Numerical Methods for One-dimensional Boundary Problems

Fernando Ruiz Ruiz
Departamento de Física Teórica I
Universidad Complutense de Madrid

Madrid, January 10th, 2012

These notes try to provide an introduction (never a detailed account) to nonlinear shooting and finite difference methods for one-dimensional boundary value problems. They are part of a course on Computational Physics that I have been lecturing to undergraduates at Universidad Complutense of Madrid.

I have tried to emphasize the practical aspects of the subject. In fact, I have first selected the exercises and then added the strictly necessary theoretical background to tackle them. Wherever I have needed a programming language to implement numerical schemes and produce figures, I have used Maple[1]. I must make clear that Maple has never been used as a solver and that it has not been my goal to write the most efficient and elegant codes but ones which illustrate how the numerical methods of interest work. Print copies of some of the used codes are included.

The material covered here is well known and established. It is divided in five paragraphs:

1. Uniqueness theorems for solutions to one-dimensional boundary problems.
2. Nonlinear shooting method for Dirichlet boundary problems.
3. Nonlinear shooting method for Neumann boundary problems.
4. Finite differences. General ideas.
5. Finite difference methods for boundary value problems.

You are free to use these notes with non-commercial purposes. My (tentative) intention is to enlarge them by including other topics covered in the course.

F. Ruiz

[1]Maple, by Maplesoft, Waterloo Maple Inc.

1. Uniqueness theorems for solutions to one-dimensional boundary problems.

In the following we will be interested in solving the boundary value problem

$$\left.\begin{array}{l} y''(x) = f(x, y(x), y'(x)) \quad a \le x \le b \\ y(a) = \alpha \\ y(b) = \beta \end{array}\right\} \quad \text{(BVP)}, \tag{1.1}$$

with f a given function of its arguments. Let us recall without proof that uniqueness of solutions is discussed by the following

Theorem 1.1. Let $D = \{(x, y, y') : \ a \le x \le b, -\infty < y, y' < \infty\}$. If

(i) f, f_y and $f_{y'}$ are continuous in D,
(ii) $f_y > 0$ in D, and
(iii) $|f_{y'}| \le const$ in D,

then the problem (BVP) has a unique solution.

For linear problems

$$\left.\begin{array}{l} y''(x) = p(x)\, y' + q(x)\, y + r(x) \quad a \le x \le b \\ y(a) = \alpha \\ y(b) = \beta \end{array}\right\} \quad \text{(Lin-BVP)},$$

the theorem takes the form

Theorem 1.2. If $p(x)$, $q(x)$ and $r(x)$ are continuous in $[a, b]$ and $q(x) > 0$ in $[a, b]$, the solution to problem (Lin-BVP) is unique.

The proofs can be found in ordinary differential equations textbooks.

2. Nonlinear shooting method for Dirichlet boundary problems.

The shooting method consists in **replacing** the boundary value problem (1.1) with the initial value problem

$$\left.\begin{array}{l} y_s''(x) = f(x, y_s(x), y_s'(x)) \quad a \le x \\ y_s(a) = \alpha \\ y_s'(a) = s \end{array}\right\} \quad \text{(IVP)}\,, \tag{2.1}$$

and **finding** the value s for which

$$y_s(b) - \beta = 0\,. \tag{2.2}$$

The boundary problem (1.1) is thus replaced by the combination of an initial problem and an equation in s. To solve this combination, one proceeds by iteration. Take an ansatz for s, say $s = s_0$, and solve the initial problem (2.1). By construction, the solution $y_0(x)$ satisfies the boundary condition $y_0(a) = \alpha$. If, in addition, it satisfies the boundary condition $y_0(b) = \beta$ at the end point $x = b$, then $y_0(x)$ is a solution to the boundary value problem (1.1) and one is done. If not, one improves the choice for s by taking $s = s_1$ and proceeds as for s_0. In this way, one ends up with a sequence of initial value problems

$$\left.\begin{array}{l} y_k''(x) = f(x, y_k(x), y_k'(x)) \quad a \le x \\ y_k(a) = \alpha \\ y_k'(a) = s_k \end{array}\right\} \quad \text{(IVP)}_k \tag{2.3}$$

that must be solved until condition

$$y_k(b) - \beta = 0 \tag{2.4}$$

is satisfied. We therefore need a way to choose the initial slopes $s_0, s_1, s_2, \ldots$ so as to converge to a solution of equation (2.12). Eq. (2.12) is an equation in s of the form $F(s) = 0$, with $F(s) = y_s(b) - \beta$, and there are many methods to solve equations of this type. Two of the most popular ones are the secant method and the Newton method. They give for the solution

$$\text{Secant:} \quad s_{k+2} = s_{k+1} - F(s_{k+1})\,\frac{s_{k+1} - s_k}{F(s_{k+1}) - F(s_k)}$$

$$\text{Newton:} \quad s_{k+1} = s_k - \frac{F(s_k)}{F'(s_k)}\,.$$

In our case, these expressions become

$$\text{Secant:} \quad s_{k+2} = s_{k+1} - \big[y_{k+1}(b) - \beta\big]\,\frac{s_{k+1} - s_k}{y_{k+1}(b) - y_k(b)} \tag{2.5}$$

$$\text{Newton:} \quad s_{k+1} = s_k - \frac{y_k(b) - \beta}{\dfrac{dy_k}{ds_k}(b)}\,. \tag{2.6}$$

The secant method requires two values s_0 and s_1 to start the iteration and is known to converge if s_0 and s_1 are such that the solution is in between them. It only uses the function $F(s)$ and not its derivative

By contrast, Newton's method only needs one value, s_0, to start the iteration, but involves the derivative $\frac{dy_k}{ds_k}$ at $x = b$. There is no rule as for what value for s_0 one should take, but it is quite usual to take (at least for nonpathological cases)

$$s_0 = \frac{\beta - \alpha}{b - a}\,. \tag{2.7}$$

To control the derivative $\frac{dy_k}{ds_k}$, we introduce the quantity

$$z_s(x) := \frac{dy_s(x)}{ds}\,.$$

Since x and s are independent, it follows that

$$\frac{dz_s}{dx} = \frac{d}{dx}\,\frac{dy_s}{ds} = \frac{d}{ds}\,\frac{dy_s}{dx} = \frac{dy_s'}{ds}$$
$$\frac{d^2z_s}{dx^2} = \frac{d}{dx}\,\frac{dy'}{ds} = \frac{d}{ds}\,\frac{dy_s'}{dx} = \frac{dy_s''}{ds}\,.$$

Using now the differential equation in (2.1), we have

$$\frac{d^2z_s}{dx^2} = \frac{d}{ds}\,f\big(x, y_s(x), y_s'(x)\big) = \frac{\partial f}{\partial y_s}\,\frac{dy_s}{ds} + \frac{\partial f}{\partial y_s'}\,\frac{dy_s'}{ds} = \frac{\partial f}{\partial y}\Big|_{y_s} z_s + \frac{\partial f}{\partial y'}\Big|_{y_s} z_s'\,.$$

Furthermore, from the initial data in (2.1), we have

$$z_s(a) = \frac{dy_s(x)}{ds}\,\Big|_{x=a} = 0$$
$$z_s'(a) = \frac{dy_s'(x)}{ds}\,\Big|_{x=a} = 1\,.$$

This gives the auxiliary initial value problem for $z_k = \frac{dy_k}{ds_k}$

$$\left.\begin{aligned} z_k''(x) &= \frac{\partial f}{\partial y_k}\,z_k + \frac{\partial f}{\partial y_k'}\,z_k' \quad a \le x \\ z_k(a) &= 0 \\ z_k'(a) &= 1 \end{aligned}\right\} \quad \text{(Aux-IVP)}_k\,. \tag{2.8}$$

The way to proceed for Newton's method is now clear: every initial value problem $(\text{IVP})_k$ must be solved together with its auxiliary problem $(\text{Aux-IVP})_k$, with the choice of s_k following the iteration law

$$s_{k+1} = s_k - \frac{y_k(b) - \beta}{z_k(b)}\,. \tag{2.9}$$

Remark In problem (1.1) the boundary conditions are of Dirichlet type. Later on we will consider other types of boundary conditions.

Advantages and disadvantages of the shooting method. The main advantage is its simplicity. Its main disadvantage is that convergence to a solution may crucially depend on the choice of the initial slope s_0, as Exercise 2.3 below illustrates.

Exercise 2.1. *Consider the nonlinear boundary value problem*

$$\left.\begin{aligned} y'' &= -\frac{y'}{y^2} + \frac{2}{x^3} + \frac{x^2-1}{(x^2+1)^2} \qquad 1 \le x \le 2 \\ y(1) &= 2 \\ y(2) &= \frac{5}{2} \end{aligned}\right\} .$$

(a) *Use the shooting method to numerically solve it. Take RK4 as initial value problem solver and the secant method to improve the choice of initial slope.*

(b) *Simple inspection shows that*

$$y_{\rm a}(x) = x + \frac{1}{x} \tag{2.10}$$

is an analytic solution. Compare it with the numerical solution obtained in (a). Note that Theorem 1.1 does not provide information about the uniqueness of the solution, since $|f_{y'}| = |1/y'^2|$ *is not bounded from above for* $y' \to 0$*, so that premise (iii) in the theorem is not met.*

Solution to 2.1. Introducing the notation

$$y_{1,k} = y_k \qquad y_{2,k} = y'_k \,,$$

the sequence of initial value problems (2.3) to be solved takes in this case the form

$$\left.\begin{aligned} y'_{1,k}(x) &= y_{2,k} \\ y'_{2,k}(x) &= -\frac{y_{2,k}}{y_{1,k}^2} + \frac{2}{x^3} + \frac{x^2-1}{(x^2+1)^2} \\ y_{1,k}(1) &= 1 \\ y_{2,k}(1) &= s_k \end{aligned}\right\} . \tag{2.11}$$

