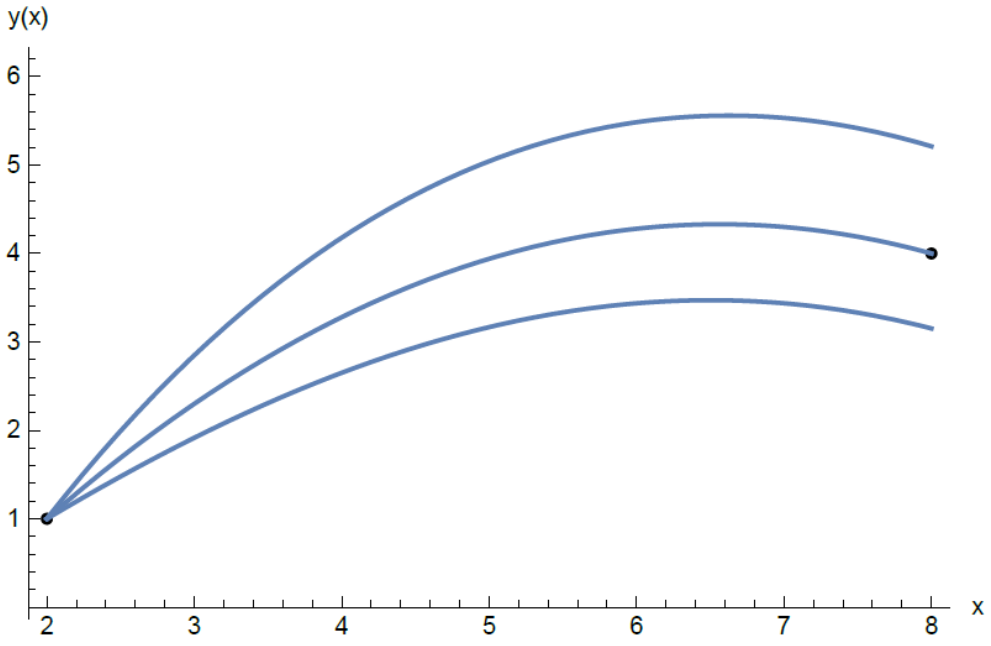
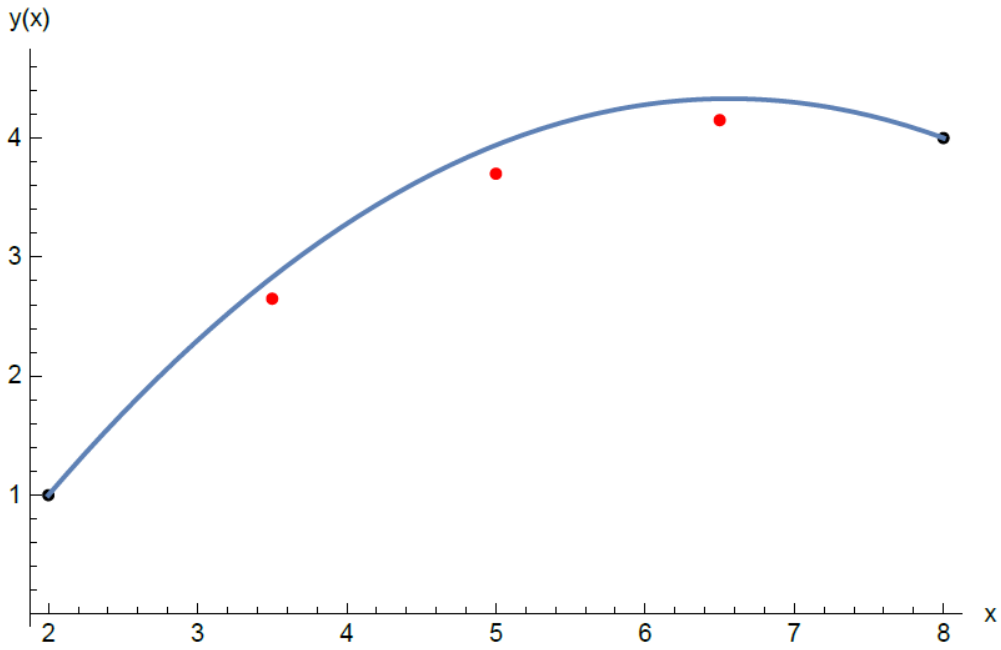
Two point boundary value problem (NR. Chap. 18). We consider the model problem:

$$\begin{cases} y''(x) = F(y'(x), y(x), x) \quad a < x < b \\ y(a) = \alpha; \quad y(b) = \beta \end{cases}$$

