Finding Minima of Functions

Potential energy and stable equilibrium

In mechanics you have met the potential energy, \( V(x) \), of a particle, which varies with the particle’s position, \( x \). A simple example is the potential \( V(x) = \frac{1}{2} k x^2 \) for a harmonic oscillator. Consider a particle moving along the x axis - the force on it in the x direction at the point \( x \) is the negative gradient (slope) of the potential energy function at that point,

\[
F(x) = -\frac{dV(x)}{dx}
\]

For the harmonic oscillator potential the negative gradient of the potential is \( F(x) = -kx \), which is Hooke’s law.

The force acting on a particle is zero at a stationary point of the function, \( dV(x)/dx = 0 \). Close to a potential energy minimum, the force is always towards the bottom of the potential well, while near a maximum the force is away from the top of the potential energy maximum. So the bottom of a potential well is a point of stable equilibrium, while a maximum of the potential is a point of unstable equilibrium, from which the particle tends to move away under the slightest perturbation. A point of inflection is also a point of unstable equilibrium. It is important to identify the states of stable equilibrium of particles or systems of particles, and this involves finding the minima of potential energy functions. For one-dimensional problems this means we must find the points where the slope of \( V(x) \) is zero, and its second derivative is positive.

The bisection method for finding roots of a function

The bisection method for finding roots for a function, \( f(x) = 0 \), is very simple: choose a value for \( x_1 \) for which \( f(x_1) \) is negative and \( x_3 \) for which \( f(x_3) \) is positive. Then evaluate \( f(x_2) \) for the average of these two values, \( x_2 = (x_1 + x_3) / 2 \). If \( f(x_2) \) is positive, replace \( x_3 \) by \( x_2 \) and if \( f(x_2) \) is negative, replace \( x_1 \) by \( x_2 \). Repeat (or iterate) this procedure until \( |f(x_2)| \) is less than some small value (tolerance) you choose.
Numerical accuracy in scientific computing

In mathematics there are analytical methods for finding solutions to many forms of equations. However, in many cases analytic methods are not available and we must seek numerical solutions. Numerical solutions are not, generally speaking, exact solutions to the equation concerned. Instead they are approximate solutions. For example, in the problem set here we are looking for solutions to the problem, \( f(x) = 0 \), where \( f \) is a quadratic function of \( x \). In this particular case the analytic solution is given by

\[
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\]

The numerical method we will use does not give this result exactly (unless we choose the initial guesses for the root so that we do end up coincidentally with the exact root). Instead we solve a similar problem, \(|f(x)| < \text{min}\), where \( \text{min} \) is some small number which we choose.

Python Scripting

(1) Open a new file with the editor of your choice (such as gedit) and name it \textit{script.py}

(2) At the top of the file enter the line

\texttt{#!/usr/bin/python}

This invokes the Python interpreter before the script begins running.

(3) A Python script is a sequence of Python commands entered in a file and usually has the descriptor \texttt{.py}. Enter the lines below after the line already entered in \textit{script.py}

\begin{verbatim}
import matplotlib.pylab as plt
import numpy as np
a = 1
b = 1
c = -6
x = np.arange(-5.0, 5.0, 0.2)
plt.plot(x, a * x * x + b * x + c)
plt.plot(x, 0.0 * x)
plt.show()
\end{verbatim}

Save \textit{script.py} using your file editor. In order to generate a list of values, \( x \), from -5.0 to 5.0 in steps of 0.2, \textit{script.py} uses the \texttt{arange} function in \textit{numpy}. The two \texttt{plot()} functions plot the
parabola determined by the values of a, b and c and a horizontal axis. The plot does not appear on screen until the `show()` command is executed.

(4) A Python script is run (or executed) by typing

`ipython script.py`

where `script.py` is the name you have given the script. Run your script and see that it produces a plot of the parabola \( f(x) = x^2 + x - 6 \)

**Functions in Python**

When a piece of code in a script is to be used more than once, it is useful to be able to refer to a predefined function. Functions are common to most, if not all, computer languages and each language has a number of predefined (or intrinsic) functions built in. For example, \( \sin(x) \), \( \cos(x) \) and \( \tan(x) \) are predefined and the script writer does not have to generate them. However, not all functions which are needed to solve a particular problem will be available as intrinsic functions and in this case it is useful for the script writer to be able to define a function.

