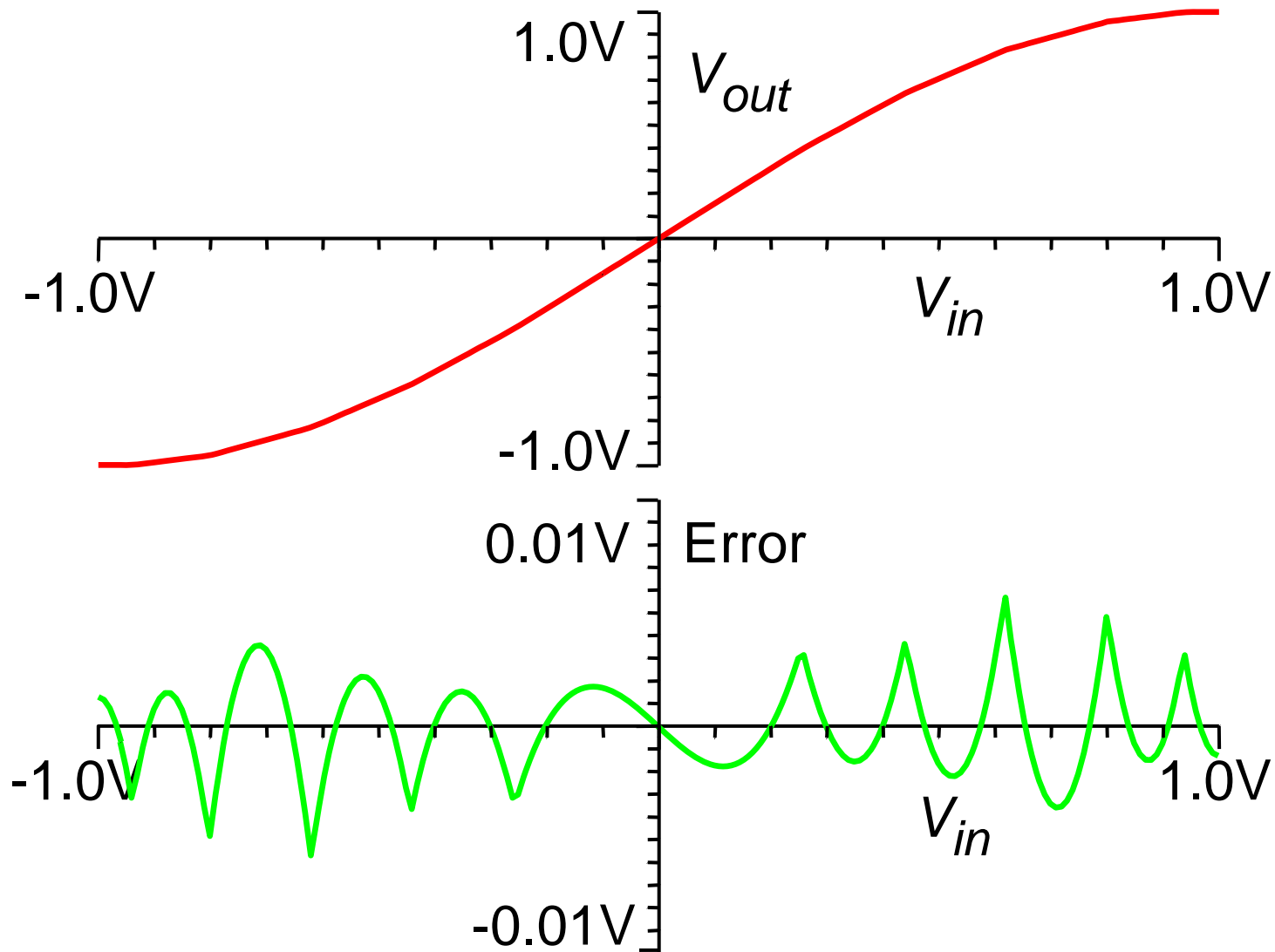
Optimisation is a true design method; any circuit that can be analysed can be optimised

Optimisation is analogous to manual adjustment of circuit properties, but may succeed where manual adjustment is impractical.

Circuit Optimisation Example



Circuit Optimisation Example



Optimisation

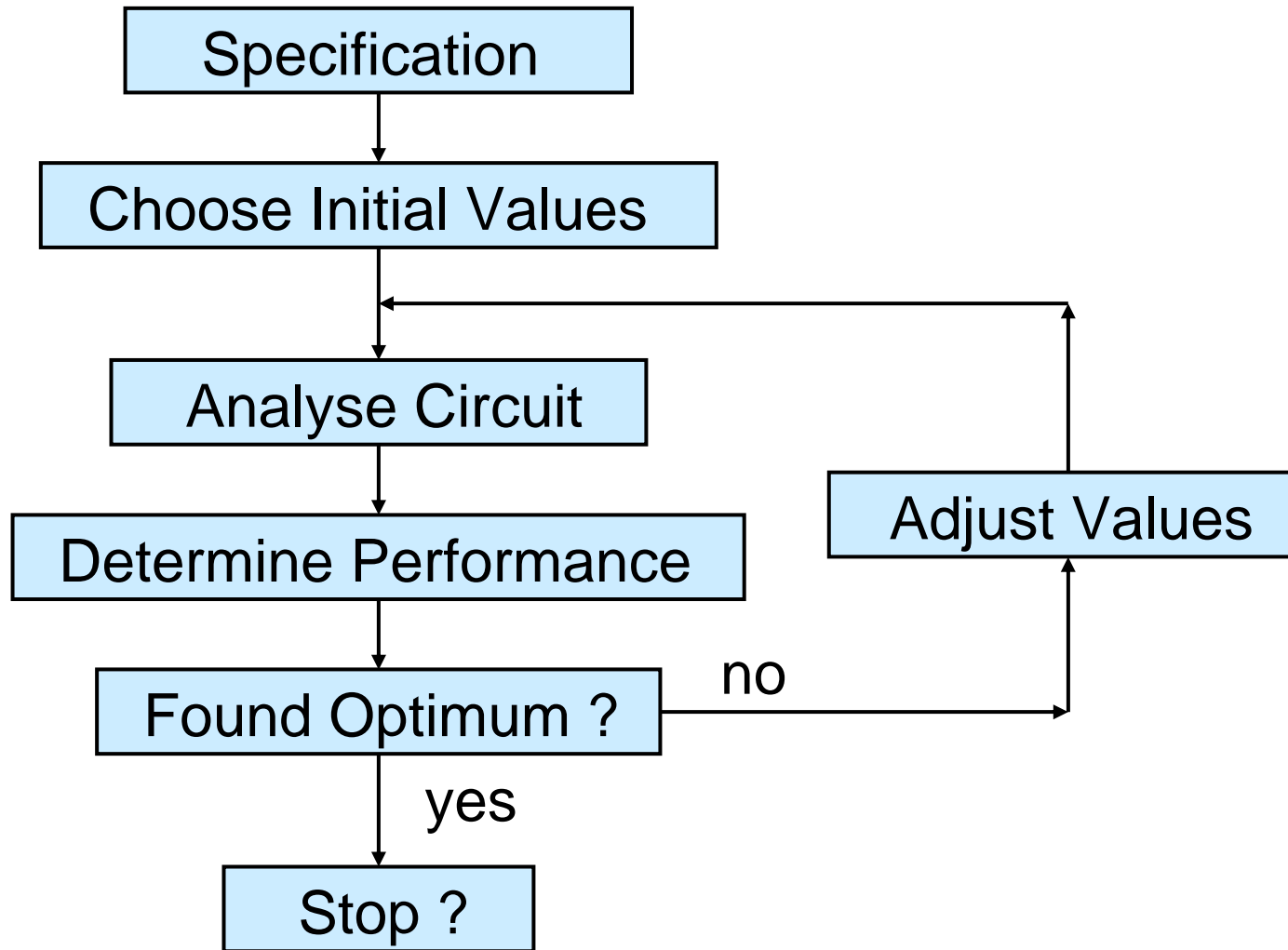
Optimisation is iterative; at each stage the performance is evaluated with a new set of parameter values.

This performance, together with those measured previously, is used to select values for the next iteration

An optimum is considered to have been reached if changing any parameter, or combination of parameters, by a small amount leads to an inferior performance

The initial parameter values have a profound effect on the efficiency of optimisation.

Optimisation Process



Optimisation

Optimisation does have some drawbacks:

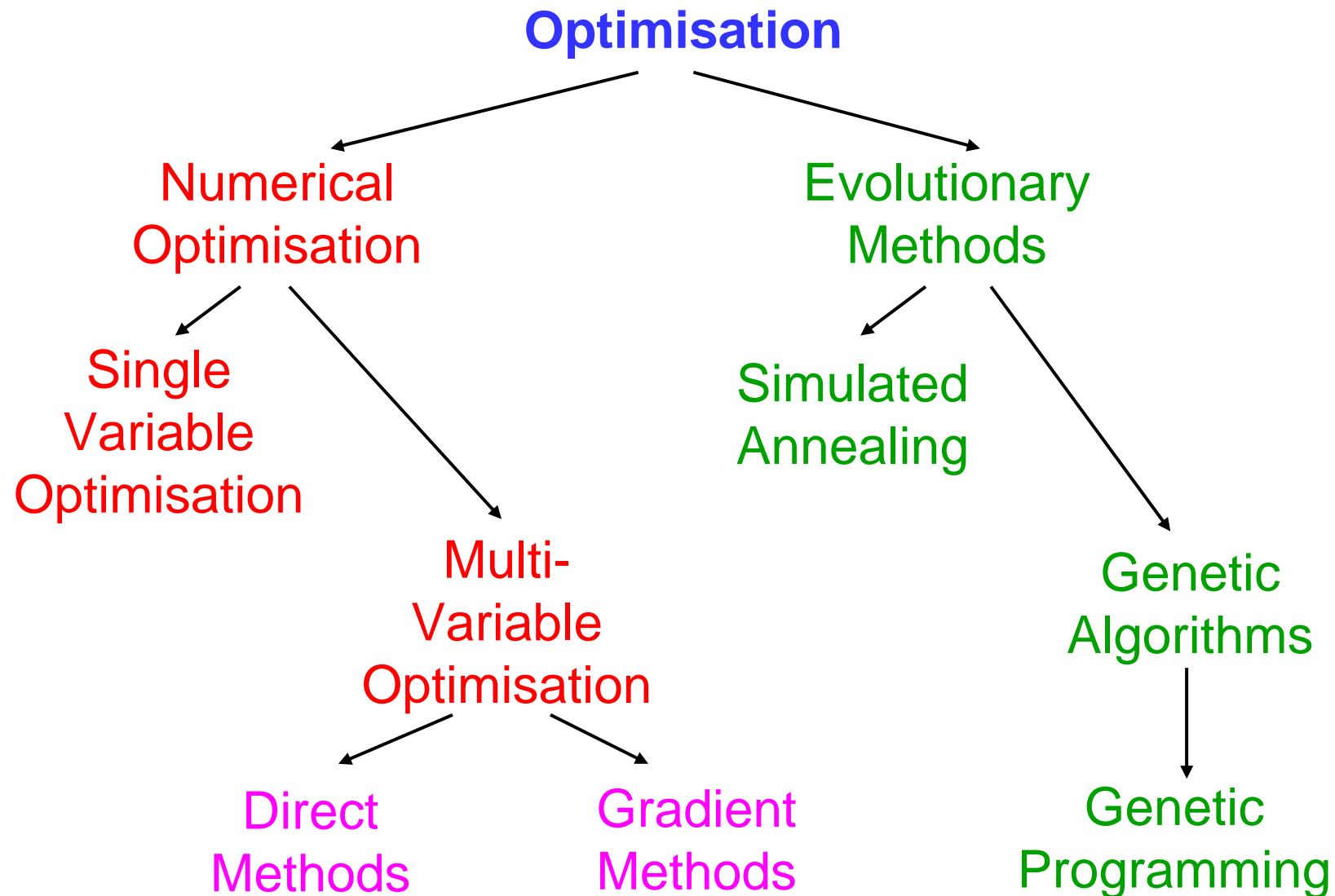
It tends to be computationally expensive. This is particularly true of genetic optimisation

It does not guarantee to find the global optimum

Numerical optimisation methods operate only on fixed circuit topologies. This restriction does not apply to genetic optimisation

Computer optimisation provides little insight into circuit operation.

Optimisation



Function Minimisation

Optimisation is normally formulated as a minimisation problem:

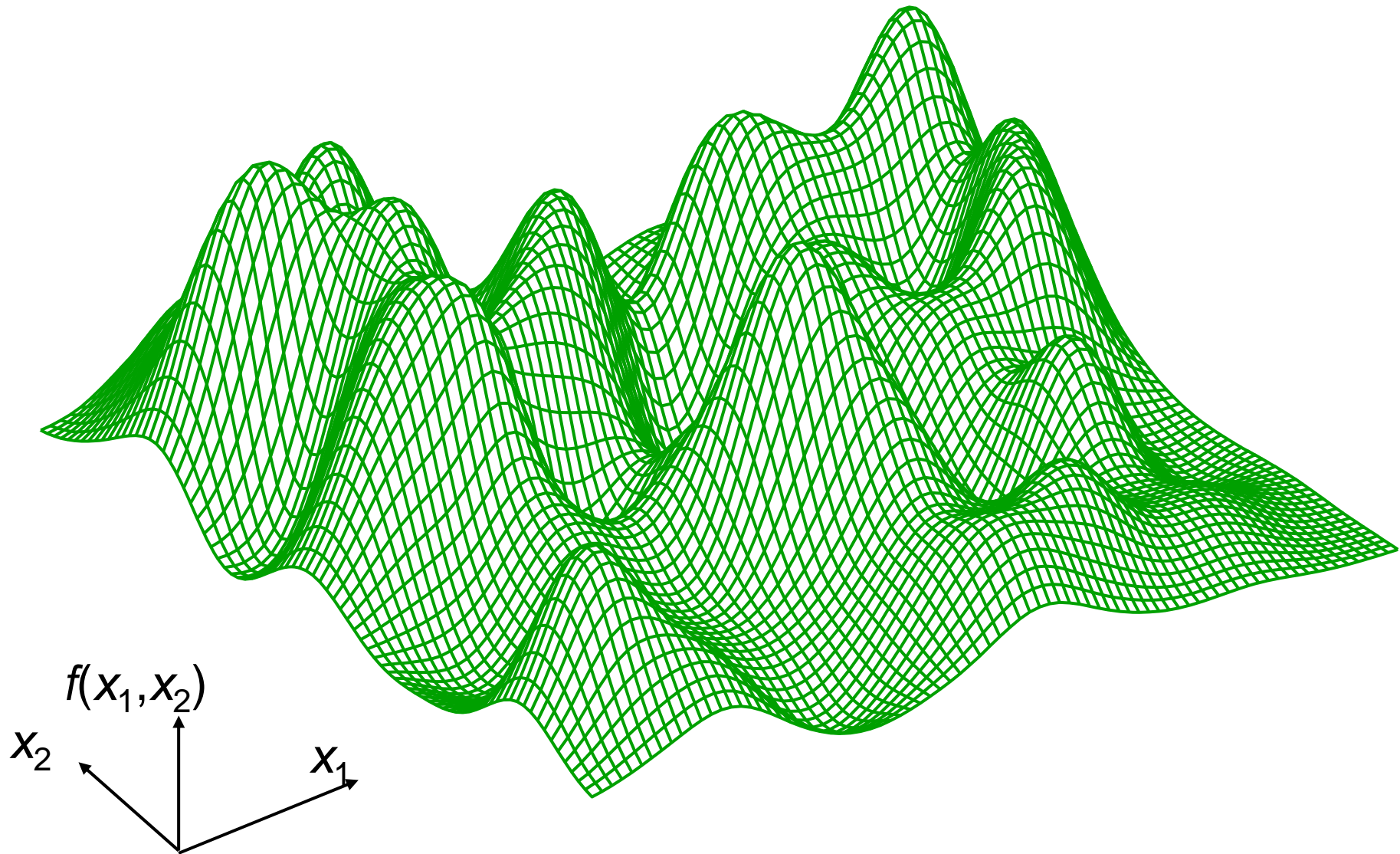
A scalar objective function f depends on the n parameter variables x_1, x_2, \dots, x_n

Find the minimum of f by adjusting the variables x_1, x_2, \dots, x_n

This does not limit the generality of the technique because:

$$\text{maximum}(f) = \text{minimum}(-f)$$

Objective Function



Vector Notation

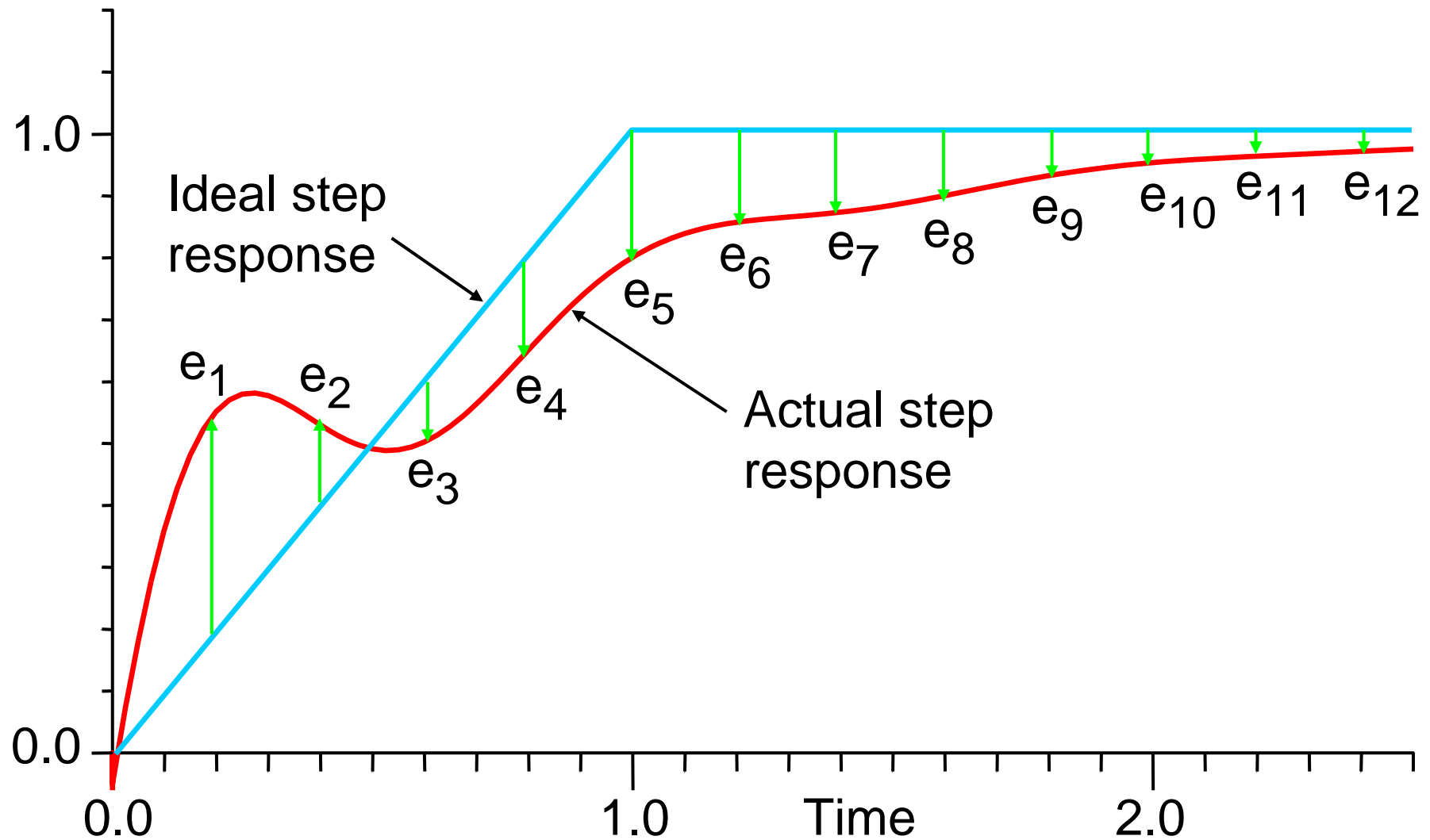
The n variables: x_1, x_2, \dots, x_n and changes in the n variables: $\Delta x_1, \Delta x_2, \dots, \Delta x_n$ will be represented using vector notation:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{bmatrix} \quad \Delta \mathbf{x} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \dots \\ \Delta x_n \end{bmatrix}$$

The objective function depends on vector \mathbf{x} :

$$f \equiv f(\mathbf{x})$$

Objective Function



Combining the Errors

Negative errors must not cancel positive errors

Increasing the modulus of any error should lead to a higher objective function

Objective function becomes zero if all errors are zero

Method of combining errors should absorb a negligible amount of computational effort

