Chapter 4
Non-linear Equations

Abstract In physics we often encounter the problem of determining the root of a function $f(x)$. Especially, we may need to solve non-linear equations of one variable. Such equations are usually divided into two classes, algebraic equations involving roots of polynomials and transcendental equations. When there is only one independent variable, the problem is one-dimensional, namely to find the root or roots of a function. Except in linear problems, root finding invariably proceeds by iteration, and this is equally true in one or in many dimensions. This means that we cannot solve exactly the equations at hand. Rather, we start with some approximate trial solution. The chosen algorithm will in turn improve the solution until some predetermined convergence criterion is satisfied. The algoritms we discuss below attempt to implement this strategy. We will deal mainly with one-dimensional problems.

In chapter 6 we will discuss methods to find for example zeros and roots of equations. In particular, we will discuss the conjugate gradient method.

4.1 Particle in a Box Potential

You may have encountered examples of so-called transcendental equations when solving the Schrödinger equation (SE) for a particle in a box potential. The one-dimensional SE for a particle with mass m is

$$-\frac{\hbar^2}{2m}\frac{d^2u}{dx^2}+V(x)u(x)=Eu(x), \tag{4.1}$$

and our potential is defined as

$$V(r)=\begin{cases}-V_0 & 0\le x<a\\ 0 & x>a\end{cases} \tag{4.2}$$

Bound states correspond to negative energy E and scattering states are given by positive energies. The SE takes the form (without specifying the sign of E)

$$\frac{d^2u(x)}{dx^2}+\frac{2m}{\hbar^2}\left(V_0+E\right)u(x)=0 \qquad x<a, \tag{4.3}$$

and

$$\frac{d^2u(x)}{dx^2}+\frac{2m}{\hbar^2}Eu(x)=0 \qquad x>a. \tag{4.4}$$

If we specialize to bound states $E<0$ and implement the boundary conditions on the wave function we obtain

$$u(r)=Asin(\sqrt{2m(V_0-|E|)}r/\hbar) \qquad r<a, \tag{4.5}$$

and

$$u(r) = B\exp\left(-\sqrt{2m|E|}r/\hbar\right) \qquad r > a, \tag{4.6}$$

where A and B are constants. Using the continuity requirement on the wave function at $r = a$ one obtains the transcendental equation

$$\sqrt{2m(V_0 - |E|)}\cot\left(\sqrt{2ma^2(V_0 - |E|)}/\hbar\right) = -\sqrt{2m|E|}. \tag{4.7}$$

This equation is an example of the kind of equations which could be solved by some of the methods discussed below. The algorithms we discuss are the bisection method, the secant and Newton-Raphson's method.

In order to find the solution for Eq. (4.7), a simple procedure is to define a function

$$f(E) = \sqrt{2m(V_0 - |E|)}\cot\left(\sqrt{2ma^2(V_0 - |E|)}/\hbar\right) + \sqrt{2m|E|}. \tag{4.8}$$

and with chosen or given values for a and V_0 make a plot of this function and find the approximate region along the $E - axis$ where $f(E) = 0$. We show this in Fig. 4.1 for $V_0 = 20$ MeV, $a = 2$ fm and $m = 938$ MeV. Fig. 4.1 tells us that the solution is close to $|E| \approx 2.2$ (the binding