We divide the domain $[1\,,2]$ in N subintervals $[x_{i-1}, x_i]$, all of length $h = 1/N$. Since RK4 has local precision h^4 (local error h^5) and N steps are necessary to go from $x_0 = 1$ to $x_N = 2$, the total precision at $x_N = 2$ will be $Nh^4 \sim h^3$. Condition (2.4) then becomes

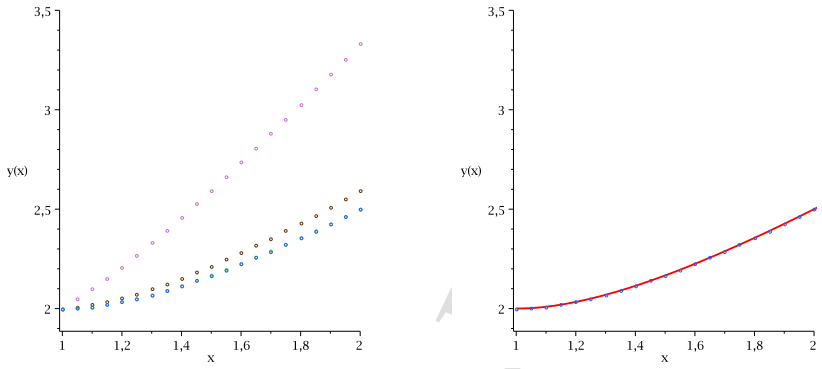
$$\left| y_{1,k}(2) - \frac{5}{2} \right| \le h^4 \,. \tag{2.12}$$

Let us take $h = 0.05$ and $s_0 = 0.1$ and $s_1 = 0.9$ as starting points for the secant iteration. The Maple code in Appendix 2.1 at the end of this section implements the shooting method for the combination RK4+secant.

RK4 applied to the initial value problem (2.11) with $s_0 = 0.1$ gives for the solution at $x_N = 2$ the value $y_{1,0}(2) = 2.5911155390$. This does not satsify condition (2.12), so we move onto the next iteration, $k = 1$. This produces $y_{1,1}(2) = 3.3326054890$, which still is not good enough, so we iterate once again, and so on. Condition (2.12) is met after five iterations, which corresponds to $k = 4$. At this point, the iteration stops and $y_{1,4}(x_i)$ is taken as the numerical solution. The table that follows is produced by the `printf` commands in the Maple code in Appendix 2.1. It lists the initial slope and the value of the solution at the endpoint $x = 2$ for every iteration:

Iteration	y'(a)=s	y(b)
0	0.1000000000	2.5911155390
1	0.9000000000	3.3326054890
2	0.0016946471	2.5015408140
3	0.0000291673	2.5000268120
4	-0.0000003273	2.5000000030

The `plot` commands in the code produce the two figures below. That on the left collects the solution $y_k(x)$, $k = 0, 1, 2, 3, 4$, for each initial value problem $(\text{IVP})_k$, while that on the right compares the numerical solution $y_4(x)$ with the analytic solution $y_a(x)$. The solutions for iterates $k = 2, 3, 4$ are so close that no difference is visually appreciated. Note in this regard that the precision required at the endpoint $x = 2$ is $h^3 \sim 10^{-4}$.



Numerical solution for the problem (2.11) using RK4+secant.
Left: plots for iterates $k = 0, 1, 2, 3, 4$. Right: numerical versus analytic

The code can be easily adapted to other boundary value problems by changing the boundary data and the differential equation. It is convenient to play with it and to change the mess h and the tolerance.

Exercise 2.2. Solve exercise 2.1.using Newton's method instead of the secant method. Take as starting point for Newton's iteration $s_0 = 0.5$.

Solution to 2.2. To implement Newton's method, the sequence of inital value problems (2.11) must be complemented with the sequence of auxiliary value problems (2.8). In the case at hand, this takes the form

$$\left.\begin{aligned} z'_{1,k}(x) &= z_2 \\ z'_{2,k}(x) &= 2\,\frac{y_{2,k}}{y_{1,k}^3}\,z_{1,k} - \frac{1}{y_{1,k}^2}\,z_{2,k} \\ z_{1,k}(1) &= 0 \\ z_{2,k}(1) &= 1 \end{aligned}\right\}, \tag{2.13}$$

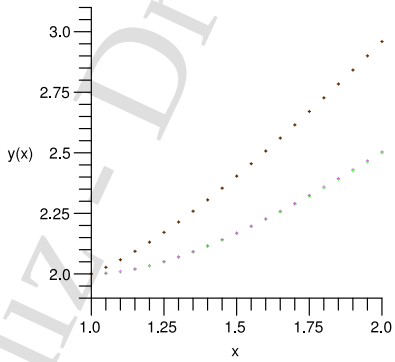
where we have set

$$z_{1,k} = z_k \qquad z_{2,k} = z'_k\,.$$

Problems (2.11) and (2.13) must be solved at the same time for every k. A Maple code doing this can be found in Appendix 2.2. Now only three iterations are needed to reach the precision required. The `printf` commands in the code produce the table

Iteration	y'(a)=s	y(b)
0	0.5000000000	2.9593806970
1	0.0039212262	2.5035650680
2	-0.0000047904	2.4999959450

The iteration plots are shown below, with the graphs for iterates $k = 1, 2$ so close to each other that they cannot been distinguised.



Iterates for the problem (2.11) using RK4+Newton

Exercise 2.3. *Consider the problem*

$$\left.\begin{aligned} y'' &= \frac{1}{2}\,y'^2 y - 4x + \frac{4}{x} + \frac{8}{x^3} - \frac{4}{x^5} \qquad 1 \le x \le 2 \\ y(1) &= 4 \\ y(2) &= 5 \end{aligned}\right\} . \tag{2.14}$$

Theorem 1.1 now applies and implies that (2.14) has a unique solution. It is very easy to check that it is given by

$$y_{\rm a} = 2\left(x + \frac{1}{x}\right).$$

(a) *Reproduce numerically this solution using nonlinear shooting based on RK4 and Newton's method, and*

(b) *Study the dependence of the method on the choice of initial slope s_0.*

Solution to 2.3. We use the same code as for Exercise 2.2 (see Appendix 2.2.) to numerically solve this problem. As step size we take $h = 0.05$ and require precision h^3 at the boundary $x = 2$. Let us take as a first guess for s_0 the slope s_0 of the analytic solution at $x = 1$, given by

$$s_0^{\rm a} = \left.\frac{dy_{\rm a}}{dx}\right|_{x=1} = 0\,.$$

We expect the shooting method to converge very quickly. And this is the case, since the solution is reached in one iteration, with

```
 Iteration      y'(a)=s          y(b)
-----------  --------------  --------------
     0        0.5000000000    5.0000358130
```

Let us take now as a second guess $s_0 = 0.01$, which is relatively close to the analytic value $s_0^{\rm a}$. The solution is then reached in six iterations, with

```
 Iteration      y'(a)=s          y(b)
-----------  --------------  --------------
     0        0.0100000000    5.1831590080
     1       -0.0001364270    4.9980439740
     2        0.0000223574    5.0003631890
     3       -0.0000069801    4.9999336570
     4       -0.0000016162    5.0000121570
     5       -0.0000025989    4.9999977760
```

This makes sense, since one expects the number of iterations to grow as the ansatz for s_0 separates from the true value $s_0^{\rm a}$. Let us now separate further from $s_0^{\rm a}$ and take $s_0 = 0.04$. The values of $y_k(x)$ at the interval's enpoint $x = 2$ now are

Iteration	y'(a)=s	y(b)
0	0.04	1.2e+16
1	-7.1e+07	NaN

The first iteration gives $y_0(2) = 1.2 \times 10^{16}$, wich is very far away from the required $y(2) = 5$. The second iteration worsens the situation since $y_1(2) \to \infty$, thus showing that the method does not converge. It is remarkable that by deviating from the analytic slope $s_0^{\mathrm{a}} = 0.00$ by 0.04, which is less than the step size $h = 0.05$, convergence is lost. This illustrates the sensitivity of the shooting method to the choice of the initial slope s_0.

Appendix 2.1. Maple program for Exercise 2.1 (RK4 + Secant).

```
> restart: with(plots):
```

Problem data.

```
> a:=1.: b:=2: alpha:=2.: beta:=5./2:
> f:=(x,y1,y2) -> array(1..2,
>                 [y2,-y2/y1^2 + 2/x^3 + (x^2-1)/(x^2+1)^2]):
```

Split domain in N subintervals of equal length h.

```
> N:=20:
> x:=array(0..N):
>    x[0]:=a: x[N]:=b:
>    h:=evalf((x[N]-x[0])/N):
>    for n from 0 to N do x[n]:=x[0]+n*h: end do:
```

Organize solution (y_1, y_2) in an array.

```
> y:=array(1..2,0..N):
>    y[1,0]:=alpha:
```

M counts secant iterations. M_{max} sets upper bound for number of iterations.
yb denotes $y_1(b)$

```
> M:=0: M_max:=100:
> s[0]:=0.1: s[1]:=0.9:
> yb:=array(0..M_max):
```