The syntax for a function in Python is illustrated by the example below which returns a list containing a Fibonacci series up to and including input argument, \( n \)

```python
def fib(n):
    # return Fibonacci series up to n
    "Return a Fibonacci series up to n"
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

The function definition begins with `def`, the function name, `fib`, and its argument(s) in parentheses, in this case the integer, \( n \). Note that the contents of the function are initially indented by 4 spaces and the content of the `while` loop is indented by a further 4 spaces. The Fibonacci sequence is stored in the integer list called `result` and the method `result.append(b)` appends the latest value of \( b \), generated by the while loop, to the list. If anything is to be returned to the calling part of the script, then a return statement is needed, along with the object to be returned. In this case it is the list called `result`.

To use the function to generate a list containing the Fibonacci series up to and including \( n = 10 \), it is ‘called’ by inserting the code

```python
defib(10)
```
fib_series = fib(10)

fib_series is also an integer list. Note that fib must be defined in the script before it is called.

Exercise 1  Python script to find the roots of a parabolic function

(1) Choose a parabola with a minimum and two real roots (other than the one just given)

(2) Write a Python script to plot your chosen parabola and the ordinate (x) axis with an appropriate ordinate and abscissa range

(3) Modify your script so that the function is evaluated within a function definition (if you did not choose that method in (2))

(4) Add variables $x_1$, $x_2$ and $x_3$ to your script with $x_1$ and $x_3$ initialised to values for which $f(x_1) < 0$ and $f(x_3) > 0$ and $x_2 = 0.5 \times (x_1 + x_3)$.

(5) Add if statements to the script which check that the variables are correctly initialised and warn the user if they are not

(6) Add if and else statements which update $x_1$ or $x_3$ to $x_2$ according to whether $f(x_2)$ is greater or less than zero and print the updated values of $x_1$ and $x_3$

(7) Plot the new point on your parabola $(x_2,f(x_2))$ in a colour different from the colour used for the parabola curve. Place the plt.show() command at the end of the script and make sure that it remains at the end when you add further commands

(8) Add a while loop to iteratively update $x_2$ using the rule $x_2 = 0.5 \times (x_1 + x_3)$ while the absolute value of $f(x_2)$ is greater than min, initialised to min = 0.0001 before the while loop. If your script does not terminate in a few seconds it may be in an infinite loop. To terminate it yourself, type ctrl-c at the keyboard

(9) Add a statement which prints the values of $x_2$ and $f(x_2)$ after the absolute value of $f(x_2)$ becomes less than min. The Python absolute value function is math.fabs(x2). Remember to indent both the while loop and the if and else loops

(10) Run the script and check that it gives you a correct root for the parabola. Modify the script so that it will give the other root and then run it again

(11) Use the script to find how the number of steps required to find the root of the parabola depends on the value of min by adding a variable nsteps into the while loop which is incremented by one each time the loop is executed. Note that the variable nsteps must be
initialised to zero

(12) Plot a graph of \( n_{\text{steps}} \) versus the logarithm of \( \min \). The Python \( \log_{10}(x) \) function is \texttt{math.log10(x)}. You will need to \texttt{import math} at the top of the script to make the log graph. Comment out commands to plot the parabola before making this new plot.

(13) Graphs from (7) and (12) should be included in your write up.

**Exercise 2 The Newton-Raphson method for finding roots of a function**

The Newton-Raphson (NR) method is based on the Taylor series expansion of a function about a specific point.

\[
f(a + h) = f(a) + h f'(a) + h^2/2! f''(a) + ...\]

where \( f(a) \), \( f'(a) \) and \( f''(a) \) are the value of \( f \) at \( a \) and its first and second derivatives, respectively. When we apply the NR method to root finding, \( a \) is the current estimate for a root and we want to know which step size and direction will take us to an improved estimate for the root. The method can be applied to find minima of a function of a single variable or to a function of many variables.