Combining the Errors

Sum of moduli:

$$\sum_{i=1}^m |e_i|$$

Sum of squares:

$$\sum_{i=1}^m e_i^2 = \mathbf{e}^T \cdot \mathbf{e}$$

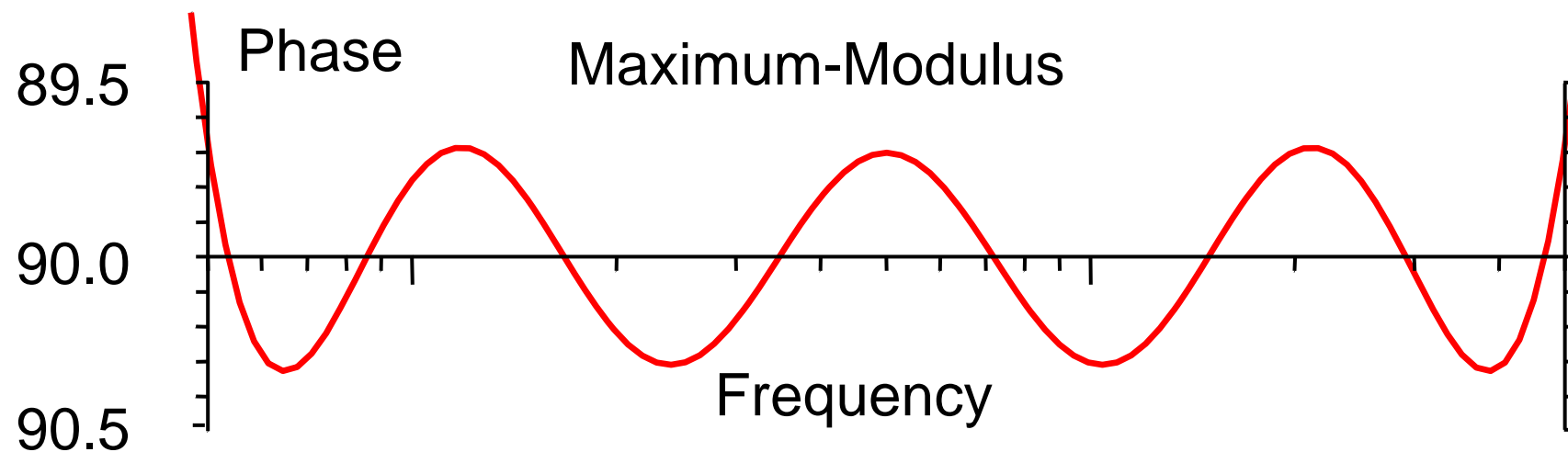
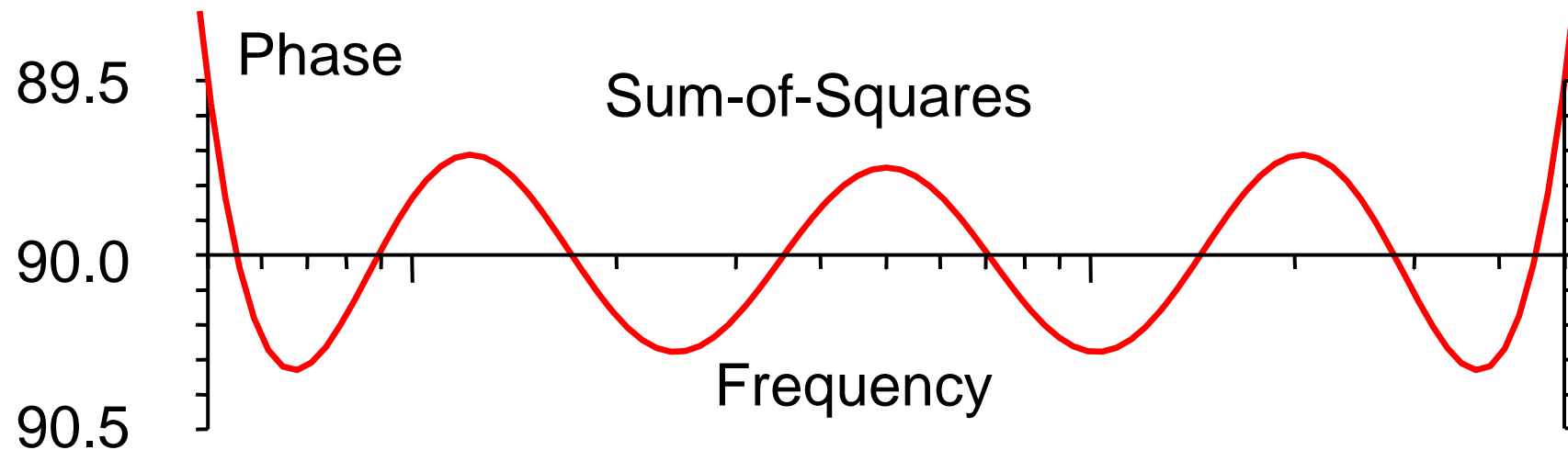
Sum of even powers:

$$\sum_{i=1}^m e_i^p$$

Maximum modulus:

$$\max_{i=1}^m |e_i|$$

Combining the Errors



Error Weighting

Not all parts of a specification are necessarily of equal importance

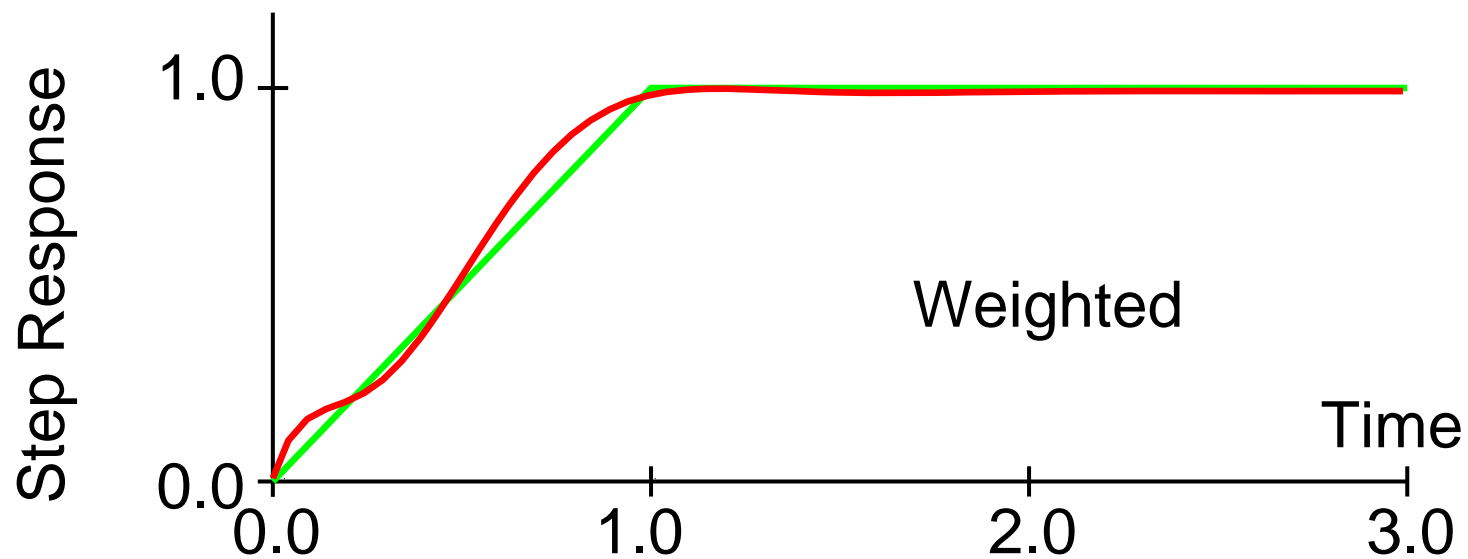
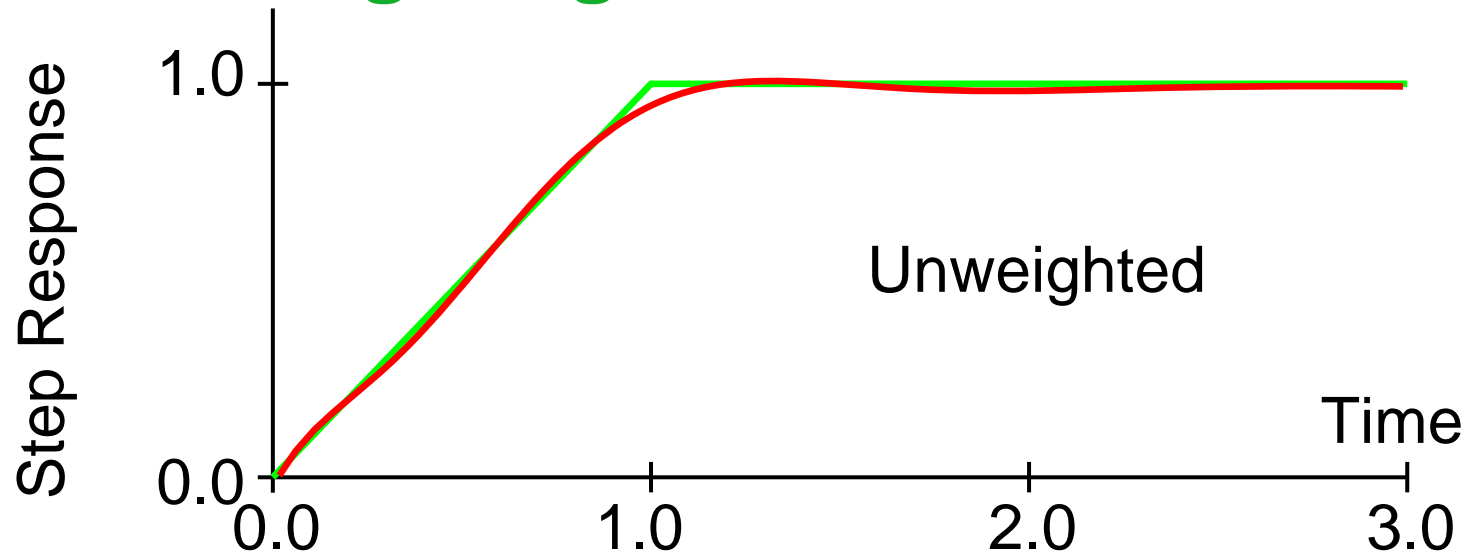
Each of the errors e_i can be given an individual weight w_i :

$$f = \sum_{i=1}^m (w_i e_i)^2$$

or:

$$f = \mathbf{e}^T \mathbf{w} \mathbf{e} \quad \text{where:} \quad \mathbf{w} = \begin{bmatrix} w_1^2 & 0 & 0 & \dots & 0 \\ 0 & w_2^2 & 0 & \dots & 0 \\ \dots & & & & \\ 0 & 0 & 0 & \dots & w_m^2 \end{bmatrix}$$

Error Weighting



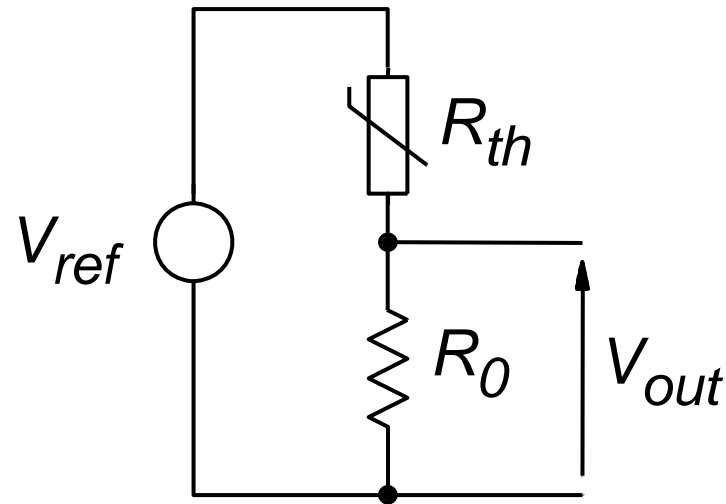
Single-Variable Optimisation

Single-variable optimisation is the minimisation of a scalar objective function: f of a single variable: x

For most purposes the best optimisation technique is the Golden-Section search method. This can cope with badly-behaved objective functions and has a fixed rate of convergence.

Before optimisation can proceed it is necessary to establish an interval in which a minimum is known to lie. This is known as bracketing

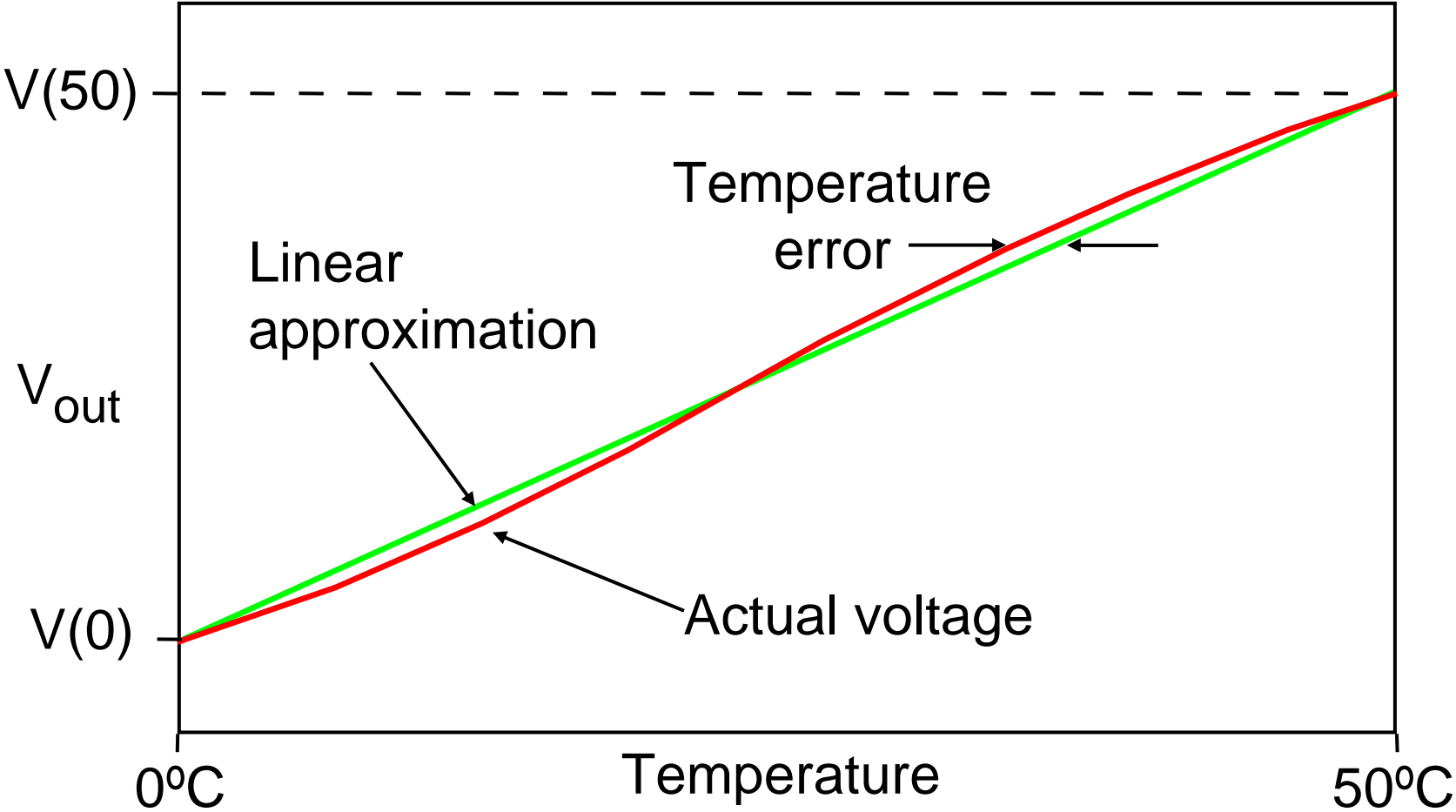
Thermistor Thermometer



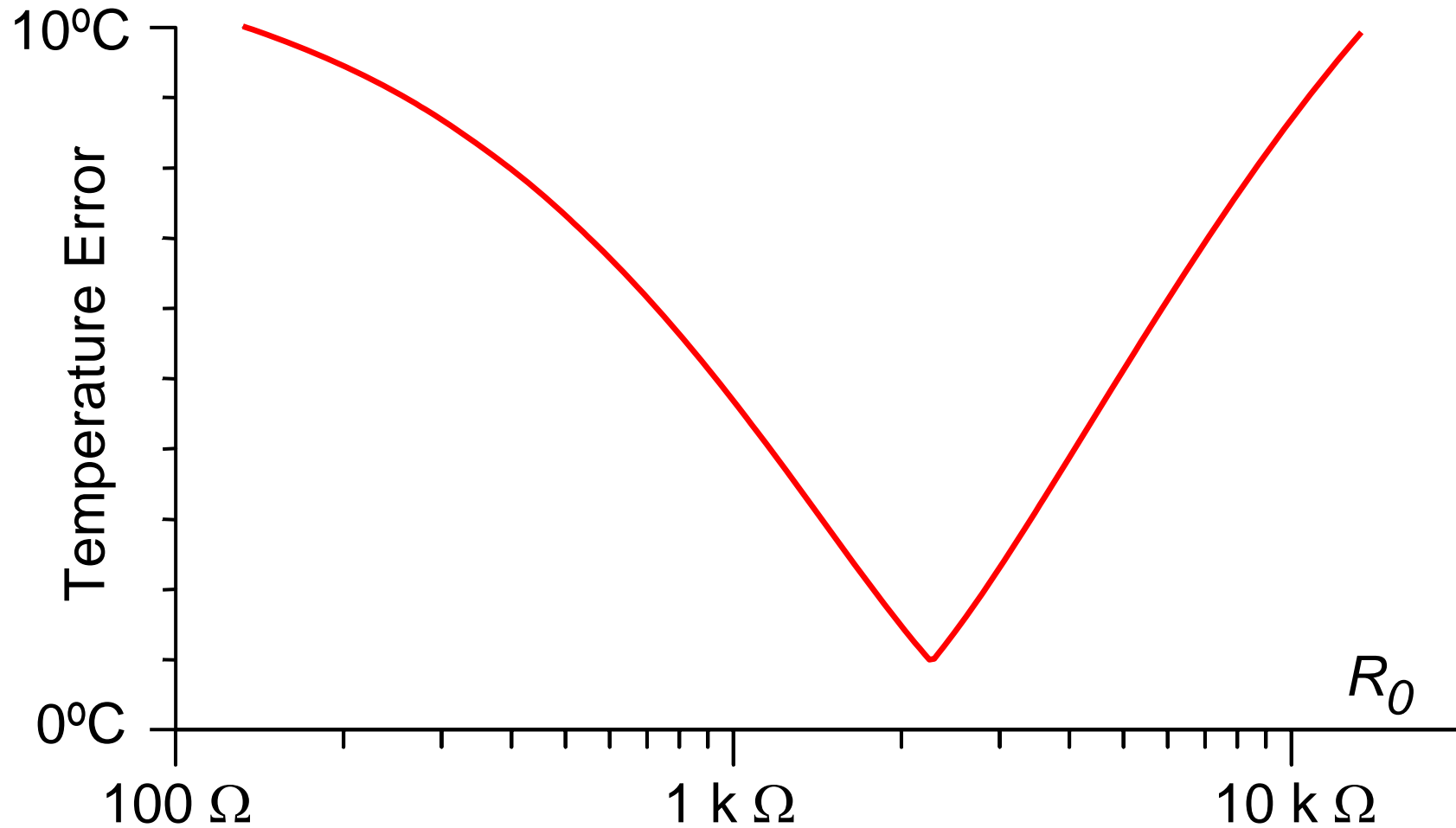
$$R_{th} = Ae^{B/T} \quad V_{out} = V_{ref} \frac{R_0}{R_0 + R_{th}}$$