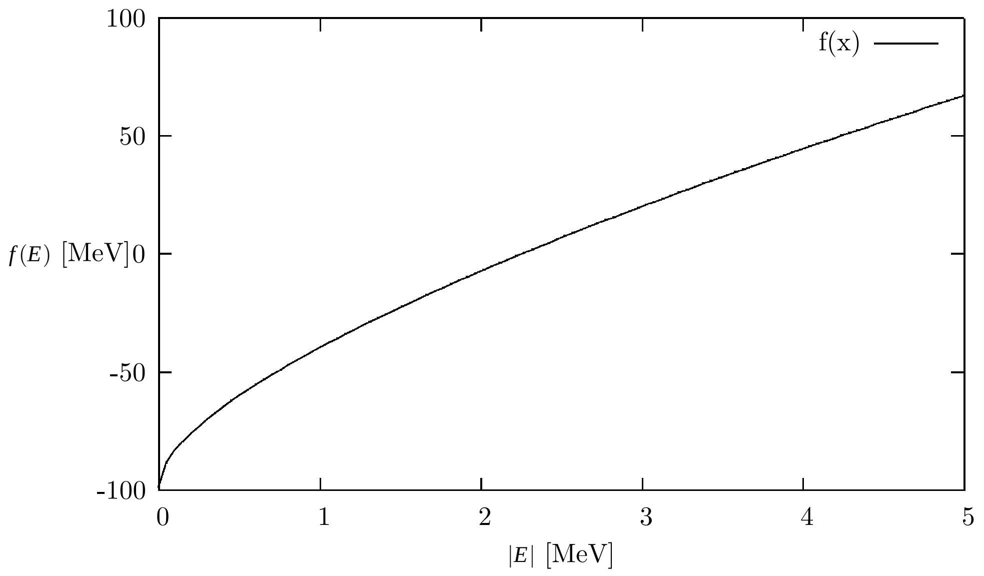

Fig. 4.1 Plot of $f(E)$ in Eq. (4.8) as function of energy |E| in MeV. Te function $f(E)$ is in units of megaelectronvolts MeV. Note well that the energy E is for bound states.

energy of the deuteron). The methods we discuss below are then meant to give us a numerical solution for E where $f(E) = 0$ is satisfied and with E determined by a given numerical precision.

4.2 Iterative Methods

To solve an equation of the type $f(x) = 0$ means mathematically to find all numbers s[1] so that $f(s) = 0$. In all actual calculations we are always limited by a given precision when doing

[1] In the following discussion, the variable s is reserved for the value of x where we have a solution.

numerics. Through an iterative search of the solution, the hope is that we can approach, within a given tolerance ε, a value x_0 which is a solution to $f(s) = 0$ if

$$|x_0 - s| < \varepsilon, \tag{4.9}$$

and $f(s) = 0$. We could use other criteria as well like

$$\left|\frac{x_0 - s}{s}\right| < \varepsilon, \tag{4.10}$$

and $|f(x_0)| < \varepsilon$ or a combination of these. However, it is not given that the iterative process will converge and we would like to have some conditions on f which ensures a solution. This condition is provided by the so-called Lipschitz criterion. If the function f, defined on the interval $[a,b]$ satisfies for all x_1 and x_2 in the chosen interval the following condition

$$|f(x_1) - f(x_2)| \leq k\,|x_1 - x_2|, \tag{4.11}$$

with k a constant, then f is continuous in the interval $[a,b]$. If f is continuous in the interval $[a,b]$, then the secant condition gives

$$f(x_1) - f(x_2) = f'(\xi)(x_1 - x_2), \tag{4.12}$$

with x_1, x_2 within $[a,b]$ and ξ within $[x_1, x_2]$. We have then

$$|f(x_1) - f(x_2)| \leq |f'(\xi)|\,|x_1 - x_2|. \tag{4.13}$$

The derivative can be used as the constant k. We can now formulate the sufficient conditions for the convergence of the iterative search for solutions to $f(s) = 0$.

1. We assume that f is defined in the interval $[a,b]$.
2. f satisfies the Lipschitz condition with $k < 1$.

With these conditions, the equation $f(x) = 0$ has only one solution in the interval $[a,b]$ and it converges after n iterations towards the solution s irrespective of choice for x_0 in the interval $[a,b]$. If we let x_n be the value of x after n iterations, we have the condition

$$|s - x_n| \leq \frac{k}{1-k}\,|x_1 - x_2|. \tag{4.14}$$

The proof can be found in the text of Bulirsch and Stoer. Since it is difficult numerically to find exactly the point where $f(s) = 0$, in the actual numerical solution one implements three tests of the type

1. $$|x_n - s| < \varepsilon, \tag{4.15}$$

   and
2. $$|f(s)| < \delta, \tag{4.16}$$
3. and a maximum number of iterations N_{maxiter} in actual calculations.

4.3 Bisection

This is an extremely simple method to code. The philosophy can best be explained by choosing a region in e.g., Fig. 4.1 which is close to where $f(E) = 0$. In our case $|E| \approx 2.2$. Choose a region $[a,b]$ so that $a = 1.5$ and $b = 3$. This should encompass the point where $f = 0$. Define then the point

$$c = \frac{a+b}{2}, \tag{4.17}$$

and calculate $f(c)$. If $f(a)f(c) < 0$, the solution lies in the region $[a,c] = [a,(a+b)/2]$. Change then $b \leftarrow c$ and calculate a new value for c. If $f(a)f(c) > 0$, the new interval is in $[c,b] = [(a+b)/2,b]$. Now you need to change $a \leftarrow c$ and evaluate then a new value for c. We can continue to halve the interval till we have reached a value for c which fulfills $f(c) = 0$ to a given numerical precision. The algorithm can be simply expressed in the following program

```
      ......
      fa = f(a);
      fb = f(b);
//   check if your interval is correct, if not return to main
      if ( fa*fb > 0) {
        cout << ``\n Error, root not in interval'' << endl;
        return;
      }
      for (j=1; j <= iter_max; j++) {
        c=(a+b)/2;
        fc=f(c)
//  if this test is satisfied, we have the root c
        if ( (abs(a-b) < epsilon ) || fc < delta ); return to main
        if ( fa*fc < 0){
          b=c ; fb=fc;
        }
        else{
          a=c ; fa=fc;
        }
      }
      ......
```