Loop for sequence of IVP's.

```
> k := array(1..2,1..4):     # Auxiliary vectors for RK4 to be used below.
> for M from 0 to M_max do
>    ### Secant's definition for initial slopes.
>    if (M>1) then
>       s[M] := s[M-1]
>       - (yb[M-1] - beta)*(s[M-1]-s[M-2])/(yb[M-1]-yb[M-2]):
>       end if:
>    ### RK4 for (y1,y2) IVP:
>    y[2,0]:=s[M];
>    for n from 0 to N-1 do
>        for i from 1 to 2 do
>           k[i,1] := f(x[n],y[1,n],y[2,n])[i]:
>           end do:
```

```
>          for i from 1 to 2 do
>             k[i,2] := f(x[n]+h/2,
>                    y[1,n]+h/2*k[1,1], y[2,n]+h/2*k[2,1])[i]:
>             end do:
>          for i from 1 to 2 do
>             k[i,3] := f(x[n]+h/2,
>                    y[1,n]+h/2*k[1,2], y[2,n]+h/2*k[2,2])[i]:
>             end do;
>          for i from 1 to 2 do
>             k[i,4] := f(x[n]+h,
>                        y[1,n]+h*k[1,3], y[2,n]+h*k[2,3])[i] ;
>             end do;
>          for i from 1 to 2 do
>             y[i,n+1] := y[i,n]
>                + h/6 * (k[i,1] + 2*k[i,2] + 2*k[i,3] + k[i,4] );
>             end do;
>      end do;
>      ### End of RK4 for (y1, y2)
>      ### Creates plot for every iterate.
>      plotpoints[M] := plot({seq([x[n],y[1,n]],n=0..N)},
>         color=COLOR(RGB, rand()/10^12, rand()/10^12, rand()/10^12),
>         style=point, symbol=circle, symbolsize=8, labels=["x","y(x)"]):
>      ### Print out s and y(b) for every iteration.
>      if (M=0) then
>         printf("   Iteration      y'(a)=s            y(b) \n");
>         printf("  -----------  -------------   --------------\n");
>         end if;
>      printf("%8g%20.10f%17.10f\n",M,s[M],y[1,N]);
>      ### Check of boundary condition at x = b.
>      yb[M]:=y[1,N];
>      if (M>=2) and (abs(yb[M]-beta)<h^4)then break; end if;
>   end do:
```

End of loop for IVP's.
Plots of iterations.

```
>   PPP:=seq(plotpoints[r],r=0..M):
>   display(PPP,view=[1..2,1.9..3.5]);
```

Comparison with analytic solution $y(x) = x + 1/x$.

```
> display(plot(x+1/x, x=1..3, color=red, view=[1..2,1.9..3.5],
>    thickness=2,labels=["x","y(x)"]),{plotpoints[M]});
```

Appendix 2. 2. Maple program for Exercise 2.2 (RK4 + Newton).

```
> restart: with(plots):
```

Problem data.

```
> a:=1.: b:=2: alpha:=2.: beta:=5./2:
> f:=(x,y1,y2) -> array(1..2,
>                 [y2,-y2/y1^2 + 2/x^3 + (x^2-1)/(x^2+1)^2]):
> g:=(x,y1,y2,z1,z2) -> array(1..2,
>                 [z2, 2*y2*z1/y1^3-z2/y1^2]):
```

Split domain in N subintervals of equal length h.

```
> N:=20:
> x:=array(0..N):
>    x[0]:=a: x[N]:=b:
>    h:=evalf((x[N]-x[0])/N):
>    for n from 0 to N do x[n]:=x[0]+n*h: end do:
```

Organize (y_1, y_2) and (z_1, z_2) in arrays.

```
> y:=array(1..2,0..N):
>    y[1,0]:=alpha:
> z:=array(1..2,0..N):
>    z[1,0]:=0:
>    z[2,0]:=1:
```

M counts Newton iterations, with M_{max} an upper bound.

```
> M:=0: M_max:=100:
> s[0]:=(beta-alpha)/(b-a):
```

Loop for sequence of IVP's

```
> k := array(1..2,1..4):     # Auxiliary vectors for RK4 to be used below.
> kz := array(1..2,1..4):    # Auxiliary vectors for RK4 to be used below.
> for M from 0 to M_max do
>    ###  RK4 for (y1, y2) IVP.
>    y[2,0]:=s[M]:
>    for n from 0 to N-1 do
>       for i from 1 to 2 do
>          k[i,1] := f(x[n],y[1,n],y[2,n])[i]:
>          end do:
```

```
>        for i from 1 to 2 do
>           k[i,2] := f(x[n]+h/2,
>                   y[1,n]+h/2*k[1,1], y[2,n]+h/2*k[2,1])[i]:
>           end do:
>        for i from 1 to 2 do
>           k[i,3] := f(x[n]+h/2,
>                   y[1,n]+h/2*k[1,2], y[2,n]+h/2*k[2,2])[i]:
>           end do:
>        for i from 1 to 2 do
>           k[i,4] := f(x[n]+h,
>                   y[1,n]+h*k[1,3], y[2,n]+h*k[2,3])[i] ;
>           end do:
>        for i from 1 to 2 do
>           y[i,n+1] := y[i,n]
>               + h/6 * (k[i,1] + 2*k[i,2] + 2*k[i,3] + k[i,4] );
>           end do:
>     end do:
>     ### End of RK4 for (y1, y2)
>     ### Creates plot for every iterate.
>     plotpoints[M] := plot( {seq([x[n],y[1,n]],n=0..N)},
>        color=COLOR(RGB, rand()/10^12, rand()/10^12, rand()/10^12),
>        style=point, symbol=cross, symbolsize=4,labels=["x","y(x)"]):
>     ### Print out s and y(b) for every iteration.
>     if (M=0) then
>        printf("   Iteration       y'(a)=s            y(b) \n");
>        printf("  -----------  -------------   --------------\n");
>        end if;
>     printf("%8g%20.10f%17.10f\n",M,s[M],y[1,N]);
>     for n from 0 to N-1 do
>     ### RK4 for (z1, z2) auxiliary IVP.
>     for n from 0 to N-1 do
>         for i from 1 to 2 do
>            kz[i,1] := g(x[n],y[1,n],y[2,n],z[1,n],z[2,n])[i]:
>            end do:
>         for i from 1 to 2 do
>            kz[i,2] := g(x[n]+h/2,y[1,n],y[2,n],
>                  z[1,n]+h/2*kz[1,1], z[2,n]+h/2*kz[2,1])[i]:
>            end do:
```

```
>          for i from 1 to 2 do
>             kz[i,3] := g(x[n]+h/2,y[1,n],y[2,n],
>                    z[1,n]+h/2*kz[1,2], z[2,n]+h/2*kz[2,2])[i]:
>             end do:
>          for i from 1 to 2 do
>             kz[i,4] := g(x[n]+h,y[1,n],y[2,n],
>                         z[1,n]+h*kz[1,3], z[2,n]+h*kz[2,3])[i] ;
>             end do:
>          for i from 1 to 2 do
>             z[i,n+1] := z[i,n]
>                + h/6 * (kz[i,1] + 2*kz[i,2] + 2*kz[i,3] + kz[i,4] );
>             end do:
>       end do:
>       ### End of RK4 for (z_1, z_2).
>       ### Check of boundary condition at x = b.
>       if (abs(y[1,N]-beta)<h^4) then break; end if;
>       ### Newton's definition for initial slopes.
>       s[M+1] := s[M] - (y[1,N] - beta)/z[1,N]:
>    end do:
```

End of loop for IVP's
Plots of iterations.

```
>    PPP:=seq(plotpoints[r],r=0..M):
>    display(PPP,view=[1..2,1.9..3.5]);
```

Comparison with analytic solution $y(x) = x + 1/x$.

```
>    display(plot(x+1/x, x=1..3, color=red, view=[1..2,1.9..3.5],
>       thickness=2,labels=["x","y(x)"]),{plotpoints[M]});
```

3. Nonlinear shooting method for Neuman boundary problems.

Our interest now is in boundary value problems

$$\left.\begin{array}{ll} y''(x) = f(x, y(x), y'(x)) & a \le x \le b \\ \text{non-Dirichlet boundary conditions at } a, b & \end{array}\right\} \text{(IVP)}\,. \tag{3.1}$$

Let us assume that the boundary conditions have the form

$$y'(a) = \gamma \qquad y(b) = \beta\,. \tag{3.2}$$

To solve this problem, we proceed as in the previous section and consider the sequence of initial value problems

$$\left.\begin{array}{l} y_k''(x) = f(x, y_k(x), y_k'(x)) \quad a \le x \\ y_k(a) = s_k \\ y_k'(a) = \gamma \end{array}\right\}, \tag{3.3}$$

together with

$$y_k(b) - \beta = 0\,. \tag{3.4}$$

Note that now the parameter s_k now specifies $y_k(a)$, whereas in the case of Dirichlet conditions discussed in the prrevios Section it specified the derivative $y_k'(a)$.

The parameters s_k can be iteratively chosen using e.g. the secant method or Newton's method. For the secant method, the s_k are taken as in eq. (2.5). For the Newton method, they follow eq (2.9). As for the sequence of auxiliary initial value problems, we introduce the variable

$$z_s(x) := \frac{dy_s(x)}{ds}$$

and proceed as for Dirichlet boundary conditions. This leads to the sequence of auxiliary initial value problems

$$\left.\begin{array}{l} z_k''(x) = \dfrac{\partial f}{\partial y_k}\, z_k + \dfrac{\partial f}{\partial y_k'}\, z_k' \quad a \le x \\ z_k(a) = 1 \\ z_k'(a) = 0 \end{array}\right\}. \tag{3.5}$$

As compared to the case of Dirichlet boundary conditions discussed in Section 2, see eqs. (2.8), the initial conditions for $z_k(x)$ have changed.

We collect these results in the second row of the table in the next page. The first row stands for Dirichlet boundary conditions. Rows third and fourth correspond to other choices of Neumann boundary conditions.

Boundary conditions for y	Initial conditions for y_k	Secant iteration	Initial conditions for z_k	Newton's teration
$y(a) = \alpha$ $y(b) = \beta$	$y_k(a) = \alpha$ $y'_k(a) = s_k$	$s_{k+2} = s_{k+1} - \left[y_{k+1}(b) - \beta\right] \dfrac{s_{k+1} - s_k}{y_{k+1}(b) - y_k(b)}$	$z_k(a) = 0$ $z'_k(a) = 1$	$s_{k+1} = s_k - \dfrac{y_k(b) - \beta}{z_k(b)}$
$y'(a) = \alpha$ $y(b) = \beta$	$y_k(a) = s_k$ $y'_k(a) = \alpha$		$z_k(a) = 1$ $z'_k(a) = 0$	
$y(a) = \alpha$ $y'(b) = \beta$	$y_k(a) = \alpha$ $y'_k(a) = s_k$	$s_{k+2} = s_{k+1} - \left[y'_{k+1}(b) - \beta\right] \dfrac{s_{k+1} - s_k}{y'_{k+1}(b) - y'_k(b)}$	$z_k(a) = 0$ $z'_k(a) = 1$	$s_{k+1} = s_k - \dfrac{y'_k(b) - \beta}{z'_k(b)}$
$y'(a) = \alpha$ $y'(b) = \beta$	$y_k(a) = s_k$ $y'_k(a) = \alpha$		$z_k(a) = 1$ $z'_k(a) = 0$	

Project 3.1. The boundary value problem for the pendulum.