Typical values: $A = 4.3 \times 10^{-3}$, $B = 4000.0$

Thermistor Thermometer



Thermistor Thermometer



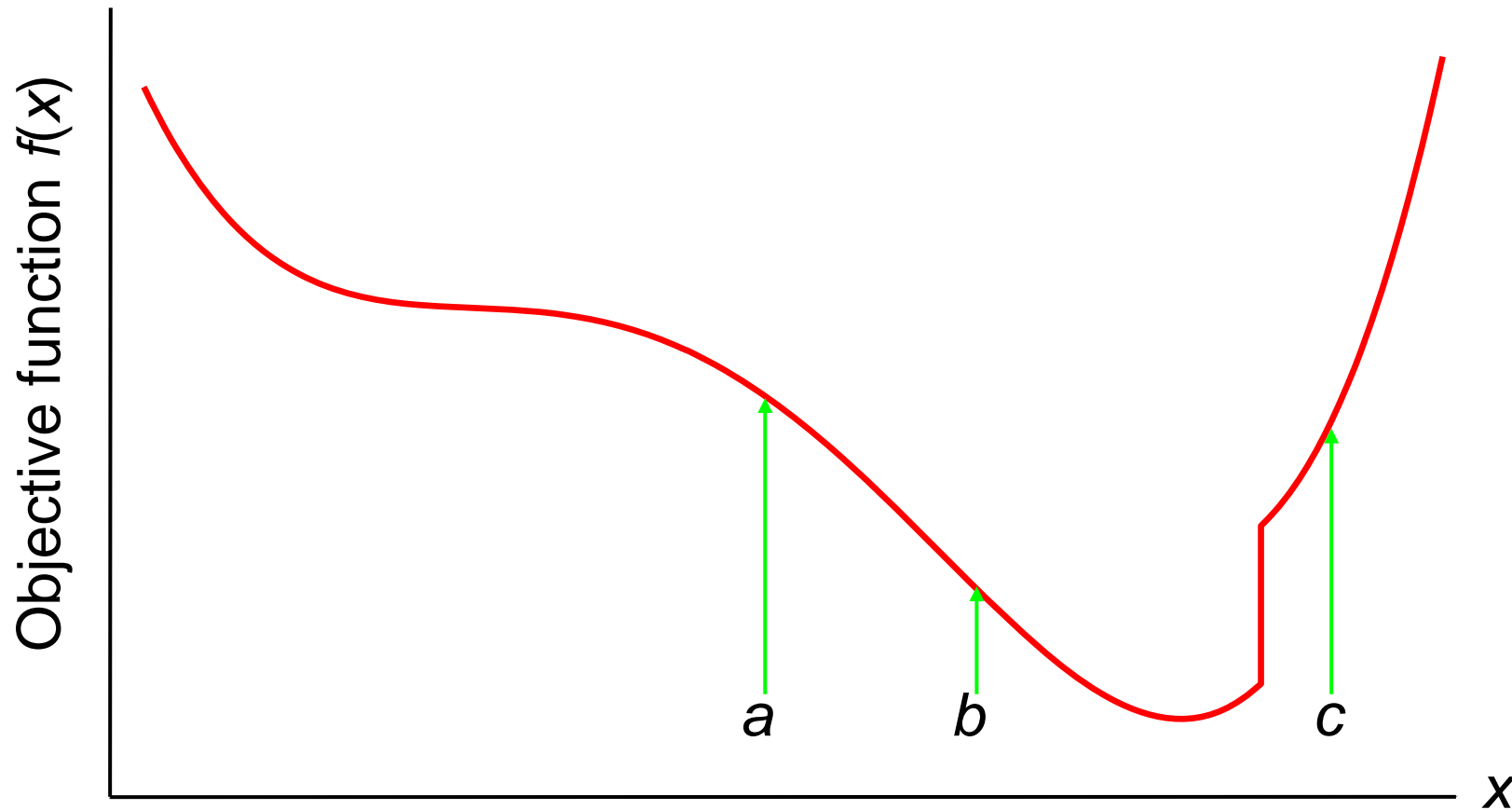
Binary Search

The binary search method (repeated bisection) that is used to solve equations can be modified to locate a minimum

An initial range (known to contain a minimum) is split into two and the minimum located in one of the sub-ranges

This process is repeated until the minimum has been located to a sufficient degree of accuracy

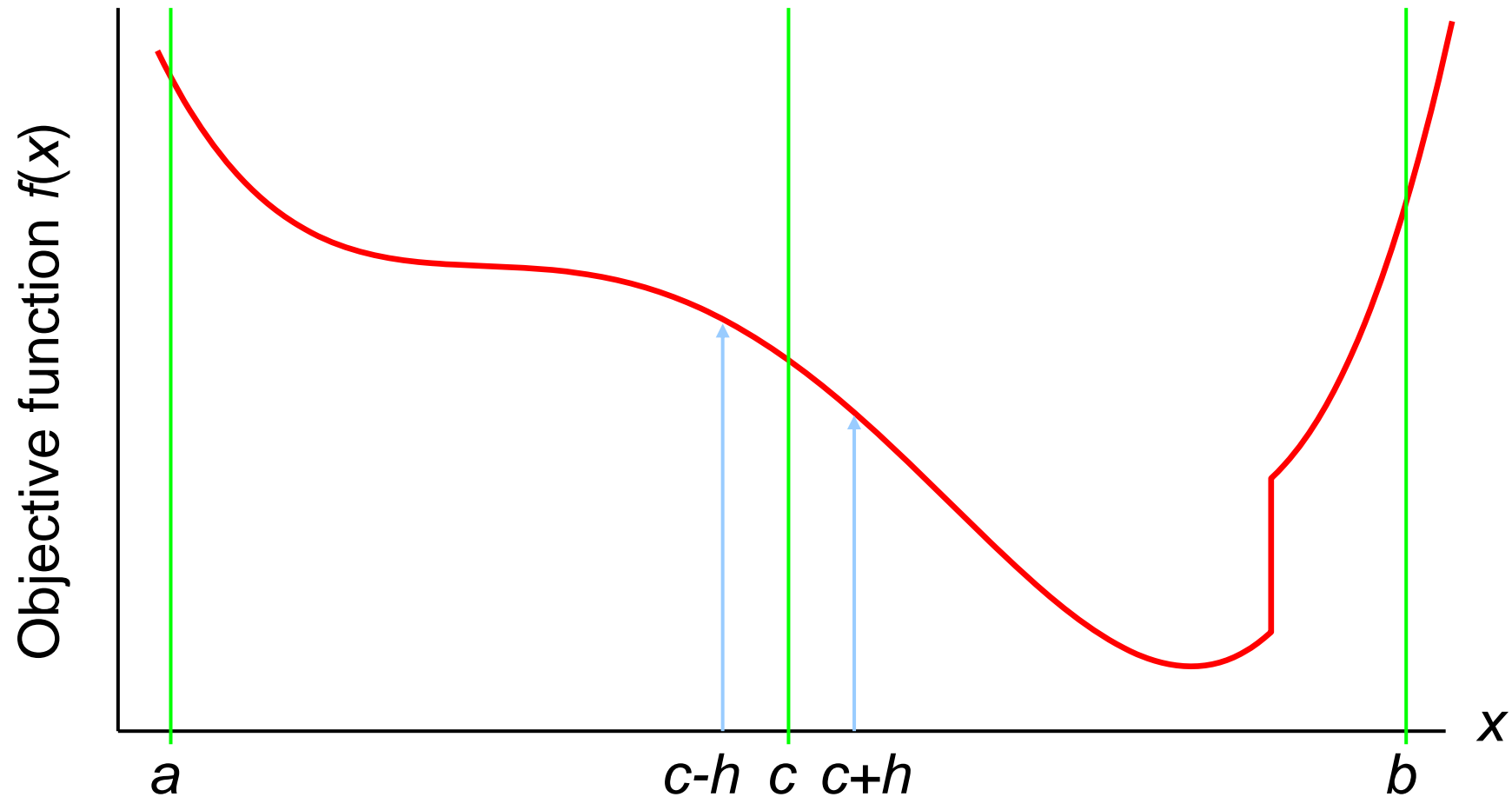
Bracketing a Minimum



Minimum is bracketed by a and c if:

$$a < b < c \quad \text{and} \quad f(b) < f(a) \quad \text{and} \quad f(b) < f(c)$$

Binary Search



Golden-Section Search

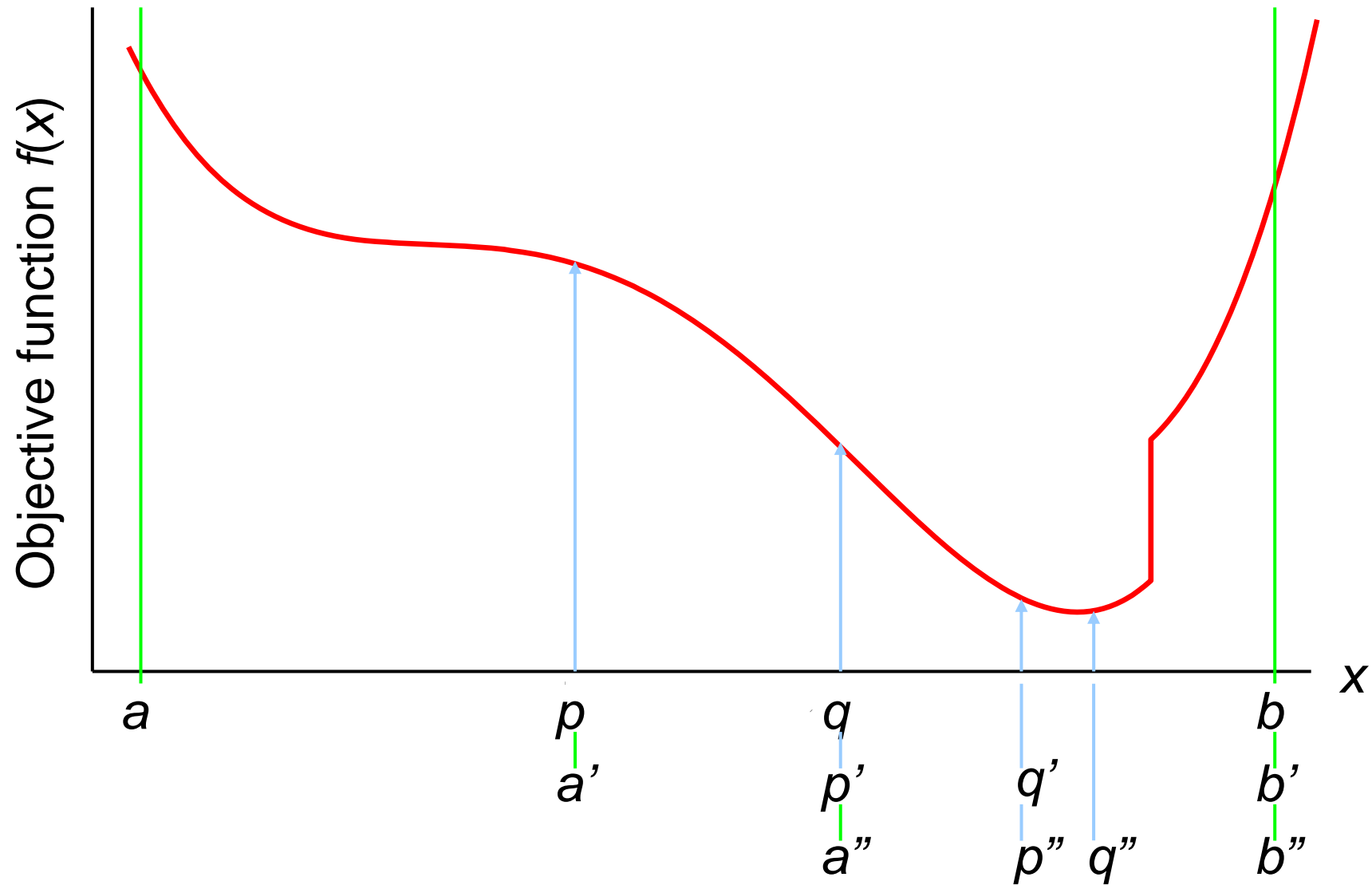
The binary search method has the disadvantage that it is difficult to choose a suitable value for h

If h is large then convergence is less than $\sqrt{2}$

If h is small then errors can occur because of rounding

These problems are overcome in the Golden-Section method, where h is large but only one objective evaluation is required at each iteration

Golden-Section Search



Golden-Section Search

Range is reduced by k at each iteration:

$$q - a = b - p = \frac{b - a}{k}$$

Second iteration:

$$a' = p \quad b' = b \quad p' = q$$

Applying same reduction factor:

$$q' - a' = b' - p' = \frac{b' - a'}{k}$$

or:

$$b - q = \frac{b - p}{k}$$

Golden-Section Search

Combining equations and using the fact that:

$$b - q = (b - a) - (q - a)$$

gives:

$$\frac{b - p}{k} = \frac{b - a}{k^2} = (b - a) - \frac{b - a}{k}$$

or:

$$\frac{1}{k^2} = 1 - \frac{1}{k}$$

thus:

$$k^2 - k - 1 = 0 \qquad k = \frac{1 + \sqrt{5}}{2} = 1.618$$

Golden-Section Search

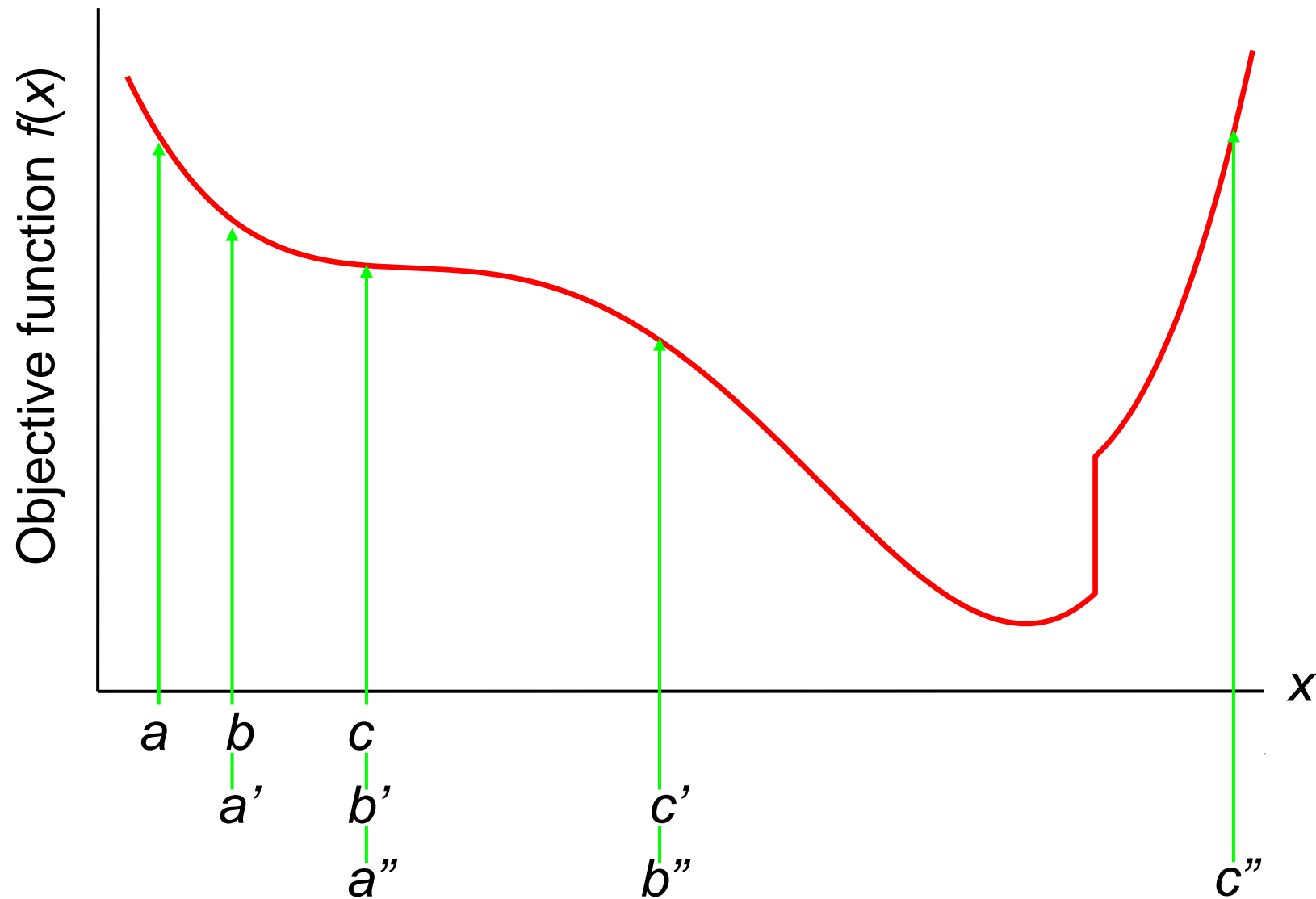
The search is continued until the search interval $a-b$ is less than the required precision

The search interval is reduced by k at each iteration. One objective evaluation is required at each iteration, so that the convergence rate is $k = 1.618$

Golden-Section search will only work if the initial search interval brackets a minimum

A bracketing routine should be used before a Golden-Section search

Bracketing a Minimum



Bracketing a Minimum

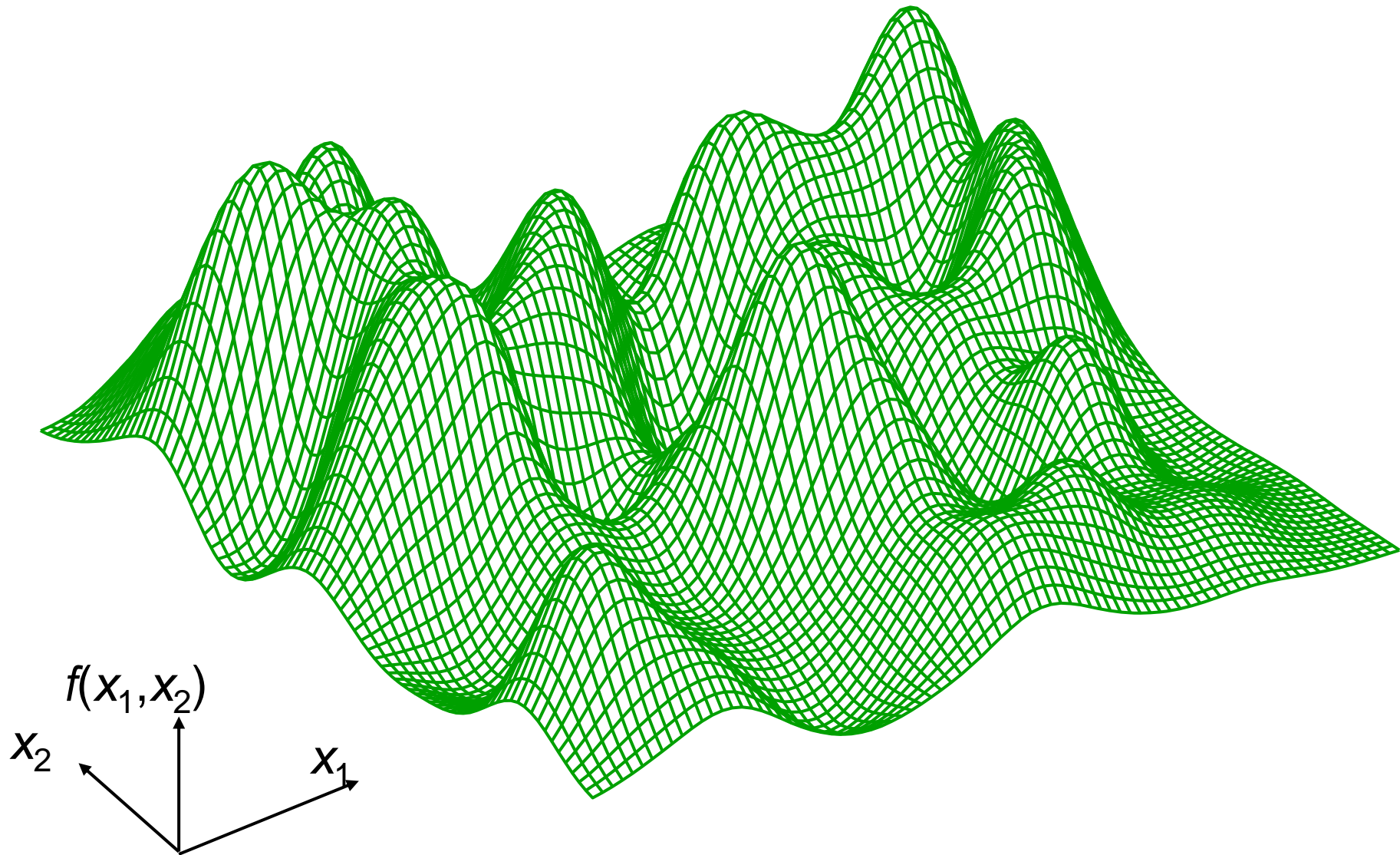
Evaluate the objective at 2 initial points a and c

If $f(c) > f(a)$ then swap a and c ; now $f(c) < f(a)$

Create a new point b between a and c , and evaluate its objective

```
While  $f(b) > f(c)$  {  
  // the points  $a$  and  $c$  do not bracket the minimum  
   $a = b$ ;  
   $b = c$ ;  
   $c = b + 2.0 \times (b - a)$ ;  
}
```

Direct vs Gradient Optimisation



Simplex Method

The Simplex method is due to Nelder and Mead.