Note that one needs to define the values of δ, ε and `iter_max` when calling this function.

The bisection method is an almost foolproof method, although it may converge slowly towards the solution due to the fact that it halves the intervals. After n divisions by 2 we have a possible solution in the interval with length

$$\frac{1}{2^n}|b-a|, \tag{4.18}$$

and if we set $x_0 = (a+b)/2$ and let x_n be the midpoints in the intervals we obtain after n iterations that Eq. (4.14) results in

$$|s-x_n| \leq \frac{1}{2^{n+1}}|b-a|, \tag{4.19}$$

since the nth interval has length $|b-a|/2^n$. Note that this convergence criterion is independent of the actual function $f(x)$ as long as this function fulfils the conditions discussed in the conditions discussed in the previous subsection.

As an example, suppose we wish to find how many iteration steps are needed in order to obtain a relative precision of 10^{-12} for x_n in the interval $[50,63]$, that is

$$\frac{|s-x_n|}{|s|} \leq 10^{-12}. \tag{4.20}$$

It suffices in our case to study $s \geq 50$, which results in

$$\frac{|s-x_n|}{50} \leq 10^{-12}, \tag{4.21}$$

and with Eq. (4.19) we obtain

$$\frac{13}{2^{n+1}50} \leq 10^{-12}, \tag{4.22}$$

meaning $n \geq 37$. The code for the bisection method can look like this

```
    /*
    ** This function
    ** calculates a root between x1 and x2 of a function
    ** pointed to by (*func) using the method of bisection
    ** The root is returned with an accuracy of +- xacc.
    */

double bisection(double (*func)(double), double x1, double x2, double xacc)
{
  int     j;
  double  dx, f, fmid, xmid, rtb;

  f   = (*func)(x1);
  fmid = (*func)(x2);
  if(f*fmid >= 0.0) {
    cout << "\n\nError in function bisection():" << endl;
    cout << "\nroot in function must be within" << endl;
    cout << "x1 ='' << x1 << ``and x2 `` << x2 << endl;
    exit(1);
  }
  rtb = f < 0.0 ? (dx = x2 - x1, x1) : (dx = x1 - x2, x2);
  for(j = 0; j < max_iterations; j++) {
    fmid = (*func)(xmid = rtb + (dx *= 0.5));
    if (fmid <= 0.0) rtb=xmid;
    if(fabs(dx) < xacc || fmid == 0.0) return rtb;
  }
  cout << "Error in the bisection:" << endl; // should never reach this point
  cout "Too many iterations!" << endl;
}
// End: function bisection
```

In this function we transfer the lower and upper limit of the interval where we seek the solution, $[x_1, x_2]$. The variable `xacc` is the precision we opt for. Note that in this function the test $f(s) < \delta$ is not implemented. Rather, the test is done through $f(s) = 0$, which is not necessarily a good option.

Note also that this function transfer a pointer to the name of the given function through `double(*func)(double)`.

4.4 Newton-Raphson's Method

Perhaps the most celebrated of all one-dimensional root-finding routines is Newton's method, also called the Newton-Raphson method. This method is distinguished from the previously discussed methods by the fact that it requires the evaluation of both the function f and its derivative f' at arbitrary points. In this sense, it is taylored to cases with e.g., transcendental equations of the type shown in Eq. (4.8) where it is rather easy to evaluate the derivative. If you can only calculate the derivative numerically and/or your function is not of the smooth type, we discourage the use of this method.

