Chapter 1

Boundary value problems

Numerical linear algebra techniques can be used for many physical problems. In this chapter we will give some examples of how these techniques can be used to solve certain boundary value problems that occur in physics.

1.1 A 1-D generalized diffusion equation

Consider a 1-D stationary state diffusion-type equation, which we will call the "generalized diffusion equation" from now on:

$$-\frac{d}{dx}\left(D(x)\frac{dy(x)}{dx}\right) + K(x)\, y(x) = q(x) \tag{1.1}$$

where $D(x) > 0$ is the diffusion coefficient, $K(x) > 0$ is some given function (its meaning will be elucidated later) and $q(x) > 0$ is a source function.

To solve this problem, we have to specify a finite domain $x_{\text{left}} \leq x \leq x_{\text{right}}$ and give the boundary conditions. For problems of this kind (diffusion-like) there are two "classical" types of boundary conditions that are usually imposed:

- *Dirichlet boundary condition:* Here you simply specify the value of the function $y(x)$ at the boundary/boundaries.
- *Neumann boundary condition:* Here you specify the value of the derivative of $y(x)$ at the boundary/boundaries.

but one can also specify a mixture of the two.

A second order ordinary differential equation (ODE) requires two boundary conditions. They could in principle be specified both at one side (either $x = x_{\text{left}}$ or $x = x_{\text{right}}$) or one at each side. The latter will turn out to be the best (and usually physically most reasonable) choice. But in the lecture course we have so far not encountered methods of solution that can deal with boundary conditions imposed at both ends of the domain. We have so far only dealt with problems that can be integrated in time or space from one position (e.g. $x = x_{\text{left}}$) to the other (e.g. $x = x_{\text{right}}$), which requires us to impose all boundary conditions at the starting point. In this chapter we deal with methods to handle problems which require us to impose boundary conditions at opposite sides of the domain. Such problem are called *boundary value problems*.

1.1.1 The discretized version of the equation

To solve this equation numerically, we have to define a *grid* in x, i.e. a discrete set of sampling points x_i with $i = 1 \cdots N$. We often regard these points as the centers of *cells*. These cells have *cell walls* located between the x_i sampling points. Usually their location are denoted with $x_{i+1/2}$, where the $i + 1/2$ is just meant to say "between i and $i + 1$" and equivalently $x_{i-1/2}$, where the $i - 1/2$ is just meant to say "between $i - 1$ and i". We can then define the cell walls to be located at:

$$x_{i+1/2} = \tfrac{1}{2}(x_i + x_{i+1}) \qquad x_{i-1/2} = \tfrac{1}{2}(x_{i-1} + x_i) \tag{1.2}$$

In a computer we cannot use "half integers" as array indices, so we will have to define for instance the variables:

- `xc[1]`$\cdots$`xc[N]`: The cell center positions x_i.
- `xi[1]`$\cdots$`xi[N+1]`: The cell wall positions $x_{i-1/2}$. Note that for N cells we have $N + 1$ walls!

Now let's make a *regularly spaced* grid:

$$x_i = x_{\text{left}} + (i - 1/2)\Delta x \tag{1.3}$$
$$x_{i-1/2} = x_{\text{left}} + (i - 1)\Delta x \tag{1.4}$$

where $i = 1 \cdots N$. We place our variables on the x_i positions. First derivatives of some function $f(x)$ at the $x_{i\pm1/2}$ positions:

$$\left(\frac{df}{dx}\right)_{i+1/2} = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} = \frac{f_{i+1} - f_i}{\Delta x} \tag{1.5}$$

This expression is accurate to second order in Δx (you can verify this by using a Taylor expansion of the true function $f(x)$ around $x = x_i$). Likewise we can express the first derivative of $f(x)$ at the x_i position is:

$$\left(\frac{df}{dx}\right)_{i} = \frac{f_{i+1} - f_{i-1}}{x_{i+1} - x_{i-1}} = \frac{f_{i+1} - f_{i-1}}{2\Delta x} \tag{1.6}$$

which is also second order accurate.