Simplex is generally held to be the most robust direct optimisation method, but is not particularly efficient

Simplex is not a single method, but a family of closely-related techniques

The name simplex derives from the simplest solid in n dimensional space. A simplex has $n+1$ vertices

Simplex Method

Create the initial simplex and evaluate the objective at the $n+1$ vertices

Determine the vertices having the highest (hi) and lowest (lo) objective functions

Construct the *centroid* of all the vertices except the highest vertex (hi)

Invert the highest vertex (hi) through the *centroid* to obtain a new point (*reflected*) and evaluate the objective at this point

Simplex Method

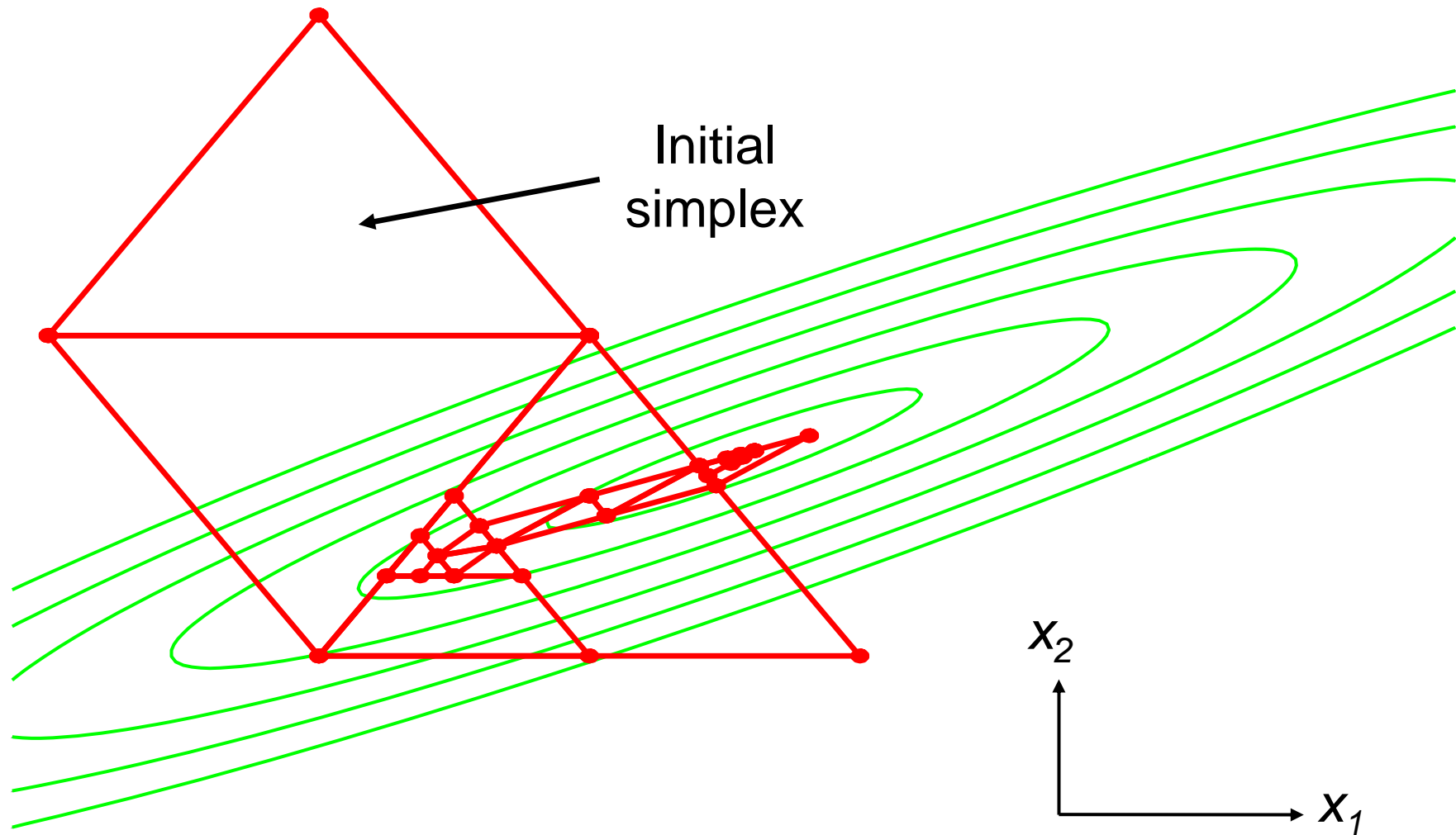
If the *reflected* point has a higher objective than the highest vertex (hi) then contract the simplex towards the lowest vertex (lo)

Otherwise, extend the line between *centroid* and *reflected* point to obtain *extended* point and evaluate the objective at this point

Replace the highest vertex by whichever of the *reflected* and *extended* points have the lowest objective

Repeat the process with the new simplex

Simplex Method



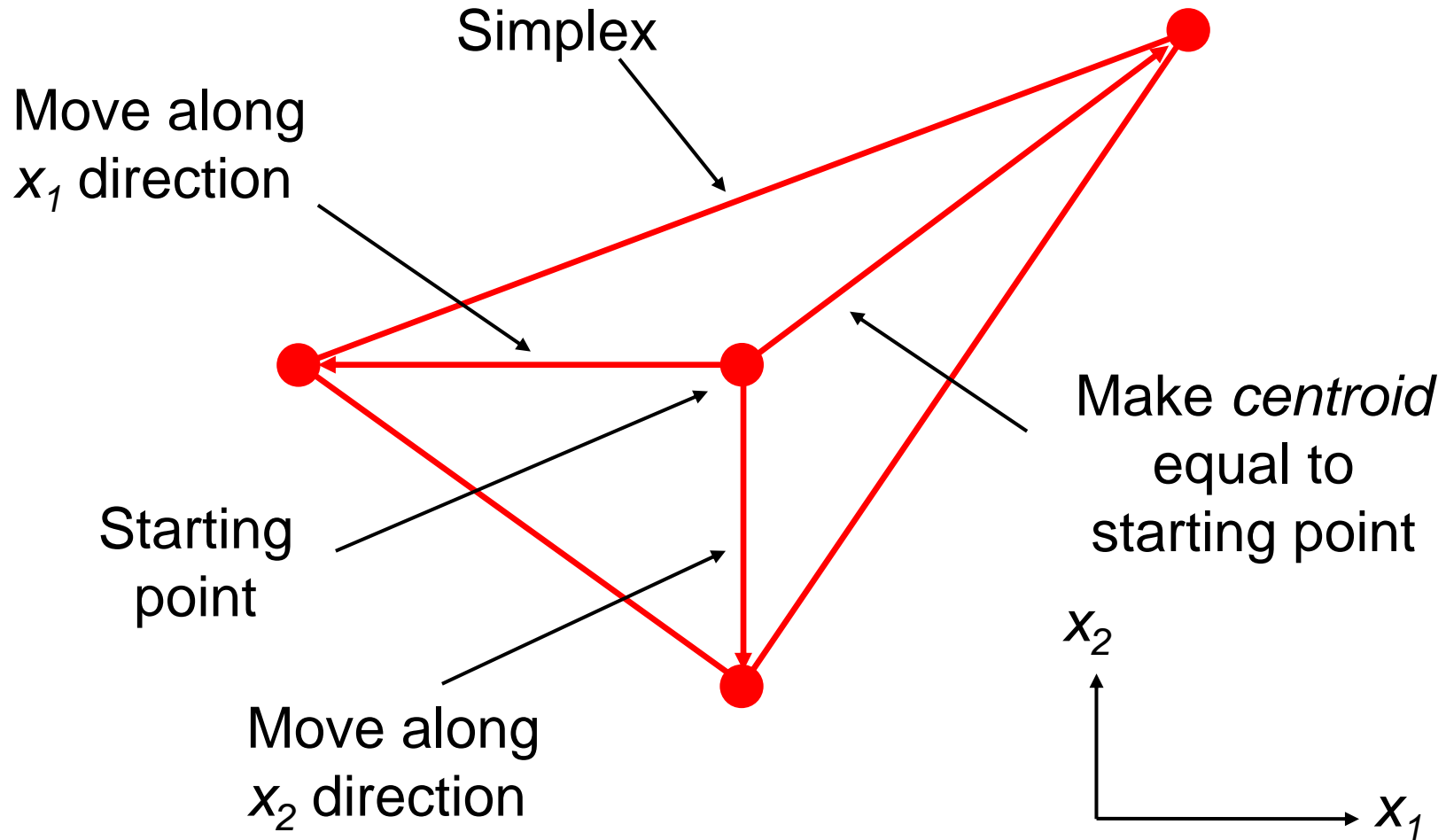
Initial Simplex

It is not essential that the initial simplex contains the optimum

The vertices must not lie on a single hyperplane, otherwise all operations in the simplex algorithm will simply generate further points on the hyperplane

Each of the first n vertices are generated by moving from the starting point along one of the axes. The $n+1$ vertex is chosen so that the *centroid* of all the vertices coincides with the starting point

Initial Simplex



Descent Methods

Descent methods use variations on the following algorithm, starting from a point \mathbf{x}_1 :

```
 $i = 1$   
while (not reached optimum) {  
    choose a search direction  $\mathbf{u}_i$   
    minimise  $f(\mathbf{x}_i + \lambda \mathbf{u}_i)$  by varying scalar  $\lambda$   
     $\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda_{\min} \mathbf{u}_i$   
     $i = i + 1$   
}
```

Descent methods differ in the way they choose the search direction \mathbf{u}_i

Classification of Optimisation Methods

	<i>Non-Gradient Methods:</i>	<i>Gradient Methods:</i>
<i>Descent Methods:</i>	One-at a time Direction set (Powell's)	Steepest-descent
<i>Non-Descent Methods:</i>	Simplex	Quasi-Newton (DFP)

One-at-a-Time Method

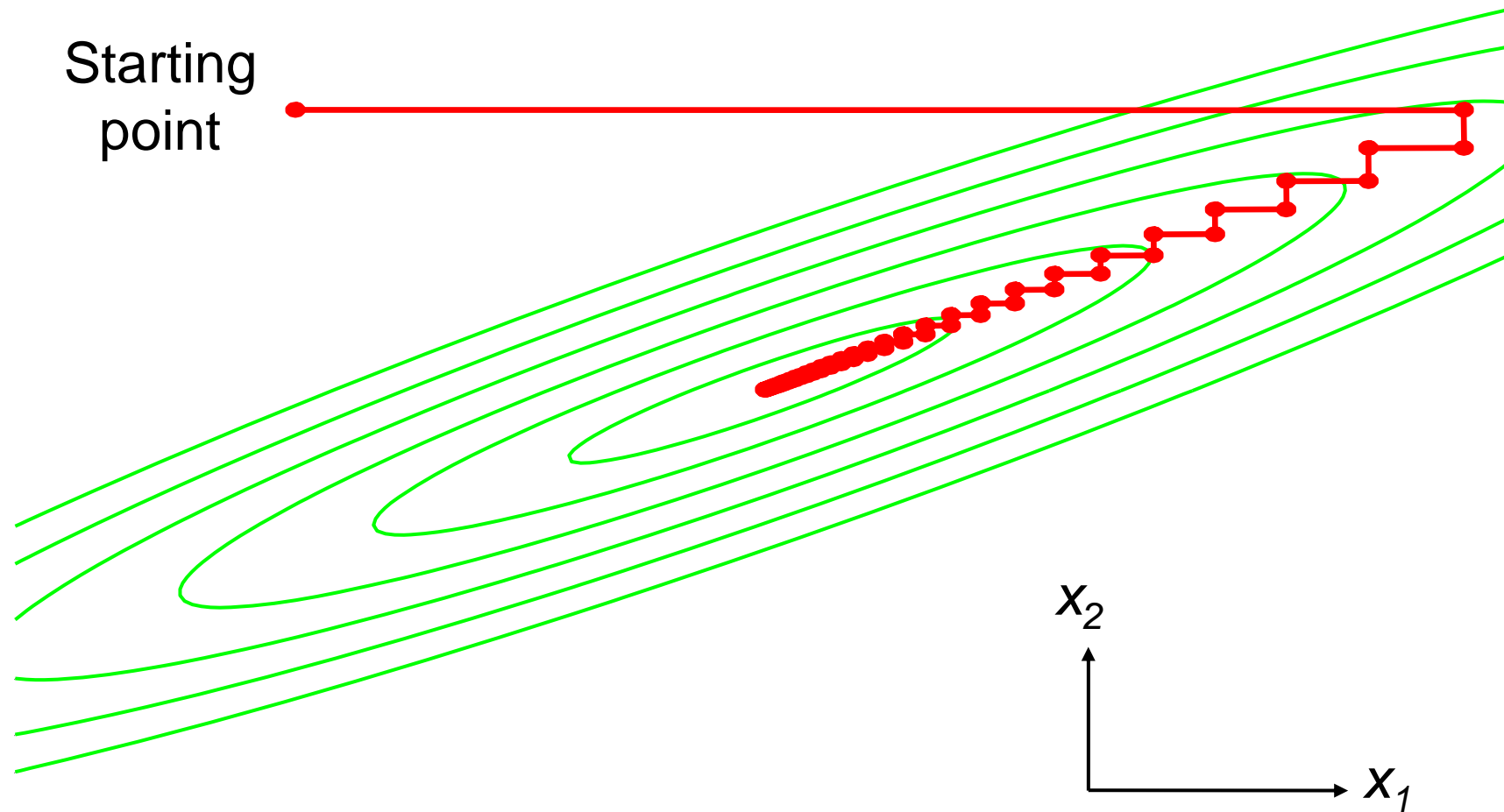
The simplest descent method is the one-at-a-time methods in which the search directions \mathbf{u}_i are the variable axes

In effect each variable is adjusted in turn, with all the others fixed, to obtain a minimum of the objective function

This is a direct method (no gradients)

For most optimisation problems this is very inefficient because a step is rarely in the direction of the minimum, and many cycles through the n variables will be necessary

One-at-a-Time Method



Gradient Optimisation

The objective function can be expanded as a Taylor series:

$$f(\mathbf{x} + \Delta\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^n \frac{\partial f}{\partial x_i} \Delta x_i + \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \frac{\partial^2 f}{\partial x_i \partial x_j} \Delta x_i \Delta x_j + \dots$$

or in vector-matrix form:

$$f(\mathbf{x} + \Delta\mathbf{x}) = f(\mathbf{x}) + \mathbf{g}^T \Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \mathbf{H} \Delta\mathbf{x} + \dots$$

Gradient Optimisation

\mathbf{g} is the Jacobian (gradient) vector and \mathbf{H} is the Hessian matrix:

$$\mathbf{g} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \dots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \dots & \dots & \dots & \dots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix}$$

Gradient Optimisation

Gradient optimisation methods which use \mathbf{g} only are known as first-order methods

Gradient optimisation methods which use both \mathbf{g} and \mathbf{H} are known as second-order methods

Normally \mathbf{g} must be obtained numerically because the objective function is too complex to be differentiated algebraically

\mathbf{H} is not evaluate directly, but is approximated from a number of evaluations of \mathbf{g}

Steepest Descent

Steepest descent is the simplest first-order gradient method

It is based on the fact that the gradient \mathbf{g} of the objective function can be used to indicate the direction of most rapid decrease of the objective function

A search is then conducted along this direction to find the minimum of the objective function

This will not normally be the optimum, but can be used as the starting point for the next iteration

Steepest Descent

Expanding the objective function as a 1st-order Taylor series:

$$f(\mathbf{x} + \Delta\mathbf{x}) = f(\mathbf{x}) + \mathbf{g}^T \Delta\mathbf{x}$$

The change Δf in the objective function resulting from a change in the variables $\Delta\mathbf{x}$ is:

$$\Delta f = \mathbf{g}^T \Delta\mathbf{x}$$

We wish to choose $\Delta\mathbf{x}$ to maximise the magnitude of Δf whilst keeping the sign of Δf negative:

$$\Delta f = \mathbf{g}^T \Delta\mathbf{x} = |\mathbf{g}| |\Delta\mathbf{x}| \cos \theta$$

For given magnitudes $|\mathbf{g}|$ and $|\Delta\mathbf{x}|$ the greatest reduction in f occurs when $\theta = \pi$, in other words when $\Delta\mathbf{x}$ is in the direction $-\mathbf{g}$

Steepest Descent

A unit vector \mathbf{u} in the direction of steepest descent $-\mathbf{g}$ is constructed:

$$\mathbf{u} = \frac{-\mathbf{g}}{|\mathbf{g}|}$$

A search is now conducted along the direction \mathbf{u} to find the minimum of f

In other words, the value of λ is found which minimises:

$$f(\mathbf{x} + \Delta\mathbf{x}) = f(\mathbf{x} + \lambda\mathbf{u})$$

This minimisation can be performed using a golden-section search

Steepest Descent Algorithm

Starting from a point \mathbf{x}_1 :

$i = 1$

while (not reached optimum) {

 determine gradient $\mathbf{g}(\mathbf{x}_i)$

$\mathbf{u}_i = -\mathbf{g}(\mathbf{x}_i) / \|\mathbf{g}(\mathbf{x}_i)\|$

 minimise $f(\mathbf{x}_i + \lambda \mathbf{u}_i)$ by varying scalar λ

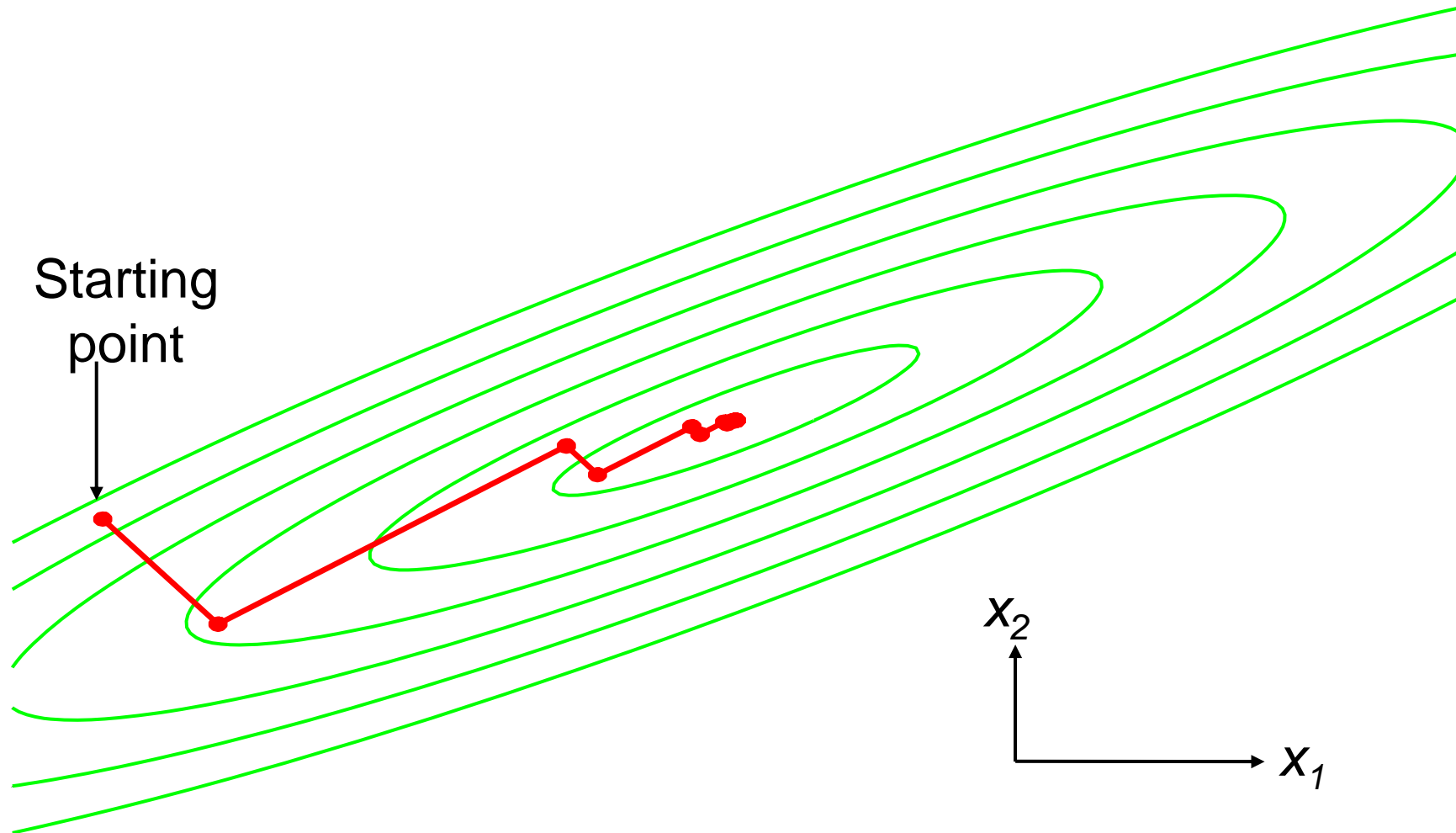
$\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda_{\min} \mathbf{u}_i$

$i = i + 1$

}

Minimisation can be performed using a single-variable optimisation method

Steepest-Descent Method



Steepest Descent

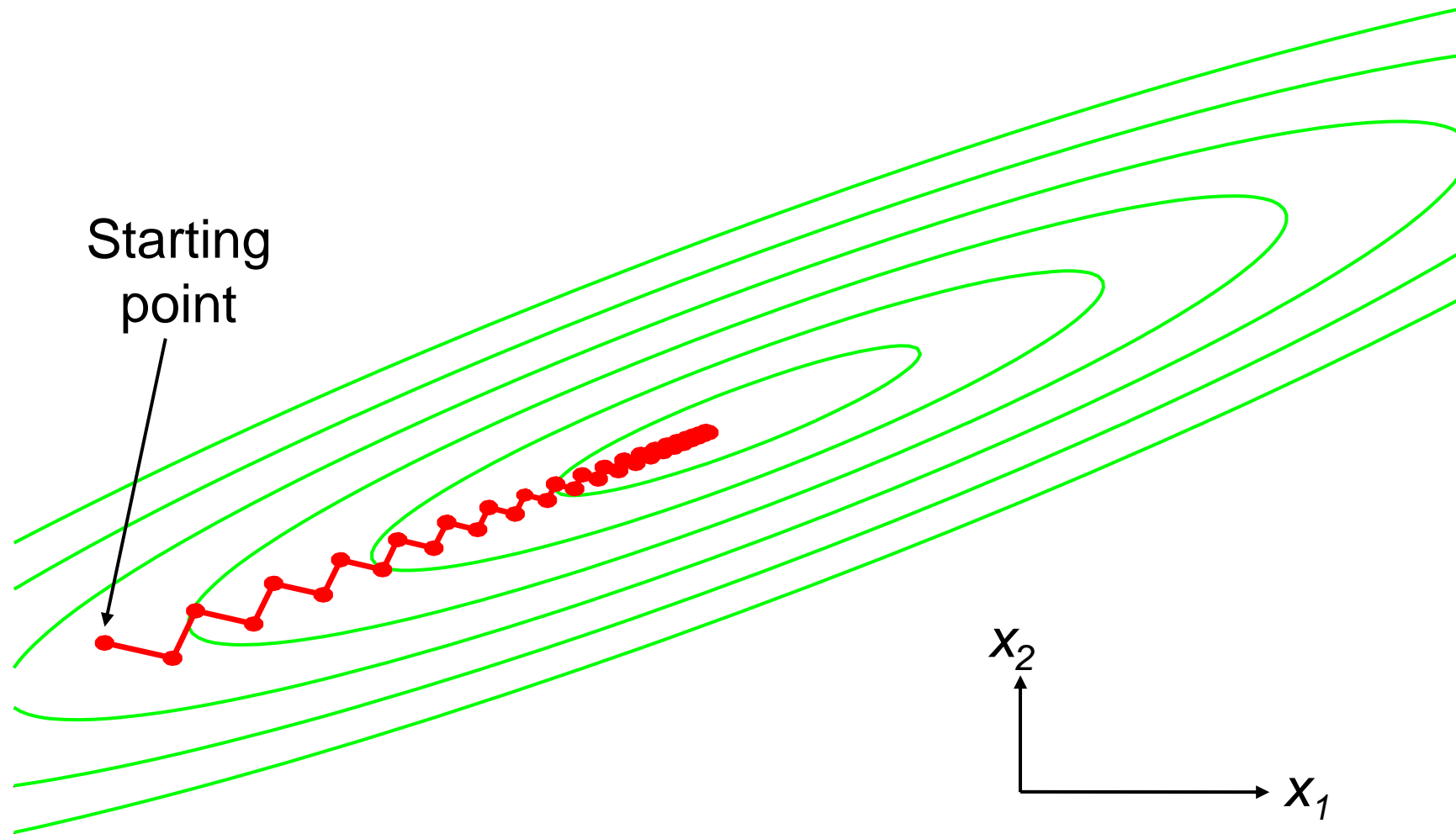
Unfortunately, the steepest descent direction does not usually point towards the optimum