The Newton-Raphson formula consists geometrically of extending the tangent line at a current point until it crosses zero, then setting the next guess to the abscissa of that zero-crossing. The mathematics behind this method is rather simple. Employing a Taylor expansion for x sufficiently close to the solution s, we have

$$f(s) = 0 = f(x) + (s-x)f'(x) + \frac{(s-x)^2}{2}f''(x) + \dots. \tag{4.23}$$

For small enough values of the function and for well-behaved functions, the terms beyond linear are unimportant, hence we obtain

$$f(x) + (s-x)f'(x) \approx 0, \tag{4.24}$$

yielding

$$s \approx x - \frac{f(x)}{f'(x)}. \tag{4.25}$$

Having in mind an iterative procedure, it is natural to start iterating with

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{4.26}$$

This is Newton-Raphson's method. It has a simple geometric interpretation, namely x_{n+1} is the point where the tangent from $(x_n, f(x_n))$ crosses the $x-$axis. Close to the solution, Newton-Raphson converges fast to the desired result. However, if we are far from a root, where the higher-order terms in the series are important, the Newton-Raphson formula can give grossly inaccurate results. For instance, the initial guess for the root might be so far from the true root as to let the search interval include a local maximum or minimum of the function. If an iteration places a trial guess near such a local extremum, so that the first derivative nearly vanishes, then Newton-Raphson may fail totally. An example is shown in Fig. 4.2

It is also possible to extract the convergence behavior of this method. Assume that the function f has a continuous second derivative around the solution s. If we define

$$e_{n+1} = x_{n+1} - s = x_n - \frac{f(x_n)}{f'(x_n)} - s, \tag{4.27}$$

and using Eq. (4.23) we have

$$e_{n+1} = e_n + \frac{-e_n f'(x_n) + e_n^2/2 f''(\xi)}{f'(x_n)} = \frac{e_n^2/2 f''(\xi)}{f'(x_n)}. \tag{4.28}$$

This gives

$$\frac{|e_{n+1}|}{|e_n|^2} = \frac{1}{2}\frac{|f''(\xi)|}{|f'(x_n)|^2} = \frac{1}{2}\frac{|f''(s)|}{|f'(s)|^2} \tag{4.29}$$

when $x_n \to s$. Our error constant k is then proportional to $|f''(s)|/|f'(s)|^2$ if the second derivative is different from zero. Clearly, if the first derivative is small, the convergence is slower. In general, if we are able to start the iterative procedure near a root and we can easily evaluate the derivative, this is the method of choice. In cases where we may need to evaluate the derivative numerically, the previously described methods are easier and most likely safer to implement with respect to loss of numerical precision. Recall that the numerical evaluation of derivatives involves differences between function values at different x_n.

We can rewrite the last equation as

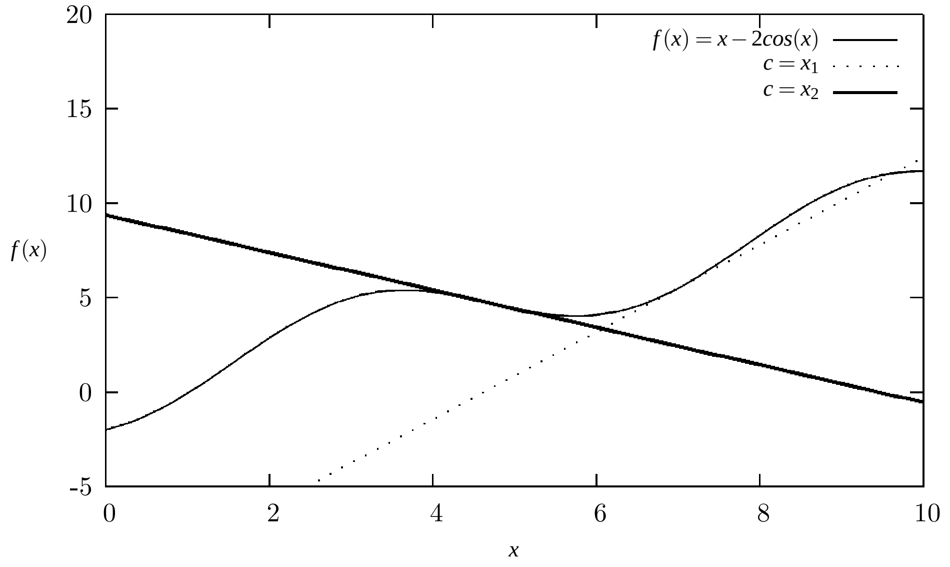

Fig. 4.2 Example of a case where Newton-Raphson's method does not converge. For the function $f(x) = x - 2cos(x)$, we see that if we start at $x = 7$, the first iteration gives us that the first point where we cross the $x-$axis is given by x_1. However, using x_1 as a starting point for the next iteration results in a point x_2 which is close to a local minimum. The tangent here is close to zero and we will never approach the point where $f(x) = 0$.

$$|e_{n+1}| = C|e_n|^2, \tag{4.30}$$

with C a constant. If we assume that $C \sim 1$ and let $e_n \sim 10^{-8}$, this results in $e_{n+1} \sim 10^{-16}$, and demonstrates clearly why Newton-Raphson's method may converge faster than the bisection method.