The *second derivative* of $f(x)$ at the $x = x_i$ position can be found by taking the derivative of Eq. (1.5):

$$\left(\frac{d^2 f}{dx^2}\right)_i = \frac{1}{\Delta x}\left[\left(\frac{df}{dx}\right)_{i+1/2} - \left(\frac{df}{dx}\right)_{i-1/2}\right] = \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2} \tag{1.7}$$

In Eq. (1.1), however, we have to not just take the second derivative, but the derivative of some function $D(x)$ *times* a derivative:

$$\begin{aligned}\left(\frac{d}{dx}\left[D(x)\frac{df}{dx}\right]\right)_i &= \frac{1}{\Delta x}\left[D_{i+1/2}\left(\frac{df}{dx}\right)_{i+1/2} - D_{i-1/2}\left(\frac{df}{\Delta x}\right)_{i-1/2}\right] \\ &= \frac{1}{\Delta x}\left[D_{i+1/2}\left(\frac{f_{i+1} - f_i}{\Delta x}\right) - D_{i-1/2}\left(\frac{f_i - f_{i-1}}{\Delta x}\right)\right]\end{aligned} \tag{1.8}$$

Using these definitions we can discretize Eq. (1.1):

$$\frac{1}{\Delta x}\left[D_{i-1/2}\left(\frac{y_i - y_{i-1}}{\Delta x}\right) - D_{i+1/2}\left(\frac{y_{i+1} - y_i}{\Delta x}\right)\right] + K_i y_i = q_i \tag{1.9}$$

We now must find a way to solve this equation numerically. We will first look at the "shooting method", but soon after we will discuss a better (more stable) method based on a tridiagonal matrix equation.

Note: Eq. (1.9) is already the fully defined discretized version of Eq. (1.1), which is second order accurate in Δx. If you would instead wish to integrate Eq. (1.1) using e.g. Runge-Kutta, then that Runge-Kutta procedure *itself* will define a discretization. Here, however, we will simply stick to the second order discretization of Eq. (1.9).

1.1.2 Discrete versions of Dirichlet or Neumann boundary conditions

While Eq. (1.9) defines the discrete version of the ODE, the discrete versions of the boundary conditions have to be specified separately. For a Dirichlet boundary condition (let's take the left boundary as an example) we simply replace Eq. (1.9) for $i = 1$ with

$$y_1 = y_{\text{left}} \tag{1.10}$$

A Neumann boundary condition would replace Eq. (1.9) for $i = 1$ with

$$\frac{y_2 - y_1}{\Delta x} = y'_{\text{left}} \tag{1.11}$$

and likewise for the right boundary.

1.1.3 The "shooting method"

One way to solve Eq. (1.9) is to use the *shooting method*. Consider a case when we wish to impose Dirichlet boundary conditions at both ends: $y(x_{\text{left}}) = y_0$ and $y(x_{\text{right}}) = y_1$. We could in principle make an initial guess for the derivative y'_{left} at $x = x_{\text{left}}$, and then, together with $y(x_{\text{left}}) = y_0$, integrate from x_{left} to x_{right}. Usually we will then obtain a value of $y(x_{\text{right}})$ that is not equal to y_1. But we can now adjust our initial guess of y'_{left} and redo the integration, again comparing the resulting $y(x_{\text{right}})$ with what we want it to be (y_1). By using a clever scheme for adjusting the guess of y'_{left} to obtain values of $y(x_{\text{right}})$ that are closer and closer to the desired value y_1 we eventually obtain the solution we seek. This is the shooting method. It allows us to use any of the numerical integration schemes we learned before (Euler, Numerov, Runge-Kutta etc) to integrate the equation from x_{left} to x_{right}. All we need to do is develop a clever method for adjusting our guess for y'_{left}. One way to do this is to create a function $f(\xi)$, with ξ being our guess for y'_{left}, where $f(\xi) = y(x_{\text{right}}) - y_1$ for the given value of ξ. We can then use any off-the-shelf root-finding routine (such as the `zbrent()` routine of Numerical Recipes) to find the value for ξ for which $f(\xi) = 0$. The function $y(x)$ obtained for that value of ξ is then the solution.

