CHEM1047 – Week 5 Lecture 1 – Numerical optimisation

☐ *Chapter 21 of Monk and Munro, "Maths for Chemistry", 2nd edition.*

☐ *Section 20.3 of Steiner, "The Chemistry Maths Book", 2nd edition.*

In modern chemical research, a new experiment is rarely attempted without prior detailed theoretical modelling. This involves finding solutions of equations or finding extrema of functions for which analytical root finding and optimisation are not available – most real-life equations either involve transcendental functions, or cannot actually be written in a closed form. Sometimes analytical solutions are not even wanted because numerical methods are faster on modern computers.

1. Numerical root finding

Many methods exist for numerical root finding. One that is simple enough to be used in manual calculations, and efficient enough to produce a solution in reasonable time, is *Newton-Raphson method*. It involves building tangent lines – linear approximations that touch the given function at one point and have the same slope at that point.

For a given function $f(x)$ at some point x_n, the value is $f(x_n)$ and the slope is $f'(x_n)$. Asking which linear function has that value and that slope at $x = x_n$ results in:

$$y = f'(x_n)(x - x_n) + f(x_n) \tag{1}$$

Isaac Newton observed in 1669 (and Joseph Raphson did, independently, in 1690) that this equation may be used to get a recursive sequence that moves closer and closer to the root.

The procedure involves finding the root of the tangent line instead of the root of the function, and then using that root as the origin of the next tangent line:

$$f'(x_n)(x_{n+1} - x_n) + f(x_n) = 0 \quad \Rightarrow \quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2}$$

At each step, this algorithm replaces the difficult problem of finding the root of the original function with the simple problem of finding the root of its tangent line (Figure 1).

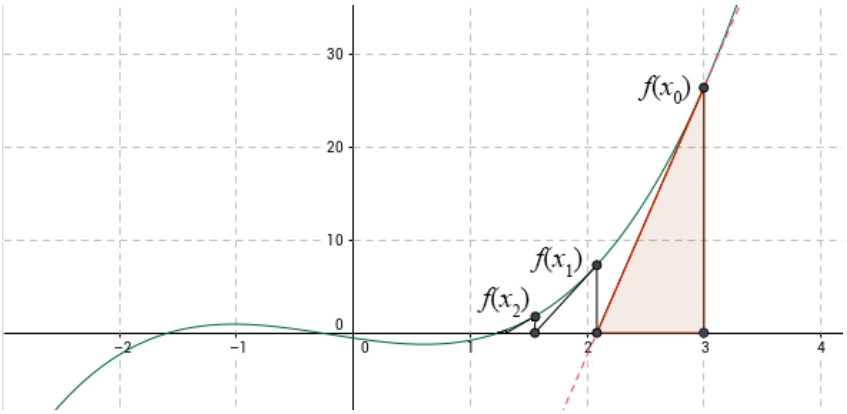

Figure 1. *A schematic illustration of the Newton-Raphson root finding method. A tangent line is drawn at the starting point, and a zero of that tangent line is found. This becomes the new starting point, and the process is repeated until the point stops changing to the accuracy required.*

The point x_n moves closer and closer to a root of $f(x)$ when n is increased. Newton-Raphson algorithm requires an initial guess x_0, and converges to the nearest root. It may fail to converge if the initial guess is not well chosen, and will fail to converge if the function has no zeroes.

Example: find the solution of $\exp(x)-5x=0$ to four decimal places using Newton-Raphson method starting at $x_0=3$.

Solution: the corresponding Newton-Raphson iteration equation is

$$x_{n+1}=x_n-\frac{\exp(x_n)-5x_n}{\exp(x_n)-5}$$

If we choose $x_0=3$ as the initial guess, we get the following sequence: $x_1=2.6629$, $x_2=2.5533$, $x_3=2.5427$, $x_4=2.5426$. The fourth decimal place stops changing beyond $n=4$, meaning that the root is found to the precision required.

2. Numerical optimisation

Numerical optimisation of univariate functions is an instance of the numerical root-finding problem described above. We need to find a point at which the derivative of the function is zero:

$$f'(x)=0 \tag{3}$$

This may be accomplished using the same Newton-Raphson iteration, in which the derivative of the function is now featured in the numerator and the second derivative in the denominator:

$$x_{n+1}=x_n-\frac{f'(x_n)}{f''(x_n)} \tag{4}$$

The Newton-Raphson method converges to the nearest stationary point. It is the user's responsibility to find out what the nature of that point is.

Example: find the nearest stationary point of $f(x)=x^3-e^x$ to $x=1$ to four decimal places and demonstrate that it is a minimum.