Summarizing, this method has a solution when f'' is continuous and s is a simple zero of f. Then there is a neighborhood of s and a constant C such that if Newton-Raphson's method is started in that neighborhood, the successive points become steadily closer to s and satisfy

$$|s - x_{n+1}| \le C|s - x_n|^2,$$

with $n \ge 0$. In some situations, the method guarantees to converge to a desired solution from an arbitrary starting point. In order for this to take place, the function f has to belong to $C^2(R)$, be increasing, convex and having a zero. Then this zero is unique and Newton's method converges to it from any starting point.

As a mere curiosity, suppose we wish to compute the square root of a number R, i.e., $\sqrt{R}$. Let $R > 0$ and define a function

$$f(x) = x^2 - R.$$

The variable x is a root if $f(x) = 0$. Newton-Raphson's method yields then the following iterative approach to the root

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{R}{x_n}\right), \tag{4.31}$$

a formula credited to Heron, a Greek engineer and architect who lived sometime between 100 B.C. and A.D. 100.

Suppose we wish to compute $\sqrt{13} = 3.6055513$ and start with $x_0 = 5$. The first iteration gives $x_1 = 3.8$, $x_2 = 3.6105263$, $x_3 = 3.6055547$ and $x_4 = 3.6055513$. With just four iterations and a not too optimal choice of x_0 we obtain the exact root to a precision of 8 digits. The above equation, together with range reduction , is used in the intrisic computational function which computes square roots.

Newton's method can be generalized to systems of several non-linear equations and variables. Consider the case with two equations

$$\begin{array}{c} f_1(x_1,x_2) = 0 \\ f_2(x_1,x_2) = 0 \end{array}, \tag{4.32}$$

which we Taylor expand to obtain

$$\begin{array}{ll} 0 = f_1(x_1+h_1,x_2+h_2) = & f_1(x_1,x_2)+h_1\partial f_1/\partial x_1+h_2\partial f_1/\partial x_2+\dots \\ 0 = f_2(x_1+h_1,x_2+h_2) = & f_2(x_1,x_2)+h_1\partial f_2/\partial x_1+h_2\partial f_2/\partial x_2+\dots \end{array}. \tag{4.33}$$

Defining the Jacobian matrix $\hat{\mathbf{J}}$ we have

$$\hat{\mathbf{J}} = \begin{pmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \end{pmatrix}, \tag{4.34}$$

we can rephrase Newton's method as

$$\begin{pmatrix} x_1^{n+1} \\ x_2^{n+1} \end{pmatrix} = \begin{pmatrix} x_1^n \\ x_2^n \end{pmatrix} + \begin{pmatrix} h_1^n \\ h_2^n \end{pmatrix}, \tag{4.35}$$

where we have defined

$$\begin{pmatrix} h_1^n \\ h_2^n \end{pmatrix} = -\hat{\mathbf{J}}^{-1} \begin{pmatrix} f_1(x_1^n,x_2^n) \\ f_2(x_1^n,x_2^n) \end{pmatrix}. \tag{4.36}$$

We need thus to compute the inverse of the Jacobian matrix and it is to understand that difficulties may arise in case $\hat{\mathbf{J}}$ is nearly singular.

It is rather straightforward to extend the above scheme to systems of more than two non-linear equations.

The code for Newton-Raphson's method can look like this

```
    /*
    ** This function
    ** calculates a root between x1 and x2 of a function pointed to
    ** by (*funcd) using the Newton-Raphson method. The user-defined
    ** function funcd() returns both the function value and its first
    ** derivative at the point x,
    ** The root is returned with an accuracy of +- xacc.
    */

double newtonraphson(void (*funcd)(double, double *, double *), double x1, double x2,
 double xacc)
{
  int   j;
  double df, dx, f, rtn;

  rtn = 0.5 * (x1 + x2);      // initial guess
  for(j = 0; j < max_iterations; j++) {
    (*funcd)(rtn, &f, &df);
    dx  = f/df;
    rtn -= dx;
    if((x1 - rtn) * (rtn - x2) < 0.0) {
      cout << "\n\nError in function newtonraphson:" << endl ;
      cout << "Jump out of interval bracket" << endl;
    }
    if (fabs(dx) < xacc) return rtn;
  }
  cout << "Error in function newtonraphson:" << endl;
  cout << "Too many iterations!" << endl;
}
// End: function newtonraphson
```

We transfer again the lower and upper limit of the interval where we seek the solution, $[x_1, x_2]$ and the variable `xacc`. Firthermore, it transfers a pointer to the name of the given function through `double(*func)(double)`.