This method works well for the classic diffusion equation:

$$-\frac{d}{dx}\left(D(x)\frac{dy(x)}{dx}\right) = q(x) \tag{1.12}$$

which is the generalized diffusion equation with $K(x) = 0$. But if $K(x) > 0$ then this shooting method diverges hopelessly! You can verify this, for instance, with the simple example of $x_{\text{left}} = -10$, $x_{\text{right}} = 10$, $D(x) = 1$, $q(x) = 1$ and $K(x) = 1$ with boundary conditions at $x = x_{\text{left}}$ of $y(x_{\text{left}}) = 1$ and $y'_{\text{left}} = 1$. This yields $y(x_{\text{right}}) \simeq 2.5 \times 10^8$. The shooting method may still work, but it is far less reliable when the equation tends to diverge so quickly. Moreover, the precision of the choice of the value of y'_{left} required to meet the boundary condition at $x = x_{\text{right}}$ may be higher than the float or double precision arithmatic can deal with.

1.1.4 Why does the "shooting method" often fail?

Consider the following version of the above generalized diffusion equation:

$$\frac{1}{3}\frac{d^2y(x)}{dx^2} = y(x) - q(x) \tag{1.13}$$

This is an equation that often appears in the theory of *radiative transfer*: the diffuse movement of radiation through opaque media, e.g. clouds in the Earth's atmosphere. To see why the shooting method tends to create divergencies, we can rewrite this equation into the form of two coupled first order differential equations:

$$\frac{dy(x)}{dx} = -3z(x) \tag{1.14}$$

$$\frac{dz(x)}{dx} = q(x) - y(x) \tag{1.15}$$

In matrix form:

$$\frac{d}{dx}\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} 0 & -3 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} y \\ z \end{pmatrix} + \begin{pmatrix} 0 \\ q \end{pmatrix} \tag{1.16}$$

The matrix has eigenvalues $\lambda = \pm\sqrt{3}$. Evidently it has one negative and one positive eigenvalue. We already know that an equation of the type $dy/dx = \lambda y$ diverges if you integrate from left to right if $\lambda > 0$. Since we have at least one such diverging eigenvalue, this must exponentially diverge! This is the cause of the problem. In fact, integrating in the opposite direction won't help: there, also, there is a diverging component.

By the way (as a side-note): If you try to do the same analysis (i.e. splitting the second order equation into a matrix equation of first order) to the classic diffusion equation (Eq. 1.12) you will encounter a *defective matrix*: A matrix for which no complete set of Eigenvectors exists. Exercise: verify this. Lesson: The above analysis of a system's diverging or converging "eigencomponents" does not always apply; but often it does. In this particular case of a classic diffusion equation the shoothing method works fine.

1.1.5 Casting the problem into a matrix equation

Clearly, for problems for which the shooting method fails, we must find an alternative method: one in which we can impose one boundary condition on one side and one on the other side from the start. This can be done by constructing the matrix equation corresponding to Eq. (1.9) and its boundary conditions. We define the N-dimensional vectors $\mathbf{y}$ and $\mathbf{q}$ as

$$\mathbf{y} = (y_1, y_2, \cdots, y_{N-1}, y_N)^T \qquad \mathbf{q} = (q_1, q_2, \cdots, q_{N-1}, q_N)^T \tag{1.17}$$

We now want to construct an $N \times N$ matrix $\mathbf{A}$ such that the equation

$$\mathbf{A} \cdot \mathbf{y} = \mathbf{q} \tag{1.18}$$

(where $\cdot$ is the matrix-vector product) represents Eq. (1.9) with the corresponding boundary conditions. You can see that Eq. (1.9) for some i contains y_{i-1}, y_i and y_{i+1}, but no other y-values. And the Dirichlet or Neumann boundary conditions only involve y_1 and y_2 (for the left boundary) and y_{N-1} and y_N (for the right boundary). In other words: the matrix $\mathbf{A}$ is a *tridiagonal matrix*:

$$\mathbf{A} = \begin{pmatrix} b_1 & c_1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{N-2} & b_{N-2} & c_{N-2} & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & a_{N-1} & b_{N-1} & c_{N-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & a_N & b_N \end{pmatrix} \tag{1.19}$$