In fact successive steepest-descent steps are at right-angles:

$$\frac{\partial}{\partial \lambda} f(\mathbf{x} + \lambda \mathbf{u}) = \mathbf{g}(\mathbf{x} + \lambda \mathbf{u}) \cdot \mathbf{u} = 0$$

Optimisation follows a zig-zag path and under worst conditions this can make steepest descent almost as inefficient as one-at-a-time optimisation

Steepest-Descent Method



Crude Steepest Descent Algorithm

Starting from a point \mathbf{x}_1 :

$$i = 1; \lambda = 1.0$$

while (not reached optimum) {

 determine gradient $\mathbf{g}(\mathbf{x}_i)$

$$\mathbf{u}_i = -\mathbf{g}(\mathbf{x}_i) / \|\mathbf{g}(\mathbf{x}_i)\|$$

$$f_{\text{old}} = f(\mathbf{x}_i)$$

$$\lambda = \lambda / 8.0$$

 while ($f(\mathbf{x}_i + \lambda \mathbf{u}_i) < f_{\text{old}}$) {

$$f_{\text{old}} = f(\mathbf{x}_i + \lambda \mathbf{u}_i)$$

$$\lambda = \lambda * 2.0$$

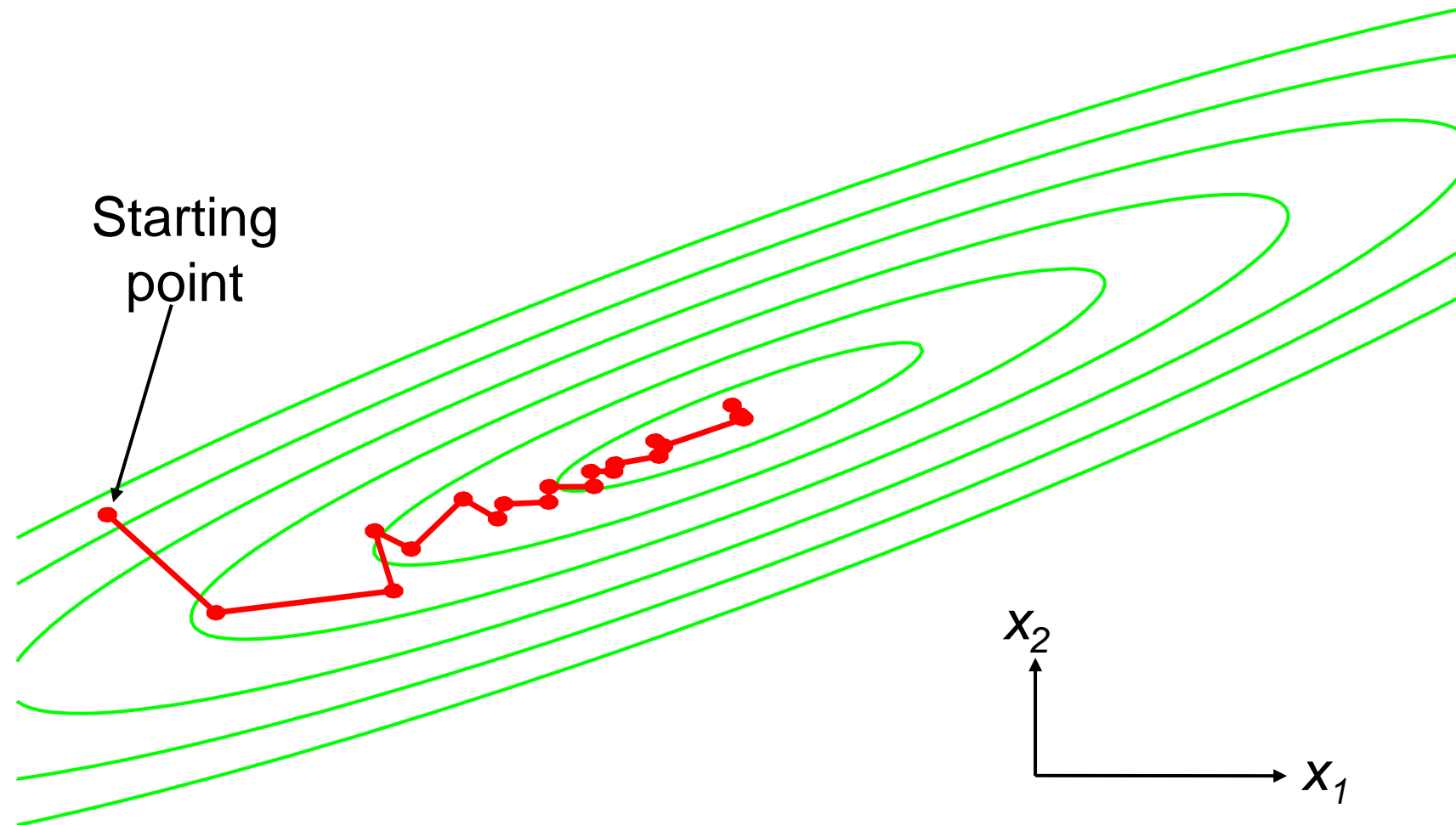
 }

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda \mathbf{u}_i / 2.0$$

$$i = i + 1$$

}

Crude Steepest Descent Algorithm



Direction-Set Methods

One-at-a-time and steepest-descent methods are inefficient because optimising with respect to one variable upsets previous optimisations

To prevent this a set of n directions \mathbf{u}_i , ($i=1..n$) is selected where optimising along any direction does not upset previous optimisations along other directions.

Directions having this property are said to be conjugate

This is the principle behind direction-set methods

Direction-Set Methods

If a set of n mutually conjugate directions can be found, then by optimising successively along the n directions the minimum will be found

Powell discovered a direction-set method which produces n mutually conjugate directions

These directions are only truly conjugate provided that the objective function can be represented by a second-order Taylor series

In practice, therefore, it is necessary to repeat the n line optimisations several times

Powell's Method

Powell's direction set algorithm:

initialise the set of directions \mathbf{u}_i ($i = 1..n$) to the basis vectors and save the starting position as \mathbf{P}_0

```
for ( $j=1..n$ ) {  
    for ( $i=1..n$ ) find minimum along  $\mathbf{u}_i$  and call this point  $\mathbf{P}_i$   
    for ( $i=1..n-1$ ) set  $\mathbf{u}_i = \mathbf{u}_{i+1}$   
     $\mathbf{u}_n = \mathbf{P}_n - \mathbf{P}_0$   
    Find minimum along direction  $\mathbf{u}_n$  and call this point  $\mathbf{P}_0$   
}
```

Powell's Method

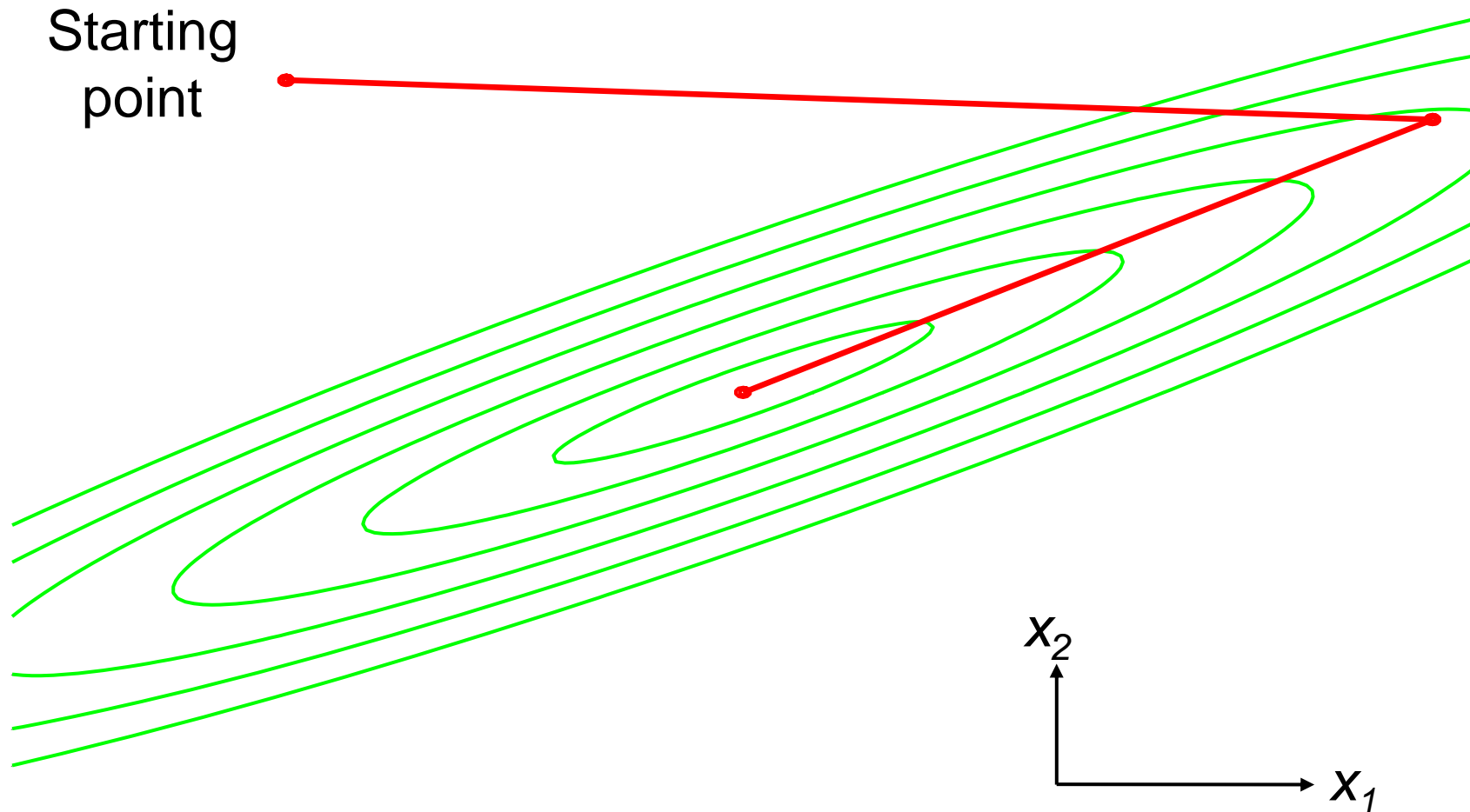
After 1 complete iteration of the algorithm ($j = 1..n$) the directions \mathbf{u}_i ($i = 1..n$) will be mutually conjugate

Minimisations will have been performed along each of the directions

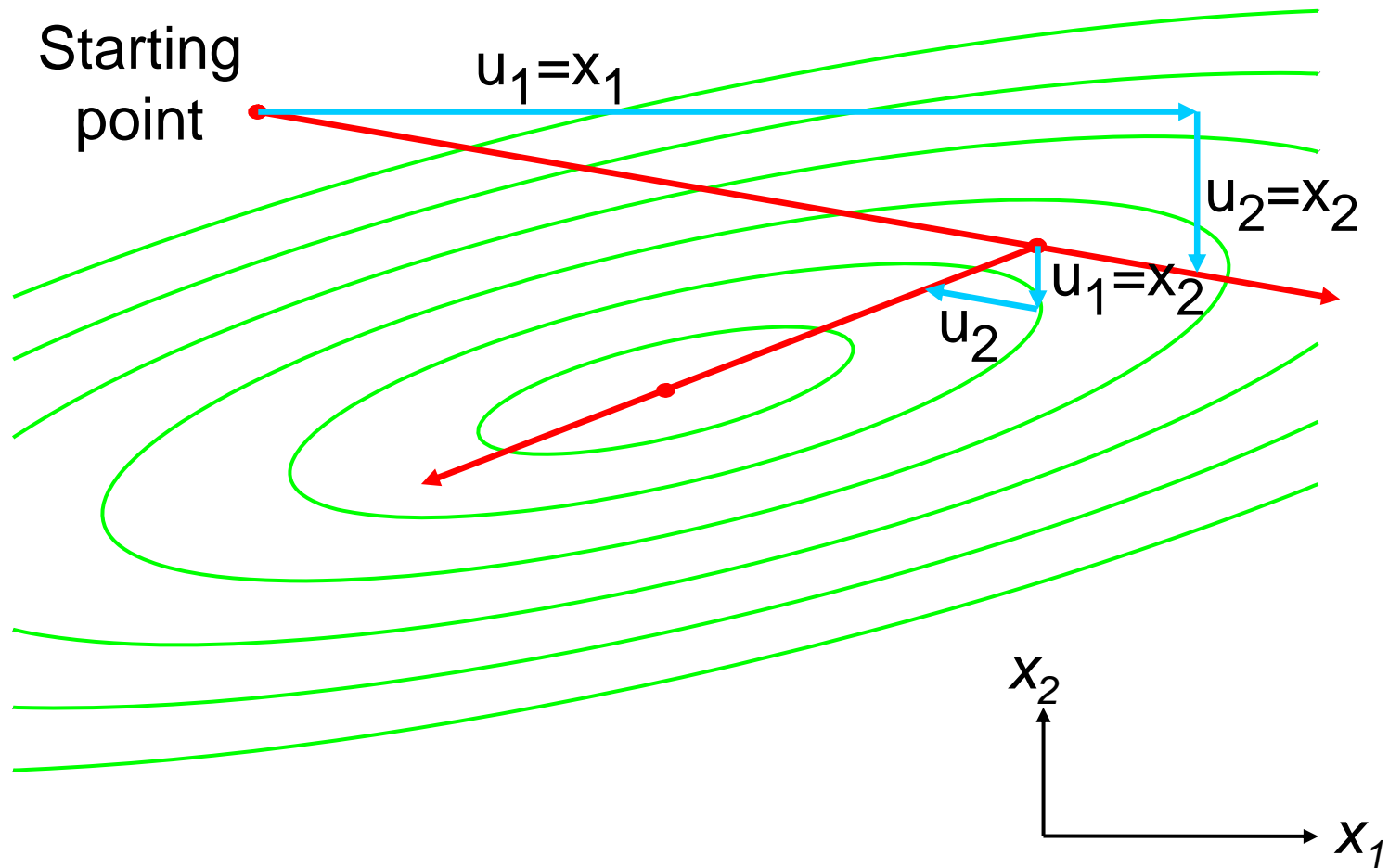
This will exactly minimise a quadratic objective function.

For realistic objective functions the algorithm must be repeated several times

Powell's Method



Powell's Method



Quasi-Newton Methods

Quasi-Newton methods are also known as Variable-Metric methods.

Quasi-Newton methods are the most efficient optimisation methods for a well-behaved objective function

Quasi-Newton methods make use of the first partial derivatives of the objective function.

There are several Quasi-Newton methods of which the best known is the Davidon-Fletcher-Powell (DFP) algorithm

Quasi-Newton Methods

The starting point for Quasi-Newton methods is the Taylor expansion of the objective function:

$$f(\mathbf{x} + \Delta\mathbf{x}) = f(\mathbf{x}) + \mathbf{g}^T(\mathbf{x})\Delta\mathbf{x} + \frac{1}{2} \Delta\mathbf{x}^T \mathbf{H}(\mathbf{x})\Delta\mathbf{x}$$

Suppose that \mathbf{x} is an approximation to the minimum, and that $\mathbf{x} + \Delta\mathbf{x}$ is the minimum

By definition, at the minimum: $\mathbf{g}(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{0}$ so that:

$$\mathbf{g}(\mathbf{x} + \Delta\mathbf{x}) = \nabla f(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{g}(\mathbf{x}) + \mathbf{H}(\mathbf{x})\Delta\mathbf{x} = \mathbf{0}$$

This equation can be solved for $\Delta\mathbf{x}$

Quasi-Newton Methods

Solving the equation:

$$\mathbf{g}(\mathbf{x}) + \mathbf{H}(\mathbf{x}) \Delta \mathbf{x} = \mathbf{0}$$

or:

$$\Delta \mathbf{x} = -\mathbf{H}^{-1}(\mathbf{x}) \mathbf{g}(\mathbf{x})$$

gives the distance $\Delta \mathbf{x}$ of the minimum from the current point \mathbf{x}

The optimum is therefore at: $\mathbf{x} + \Delta \mathbf{x}$

In practice, because the 2nd-order Taylor expansion is only an approximation, the point $\mathbf{x} + \Delta \mathbf{x}$ is not the exact minimum but is closer to it than \mathbf{x}

Quasi-Newton Methods

The equation for $\Delta\mathbf{x}$ requires the gradient vector \mathbf{g} and the Hessian matrix \mathbf{H} to be evaluated

The gradient \mathbf{g} can be determined numerically

The Hessian matrix \mathbf{H} could in principle be evaluated numerically, but this procedure would be time-consuming and of limited accuracy

Quasi-Newton methods do not use the actual Hessian matrix, but instead an approximation to it

Quasi-Newton Methods

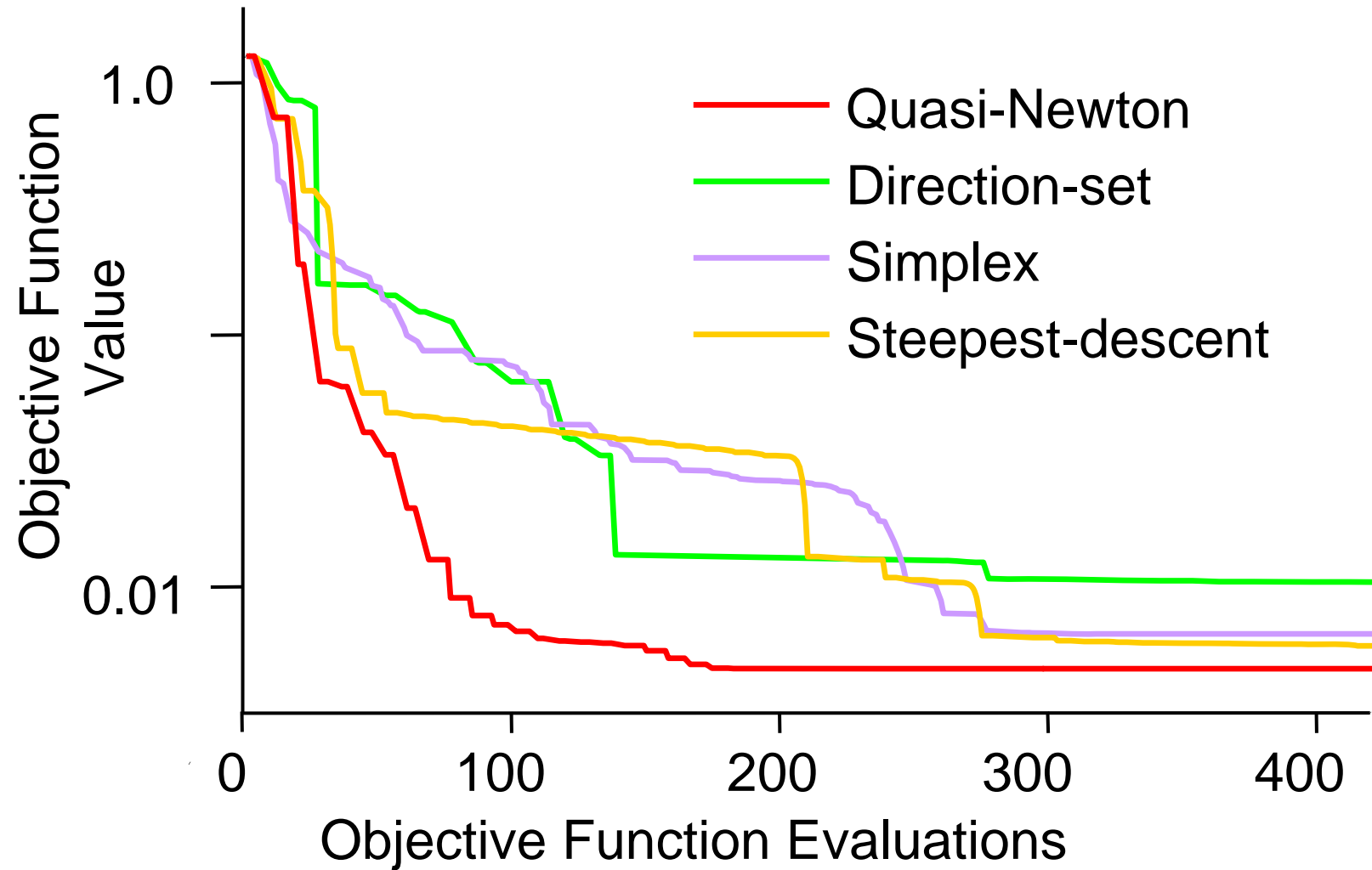
Quasi-Newton methods differ in the way that they update the Hessian matrix \mathbf{H} at each iteration from \mathbf{x} and \mathbf{g}