4.5 The Secant Method

For functions that are smooth near a root, the methods known respectively as false position (or regula falsi) and secant method generally converge faster than bisection but slower than Newton-Raphson. In both of these methods the function is assumed to be approximately linear in the local region of interest, and the next improvement in the root is taken as the point where the approximating line crosses the axis.

The algorithm for obtaining the solution for the secant method is rather simple. We start with the definition of the derivative

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

and combine it with the iterative expression of Newton-Raphson's

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

to obtain

$$x_{n+1} = x_n - f(x_n)\left(\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}\right), \tag{4.37}$$

which we rewrite to

$$x_{n+1} = \frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})}. \tag{4.38}$$

This is the secant formula, implying that we are drawing a straight line from the point $(x_{n-1}, f(x_{n-1}))$ to $(x_n, f(x_n))$. Where it crosses the $x-axis$ we have the new point x_{n+1}. This is illustrated in Fig. 4.3.

In the numerical implementation found in the program library, the quantities x_{n-1}, x_n, x_{n+1} are changed to a, b and c respectively, i.e., we determine c by the point where a straight line from the point $(a, f(a))$ to $(b, f(b))$ crosses the $x-axis$, that is

$$c = \frac{f(b)a - f(a)b}{f(b) - f(a)}. \tag{4.39}$$

We then see clearly the difference between the bisection method and the secant method. The convergence criterion for the secant method is

$$|e_{n+1}| \approx A|e_n|^{\alpha}, \tag{4.40}$$

with $\alpha \approx 1.62$. The convergence is better than linear, but not as good as Newton-Raphson's method which converges quadratically.

While the secant method formally converges faster than bisection, one finds in practice pathological functions for which bisection converges more rapidly. These can be choppy, discontinuous functions, or even smooth functions if the second derivative changes sharply near

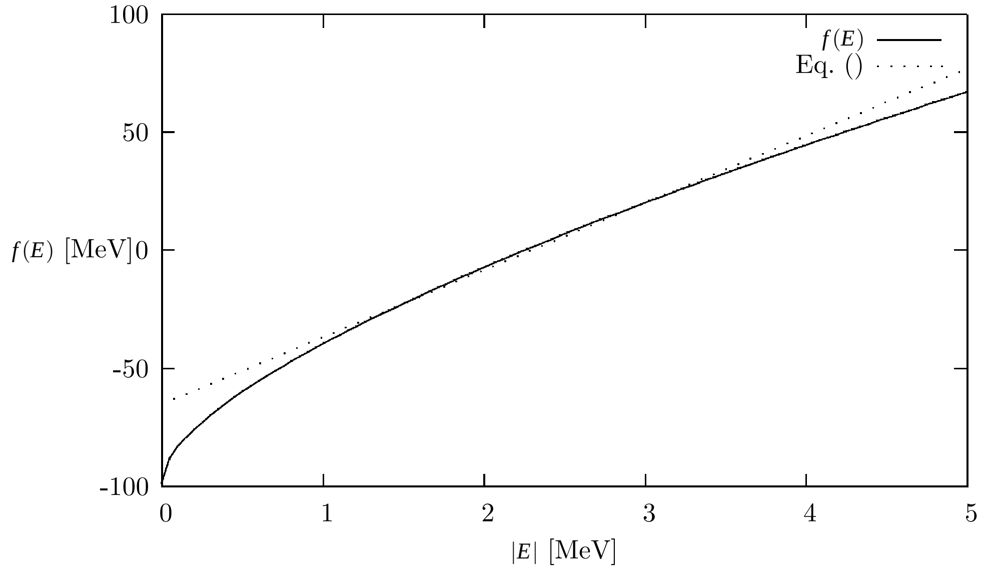

Fig. 4.3 Plot of $f(E)$ Eq. (4.8) as function of energy |E|. The point c is determined by where the straight line from $(a, f(a))$ to $(b, f(b))$ crosses the $x-axis$.

the root. Bisection always halves the interval, while the secant method can sometimes spend many cycles slowly pulling distant bounds closer to a root. We illustrate the weakness of this method in Fig. 4.4 where we show the results of the first three iterations, i.e., the first point is $c = x_1$, the next iteration gives $c = x_2$ while the third iterations ends with $c = x_3$. We may risk that one of the endpoints is kept fixed while the other one only slowly converges to the desired solution.