The matrix elements and right-hand-side elements can be found from Eq. (1.9). For $2 \leq i \leq N-1$ we have

$$a_i = -\frac{D_{i-1/2}}{\Delta x^2} \tag{1.20}$$

$$b_i = \frac{D_{i-1/2} + D_{i+1/2}}{\Delta x^2} + K_i \tag{1.21}$$

$$c_i = -\frac{D_{i+1/2}}{\Delta x^2} \tag{1.22}$$

$$r_i = q_i \tag{1.23}$$

Now suppose we put a Neumann boundary condition on the left side and a Dirichlet boundary condition on the right side. We then get:

$$b_1 = 1/\Delta x \tag{1.24}$$

$$c_1 = -1/\Delta x \tag{1.25}$$

$$r_1 = y'_{\text{left}} \tag{1.26}$$

$$a_N = 0 \tag{1.27}$$

$$b_N = 1 \tag{1.28}$$

$$r_N = y_{\text{right}} \tag{1.29}$$

We can now use a tridiagonal matrix equation solver (see exercises) to solve this system of equations. This immediately gives us our solution, because we have a linear problem.

1.2 Heat conduction

The generalized diffusion equation has many applications. One of them is heat transport. It is an every-day experience that heat tends to move from hot regions to cooler regions. One can express this quantitatively as

$$\vec{F}(\vec{x}) = -D(\vec{x})\vec{\nabla}T(\vec{x}) \tag{1.30}$$

where $\vec{x}$ is the spatial position vector and $\vec{F}(\vec{x})$ is the heat flux and $D(\vec{x})$ the heat conductivity coefficient (which we allow to vary in space). Consider now a source function $q(\vec{x})$. The stationary-state conservation of energy now reads:

$$\vec{\nabla} \cdot \vec{F}(\vec{x}) = -\vec{\nabla} \cdot [D(\vec{x})\vec{\nabla}T(\vec{x})] = q(\vec{x}) \tag{1.31}$$

In 1-D this equation becomes

$$-\frac{d}{dx}\left(D(x)\frac{dT(x)}{dx}\right) = q(x) \tag{1.32}$$

We can now solve this using the above matrix equation. In the exercise class we will work out an explicit example.

1.3 2-D Diffusion problem

So what about a 2-D heat conduction problem? The equation is

$$-\frac{d}{dx}\left(D(x,y)\frac{dT(x,y)}{dx}\right) - \frac{d}{dy}\left(D(x,y)\frac{dT(x,y)}{dy}\right) = q(x,y) \tag{1.33}$$

Let us, for simplicity, take $D(x,y) = D$ =constant, so that the equation becomes

$$-D\left(\frac{d^2T(x,y)}{dx^2} + \frac{d^2T(x,y)}{dy^2}\right) = q(x,y) \tag{1.34}$$

How do we cast this into matrix form? Let us first define a grid in x and y: We have x_i with $1 \leq i \leq N$ and y_j with $1 \leq j \leq M$. We construct both the x and y grids in the same way as we did for the 1-D case (Section 1.1.1). We thus have an $N \times M$ grid. We wish to solve for $T_{i,j}$. In order to be able to make a matrix equation for this, we must define a large vector $\mathbf{z}$:

$$\mathbf{z} = (T_{1,1}, \cdots, T_{N,1}, T_{1,2}, \cdots, T_{N,2}, \cdots, \cdots, T_{1,M-1}, \cdots, T_{N,M-1}, T_{1,M}, \cdots, T_{N,M})^T \tag{1.35}$$

i.e. a vector with NM components. So for a grid of 20×30 (i.e. $N = 20$, $M = 30$) we have a vector of 600 components: $z_1, \cdots z_{600}$. We also create a vector $\mathbf{r}$ for the right-hand-side of the equation:

$$\mathbf{r} = (q_{1,1}, \cdots, q_{N,1}, q_{1,2}, \cdots, q_{N,2}, \cdots, \cdots, q_{1,M-1}, \cdots, q_{N,M-1}, q_{1,M}, \cdots, q_{N,M})^T \tag{1.36}$$