Example with boundary:
a=2; b=8; alpha=1; beta=4

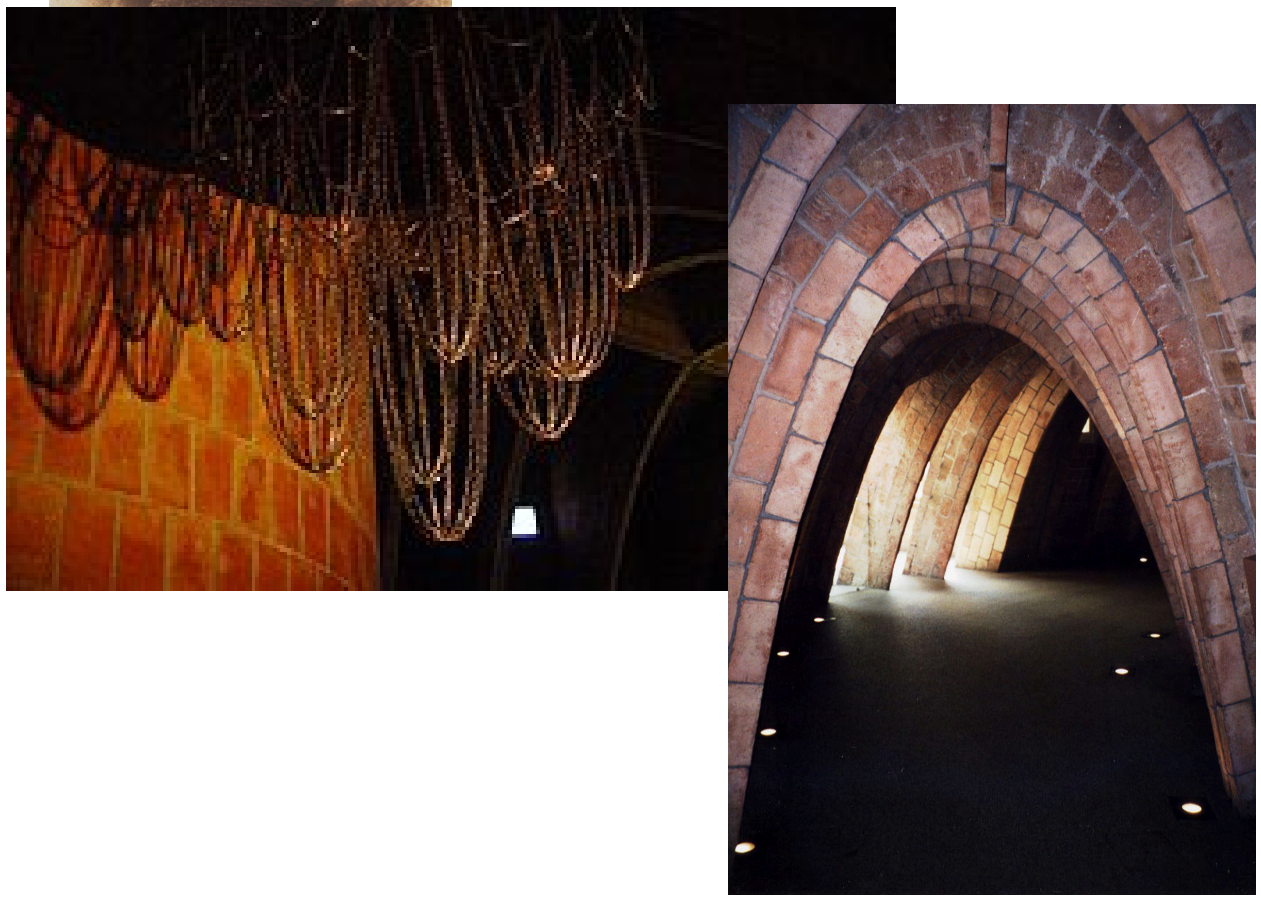


Shooting Method



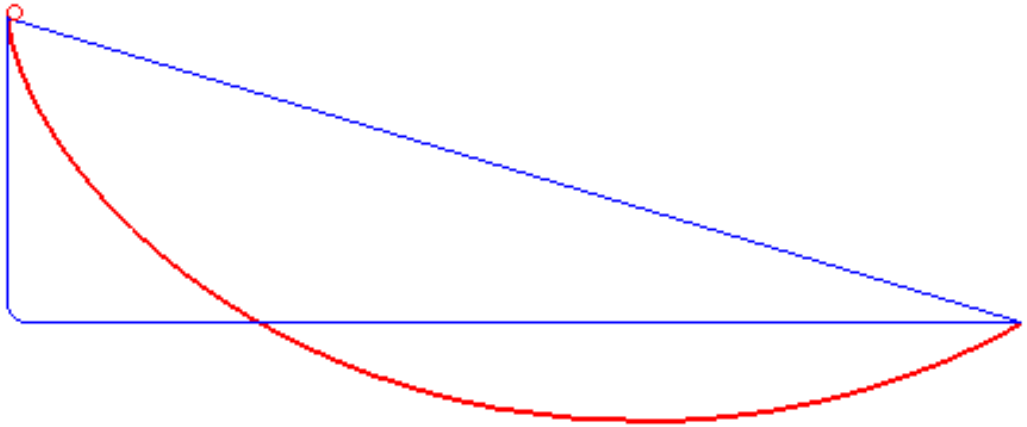
Finite Difference Method

Antoni Gaudí (1852—1926)



The Brachistochrone curve

(fastest frictionless movement between two points)

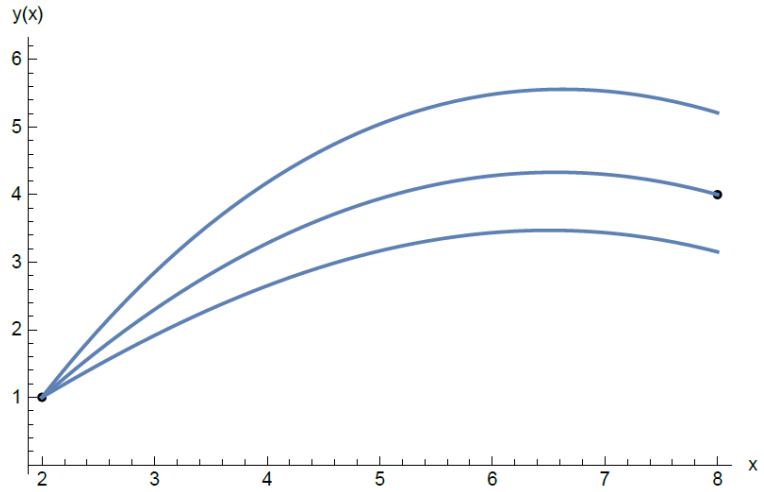


Catenary curves. Example from Casa Mila (Gaudi), Barcelona

Two point boundary value problem (NR. Chap. 18). We consider the model problem:

$$\begin{cases} y''(x) = F(y'(x), y(x), x) & a < x < b \\ y(a) = \alpha; \quad y(b) = \beta \end{cases}$$

Shooting method. NR 18.2



Choose $y'(a) = s$. Then solve the initial value problem

$$\begin{aligned} y'_{[0]}(x) &= y_{[1]}(x) \quad y_{[0]}(a) = \alpha \\ y'_{[1]}(x) &= F(y_{[1]}(x), y_{[0]}(x), x) \quad y_{[1]}(a) = s \end{aligned}$$

with a some selected method and a stepsize h so that $b - a = Mh$ for some integer M. You then obtain a numerical approximation $(y_M)_{[0]}$ to $y_{[0]}(b)$. This is a function of the "shooting value" s and the stepsize h.