Consider the boundary problem

$$\left.\begin{aligned} \theta''(x) + \sin\theta(x) &= 0 \qquad 0 \le x \le 2\pi \\ \theta(0) &= 0.7 \\ \theta(2\pi) &= 0.7 \end{aligned}\right\} . \tag{3.6}$$

This is the pendulum's differential equation with boundary conditions instead of initial conditions. In the notation of eq. (1.1), the function f reads $f(x,\theta,\theta') = -\sin\theta$. It follows that $f_\theta = -\cos\theta$, which is not positive for all θ, so Theorem 1.1 does not apply and it does not provide information on the uniqueness of the solution. In what follows we solve this differential problem using nonlinear shooting and show that the solution is not unique. In fact, we provide three different solutions.

First we note that multiplying the differential equation with $\theta'(x)$ and integrating over x, we obtain

$$\frac{1}{2}\,(\theta')^2 - \cos\theta = \text{const} := E\,. \tag{3.7}$$

Every solution then has then a constant of integration E, which we will call 'energy'. Different energies will correspond to different solutions.

Next we remark that if $\omega(x)$ is a solution of the equation $\omega''(x) + \sin\omega(x) = 0$ in the half interval $[\pi, 2\pi]$, its **symmetric extension**

$$\theta(x) = \begin{cases} \omega(\pi - x) & 0 \le x \le \pi \\ \omega(x) & \pi \le x \le 2\pi \end{cases}$$

with respect to $x = \pi$ is a solution to the same equation in the whole interval $[0, 2\pi]$. Since the boundary conditions in (3.6) are symmetric with respect to $x = \pi$, the problem can be solved in two steps as follows:

(1) Solve the boundary value problem

$$\left.\begin{aligned} \omega''(x) + \sin\omega(x) &= 0 \qquad \pi \le x \le 2\pi \\ \omega'(\pi) &= 0 \\ \omega(2\pi) &= 0.7 \end{aligned}\right\} \tag{3.8}$$

(2) **Symmetrically extend** the resulting solution to $[0, 2\pi]$.

Note that the boundary value problem (3.8) is of the type in eqs. (3.1)-(3.2). To solve it, we must solve the corresponding sequence of initial value problems (3.3) together with equation (3.4). That is, we must solve

$$\left.\begin{aligned} \omega_k''(x) + \sin\omega_k(x) &= 0 \qquad \pi \le x \\ \omega_k(\pi) &= s_k \\ \omega_k'(\pi) &= 0 \end{aligned}\right\} \tag{3.9}$$

and

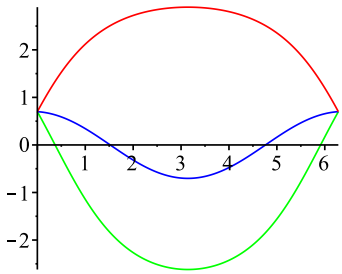
$$\omega_k(2\pi) - 0.7 = 0\,.$$

So let us make an ansatz for s_0 and say that a solution $\omega(x) = \omega_{K_1}(x)$ to (3.8) is reached after $K_1 + 1$ iterations. Since the energy is constant, it can be computed at $x = \pi$. Using that $\omega_k'(\pi) = 0$ for all k, we obtain

$$E_{K_1} = E_{K_1}(\pi) = -\cos\omega_{K_1}(\pi)\,.$$

The idea is to play with s_0 so that different choices for s_0 lead to different solutions $(\omega_K,\, s_K)$. One would naïvely expect to obtain different solutions by starting from separated values of s_0.

The Maple code in Appendix 3.1 implements this analysis. For $s_0 = 0.5$, a solution is reached in 4 iterations. This is the solution in blue in the figure below. For $s_0 = 3.0$, the solution in red is reached after 5 iterations. Finally, for $s_0 = -3.0$, the green solution is reached in 7 iterations.



Three different solutions $\theta(x)$ to the IVP problem (3.6).

The 'energies' corresponding to these solutions are collected in the following table, produced by the `printf` at the end of the code:

Solution	s[0]	Energy
Blue	0.5	-0.7703863786
Red	3.0	0.9800800693
Green	-3.0	0.8755489488

Note that, as anticipated, every solution has a different 'energy'. We remark that we are calling 'energy' to the constant of integration in eq. (3.7) but that strictly speaking it does not have the meaning of a physical energy, so that there is no problem with negative 'energies'.

Comment. The following shows, though, that things are not as simple as they look at first sight. Take $s_0 = 4.2$ and run the Maple code. After 18 iterations, one obtains the solution in red. Change now s_0 by a very small amount. For example, take $s_0 = 4.2 + h^3 \approx 4.20003$. One would expect to obtain the same solution. However, after 10^3 Newton iterations, nonlinear shooting for $\omega(x)$ does not converge.

Appendix 3.1. Maple program for Project 3.1 (RK4 + Newton).

```
> restart: with(plots):
```

Problem data.

```
> a:=evalf(Pi): b:=evalf(2*Pi): alpha:=0.7: beta:=0.7:
> f:=(x,y1,y2) -> array(1..2,[y2,-sin(y1)]):
> g:=(x,y1,y2,z1,z2) -> array(1..2,[z2,-cos(y1)*z1]):
```

Split domain in N subintervals of equal length h:

```
> N:=100:
> x:=array(0..N):
>    x[0]:=a: x[N]:=b:
>    h:=evalf((x[N]-x[0])/N):
>    for n from 0 to N do x[n]:=x[0]+n*h: end do:
```

Organize solutions (y_1, y_2) and (z_1, z_2) in arrays

```
> y:=array(1..2,0..N):
>    y[2,0]:=0:
> z:=array(1..2,0..N):
>    z[1,0]:=1: z[2,0]:=0:
```

Array for energy

```
> E:=array(0..N):
```

M counts Newton iterations. $M_{\max}$ sets upper bound for M.

```
> M:=0: M_max:=100:
```

These are three different choices for s_0.

```
> # s[0]:=0.5:
> # s[0]:=3.:
> s[0]:=-3.:
```

Auxiliary vectors in RK4:

```
> k := array(1..2,1..4):
> kz := array(1..2,1..4):
> for M from 0 to M_max do  ## Loop over Newton's iterations starts
```

RK4 for (y_1, y_2):

```
> y[1,0]:=s[M];
> for n from 0 to N-1 do
>     for i from 1 to 2 do
>         k[i,1] := f(x[n],y[1,n],y[2,n])[i]:
>     end do:
>     for i from 1 to 2 do
>         k[i,2] := f(x[n]+h/2,
>                     y[1,n]+h/2*k[1,1], y[2,n]+h/2*k[2,1])[i]:
>     end do:
>     for i from 1 to 2 do
>         k[i,3] := f(x[n]+h/2,
>                     y[1,n]+h/2*k[1,2], y[2,n]+h/2*k[2,2])[i]:
>     end do;
>     for i from 1 to 2 do
>         k[i,4] := f(x[n]+h,
>                     y[1,n]+h*k[1,3], y[2,n]+h*k[2,3])[i]:
>     end do:
>     for i from 1 to 2 do
>         y[i,n+1] := y[i,n]
>                   + h/6 * (k[i,1] + 2*k[i,2] + 2*k[i,3] + k[i,4]):
>     end do:
> end do:
```

Create plots for every iterate:

```
> plotpoints[M] := plot({seq([x[n],y[1,n]],n=0..N)},
>    color=COLOR(RGB, rand()/10^12, rand()/10^12, rand()/10^12),
>    style=point):
```

Create plots for every iterate's energy:

```
> for j from 0 to N do E[j]:=1/2*(y[2,j])^2 -cos(y[1,j]) end do;
```

We take for the energy the average of the energies E_n at x_n.
The graphics obtained (see below) show that E_n does not change with x_n

```
> E_av[M]:=1/N*sum(E[q],q=0..N);
> plotenergy[M] := plot({seq([x[p],E[p]],p=0..N)},
>    color=COLOR(RGB, rand()/10^12, rand()/10^12, rand()/10^12),
>    style=point):
```

Print out s_k and the value of the solution at $x = b$ for every iterate:

```
> if (M=0) then
>    printf("   Iteration     y_k(a)=s[k]         y(b)          Energy\n");
>    printf(" ----------  --------------   --------------   -------------\n");
> end if;
```

RK4 for (z_1, z_2):

```
> for n from 0 to N-1 do
>    for i from 1 to 2 do
>       kz[i,1] := g(x[n],y[1,n],y[2,n],z[1,n],z[2,n])[i]:
>    end do:
>    for i from 1 to 2 do
>       kz[i,2] := g(x[n]+h/2,y[1,n],y[2,n],
>                    z[1,n]+h/2*kz[1,1], z[2,n]+h/2*kz[2,1])[i]:
>    end do:
>    for i from 1 to 2 do
>       kz[i,3] := g(x[n]+h/2,y[1,n],y[2,n],
>                    z[1,n]+h/2*kz[1,2], z[2,n]+h/2*kz[2,2])[i]:
>    end do;
>    for i from 1 to 2 do
>       kz[i,4] := g(x[n]+h,y[1,n],y[2,n],
>                    z[1,n]+h*kz[1,3], z[2,n]+h*kz[2,3])[i]:
>    end do:
>    for i from 1 to 2 do
>    z[i,n+1] := z[i,n] + h/6 * (kz[i,1] + 2*kz[i,2] + 2*kz[i,3] + kz[i,4]):
>    end do:
> end do:
```