We then want to construct a matrix $\mathbf{A}$ such that

$$\mathbf{A} \cdot \mathbf{z} = \mathbf{r} \tag{1.37}$$

is the matrix-equation corresponding to Eq. (1.34). It turns out that, like the case of a tridiagonal matrix, also here most of the elements of the matrix $\mathbf{A}$ are zero, except for some special locations. However, in this case the matrix is not tridiagonal anymore. It is "tridiagonal with sidebands":

$$A_{k,k} = 4D/\Delta x^2 \tag{1.38}$$
$$A_{k+1,k} = -D/\Delta x^2 \tag{1.39}$$
$$A_{k-1,k} = -D/\Delta x^2 \tag{1.40}$$
$$A_{k+N,k} = -D/\Delta x^2 \tag{1.41}$$
$$A_{k-N,k} = -D/\Delta x^2 \tag{1.42}$$

for $k = i + (j-1)N$ with $2 \leq i \leq N-1$ and $2 \leq j \leq M-1$. The boundary conditions have to be specified all around the domain, i.e. at all y_j for $x_i = x_1$, all y_j for $x_i = x_N$, all x_i for $y_j = y_1$ and all x_i for $y_j = y_M$.

In spite of the fact that also this matrix $\mathbf{A}$ has mostly 0-elements (it is a so-called *sparse matrix*), the presence of the side-bands now makes it less easy to solve the matrix equation than for a tridiagonal matrix. So we are forced back to the LU-decomposition method. But for very large N and M this may become extremely computationally expensive.

Fortunately there is a whole class of methods that can solve such *sparse matrix equations* relatively efficiently even for large NM. Most of these are based on iteration. Some famous methods are `GMRES`, `BiCG`, `BiCGSTAB`. We will not go into these methods here. It is just important to remember that for large multi-dimensional problems, the corresponding matrix equations usually are very large sparse matrix equations that require such special methods such as `GMRES`, `BiCG` or `BiCGSTAB` to be solved within reasonable time.

Note that in a similar manner one can do to 3-D. This adds two more side bands, even further away from the diagonal.

1.4 The Cable Equation

The *cable equation* is a simple model of signal propagation through neuronal fibres (dendrites/axons). The model is related to the *telegraph equation* for signal propagation through electric wires.

The model describes a neuronal fibre as a cylinder of radius a containing intracellular fluid (cytosol). The surface of the cylinder consists of two membranes, the phospholipid bilayer, that act as a capacitance that can store charge. We write the capacitance per unit surface of the pair of membranes as c_m (unit: Farad/m^2). The bilayer is a reasonably good insulator, but not perfect. It has a certain resistance times unit surface r_m (unit: Ohm·m^2). This means that some of the current that flows through the fibre can leak out sideways through the double membrane of the fibre. Let us call this current-per-unit-surface i_m (unit: Ampère/m^2), and define it positive for current flowing out of the fibre. Let us define the voltage at the inside of the bilayer as V while the voltage outside the bilayer as 0. Ohm's law then states

$$i_m = \frac{V}{r_m} \tag{1.43}$$

If V changes with time, then part of this current is consumed in readjusting the charge in the capacitor. This is called the displacement current, which we will write as i_d:

$$i_d = -c_m \frac{\partial V}{\partial t} \tag{1.44}$$

The remainder of i_m is actual current from the cytosol to the outside of the fibre, i.e. this is the leaking current i_l:

$$i_l = i_m - i_d \tag{1.45}$$

Now let us look at how a current moves *along* the cylinder. Define the current in the cylinder as $I(x,t)$, where x is the coordinate along the cylinder and t is time, and $I > 0$ for current flowing toward positive x. Define the resistance along the cable as R. If we first assume that there is no loss through the membranes $r_m = \infty$ and that the membranes

have zero capacity $c_m = 0$, then we can look at a steady-state situation. The voltage $V(x)$ and current I must obey

$$\frac{\partial V(x)}{\partial x} = -RI \tag{1.46}$$

where $I(x) = I$ =constant. However, *with* the membrane capacitance and finite resistance the equation becomes more complex.