We now define the deviation between the found numerical approximation and the desired value

$$\phi_h(s) = (y_M)_{[0]} - \beta; \quad \text{when shooting with } y_{[1]}(a) = s$$

We now wish to solve

$$\phi_h(s) = 0$$

This leads to an algorithm called "Shooting". Select h and two shooting values s_0 and s_1, and compute $\phi_h(s_0)$ and $\phi_h(s_1)$ by integrating the ODE's.

Then compute

$$s_2 = s_1 - \phi_h(s_1)\frac{s_1 - s_0}{\phi_h(s_1) - \phi_h(s_0)}$$

which you will recognize as the Secant method. Now use s_2 as the new shooting value, integrate the ODE's to obtain $\phi_h(s_2)$ and do Secant with s_1 and s_2.

When the Secant method converges, you have found a solution that still depends on h. You may then divide the stepsize by 2, and run again. Here, you can use the last two s_i's from the old stepsize as the initial guesses for the Secant method.

The number of function calls is proportional to $N_0(n_0^S + 2n_1^S + 4n_2^S + \ldots + 2^{iMax}n_{iMax}(s))$ where N_0 is the number of steps for the largest h and n_i^S is the number of secant iterations for h_i (corresponding to $N_0 2^i$ steps.)

NOTICE: You have two loops with related errors. In the inner loop you solve $\phi_h(s) = 0$ with the secant method, leading to a secant method error $e_S(h)$ for the fixed h. In the outer loop, you subdivide h by 2 and obtain the ODE numerical integration error $e_I(h)$. You can typically ensure that $e_S(h) << e_I(h)$ and then you only have the integration error.

Boundary value problem. Model problem

$$\begin{cases} y''(x) = F(y'(x), y(x), x) \quad a < x < b \\ y(a) = \alpha; \quad y(b) = \beta \end{cases}$$

Finite difference method (similar to Relaxation method NR 18.3)

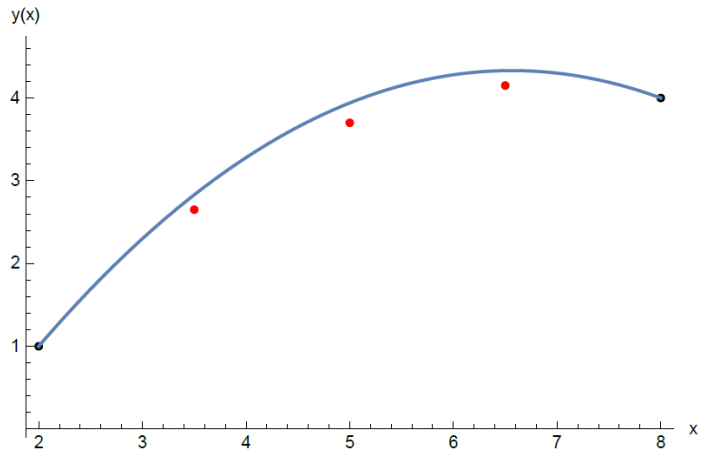
Use finite differences $h = (b-a)/N; \quad x_i = a + ih; \quad y_i \simeq y(x_i)$

$$y_i'' = F(y_i', y_i, x_i) \qquad i = 1, \ldots, N-1$$
$$y_0 = \alpha; \quad y_N = \beta$$

$$y'(x_i) = \frac{y_{i+1} - y_{i-1}}{2h} + \mathcal{O}(h^2) \qquad y''(x_i) = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + \mathcal{O}(h^2)$$

Turned to set of nonlinear equations

$$\begin{cases} y_{i+1} - 2y_i + y_{i-1} = h^2 F\left(\frac{1}{2h}(y_{i+1} - y_{i-1}), y_i, x_i\right), \quad i = 1, \ldots, N-1 \\ y_0 = \alpha, \quad y_N = \beta \end{cases}$$