In order to have a linear equation to solve for an improved approximation for a root, the Taylor series for \( f \) about the point \( a \) is truncated at the term which is linear in \( h \),

\[
f(a + h) = f(a) + h f'(a) + O(h^2)\]

The term \( O(h^2) \) in this equation indicates that the series was truncated and terms of order \( h^2 \) and higher were omitted.

Now set \( f(a+h) = 0 \) so that \( a + h \) is an approximation for a root. It is not the exact root since the Taylor series for \( f \) was truncated.

From the truncated Taylor series, the condition \( f(a + h) = 0 \) is equivalent to

\[
f(a) + h f'(a) = 0\]

from which we obtain

\[
h = - f(a) / f'(a)\]

Hence, given a current estimate of the root, \( a \), an improved estimate for the root is
\[ a + h = a - f(a) / f'(a) \]

(1) Write a new Python script which plots the parabola you chose for the previous exercise and its derivative using function definitions for the functions. Note that you do not need the constant \( c \) to evaluate the derivative of the parabola and so the arguments for the derivative function definition should be \((x, a, b)\) only.

(2) Add a variable \( x_1 \) and initialise it to 1.

(3) Add a statement which updates the value of \( x_1 \) using the Newton-Raphson rule:

\[ x_1 = x_1 - f(x_1) / f'(x_1) \]

(4) Plot the new point on your parabola \((x_1, f(x_1))\) in a different colour from the colour used to draw the parabola. Comment out the previous `plt.show()` command from (1) and add a new `plt.show()` command at the end of the script in order to see the complete graph and the NR generated point.

(5) Add a `while` loop to update \( x_1 \) iteratively using the Newton-Raphson rule while the absolute value of \( f(x_1) \) is greater than \( min \), where \( min = 0.0001 \). Remember that you need to indent the code within the `while` loop.

(6) Print the values of \( x_1 \) and \( f(x_1) \) after the absolute value of \( f(x_1) \) becomes less than \( min \).

(7) Run the script and check that it gives you a correct root for the parabola.

(8) Modify the script so that it will give the other root and then run it again to obtain that root. If you cannot think how to modify the script to do this, think about how the NR algorithm works and therefore what you need to change in order to obtain the other root.

(9) Use the script to find how the number of steps required to find the root of the parabola depends on the value of \( min \) by adding a variable \( nsteps \) into the while loop which is incremented each time the loop is executed.

(10) Plot a graph of \( nsteps \) versus the logarithm of \( min \). The Python \( \log_{10}(x) \) function is `math.log10(x)`.

(11) Compare the efficiencies of the bisection and Newton-Raphson methods in your write up by comparing graphs of the number of iterations taken to reach a root for a given value of \( min \).

(12) Experiment with the value of \( min \) to find the greatest accuracy that is achievable with this method.
Exercise 3  Python script to find roots of a potential energy function

The interaction potential between two ions such as Na⁺ and Cl⁻ is given by

\[ V(x) = A e^{-x/p} - \frac{e^2}{4\pi\varepsilon_0 x} \]

The two terms represent short range, Pauli repulsion of electron clouds and long range electrostatic attraction of oppositely charged ions. The numerical values of \( e^2/4\pi\varepsilon_0 \), \( A \) and \( p \) are 1.44 eV nm, 1090 eV and 0.033 nm, respectively. Use these values in your script, to be described below.

Previously you used the Newton-Raphson method to find roots of functions, \( f(x) \), by solving \( f(x) = 0 \). It can also be used to find minima or maxima of functions by solving \( df/dx = 0 \) for \( x \), since the derivative of the function vanishes at a maximum or minimum.

To apply the NR method to find the root of \( dV/dx = 0 \) we need the rule

\[ x_1 = x_0 - \frac{V'(x_0)}{V''(x_0)} \]

where \( V' \) and \( V'' \) are the first and second derivatives of \( V(x) \).

(1) Write a Python script to plot the function \( V(x) \). First, define the function \( V(x) \) in your script.

(2) The Python function \( \text{math.exp}(x) \) takes single, scalar values as arguments, not a range of values as generated by \( \text{arange} \) in \( \text{numpy} \). Therefore \( \text{arange} \) cannot be used to generate a list of ordinate values for input to \( \text{math.exp}(x) \) as we did previously when plotting a parabola. However, the exponential function, \( \text{numpy.exp}(x) \), in \( \text{numpy} \) does take a list of values as input.