Solution: the Newton-Raphson iteration for this problem is

$$x_{n+1}=x_n-\frac{3x_n^2-e^{x_n}}{6x_n-e^{x_n}}$$

Starting from $x_0=1.0000$, the iterations are $x_1=0.9142$, $x_2=0.9100$ and $x_3=0.9100$, at which point the fourth decimal place stops changing. The second derivative of the function at this point is positive, $f''(0.9100)=2.9757$, and therefore the point is a minimum.

An extension of this procedure to multivariate functions does exist, but tends to be problematic because the denominator in the multivariate version on Equation (4) has singularities at points where second derivatives are zero. It turns out that too many such points exist for multivariate functions. A simpler procedure is therefore often used, where the location of stationary points is determined using only first derivative information: the computer starts at some user-specified location and moves up or down the direction of steepest ascent. This direction is called *gradient vector* $\vec{\nabla}f$:

$$\vec{\nabla}f(\vec{r})=\mathrm{grad}\left[f(\vec{r})\right]=\left(\frac{\partial f}{\partial x}\quad\frac{\partial f}{\partial y}\quad\ldots\right) \tag{5}$$

In practice, the gradient is obtained by computing first partial derivatives at each point in the optimisation trajectory. Algorithms that descend along the gradient direction are called *gradient descent* methods. Given a starting point $\vec{r}_0$, gradient descent methods proceed in the following way:

$$\vec{r}_{n+1} = \vec{r}_n - \alpha \cdot \vec{\nabla} f\left(\vec{r}_n\right) \tag{6}$$

where α is a step length. In the simplest case it is fixed, but more sophisticated methods exist that estimate the optimal step length using local curvature information.

3. Non-linear least squares

The *non-linear least squares* method is a simple extension of the linear case to an arbitrary function:

$$\Omega(a,b,\ldots) = \sum_n \left[f\left(a,b,\ldots,x_n\right) - y_n \right]^2 \tag{7}$$

The error functional is the same – the sum of squares of deviations. An example is shown in Figure 2.

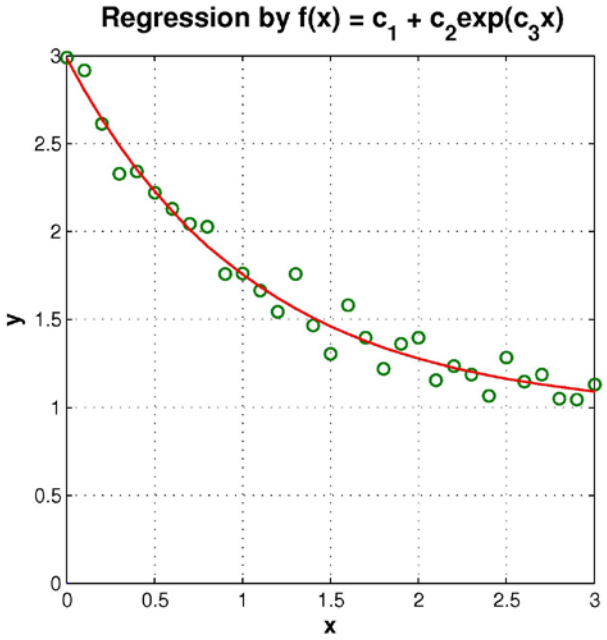

Figure 2. *An example of a non-linear least squares fit to a function with three parameters, performed numerically in Matlab.*

The minimum of the functional in Equation (7) is no longer easy to find analytically. A simple expression for $f\left(a,b,\ldots,x_n\right)$ may not even exist – the function may be an output of a numerical simulation or the readout from some instrument. In such cases, the minimisation proceeds using gradient descent.

4. Molecular geometry optimisation

The task of refining a molecular geometry from some initial guess (provided by chemical intuition) amounts to finding the minimum of the energy with respect to the atomic coordinates:

$$\left\{\vec{r}_1,\vec{r}_2,\ldots\right\} = \underset{\{\vec{r}_1,\vec{r}_2,\ldots\}}{\arg\min}\, E\left(\vec{r}_1,\vec{r}_2,\ldots\right) \tag{8}$$

where *argmin* stands for "argument at the minimum" and $\vec{r}_k = \begin{bmatrix} x_k & y_k & z_k \end{bmatrix}$ are Cartesian coordinates of the *k*-th atom. It is common to use $\vec{R}$ symbol to denote a vector containing all coordinates of all atoms. The minimum of the energy with respect to $\vec{R}$ is called *equilibrium geometry* because static forces on

each atom at the energy minimum are zero. Basic quantum chemistry software packages refine molecular geometries using gradient descent:

$$\vec{R}^{(n+1)} = \vec{R}^{(n)} - k \cdot \mathrm{grad}\left[E\left(\vec{R}\right)\right] \tag{9}$$

where the energy and its gradient are obtained by solving Schrödinger's equation. Minima of the molecular energy correspond to stable molecular geometries, and saddle points to chemical and conformational transition states.