Blue curve is the solution y(x). Red dots are (x_i,y_i)'s and the black dots are the boundary values (a, α) and (b, β).

$$
\begin{aligned}
y_0 &= \alpha \\
-y_{i-1} + 2y_i - y_{i+1} + h^2 F(\frac{y_{i+1} - y_{i-1}}{2h}, y_i, x_i) &= 0 \quad i = 1, \ldots, N-1 \\
y_N &= \beta
\end{aligned}
$$

The boundary conditions are incorporated in the equations for the neighbour points ($i = 1$ and $i = N - 1$):

$$
\left.
\begin{aligned}
-\alpha + 2y_1 - y_2 + h^2 F(\tfrac{y_2 - \alpha}{2h}, y_1, x_1) &= 0 \\
-y_{i-1} + 2y_i - y_{i+1} + h^2 F(\tfrac{y_{i+1} - y_{i-1}}{2h}, y_i, x_i) &= 0 \quad i = 2, \ldots, N-2 \\
-y_{N-2} + 2y_{N-1} - \beta + h^2 F(\tfrac{\beta - y_{N-2}}{2h}, y_{N-1}, x_{N-1}) &= 0
\end{aligned}
\right\} \phi(\mathbf{y}) = 0
$$

where $\mathbf{y} \equiv (y_1, \ldots, y_{N-1})$.

This is a set of non-linear equations in $\mathbf{y}$. We will apply Newtons method:

$$
\begin{aligned}
\text{Solve} \quad J(\mathbf{y}^{(n)})\Delta\mathbf{y} &= -\phi(\mathbf{y}^{(n)}) \\
\mathbf{y}^{(n+1)} &= \mathbf{y}^{(n)} + \Delta\mathbf{y}
\end{aligned}
$$

where $J(\mathbf{y})$ is the Jacobian to $\phi(\mathbf{y})$.

$$
\begin{aligned}
-\alpha + 2y_1 - y_2 + h^2 F(\frac{y_2 - \alpha}{2h}, y_1, x_1) &= 0 \\
-y_{i-1} + 2y_i - y_{i+1} + h^2 F(\frac{y_{i+1} - y_{i-1}}{2h}, y_i, x_i) &= 0 \quad i = 2, \ldots, N-2 \\
-y_{N-2} + 2y_{N-1} - \beta + h^2 F(\frac{\beta - y_{N-2}}{2h}, y_{N-1}, x_{N-1}) &= 0
\end{aligned}
$$

To facilitate notation, we write

$$
\begin{aligned}
F_y(y', y, x) &\equiv \frac{\partial F(y', y, x)}{\partial y} \\
F_{y'}(y', y, x) &\equiv \frac{\partial F(y', y, x)}{\partial y'}
\end{aligned}
$$

For the first equation, we get nonzero Jacobian elements

$$
\begin{aligned}
J_{1,1} &= 2 + h^2 F_y(\frac{y_2 - \alpha}{2h}, y_1, x_1) \\
J_{1,2} &= -1 + \frac{h}{2} F_{y'}(\frac{y_2 - \alpha}{2h}, y_1, x_1)
\end{aligned}
$$

For the last equation, we get nonzero Jacobian elements

$$
\begin{aligned}
J_{N-1,N-2} &= -1 - \frac{h}{2} F_{y'}(\frac{\beta - y_{N-2}}{2h}, y_{N-1}, x_{N-1}) \\
J_{N-1,N-1} &= 2 + h^2 F_y(\frac{\beta - y_{N-2}}{2h}, y_{N-1}, x_{N-1})
\end{aligned}
$$

and for the internal equations $i = 2, \ldots, N-2$, we get nonzero Jacobian elements

$$
\begin{aligned}
J_{i,i-1} &= -1 - \frac{h}{2} F_{y'}(\frac{y_{i+1} - y_{i-1}}{2h}, y_i, x_i) \\
J_{i,i} &= 2 + h^2 F_y(\frac{y_{i+1} - y_{i-1}}{2h}, y_i, x_i) \\
J_{i,i+1} &= -1 + \frac{h}{2} F_{y'}(\frac{y_{i+1} - y_{i-1}}{2h}, y_i, x_i)
\end{aligned}
$$

$$\text{Solve} \quad J(\mathbf{y}^{(n)})\Delta\mathbf{y} = -\phi(\mathbf{y}^{(n)})$$
$$\mathbf{y}^{(n+1)} = \mathbf{y}^{(n)} + \Delta\mathbf{y}$$