Let us introduce the current through the membrane per unit length of cable I_m, as well as I_l and I_d as

$$I_m = 2\pi a i_m \qquad I_l = 2\pi a i_l \qquad I_d = 2\pi a i_d \tag{1.47}$$

the capacitance per unit length of cable C_m as

$$C_m = 2\pi a \; c_m \tag{1.48}$$

and the membrane resistance times unit length of cable R_m as

$$R_m = \frac{r_m}{2\pi a} \tag{1.49}$$

We can then write the continuity equation for $I(x)$ (ignoring the $\partial/\partial t$ term) as

$$\frac{\partial I(x)}{\partial x} = -I_l = I_d - I_m = -C_m \frac{\partial V}{\partial t} - \frac{V}{R_m} \tag{1.50}$$

and inserting Eq. (1.46) we obtain

$$\frac{1}{R}\frac{\partial^2 V}{\partial x^2} = C_m \frac{\partial V}{\partial t} + \frac{V}{R_m} \tag{1.51}$$

where we write V instead of $V(x,t)$ just for clarity of the equation. This equation is called the *cable equation*.

We can make this equation dimensionless by introducing a length scale $\lambda = \sqrt{R_m/R}$ and a time scale $\tau = C_m R_m$. Then the equation becomes

$$\lambda^2 \frac{\partial^2 V}{\partial x^2} = \tau \frac{\partial V}{\partial t} + V \tag{1.52}$$

1.4.1 Stationary state solutions

For a steady-state situation the cable equation becomes

$$\lambda^2 \frac{\partial^2 V}{\partial x^2} = V \tag{1.53}$$

which is, for the choice $\lambda = 1/\sqrt{3}$ identical to the diffusion equation for radiative transfer, Eq. (1.13) with $q(x) = 0$. So also for the cable equation we expect that a shooting method approach will fail, so we need to solve a matrix equation instead. This goes perfectly analogous to what we did in Section 1.1.5.

1.4.2 Time-dependent solutions

A straightforward discretization of the cable equation for time-dependent Euler integration reads

$$\tau \frac{V_i^{n+1} - V_i^n}{\Delta t} = \lambda^2 \frac{V_{i+1}^n - 2V_i^n + V_{i-1}^n}{\Delta x^2} - V_i^n \tag{1.54}$$

where V_i^n is the potential at position i and time step n. Finding V_i^{n+1} in this manner can work, but only if $\Delta t \ll \tau \Delta x^2/\lambda^2$. For $\Delta t \gtrsim \tau \Delta x^2/\lambda^2$ this direct forward Euler integration becomes numerically violently unstable! It means that one may need to do very many time steps, meaning a computationally expensive integration.

An often-used trick to speed up the calculation is to rewrite the equation in the following way:

$$\tau \frac{V_i^{n+1} - V_i^n}{\Delta t} = \lambda^2 \frac{V_{i+1}^{n+1} - 2V_i^{n+1} + V_{i-1}^{n+1}}{\Delta x^2} - V_i^{n+1} \tag{1.55}$$

i.e. take as the right-hand-side the *future* variables instead of the current ones. Putting all future variables to the left and all current ones to the right, we obtain

$$\tau \frac{V_i^{n+1}}{\Delta t} - \lambda^2 \frac{V_{i+1}^{n+1} - 2V_i^{n+1} + V_{i-1}^{n+1}}{\Delta x^2} + V_i^{n+1} = \tau \frac{V_i^n}{\Delta t} \tag{1.56}$$

which can be cast into a matrix equation and solved for V_i^{n+1}. In other words, the matrix equation gives us the values of V_i at the next time step. This is called *implicit integration* (or *implicit differencing*) of the partial differential equation. It turns out that with this method the integration is numerically stable, even for relatively large Δt.

1.5 Afterword

In this chapter we have dealt with *linear* boundary value problems. However, in practice, many boundary value problems are *non-linear*. Nevertheless one can use the same methods as above, but now in an iterative scheme. For each iteration we write the vector $\mathbf{y} = \mathbf{y}_{\text{prev}} + \delta\mathbf{y}$, where $\mathbf{y}_{\text{prev}}$ is the vector $\mathbf{y}$ from the previous iteration. We then linearize the equations in $\delta\mathbf{y}$ and solve for $\delta\mathbf{y}$. We will not go into detail on this, because the principles are identical to those of *Newton-Raphson iteration* for root-finding.