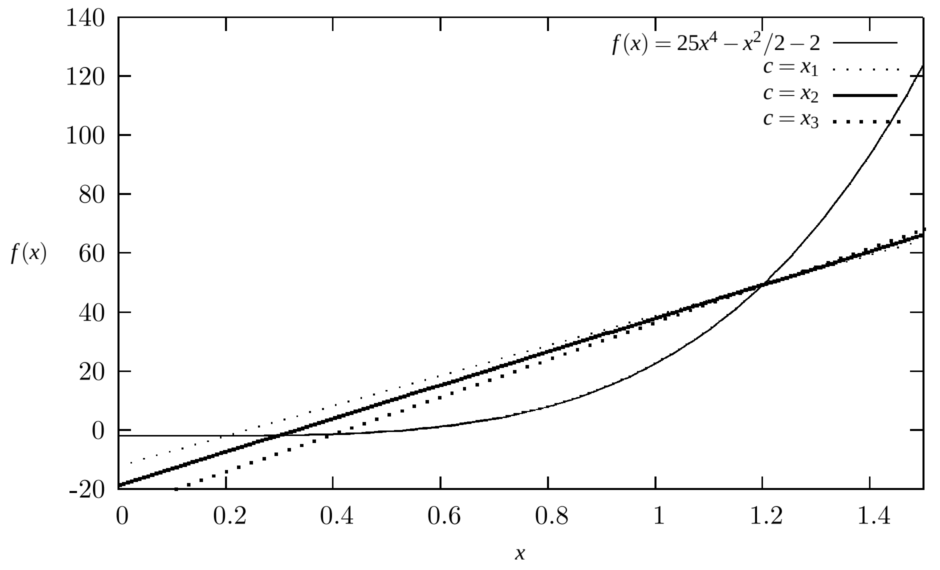

Fig. 4.4 Plot of $f(x) = 25x^4 - x^2/2 - 2$. The various straight lines correspond to the determination of the point c after each iteration. c is determined by where the straight line from $(a, f(a))$ to $(b, f(b))$ crosses the $x-axis$. Here we have chosen three values for c, x_1, x_2 and x_3 which refer to the first, second and third iterations respectively.

The search for the solution s proceeds in much of the same fashion as for the bisection method, namely after each iteration one of the previous boundary points is discarded in favor of the latest estimate of the root. A variation of the secant method is the so-called false position method (regula falsi from Latin) where the interval [a,b] is chosen so that $f(a)f(b) < 0$, else there is no solution. This is rather similar to the bisection method. Another possibility is to determine the starting point for the iterative search using three points $(a, f(a))$, $(b, f(b))$ and $(c, f(c))$. One can thenuse Lagrange's interpolation formula for a polynomial, see the discussion in the previous chapter.

4.5.1 Broyden's Method

Broyden's method is a quasi-Newton method for the numerical solution of nonlinear equations in k variables.

Newton's method for solving the equation $f(x) = 0$ uses the Jacobian matrix and determinant J, at every iteration. However, computing the Jacobian is a difficult and expensive operation. The idea behind Broyden's method is to compute the whole Jacobian only at the first iteration, and to do a so-called rank-one update at the other iterations.

The method is a generalization of the secant method to multiple dimensions. The secant method replaces the first derivative $f'(x_n)$ with the finite difference approximation

$$f'(x_n) \simeq \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}},$$

and proceeds using Newton's method

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

Broyden gives a generalization of this formula to a system of equations $F(x) = 0$, replacing the derivative f' with the Jacobian J. The Jacobian is determined using the secant equation (using the finite difference approximation):

$$J_n \cdot (x_n - x_{n-1}) \simeq F(x_n) - F(x_{n-1}).$$

However this equation is underdetermined in more than one dimension. Broyden suggested using the current estimate of the Jacobian J_{n-1} and improving upon it by taking the solution to the secant equation that is a minimal modification to J_{n-1} (minimal in the sense of minimizing the Frobenius norm $\|J_n - J_{n-1}\|_F$))

$$J_n = J_{n-1} + \frac{\Delta F_n - J_{n-1}\Delta x_n}{\|\Delta x_n\|^2} \Delta x_n^T,$$