For example, the DFP method uses the formula:

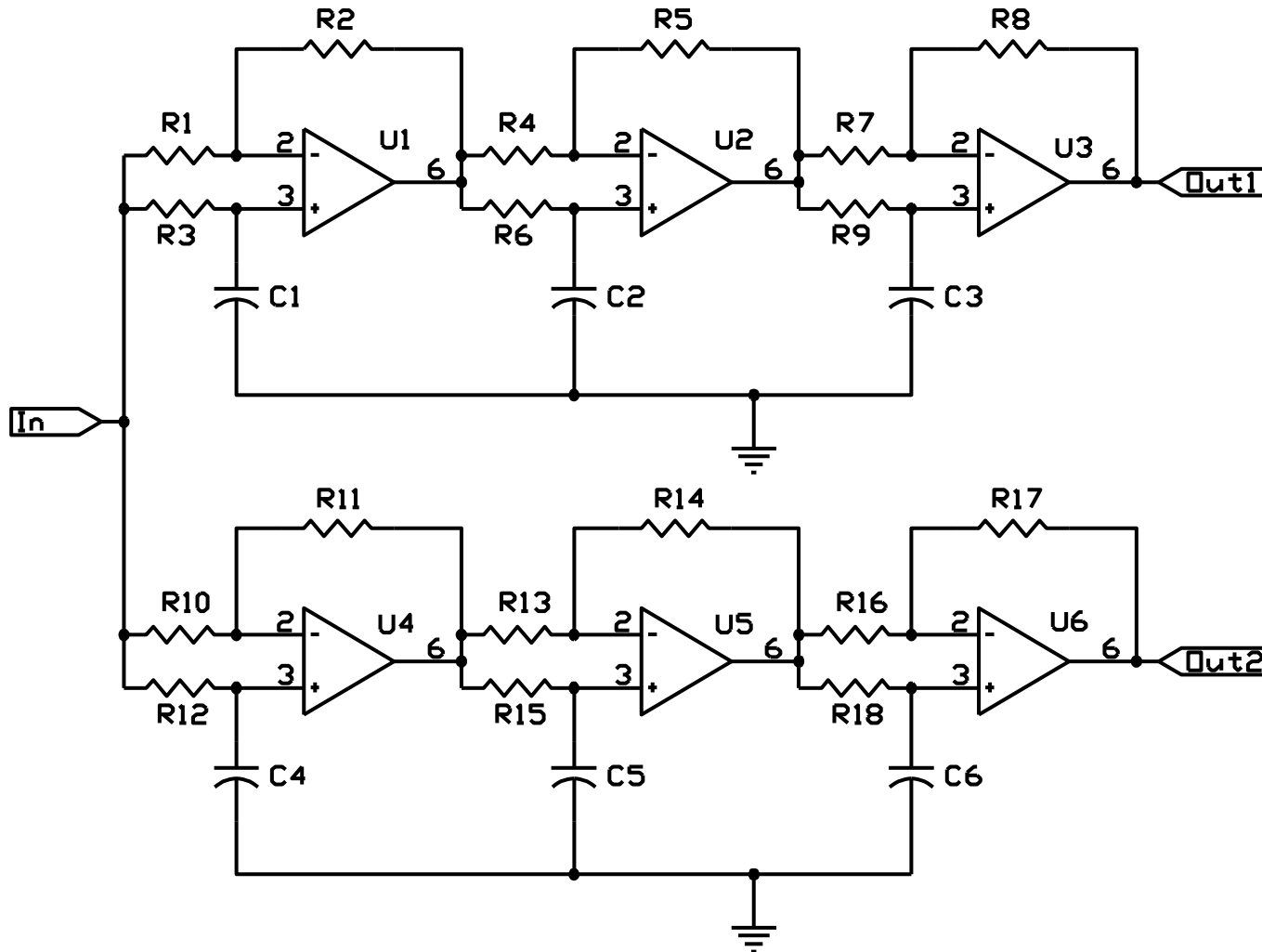
$$\mathbf{H}_{i+1} = \mathbf{H}_i + \frac{(\mathbf{x}_{i+1} - \mathbf{x}_i) \otimes (\mathbf{x}_{i+1} - \mathbf{x}_i)}{(\mathbf{x}_{i+1} - \mathbf{x}_i) \cdot (\mathbf{g}_{i+1} - \mathbf{g}_i)} - \frac{[\mathbf{H}_i \cdot (\mathbf{g}_{i+1} - \mathbf{g}_i)] \otimes [\mathbf{H}_i \cdot (\mathbf{g}_{i+1} - \mathbf{g}_i)]}{(\mathbf{g}_{i+1} - \mathbf{g}_i) \cdot \mathbf{H}_i \cdot (\mathbf{g}_{i+1} - \mathbf{g}_i)}$$

where \otimes denote the outer product of two vectors

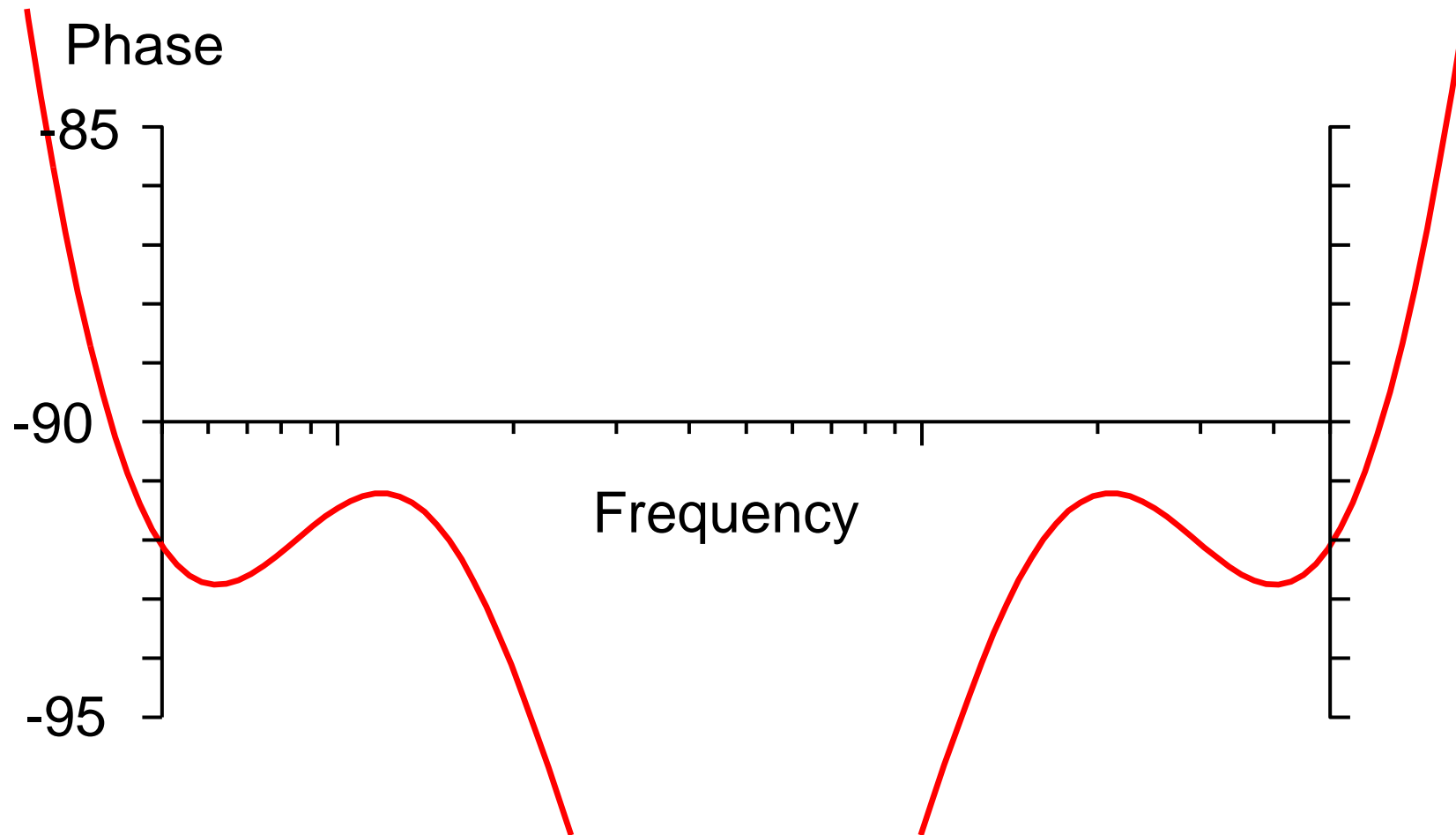
Convergence



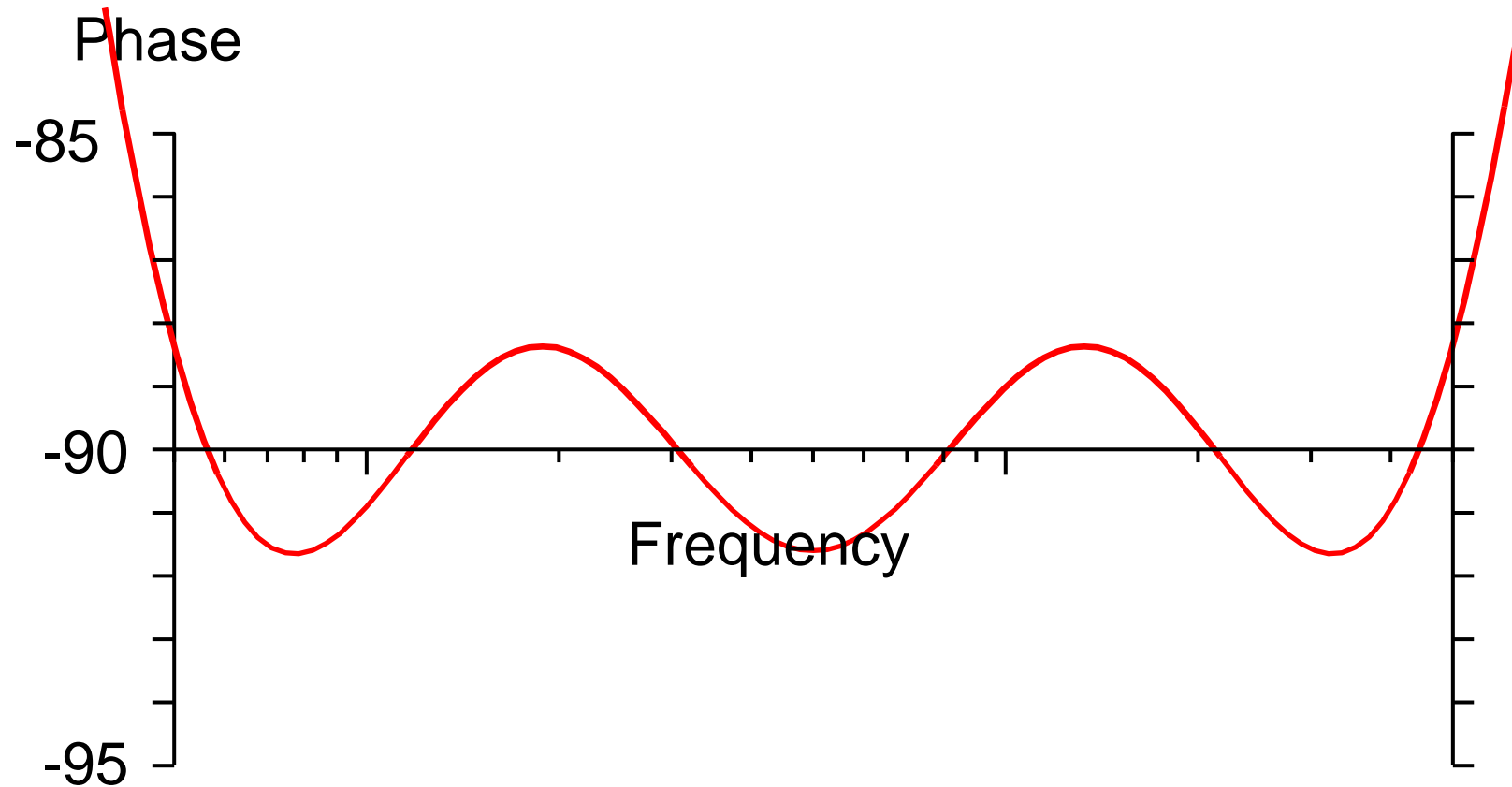
Example of Optimisation



Example of Optimisation



Example of Optimisation



Constrained Variables

Unconstrained optimisation may lead to impractical values for the variables

For example, the value of a resistor may become negative, or the value of a capacitor exceed 1 F

There are two ways of preventing this:

1. Add a penalty function to the objective
2. Transform the optimisation variables

Penalty Function

A penalty function is used to artificially increase the objective function f when a constraint violation occurs

Suppose that the variable x_i must not become negative. A possible penalty function would be:

$$\begin{aligned} p_i &= 0.0 && \text{for } x_i \geq 0.0 \\ &= 1000.0 && \text{for } x_i < 0.0 \end{aligned}$$

A better penalty function would be:

$$p_i = e^{-x_i}$$

Variable Transformations

Normally the optimisation variable x_j can take on any value from $-\infty$ to ∞

Suppose that the corresponding component value R_j must be constrained to be positive

Instead of the component value R_j being equal to the optimisation variable x_j , it can be derived from x_j by a transformation:

$$R_j = x_j^2$$

Now R_j is restricted to the range 0 to ∞

Variable Transformations

Suppose that a component value C_i must be constrained to lie in the range 0 to 1:

$$C_i = \frac{1}{1 + x_i^2} \quad \text{or} \quad C_i = \frac{1}{2}(1 + \sin x_i)$$

The simplest transformation that achieves the required result should be used

A complex transformation leads to a more complicated objective function, which in turn make optimisation less efficient.

Genetic Optimisation

Genetic optimisation attempts to exploit the remarkable success of natural selection in adapting organisms to suit their environment

A population is maintained, each individual of which has a unique set of genes (genome) and corresponds to a design solution

Genetic optimisation improves the general fitness of the population by selective breeding

Genetic Optimisation

A set of genes contains all the information necessary to construct an individual

Genes are stored in a “chromosome”

Pairs are randomly selected from the population for breeding but with a bias towards the fittest

Pairs reproduce by merging their chromosomes

Reproduction is sometimes followed by mutation

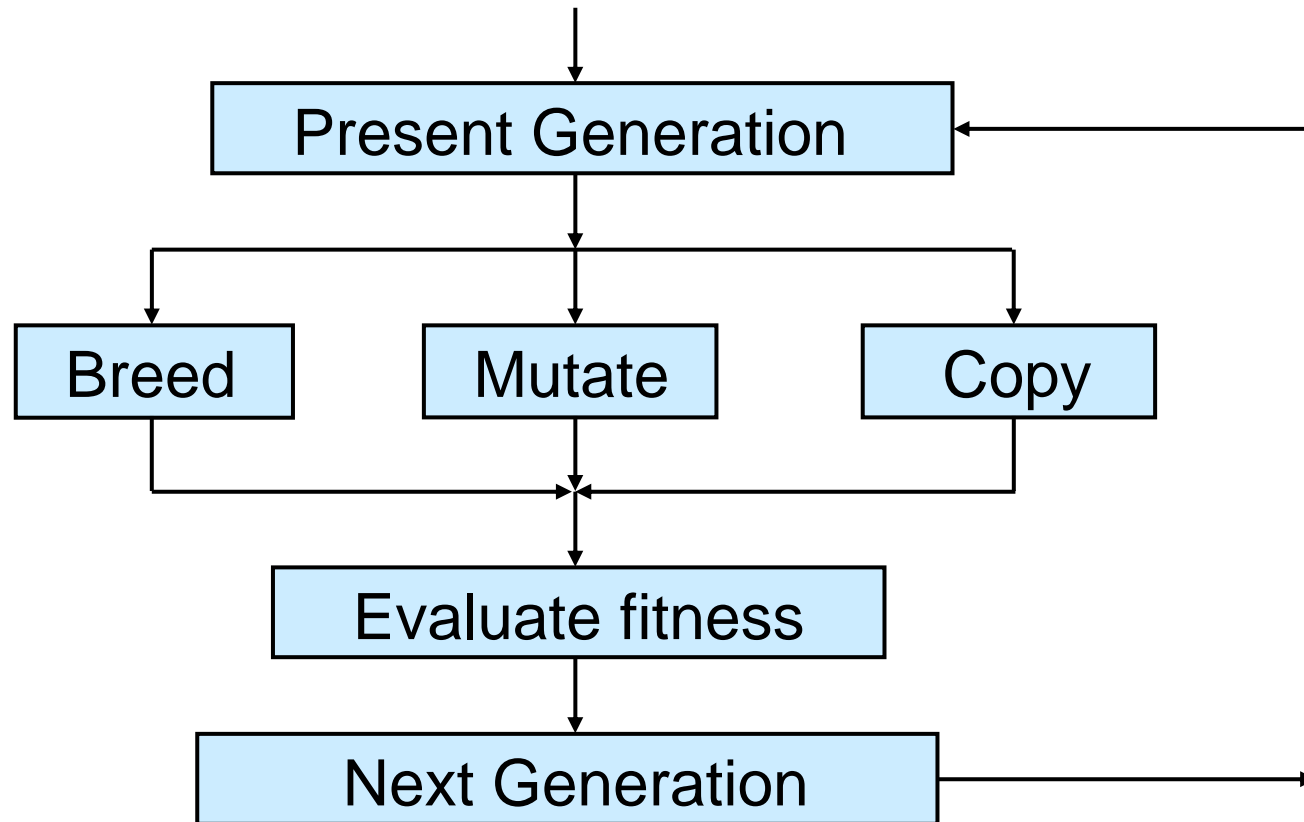
Genetic Optimisation

Genetic optimisation is simple and robust. It can be used to optimise systems which resist traditional optimisation methods (epistatic systems)

Genetic optimisation has been applied to such diverse problems as:

- Structural design
- Class-room scheduling
- Pattern recognition
- Cutting shapes from cloth
- VLSI layout