Newton's choice of s_k:

```
> if (abs(y[1,N]-beta)<h^4) then break;  end if;
>    s[M+1] := s[M] - (y[1,N] - beta)/z[1,N]:
> end do:
```

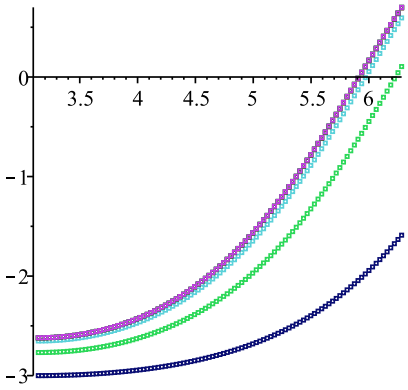
Loop over Newton's iterations ends

Output of the `printf` command

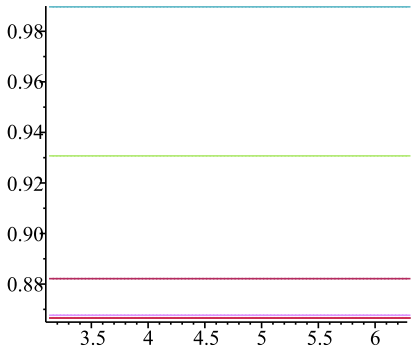
Iteration	y(a)=s[k]	y(b)	Energy
0	-3.0000000000	-1.5890492410	0.9998924222
1	-2.7680514430	0.1049608761	0.9403513548
2	-2.6518689410	0.5937544687	0.8912874822
3	-2.6219451230	0.6929498298	0.8766742019
4	-2.6197634160	0.6998194810	0.8755779027
5	-2.6197071690	0.6999959572	0.8755495824
6	-2.6197059090	0.6999999146	0.8755489488

Plots of iterates for $\omega(x)$ and their energies as functions of x_n

```
> PlotsOm:=seq(plotpoints[r],r=0..M): display(PlotOm):
> PlotsE:=seq(plotenergy[r],r=0..M): display(PlotE):
```



Iterates $\omega_k(x),\ k = 0, \ldots, 6$



Energies $E_k(x)$ *of iterates*

The energy of every iterate does not depend on x_n
Construction of $\theta(x)$ through symmetric extension to $[0, 2\pi]$:

```
> for n from 0 to N do
>    theta[n] := y[1,n]:
>    theta[-n] := y[1,n]:
> end do:
```

Plot of the extended solution. Run the program for different values of $s[0] = s_0$
and save the solutions. Take for example $s_0 = 0.5, 3.0, -3.0$

```
> # Sol[1]:=plot({seq([n*h+evalf(Pi),theta[n]],n=-N..N)},color=blue):
> #   E1:= E_av[M]:
> # Sol[2]:=plot({seq([n*h+evalf(Pi),theta[n]],n=-N..N)},color=red):
> #   E2:= E_av[M]:
> Sol[3]:=plot({seq([n*h+evalf(Pi),theta[n]],n=-N..N)},color=green):
>   E3:= E_av[M]:
```

Display the solutions and print a summary table

```
> display({Sol[1],Sol[2],Sol[3]});
> printf("   Solution       s[0]     Energy\n");
> printf("   Blue           0.5      "), printf("%14.10f\n",E1);
> printf("   Red            3.0      "), printf("%14.10f\n",E2);
> printf("   Green          -3.0     "), printf("%14.10f\n",E3);
```

4. Finite differences. General ideas.

The two basic finite difference operators are the forward finite difference Δ_+ and the backward finite difference Δ_-, whose action on a function $f(x)$ is defined for a mesh size h as

$$\text{Forward:} \quad \Delta_+ f(x) = f(x+h) - f(x) \tag{4.1}$$

$$\text{Backward:} \quad \Delta_- f(x) = f(x) - f(x-h)\,. \tag{4.2}$$

Linear combinations of products of powers of them can be formed to produce new finite difference operators. One such combination of particular interest is the sum, known as the centered finite difference

$$\text{Centered:} \quad \Delta_0 f(x) = (\Delta_+ + \Delta_-)\, f(x) = f(x+h) - f(x-h)\,. \tag{4.3}$$

Expansion of $f(x \pm h)$ in powers of h yields

$$\begin{aligned} f(x \pm h) &= f(x) \pm h\, f'(x) + \frac{1}{2!}\, h^2 f''(x) \pm \frac{1}{3!}\, h^3 f'''(x) + \cdots \\ &= \left[1 + \Big(\pm\, h\, \frac{d}{dx}\Big) + \frac{1}{2!}\Big(\pm\, h\, \frac{d}{dx}\Big)^2 + \frac{1}{3!}\Big(\pm\, h\, \frac{d}{dx}\Big)^3 + \cdots\right] f(x) \\ &= \left[\exp\Big(\pm\, h\, \frac{d}{dx}\Big)\right] f(x)\,. \end{aligned} \tag{4.4}$$

The finite difference operators $\Delta_\pm$ can then be written in terms of the derivative $\frac{d}{dx}$ as

$$\Delta_\pm = \pm\left[\exp\Big(\pm\, h\, \frac{d}{dx}\Big) - 1\right].$$

Formal inversion leads to

$$h\, \frac{d}{dx} = \pm \ln\left(1 \pm \Delta_\pm\right) = \Delta_\pm \mp \frac{(\Delta_\pm)^2}{2} + \frac{(\Delta_\pm)^3}{3} + \cdots \tag{4.5}$$

This equation can be used to obtain approximations for the first derivative with decreasing truncation error. For example, for Δ_+, by including more an more terms on the right-hand side of eq. (4.5), we obtain

$$\begin{aligned} \frac{1}{h}\, \Delta_+ f(x) &= \frac{f(x+h) - f(x)}{h} = f'(x) + \frac{h}{2}\, f''(x) + \cdots \\ \frac{1}{h}\left(\Delta_+ - \frac{\Delta_+^2}{2}\right) f(x) &= -\frac{f(x+2h) - 4f(x+h) + 3f(x)}{2h} = f'(x) - \frac{h^2}{3}\, f'''(x) + \cdots \\ \frac{1}{h}\left(\Delta_+ - \frac{\Delta_+^2}{2} + \frac{\Delta_+^3}{3}\right) f(x) &= \frac{2\, f(x+3h) - 9\, f(x+2h) + 18\, f(x+h) - 11 f(x)}{6h} \\ &= f'(x) + \frac{h^3}{4}\, f''''(x) + \cdots\,, \\ &\vdots \end{aligned}$$

where we have explicitly written the term carrying the truncation error. To achieve truncation error h^n, we need to go up to the power Δ_+^n. An analaogues analysis can be carried for Δ_-. In this case, the approximations for $\frac{d}{dx}$ are most easily obtainded by replacing h with $-h$ in the expressions above, since

$$-\frac{1}{h}\,\ln(1-\Delta_-) = \left[\frac{1}{h}\,\ln(1+\Delta_+)\right]_{h\to -h}.$$

Thus the approximation with truncation error h^2 reads

$$\frac{1}{h}\left(\Delta_- + \frac{\Delta_-^2}{2}\right) f(x) = \frac{3f(x) - 4f(x-h) + f(x-2h)}{2h} = f'(x) - \frac{h^2}{3}\,f'''(x) + \cdots$$

A similar analysis can be performed for the centered finite difference Δ_0. From the Taylor series (4.4) it follows that

$$\Delta_0 = 2\,\sinh\left(h\,\frac{d}{dx}\right),$$

hence

$$h\,\frac{d}{dx} = \operatorname{arg\,sinh}\left(\frac{\Delta_0}{2}\right) = \frac{1}{2}\left(\Delta_0 - \frac{1}{24}\,\Delta_0^3 + \frac{3}{640}\,\Delta_0^5 + \cdots\right).$$

We now have

$$\frac{1}{2h}\,\Delta_0 f(x) = \frac{f(x+h) - f(x-h)}{2h} = f'(x) + \frac{h^2}{6}\,f'''(x) + \cdots \tag{4.6}$$

$$\begin{aligned}\frac{1}{2h}\left(\Delta_0 - \frac{\Delta_0^3}{24}\right) f(x) &= -\,\frac{f(x+3h) - 27\,f(x-h) + 27\,f(x-h) - f(x-3h)}{48\,h} \\ &= f'(x) - \frac{9h^4}{5}\,f''''(x) + \cdots,\end{aligned}$$

etc. Derivatives of higher order can be approximated in terms of finite differences as follows.

Exercise 4.1. Approximate **the second derivative $f''(x)$** in terms of $f(x+h)$, $f(x)$ and $f(x-h)$.

Solution. We must find a linear combination

$$\Delta_2 f(x) = a\,f(x+h) + b\,f(x) + c\,f(x-h)$$

such that, for suitable values of a, b and c, it is equal to $f''(x)$ with some truncation error. Expanding the right-hand side in powers of h, we write

$$\begin{aligned}\Delta_2 f(x) = (a+b+c)\,f(x) + h\,(a-c)\,f'(x) + \frac{h^2}{2!}\,(a+c)\,f''(x) \\ + \frac{h^3}{3!}\,(a-c)\,f'''(x) + \frac{h^3}{4!}\,(a+c)\,f''''(x) + \cdots\end{aligned} \tag{4.7}$$

The coeffcients of $f(x)$ and $f'(x)$ must vanish, and that of $f''(x)$ must be equal to one, that is

$$a+b+c=0 \qquad h\,(a-c)=0 \qquad \frac{h^2}{2}\,(a+c)=1\,.$$

The only solution to these equations is

$$a=\frac{1}{h^2} \qquad b=-\frac{2}{h^2} \qquad c=\frac{1}{h^2}\,.$$

Subsituting back in (4.7), we obtain

$$\frac{1}{h^2}\left[\,f(x+h)-2\,f(x)+f(x-h)\,\right]=f''(x)+\frac{h^2}{12}\,f''''(x)+\cdots\,, \tag{4.8}$$

which gives an approximation with truncation error h^2. It is trivial to see that the finite difference operator Δ_2 is the square of the Δ_0^2 of the centered finite difference.