and then apply Newton's method

$$x_{n+1} = x_n - J_n^{-1} F(x_n).$$

In the formula above $x_n = (x_1[n], \ldots, x_k[n])$ and $F_n(x) = (f_1(x_1[n], \ldots, x_k[n]), \ldots, f_k(x_1[n], \ldots, x_k[n]))$ are vector-columns with k elements for a system with k dimensions. We obtain then

$$\Delta x_n = \begin{bmatrix} x_1[n] - x_1[n-1] \\ \ldots \\ x_k[n] - x_k[n-1] \end{bmatrix} \quad \text{and} \quad \Delta F_n = \begin{bmatrix} f_1(x_1[n], \ldots, x_k[n]) - f_1(x_1[n-1], \ldots, x_k[n-1]) \\ \ldots \\ f_k(x_1[n], \ldots, x_k[n]) - f_k(x_1[n-1], \ldots, x_k[n-1]) \end{bmatrix}.$$

Broyden also suggested using the Sherman-Morrison formula to update directly the inverse of the Jacobian

$$J_n^{-1} = J_{n-1}^{-1} + \frac{\Delta x_n - J_{n-1}^{-1}\Delta F_n}{\Delta x_n^T J_{n-1}^{-1}\Delta F_n}(\Delta x_n^T J_{n-1}^{-1})$$

This method is commonly known as the "good Broyden's method". Many other quasi-Newton schemes have been suggested in optimization, where one seeks a maximum or minimum by finding the root of the first derivative (gradient in multi dimensions). The Jacobian of the gradient is called Hessian and is symmetric, adding further constraints to its upgrade.

4.6 Exercises

4.1. Write a code which implements the bisection method, Newton-Raphson's method and the secant method.

Find the positive roots of

$$x^2 - 4x\sin x + (2\sin x)^2 = 0,$$

using these three methods and compare the achieved accuracy number of iterations needed to find the solution. Give a critical discussion of the methods.

4.2. Make thereafter a class which includes the above three methods and test this class against selected problems.

4.3. We are going to study the solution of the Schrödinger equation (SE) for a system with a neutron and proton (the deuteron) moving in a simple box potential.

We begin our discussion of the SE with the neutron-proton (deuteron) system with a box potential $V(r)$. We define the radial part of the wave function $R(r)$ and introduce the definition $u(r) = rR(R)$ The radial part of the SE for two particles in their center-of-mass system and with orbital momentum $l = 0$ is then

$$-\frac{\hbar^2}{m}\frac{d^2u(r)}{dr^2} + V(r)u(r) = Eu(r),$$

with

$$m = 2\frac{m_p m_n}{m_p + m_n},$$

where m_p and m_n are the masses of the proton and neutron, respectively. We use here $m = 938$ MeV. Our potential is defined as

$$V(r) = \begin{cases} -V_0 & 0 \le r < a \\ 0 & r > a \end{cases}$$

Bound states correspond to negative energy E and scattering states are given by positive energies. The SE takes the form (without specifying the sign of E)

$$\frac{d^2u(r)}{dr^2} + \frac{m}{\hbar^2}(V_0 + E)u(r) = 0 \qquad r < a,$$

and

$$\frac{d^2u(r)}{dr^2} + \frac{m}{\hbar^2}Eu(r) = 0 \qquad r > a.$$

We are now going to search for eventual bound states, i.e., $E < 0$. The deuteron has only one bound state at energy $E = -2.223$ MeV. Discuss the boundary conditions on the wave function and use these to show that the solution to the SE is

$$u(r) = Asin(kr) \qquad r < a,$$

and

$$u(r) = B\exp(-\beta r) \qquad r > a,$$

where A and B are constants. We have also defined

$$k = \sqrt{m(V_0 - |E|)}/\hbar,$$

and

$$\beta = \sqrt{m|E|}/\hbar.$$

Show then, using the continuity requirement on the wave function that at $r = a$ you obtain the transcendental equation

$$kcot(ka) = -\beta. \tag{4.41}$$

Insert values of $V_0 = 60$ MeV and $a = 1.45$ fm (1 fm = 10^{-15} m) and make a plot plotting programs) of Eq. (4.41) as function of energy E in order to find eventual eigenvalues. See if these values result in a bound state for E.

When you have localized on your plot the point(s) where Eq. (4.41) is satisfied, obtain a numerical value for E using the class you programmed in the previous exercise, including the Newton-Raphson's method, the bisection method and the secant method. Make an analysis of these three methods and discuss how many iterations are needed to find a stable solution.

What is smallest possible value of V_0 which gives a bound state?