Genetic Optimisation

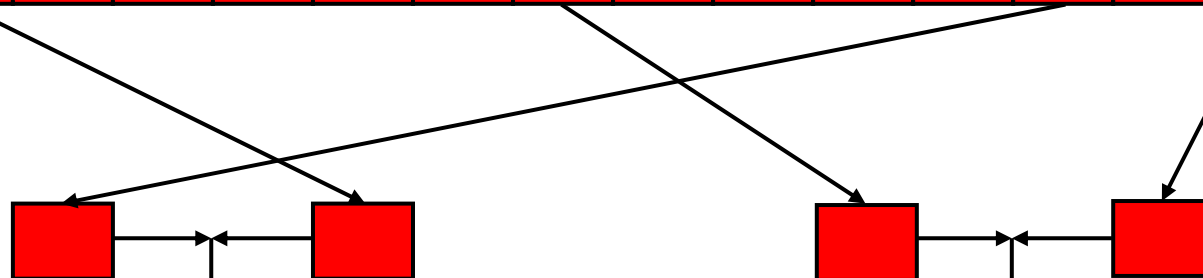


Selection of Parents

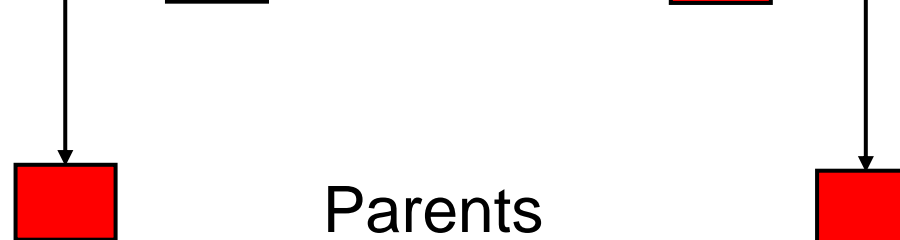
Population



Random selection



Tournament



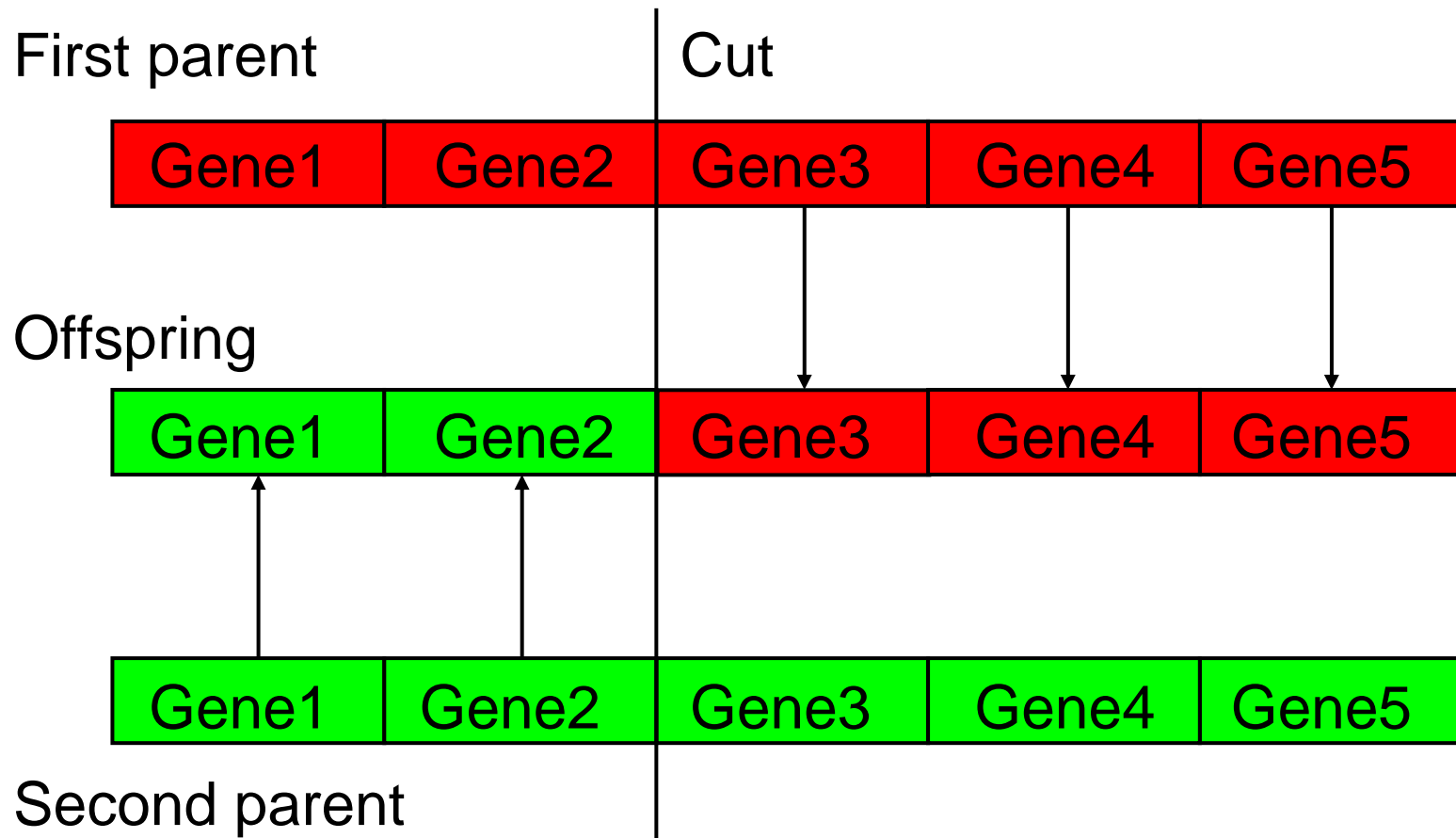
Parents

Genetic cross-over



Offspring

Single-Point Cross-over

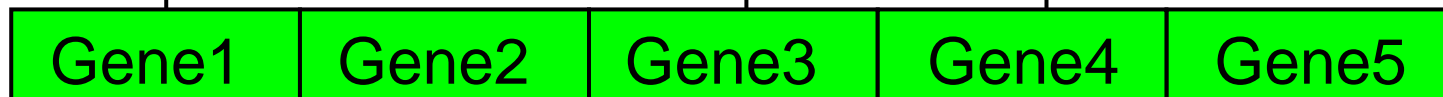
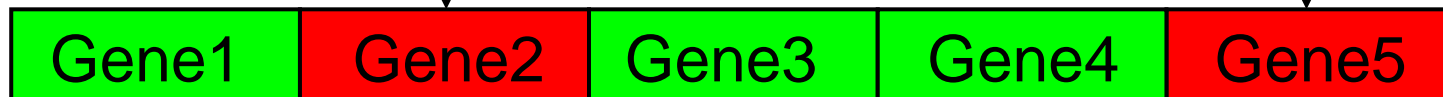


Uniform Cross-over

First parent

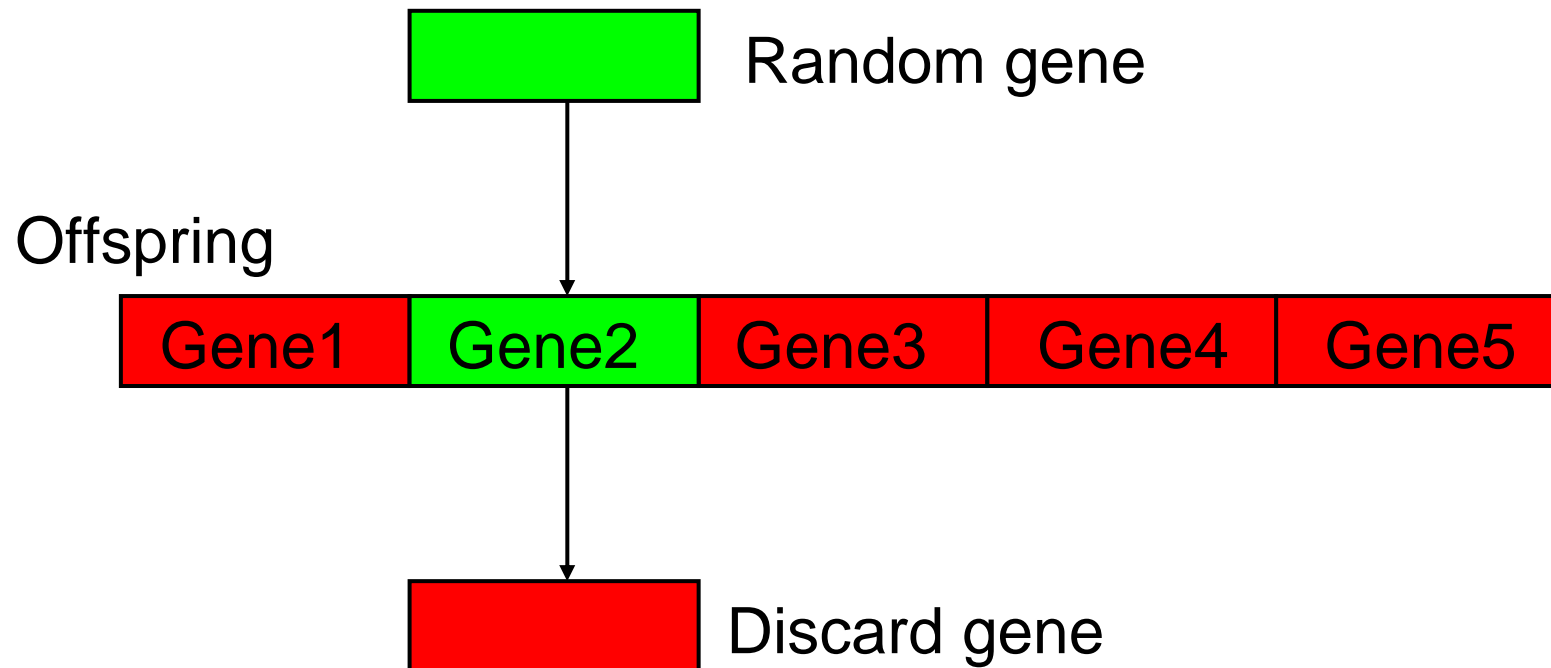


Offspring

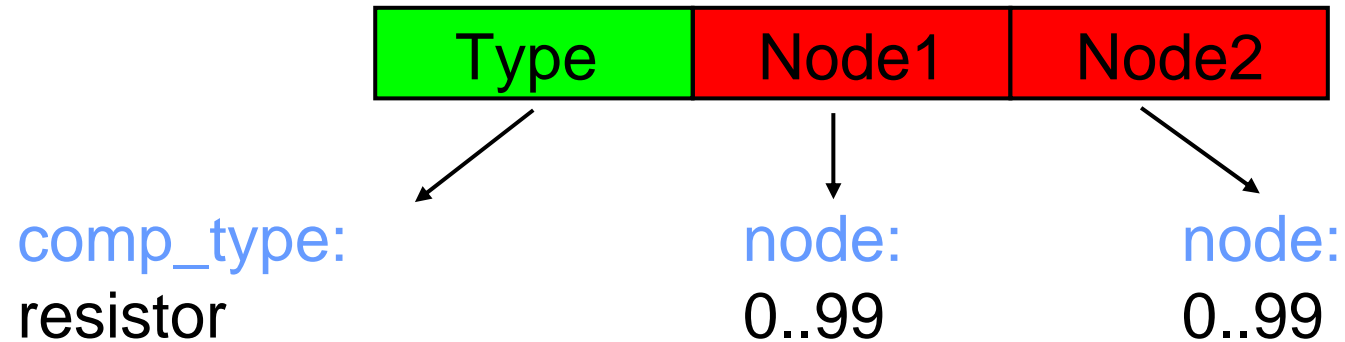


Second parent

Mutation



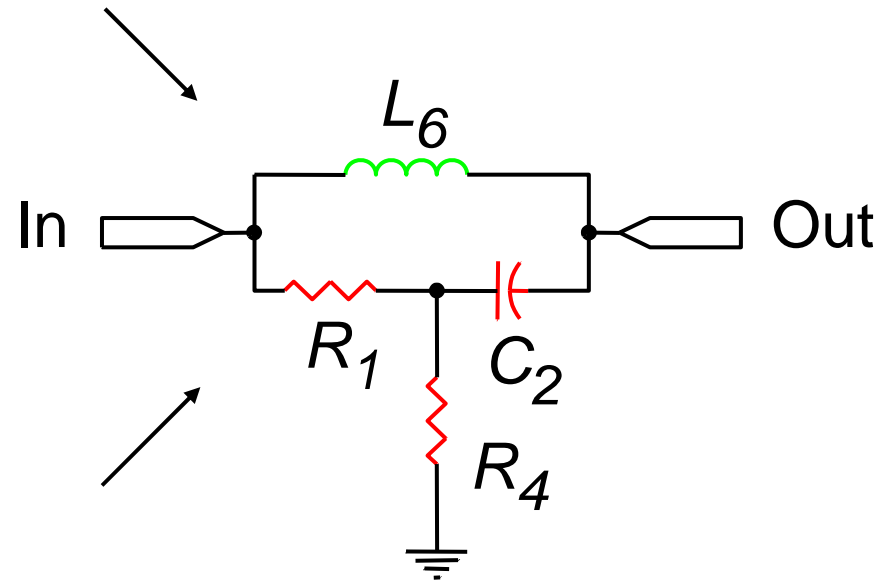
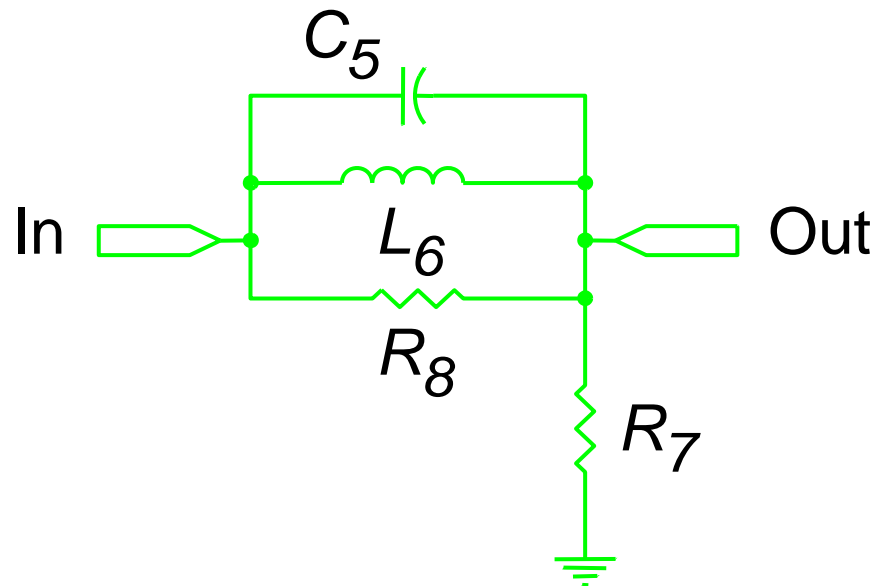
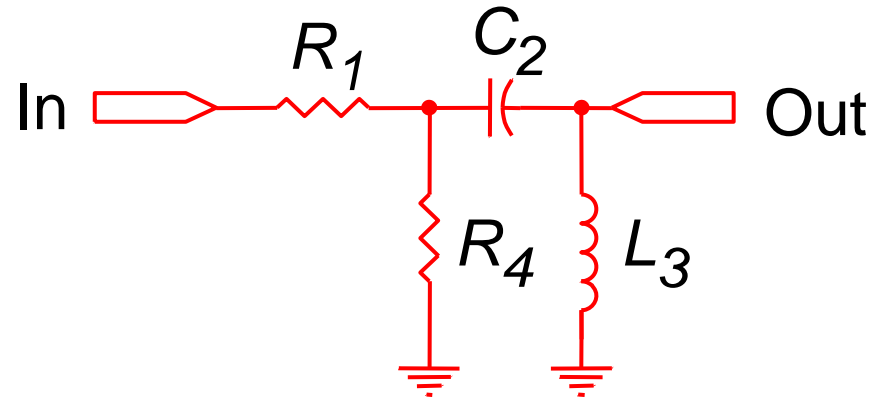
Gene Structure



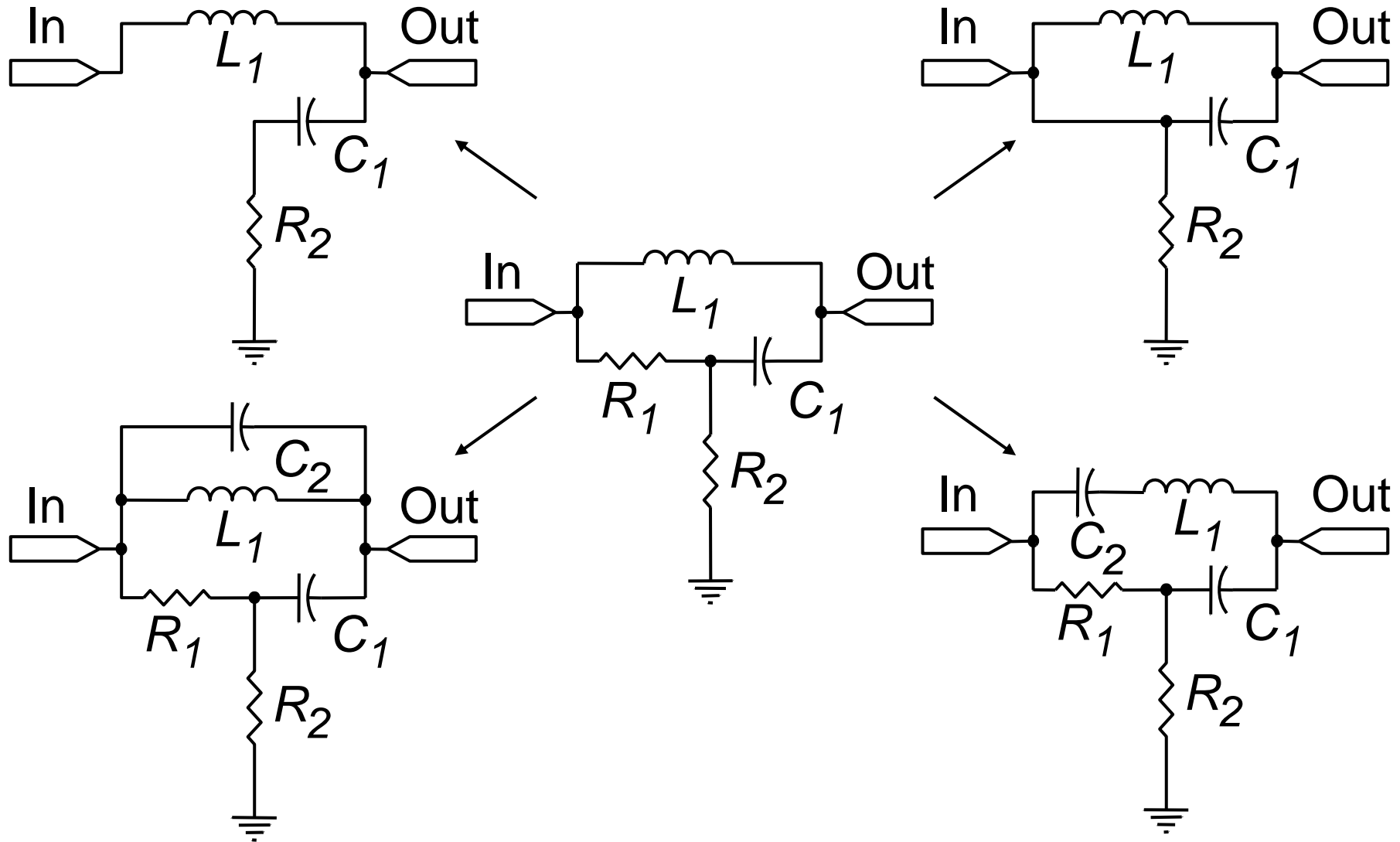
comp_type:
resistor
capacitor
inductor
null

```
class gene {  
private:  
    comp_type type;  
    node n1, n2;  
public:  
    gene();  
    ~gene();  
    . . . . .  
};
```

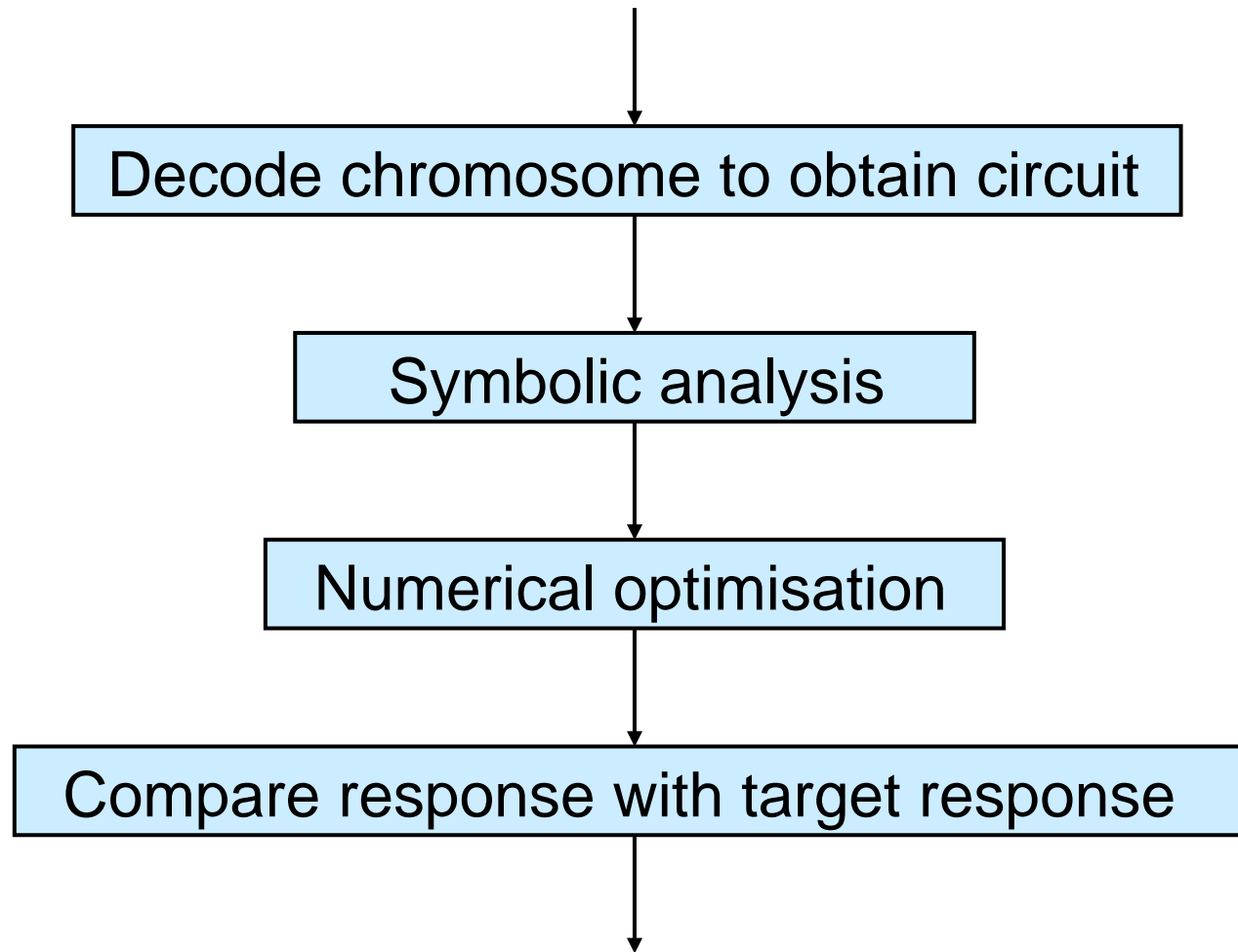
Circuit Cross-Over



Circuit Mutations



Fitness Evaluation



Genetic Optimisation

Low-Pass filter specification:

Pass-band edge = 1 Hz

Stop-band edge = 1.5 Hz

Max pass-band gain = -6 dB

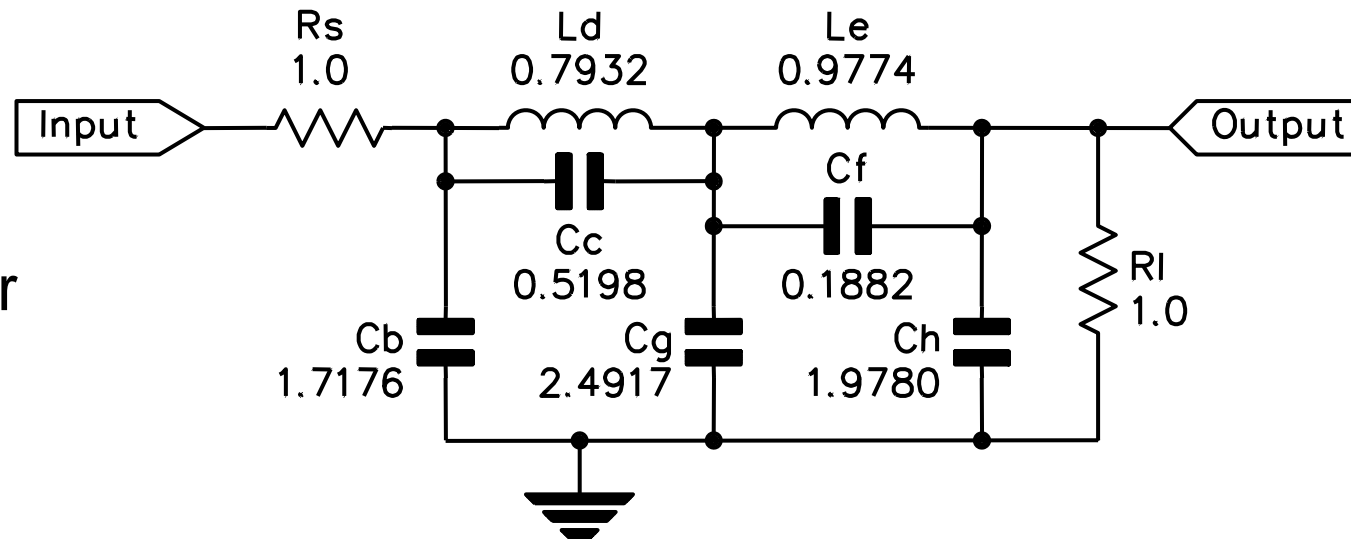
Min pass-band gain = -7 dB

Max stopband gain = -52 dB

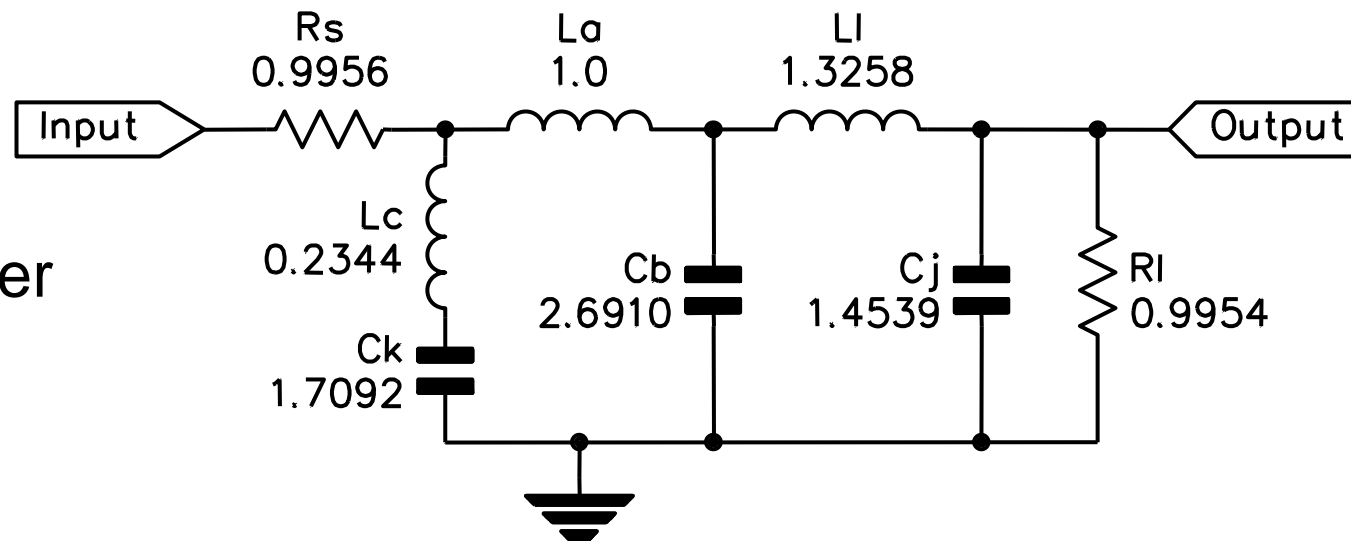
Traditional design methods would lead to a 5th-order elliptic filter which could be implemented as an equally-terminated ladder

Genetic Optimisation

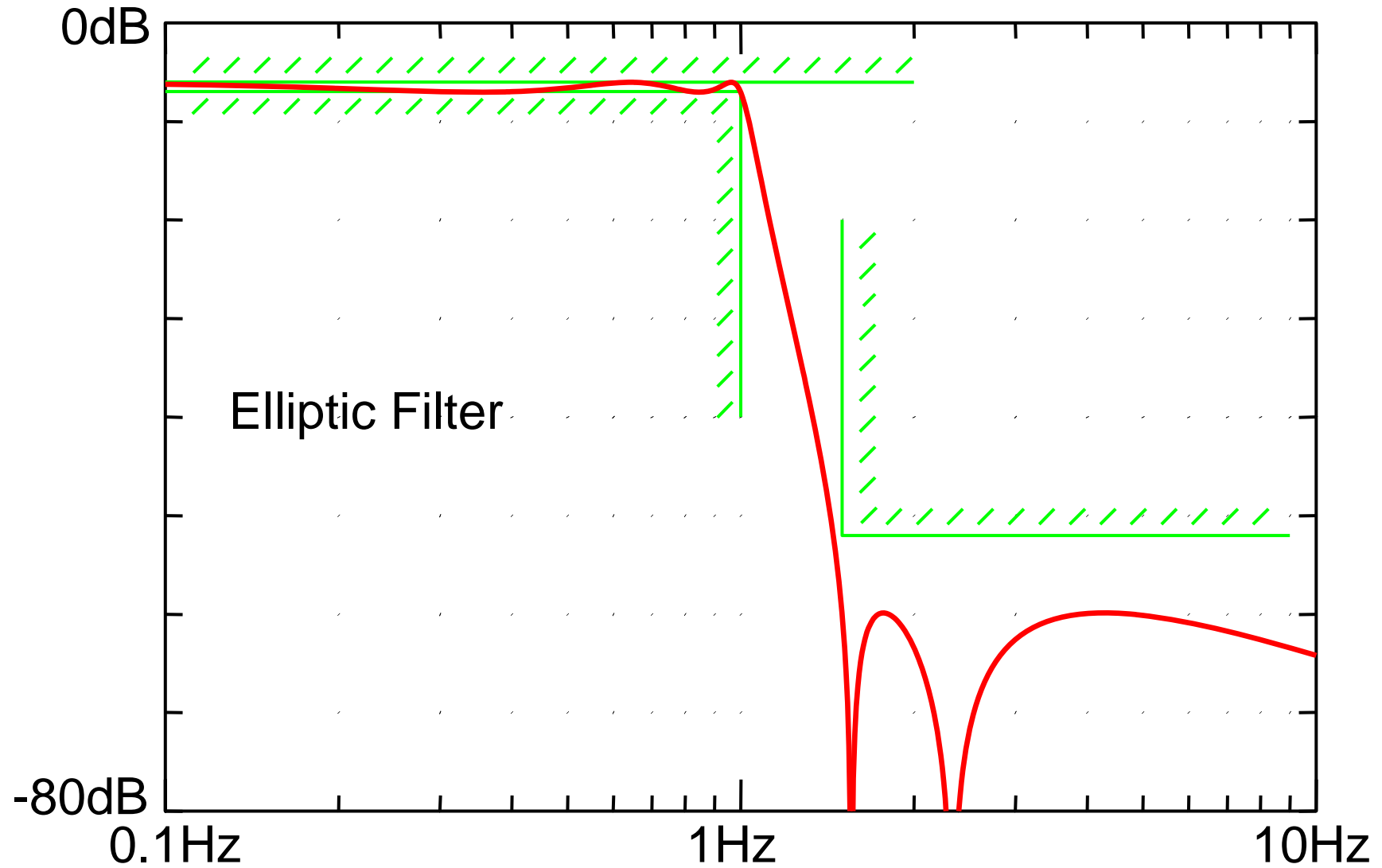
Elliptic Filter



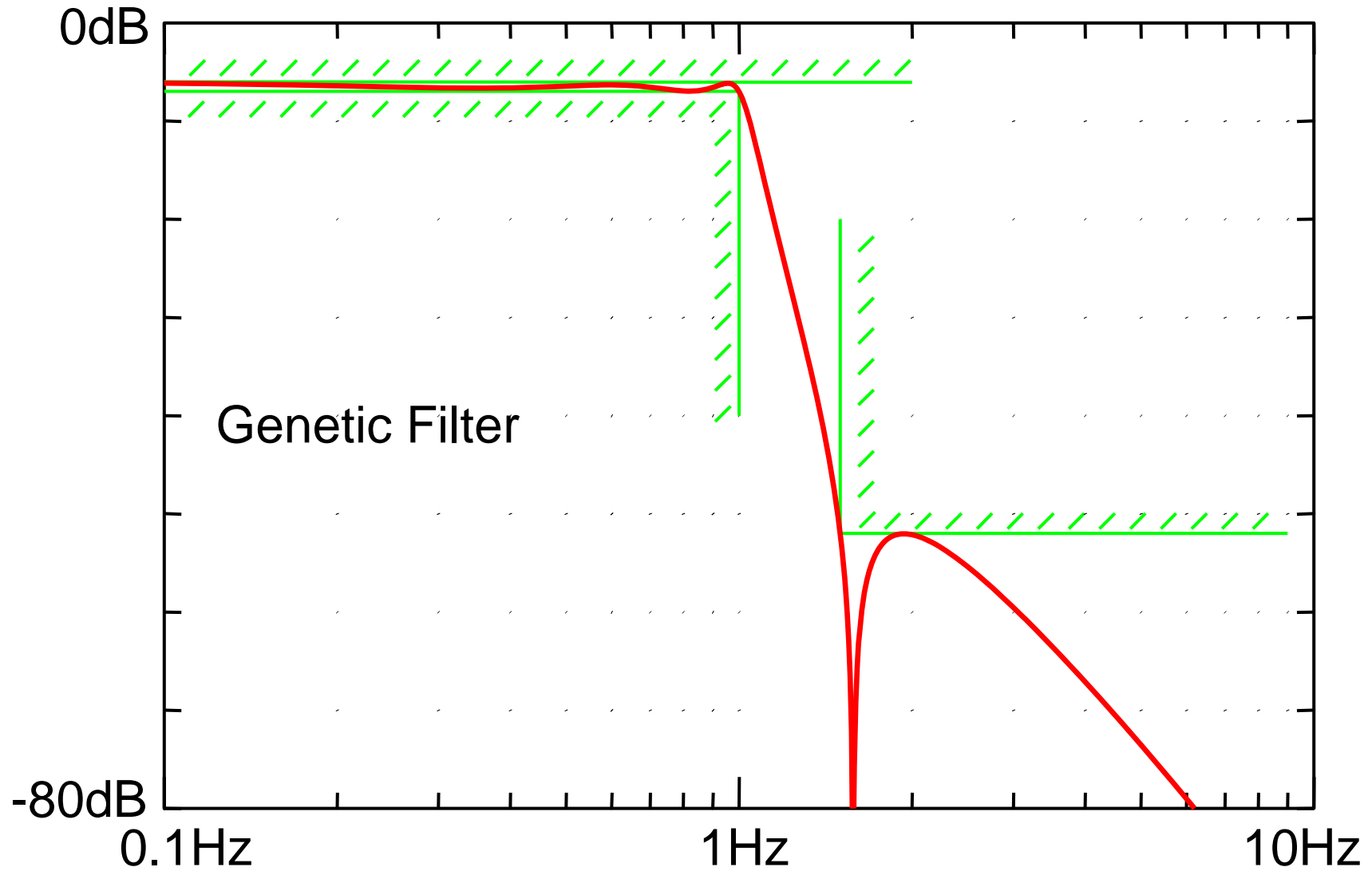
Genetic Filter



Genetic Optimisation



Genetic Optimisation



Genetic Optimisation

Highly-asymmetric band-pass filter specification for a modem application:

Pass-band: 31.2 kHz to 45.6 kHz

Maximum pass-band ripple: 0.6 dB

Lower stop-band edge: 20 kHz

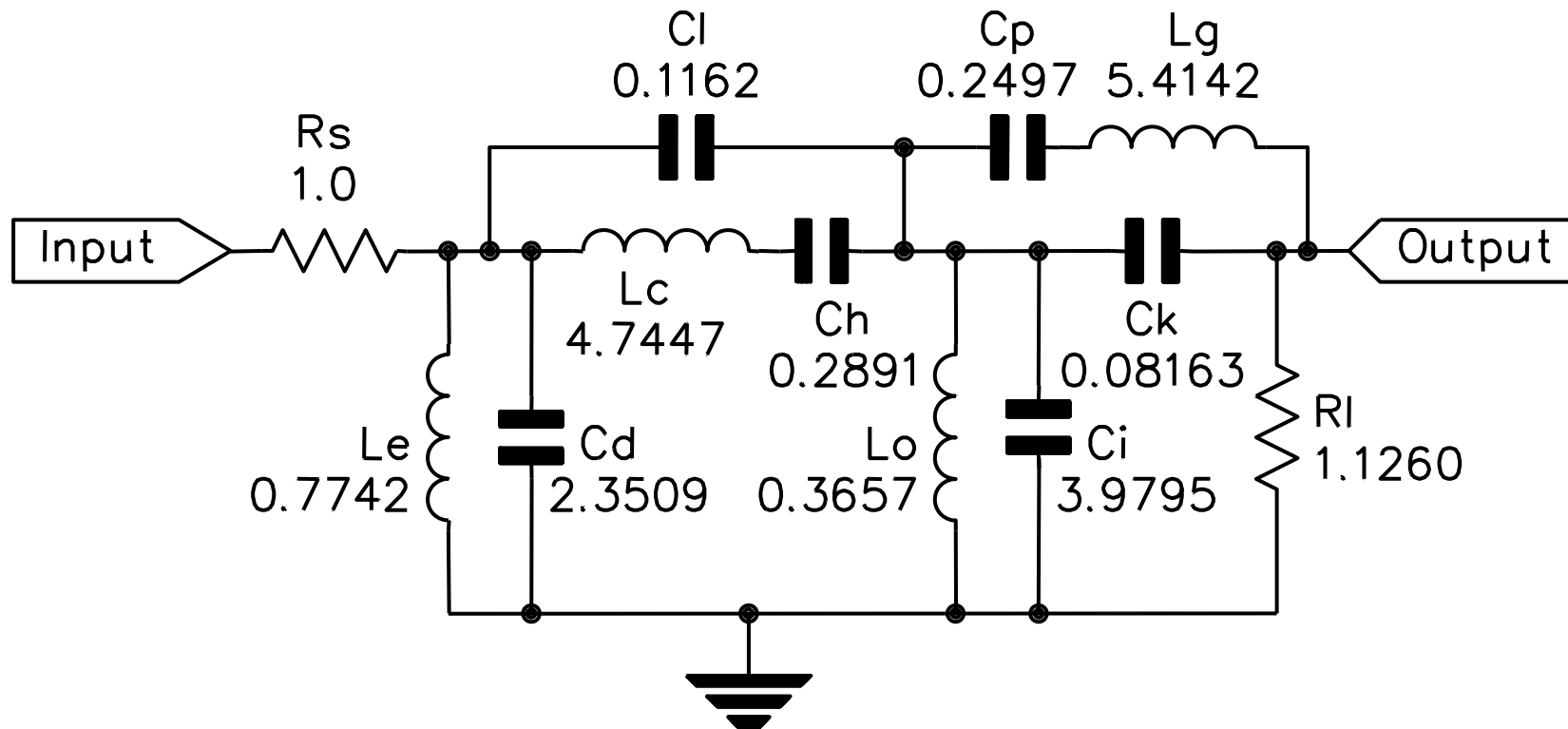
Lower stop-band gain: < -38 dB

Upper stop-band: 69.6 kHz to 84.0 kHz

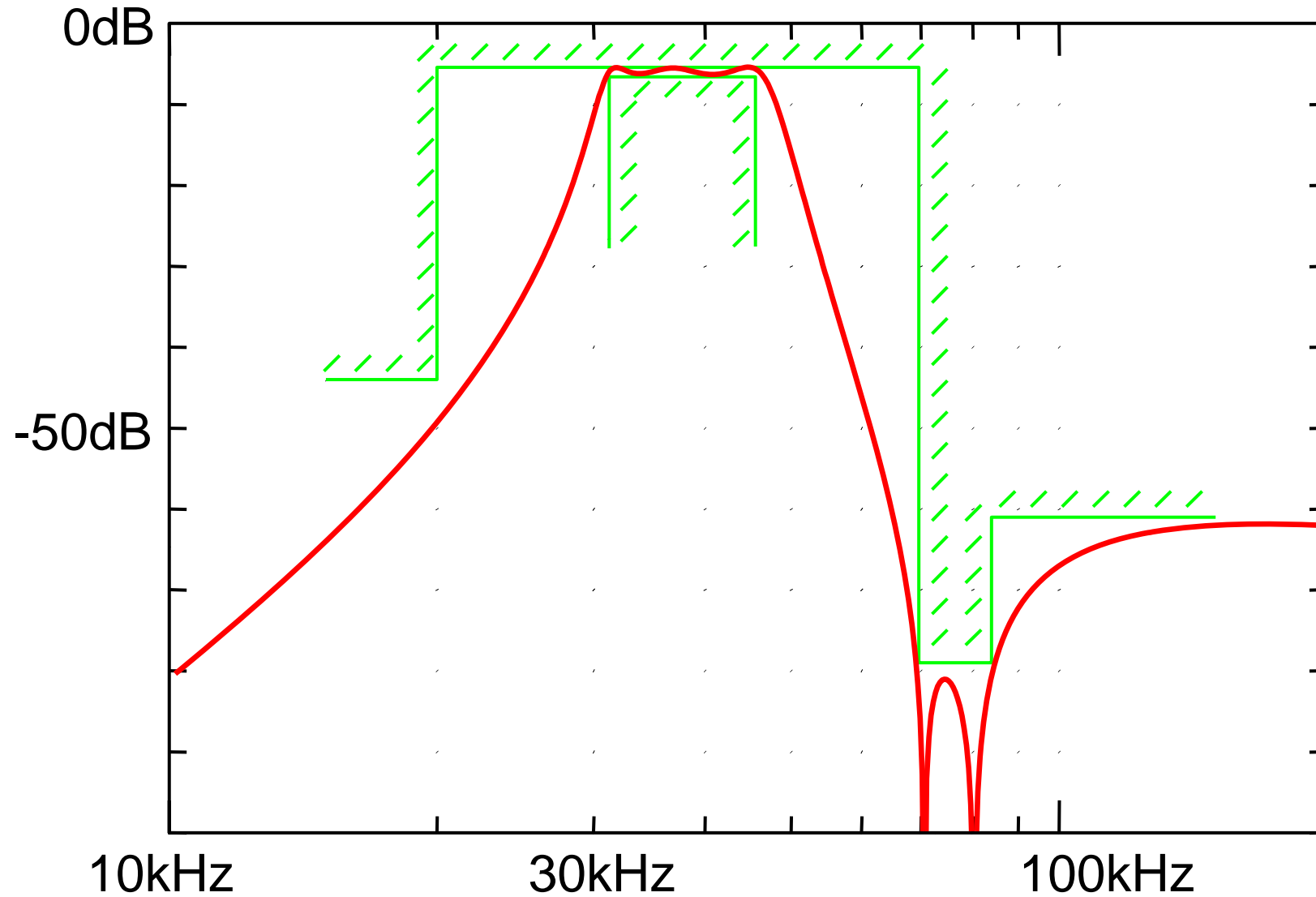
Upper stop-band gain: < -73 dB

Gain above stop-band: < -55 dB

Genetic Optimisation



Genetic Optimisation



Circuit Optimisation

© J. B. Grimbleby, October 07