Exercise 4.2. Approximate $f''(x)$ with $f(x)$, $f(x-h)$ and $f(x-2h)$.

Solution. In this case we must look for values $\tilde{a}_2$, $\tilde{b}_2$ and $\tilde{c}_2$ such that

$$\tilde{\Delta}_2 f(x) := \tilde{a}_2\,f(x)+\tilde{b}_2\,f(x-h)+\tilde{c}_2\,f(x-2h)$$

is equal to $f''(x)$ with some truncation error. Expanding $\tilde{\Delta}_2 f(x)$ in powers of h,

$$\begin{aligned}\tilde{\Delta}_2 f(x) = {}& (\tilde{a}_2+\tilde{b}_2+\tilde{c}_2)\,f(x)-h\,(\tilde{b}_2+2\tilde{c}_2)\,f'(x)\\ &+\frac{h^2}{2}\,(\tilde{b}_2+4\tilde{c}_2)\,f''(x)+\frac{h^3}{6}\,(\tilde{b}_2+8\tilde{c}_2)\,f'''(x)+\cdots\,,\end{aligned} \tag{4.9}$$

and identifying coefficients, we arrive at

$$\tilde{a}_2+\tilde{b}_2+\tilde{c}_2=0 \qquad -h\,(\tilde{b}_2+2\tilde{c}_2)=0 \qquad \frac{h^2}{2}\,(\tilde{b}_2+4\tilde{c}_2)=1\,.$$

Its only solution is

$$\tilde{a}_2=\frac{1}{h^2} \qquad \tilde{b}_2=-\frac{2}{h^2} \qquad \tilde{c}_2=\frac{1}{h^2}\,.$$

Taking these to eq. (4.9), we now have

$$f''(x)=\frac{1}{h^2}\left[\,f(x)-2\,f(x-h)+f(x-2h)\,\right]+h\,f'''(x)+\cdots\,, \tag{4.10}$$

which has truncation error h. In this case, $\tilde{\Delta}_2=\Delta_-^2$. One may improve the truncation error by including more terms in the finite difference operator, as the following exercise shows.

Exercise 4.3. Approximating $f''(x)$ with truncation error h^2 in terms of $f(x)$, $f(x+h)$, $f(x+2h)$ and $f(x+3h)$.

Solution. The finite difference operator will be

$$\begin{aligned}\bar{\Delta}_2 f(x) &= \bar{a}_2\, f(x) + \bar{b}_2\, f(x-h) + \bar{c}_2\, f(x-2h) + \bar{d}_2\, f(x-3h)\\ &= \text{expanding in powers of } h\\ &= \left(\bar{a}_2 + \bar{b}_2 + \bar{c}_2 + \bar{d}_2\right) f(x) - h\left(\bar{b}_2 + 2\,\bar{c}_2 + 3\,\bar{d}_2\right) f'(x)\\ &+ \frac{h^2}{2!}\left(\bar{b}_2 + 4\,\bar{c}_2 + 9\,\bar{d}_2\right) f''(x) - \frac{h^3}{3!}\left(\bar{b}_2 + 8\,\bar{c}_2 + 27\,\bar{d}_2\right) f'''(x)\\ &+ \frac{h^4}{4!}\left(\bar{b}_2 + 16\,\bar{c}_2 + 81\,\bar{d}_2\right) f''''(x) + \cdots\end{aligned}$$

Demanding

$$\left.\begin{aligned}\bar{a}_2 + \bar{b}_2 + \bar{c}_2 + \bar{d}_2 &= 0\\ \bar{b}_2 + 2\,\bar{c}_2 + 3\,\bar{d}_2 &= 0\\ \frac{h^2}{2!}\left(\bar{b}_2 + 4\,\bar{c}_2 + 9\,\bar{d}_2\right) &= 1\\ \bar{b}_2 + 8\,\bar{c}_2 + 27\,\bar{d}_2 &= 0\end{aligned}\right\} \quad \Rightarrow \quad \bar{a}_2 = -\frac{8}{h^2} \quad \bar{b}_2 = \frac{5}{h^2} \quad \bar{c}_2 = \frac{4}{h^2} \quad \bar{d}_2 = -\frac{1}{h^2}\,,$$

we have

$$f''(x) = -\frac{1}{h^2}\left[\,8\,f(x) - 5\,f(x-h) - 4\,f(x-2h) + f(x-3h)\,\right] - \frac{h^2}{2}\,f''''(x) + \cdots$$

With respect to exercise 2, the truncation error has improved but the finite difference operator requires, in addition, from $f(x-3h)$.

It is clear that to approximate the n-th derivative $f^{(n)}(x)$ one needs at least $f(x+k_i h)$ at $n+1$ different points $x+k_1 h, \ldots, x+k_{n+1} h$.

5. Finite difference methods for boundary value problems.

The presentation here closely follows R. J. Le Veque, "Finite difference methods for ordinary and partial differential equations, steady states and time dependent problems" (SIAM 2007).

We wish to solve the boundary value problem

$$\left.\begin{array}{ll} y''(x) = f(x, y, y') & a \le x \le b \\ y(a) = \alpha & \\ y(b) = \beta & \end{array}\right\}, \tag{5.1}$$

using finite differences., under the assumption that f, f_y and $f_{y'}$ are continuous in the domain

$$D = \{(x, y, y') : \ a \le x \le b, \ -\infty < y, y' < \infty\}\,.$$

To do this, we partition the domain $[a, b]$ in $N + 1$ subintervals all of length h, so that

$$h = \frac{b-a}{N+1}, \quad x_n := a + nh, \quad n = 0, 1, \ldots, N+1\,,$$

and introduce the notation

$$y_n := y(x_n), \quad n = 0, 1, \ldots, N+1\,.$$

For the first and second derivative we use the finite difference approximations (4.6) and (4.8), namely

$$\begin{aligned} y'_n &= \frac{1}{2h}\,(y_{n+1} - y_{n-1}) + O(h^2) \\ y''_n &= \frac{1}{h^2}\,(y_{n+1} - 2y_n + y_{n-1}) + O(h^2)\,. \end{aligned}$$

This transforms the differential problem (5.1) into a system of algebraic equations

$$\left.\begin{array}{rl} \dfrac{y_{n+1} - 2\,y_n + y_{n-1}}{h^2} - f\Big(x_n, y_n, \dfrac{y_{n+1} - y_{n-1}}{2h}\Big) = 0 & n = 1, \ldots, N \\ y_0 = \alpha & \\ y_{N+1} = \beta & \end{array}\right\}, \tag{5.2}$$

where $h, x_0, x_1, \ldots, x_{N+1}$ are data and $y_1, \ldots, y_N$ are the unknowns. This is in general a system of nonlinear equations. All that remains is to solve it. There is an ample variety of methods in the literature to do this. In Exercise 1 below, the Newton-Raphston method is used to to solve the nonlinear algebraic system that results from the boundary value problem for the pendulum.

Let us briefly review the **Newton-Raphston method**. Consider the system of nonlinear equations

$$\mathcal{F}(Y) = 0 \quad \Leftrightarrow \quad \mathcal{F}_n(y_1, \ldots, y_N) = 0\,. \tag{5.3}$$

The method uses iteration to provide a solution

$$Y = Y^{[0]} + Y^{[1]} + Y^{[2]} + \cdots\,,$$

where $Y^{[0]}$ is an initial ansatz and the contribution $Y^{[k]}$ $(k = 1, 2, \ldots)$ is given in terms of $Y^{[k-1]}$ as the solution of

$$\mathcal{F}\Big(\sum_{r=0}^{k-1} Y^{[r]}\Big) + \mathrm{Jac}_{\mathcal{F}}\Big(\sum_{r=0}^{k-1} Y^{[r]}\Big)\, Y^{[k]} = 0\,, \tag{5.4}$$

with $\mathrm{Jac}_{\mathcal{F}}\big(Z\big)$ the Jacobian matrix of $\mathcal{F}$ at Z. Iteration ends at a $k_{\max}$, for which a required tolerance ϵ is met, i.e.

$$\Big\|\mathcal{F}\Big(\sum_{r=0}^{k_{\max}} Y^{[r]}\Big)\Big\|_{\infty} := \max_{1\le n\le N}\Big|\mathcal{F}_n\Big(\sum_{r=0}^{k_{\max}} Y^{[r]}\Big)\Big| < \epsilon\,. \tag{5.5}$$

We note that eq. (5.4) is linear and that its solution $Y^{[k]}$ is unique for non-singular $\mathrm{Jac}_{\mathcal{F}}\big(Y^{[k-1]}\big)$. To find it, there are various methods. We will use LU decomposition in Doolittle's form. This consists of three steps:

i) Writing the Jacobian as the product of a lower triangular matrix $L^{[k]}$ (with all its entries in the diagonal equal to one) times an upper triangular matrix $U^{[k]}$,

$$\mathrm{Jac}_{\mathcal{F}}\Big(\sum_{r=0}^{k-1} Y^{[r]}\Big) = L^{[k]}U^{[k]}, \quad \text{with} \quad \begin{cases} \ell^{[k]}_{ii} = 1, & \ell^{[k]}_{i>j} = 0 \\ u^{[k]}_{i\le j} = 0 \end{cases}\,.$$

This decomposition is unique, so that $L^{[k]}$ and $U^{[k]}$ are uniquely determined.

ii) Solving the equation $L^{[k]}Z^{[k]} = \mathrm{Jac}_{\mathcal{F}}\Big(\sum_{r=0}^{k-1} Y^{[r]}\Big)$ for $Z^{[k]}$ through forward substitution.

iii) Solving the equation $U^{[k]}Y^{[k]} = Z^{[k]}$ for $Y^{[k]}$ through backward substitution.