\[ x = \text{numpy.arange}(0.01, 1.00, 0.01) \]
\[ \text{listx} = \text{numpy.exp}(x) \]

will generate a list of values of \( \exp(x) \) in the range \( 0.01 < x < 1.00 \) in \( \text{listx} \), which can be plotted using the Matplotlib \( \text{plot} \) command.

Now add code to your script to plot a graph of \( V(x) \) in the range 0.01 to 1.00.

(3) Differentiate \( V(x) \) with respect to \( x \) (on paper) and plot a graph of \(-dV/dx\). This is the force acting on the particle. Add a statement to your script so that it also plots \(-dV/dx\). Observe that \(-dV/dx\) is zero at the minimum of \( V(x) \). Explain why this is so in your write up.

(4) Differentiate \( dV/dx \) to obtain \( V''(x) \) and state what you obtain for \( V'(x) \) and \( V''(x) \) in your
write up

(5) Add code to your script so that it finds the value of \( x \) for which \( V(x) \) is a minimum using a NR method. Note that the minimum is close to 0.2 so that \( x = 0.2 \) is a good value to initialise \( x \) to. State the value of \( x \) that you obtain for the minimum of \( V(x) \) in your write up

**Supplementary Exercise  Equilibrium positions of charges on a ring**

This exercise is not compulsory and no extra marks will be awarded if you complete it. It is included in this lab class to allow you to test yourself on a harder problem and to help build your coding skills. The case of three inequivalent charges on a ring is an example of a problem which cannot be solved analytically but can be solved easily using numerical methods.

Three charges of equal sign but varying magnitude are confined to move on a ring of radius, \( R \). The equilibrium arrangement for the charges is one in which the potential energy function is a minimum. Since the potential energy does not change if all charges are moved on the ring through the same angle, we need to find the angles subtended by lines from the centre of the circle to the charges.

It is convenient to think of one charge as fixed on the \( x \) axis, and identify the positions of the other two by their polar coordinates \((R, \phi_1)\) and \((R, \phi_2)\). The potential energy associated with each pair is inversely proportional to the separation of the two charges. The potential energy of the 1st and 2nd charges is

\[
V(r_{12}) = \frac{q_1 q_2}{4 \pi \varepsilon_0 r_{12}} = \frac{q_1 q_2}{4 \pi \varepsilon_0} \left|\frac{2R \sin(\phi_1/2)}{2R \sin(\phi_2/2)}\right|
\]

where \( r_{12} = 2R \left|\sin(\phi_1/2)\right| \) is the distance between the two electrons, which is a positive number. Note that the separation of the 2nd and 3rd electrons is \( \left|2R \sin(\phi_2 - \phi_1)\right|/2\).

(1) Show that \( r_{12} = 2R \left|\sin(\phi_1/2)\right| \) is the distance between two charges on the ring if \( \phi_1 \) is the angle between charges. You will need the formula which relates the three sides of a triangle and one angle in the triangle as well as the half-angle formula for the square root of \( 1 - \cos(2\phi_1) \).

(2) Find expressions for the force and derivative of the force between pairs of charges using the potential energy function given above.

(3) Write a script to plot the potential energy and the force acting on one charge as a function of \( \phi_1 \) for two equal charges confined to a ring. The plot range should avoid the singularity in both functions at \( \phi_1 = 0 \). The Python commands for absolute value, \( \sin(x) \), \( \cos(x) \) and \( \tan(x) \) are `math.fabs(x)`, `math.sin(x)`, `math.cos(x)` and `math.tan(x)`, respectively. The value of \( \pi \) is
math.pi. Explain the shapes of the two functions in your write up

(4) Modify your script so that it can calculate the total energy of two charges of magnitude $q_1$ and $q_2$ on a ring and print the energy. $q_1$ and $q_2$ should both be initialised to 1 for this part

(5) Modify your script so that it calculates the forces acting on both charges and prints them. Note that since both charges are of the same sign, the forces should be repulsive

(6) Use the Newton-Raphson method to find the equilibrium configuration for three inequivalent charges. You will need nested while loops which find equilibrium configurations for all three pairs of charges