To start the Newton-Raphston method, an initial ansatz $Y^{[0]}$ is needed. The method can be proved to converge if the ansatz is sufficiently close to a solution of $\mathcal{F}(Y) = 0$. The Newton-Kantorovich theorem (see Appendix 5.3 below) discusses how close the ansatz must be for the method to converge. Since the equation $\mathcal{F}(Y) = 0$ is nonlinear, it may have more than one solution. Different ansatze for $Y^{[0]}$ may (but not necessarily) lead to different solutions. The choice of the inital guess $Y^{[0]}$ is thus the key point for finding different solutions. In a sense the situatin is analogous to the problem of the guess for the initial slope in nonlinear shooting. In the project below we illuistrate all this.

Project 5.1. The boundary value problem for the pendulum.

The purpose of this project is to solve the boundary value problem for the pendulum (3.6).

We split the domain $[0, 2\pi]$ in $N+1 = 100$ subintervals, so that $h = 2\pi/100$. The system of equations (5.2) takes the form

$$\left.\begin{aligned} \theta_0 &= 0.7 \\ \mathcal{F}_n(\theta_1, \ldots, \theta_N) &= 0 \qquad n = 1, \ldots, N \\ \theta_{N+1} &= 0.7 \end{aligned}\right\}, \tag{5.6}$$

where the functions $\mathcal{F}_n$ are given by

$$\mathcal{F}_n := \frac{\theta_{n-1} - 2\,\theta_n + \theta_{n+1}}{h^2} + \sin\theta_n\,. \tag{5.7}$$

In this case the Jacobian matrix is

$$\begin{aligned} \mathrm{Jac}_{\mathcal{F}}(\Theta) &= \left[\frac{\partial \mathcal{F}_n}{\partial \theta_m}\right](\theta_1, \ldots, \theta_N) \\ &= \begin{pmatrix} \cos\theta_1 - \frac{2}{h^2} & \frac{1}{h^2} & 0 & \ldots & 0 & 0 & 0 \\ \frac{1}{h^2} & \cos\theta_2 - \frac{2}{h^2} & \frac{1}{h^2} & \ldots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & \frac{1}{h^2} & \cos\theta_{N-1} - \frac{2}{h^2} & \frac{1}{h^2} \\ 0 & 0 & 0 & \ldots & 0 & \frac{1}{h^2} & \cos\theta_N - \frac{2}{h^2} \end{pmatrix}. \end{aligned} \tag{5.8}$$

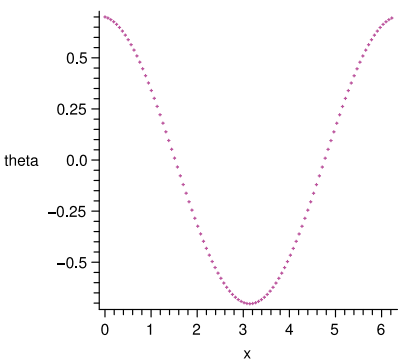
Since the approximation used for the second derivative [see eq. (4.8)] has truncation error h^2, we require precision h in the solution, so that ϵ in (5.5) is $\epsilon = h^2$.

The Maple code in Appendix 5. 1 solves the algebraic problem (5.6)-(5.7) using the Newton-Raphston method. We note that the Jacobian (5.8) is tridiagonal and that for matrices of this type LU decomposition is very simpe. In fact, formuli for L and U in terms of the the Jacobain matrix entries can be directly written. In the code in Appendix 5.2, we however use the general expression for LU deocmposition so that the code is usable for arbitrary non-tridiagonal Jacobians.

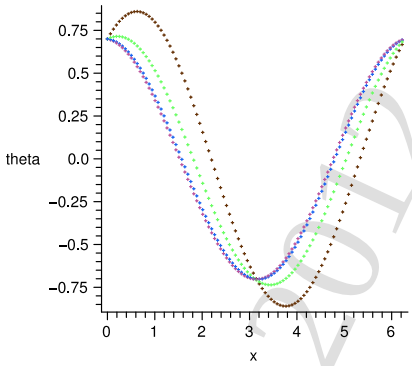
Let us first consider the ansatz

$$\text{Ansatz 1:} \quad \theta_n^{[0]} = 0.7\,\cos(x_n) + 0.5\,\sin(x_n)\,. \tag{5.9}$$

In this case, 4 iterations are needed. The graphs for the solution $\theta(x) = \theta^{[3]}(x)$ and for the four iterations $\theta^{[k]}(x)$ leading to it are displayed in the following two figures:



Solution $\theta(x) = \theta^{[3]}(x)$

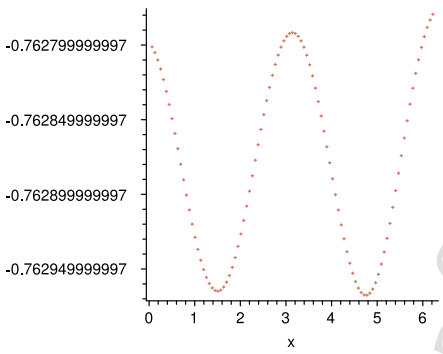


$\theta^{[k]}(x)$ for $k = 0, 1, 2, 3$

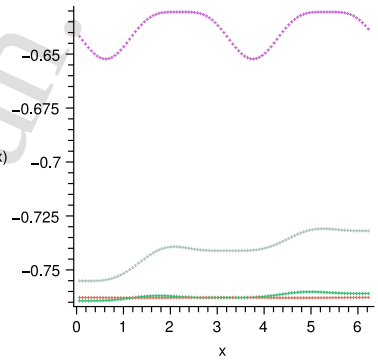
Their energies $E^{[k]}(x_n) = E_n^{[k]}$, as functions of x, can be defined as

$$E_n^{[k]} := \frac{1}{2}\left(\frac{\theta_{n+1}^{[k]} - \theta_{n-1}^{[k]}}{2h}\right)^2 - \cos(\theta_n^{[k]})$$

and graphically look like



Energy of solution $\theta(x) = \theta^{[3]}(x)$



$E^{[k]}(x)$ for $k = 0, 1, 2, 3$

Note that $E_n^{[k]}$ is constant if the corresponding $\theta_n^{[k]}$ is a solution. In other words, as $\theta_n^{[k]}$ gets closer to the solution, $E_n^{[k]}$ approaches a constant. We may estimate the solution's energy as the average

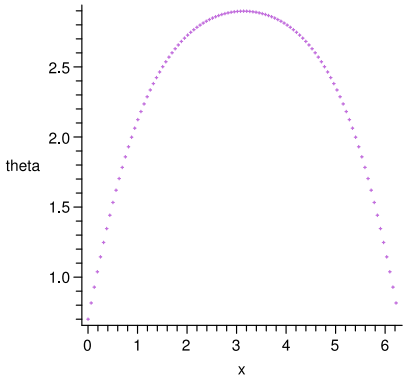
$$E = \frac{1}{N}\sum_{n=1}^{N} E_n^{[k_{\max}]},$$

for the solution (=last iterate). This gives for the ansatz being considered ($k_{\max} = 3$) an energy of `-0.7628804197`.

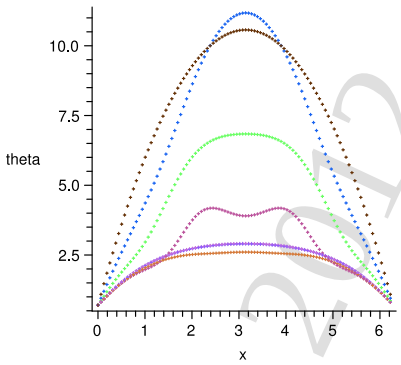
We now move on to consider a different ansatz for $\theta_n^{[0]}$, namely

$$\text{Ansatz 2:} \quad \theta_n^{[0]} = 0.7 + x_n(2\pi - x_n)\,. \tag{5.10}$$

In this case we obtain a different solution. It is reached in `7 iterations`. The figure on the left below depicts the solution $\theta(x) = \theta^{[6]}(x)$, whereas that on the right the seven Newton-Raphston iterations:

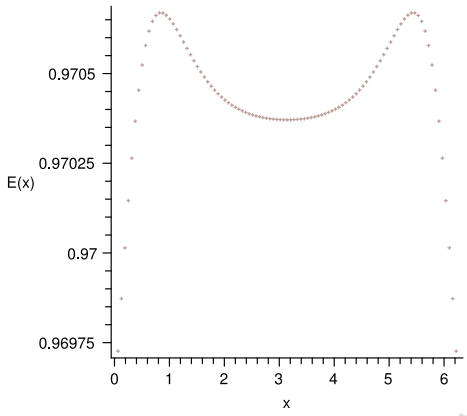


Solution $\theta(x)$

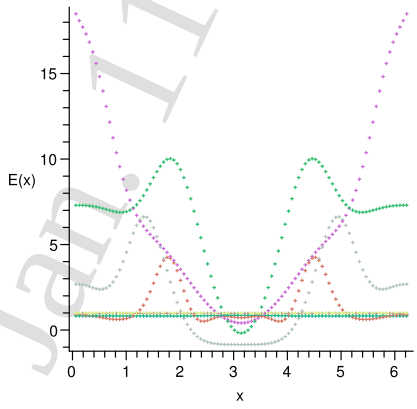


$\theta^{[k]}(x_n),\ k = 0, 1, \ldots, 6,$

The energy graphs look also very different to those of ansatz 1:



Energy of solution $\theta(x)$



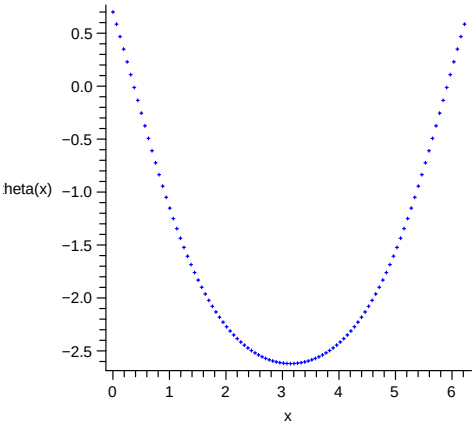
$E^{[k]}(x_n)\ , k = 0, 1, \ldots, 6$

The energy now takes the value 0.9704366283. We thus see that the boundary value problem (3.6) has at least two solutions. This is not a suprise since the hypotheses in Theorem 1.1 do not hold and there is no reason for the solution to be unique.

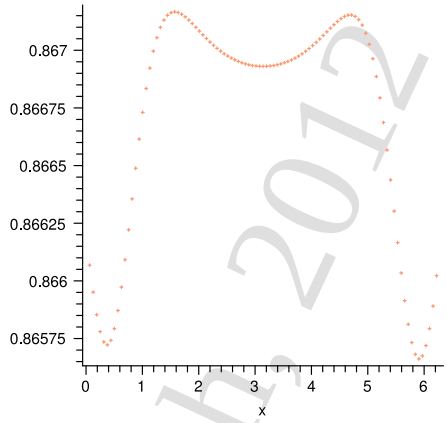
Let us produce yet another solution. Take

$$\text{Ansatz 3:}\quad \theta_n^{[0]} = 0.7 + 100\, x_n(2\pi - x_n)\,. \tag{5.11}$$

Note the coeffcient 100 in front of the second term as compared to Case 2 above. The solution is now reached in 15 iterations and its energy is 0.8667053077. The solution's plot is as in the figure on the left below, and its energy x-dependence is as in the figure on the right:



Solution $\theta(x) = \theta^{[14]}(x)$



$E^{[14]}(x)$

So far we have obtained three different solutions (with differnet enaergies). We must warn however that, in accordance with the Newton-Kantorovich theorem, not every ansatz for $\theta_n^{[0]}$ leads to a new solution. For example,

$$\text{Ansatz 4:} \quad \theta_n^{[0]} = 0.7 + \sin(x_n)\,. \tag{5.12}$$

leads to the same solution as for ansatz 1, whereas

$$\text{Ansatz 5:} \quad \theta_n^{[0]} = 0.7 + \sin(x_n/2) \tag{5.13}$$

yields the solution of ansatz 2.

Appendix 5.1. Maple code for Project 5.1 (Fin. diff. + Newton-Raphston).

```
> restart:  with(plots):
```

Problem data.

```
> a:=0:  b:=evalf(2*Pi:
> alpha:=0.7:  beta:=0.7:
```

Split domain in $N+1$ subintervals of equal length h.

```
> N:=99:
> x:=array(0..N+1):
>    x[0]:=a:  x[N+1]:=b:
>    h:=(x[N+1]-x[0])/(N+1):
>    for n from 1 to N do x[n]:=x[0]+n*h:  end do:
```

System of equations $F(y) = 0$.

```
> y:=array(0..N+1):
>    y[0]:=alpha:  y[N+1]:=beta:
> F:=array(1..N):
>    F[1] := (u,v,w) -> 1/h^2*(y[0]-2*v+w)+sin(v):
>    for n from 2 to N-1 do
>       F[n] := (u,v,w)  -> 1/h^2*(u-2*v+w)+sin(v):
>    end do:
>    F[N] := (u,v,w) -> 1/h^2*(u-2*v+y[N+1])+sin(v):
```

Ansatz for Newton-Raphston's first iteration.

```
> Z:=array(0..N+1):
>    for n from 0 to N+1 do
>       Z[n] := 0.7*cos(x[n])+0.5*sin(x[n]):          # Case 1
>       # Z[n] := 0.7+2*x[n]*(evalf(Pi)-x[n]):       # Case 2
>       # Z[n] := 0.7+sin(x[n]):                     # Case 4
>       # Z[n] := 0.7+sin(x[n]/2):                   # Case 5
>    end do:
```

Solution is recursively constructed as $Z + Z_1$.
Set a value for Z_1 to start iteration test.

```
> Z:=array(1..N):
> for n from 1 to N do Z1[n]:=h:  end do:
```

N_{iter} counts NR iterations.

```
> N_iter:=0:
```

Loop for Newton-raphston iterations.

```
> while (max(seq(abs(Z1[n]),n=1..N)) > h^2) do
>    ### Create plot data for each iterate.
>    plotpoints[N_iter] := plot( {seq([x[n],Z[n]],n=0..N)},
>       color=COLOR(RGB, rand()/10^12, rand()/10^12, rand()/10^12),
>       style=point, symbol=cross,symbolsize=4 ):
>    ### Calculation of energy as function of x and of its average.
>    E:=array(1..N):
>    for n from 1 to N do
>       E[n]:=1/2*((Z[n+1]-Z[n-1])/2/h)^2-cos(Z[n]):
>    end do:
>    E_av[N_iter]:=sum(E[r],r=1..N)/N:
>    ### Create plot data for every iterate's energy function.
>    energy[N_iter]:=plot({seq([x[i],E[i]],i=1..N)},
>       color=COLOR(RGB, rand()/10^12, rand()/10^12, rand()/10^12),
>       style=point, symbol=cross,symbolsize=4 ):
>    ### Jacobian of F(y).
>    Jac:=array(1..N,1..N):
>    for n from 1 to N do
>       for m from 1 to N do Jac[n,m]:=0:
>       end do:
>    end do:
>    for n from 1 to N do Jac[n,n]:=-2/h^2 + cos(Z[n]):
>    end do:
>    for n from 1 to N-1 do Jac[n,n+1]:=1/h^2:
>    end do:
>    for n from 2 to N do Jac[n,n-1]:=1/h^2:
>    end do:
>    ### Doolittle's LU factorization: Jac_F = LU, with L_ii = 1.
>    A:=Jac:
>    L:=array(1..N,1..N):
>    U:=array(1..N,1..N):
>    ### Trivial entries for L and U.
>    for i from 1 to N do L[i,i]:=1:
>       for j from i+1 to N do L[i,j]:=0; end do:
>    end do:
>    for j from 1 to N do
>       for i from 1 to j-1 do U[i,j]:=0; end do:
>    end do:
>    ### Calculation of nontrivial entries in L and U.
```

```
>     for k from 1 to N do
>        U[k,k] := A[k,k] - sum(L[k,s]*U[s,k],s=1..k-1):
>        for j from k+1 to N do
>           U[k,j] := A[k,j] - sum(L[k,r]*U[r,j],r=1..k-1):
>        end do:
>        for i from k+1 to N do
>           L[i,k]:=1/U[k,k]*(A[i,k] - sum(L[i,q]*U[q,k],q=1..k-1)):
>        end do:
>     end do:
>     ### Solution of equation Jac_F Z^[r+1] = -F(Z^[r]).
>     ### Solve LW^[r+1] = -F(Z^[r]) by forward substitution: W^[r+1] = UZ^[r+1].
>     W:=array(1..N):
>     for i from 1 to N do
>        W[i] := - F[i](Z[i-1],Z[i],Z[i+1])- sum(L[i,s]*W[s],s=1..i-1:
>     end do:
>     ### Solve next UZ^[r+1] = W^[r+1] by backward substitution.
>     for i from 0 to N-1 do
>        Z1[N-i] := 1/U[N-i,N-i]*( W[N-i] - sum(U[N-i,s]*Z1[s],s=N-i+1..N)):
>     end do:
>     ### Next iteration.
>     for n from 1 to N do Z[n]:=Z[n]+Z1[n]:  end do:
>     N_iter:=N_iter+1:
>     if (N_iter=30) then break; end if:     # To aovid an infinite loop
>  end do:
###### End of Newton-Raphston loop.
###### Results and plots:
>  printf("Convergence with precision h in %2d iterations \n", N_iter):
###### Solution and iterations leading to it.
>  display(plotpoints[N_iter-1],axes=framed,labels=["x","theta"]);
>  PPP:=seq(plotpoints[r],r=0..N_iter-1):
>  display(PPP, axes=framed,labels=["x","theta(x)"]);
###### Energy of solutions and of iterations as function of x.
>  display(energy[N_iter-1],axes=framed,labels=["x","E(x)"]);
>  EEE:=seq(energy[r],r=0..N_iter-1):
>  display(EEE, axes=framed,labels=["x","E(x)"]);
>    ###### Energy of solutions and of iterations as function of x.
>  printf("Energy of soution = %14.12f", E_av[N_iter-1]):
```

Appendix 5.2. The Newton-Kantorovich theorem (outside the scope of these notes).

Given a mapping $f:\ \mathbf{R}^n \to \mathbf{R}^n$ and a covex set $\mathbf{C} \subset \mathbf{R}$, let $D_f(x)$ be the Jacobian of f at x and satisfy:

(a) $\|D_f(x) - D_f(y)\| \le \gamma \|x - y\|$ *for all* $x, y \in \mathbf{C}$,
(b) $\|D_f^{-1}(x_0)\, f(x_0)\| \le \alpha$,
(c) $\|D_f^{-1}(x_0)\| \le \eta$

for some $x_0 \in \mathbf{C}$. Consider the quantities

$$\delta = \alpha\gamma\eta \qquad \text{and} \qquad \rho = \frac{1 - \sqrt{1 - 2\delta}}{\delta}\,\alpha\,.$$

If $\delta \le 1/2$ and the ball $\mathbf{B}_{\boldsymbol{\rho}}(\boldsymbol{x_0})$ centered at x_0 of radius ρ is a subset of $\mathbf{C}$, then the sequence

$$\{x_k\}_{k=0,1,\ldots} \qquad x_{k+1} = x_k - D_f^{-1}(x_k)\, f(x_k)$$

remains in $\mathbf{B}_{\boldsymbol{\rho}}(\boldsymbol{x_0})$ and converges to the unique zero of $f(x)$ in $\mathbf{B}_{\boldsymbol{\rho}}(\boldsymbol{x_0})$.