As with any other method using a step size h, we start with a rather large step size. For this largest step size use the initial guess $\mathbf{y}^{(0)}$ as

$$y_i = \alpha + \frac{i}{N}(\beta - \alpha); \quad i = 1, \ldots, N-1$$

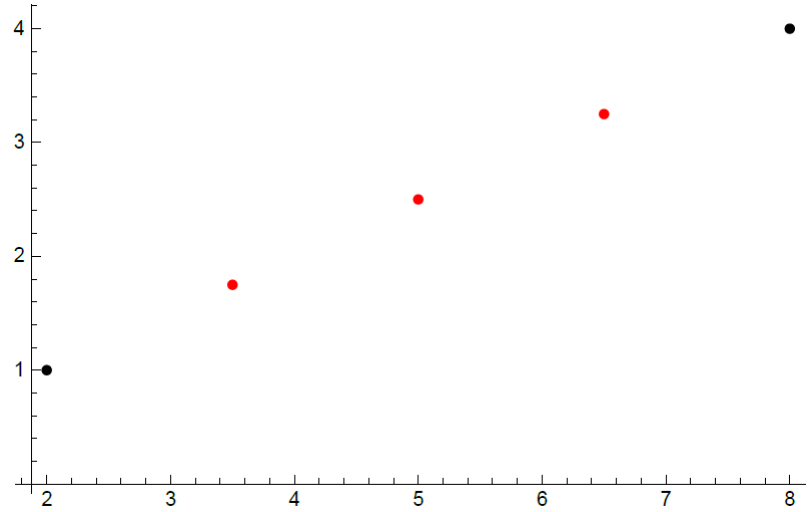
which is simply a linear interpolation between the end points.

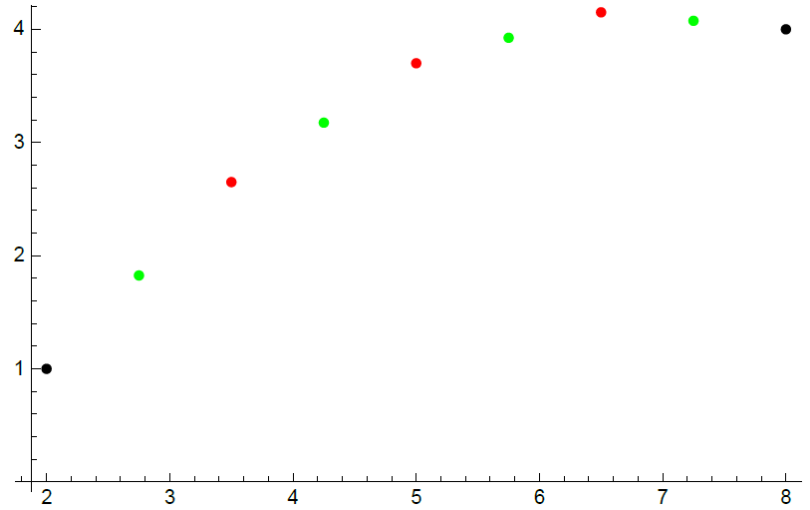
For the subsequent h values, record the result $\mathbf{y}^{old}$ from the previous h-value. Then use as the initial guess $\mathbf{y}^{(0)}$ as the interpolation

$$y_i = \begin{cases} y^{old}_{i/2} & i \text{ even} \\ \frac{1}{2}\left(y^{old}_{(i-1)/2} + y^{old}_{(i+1)/2}\right) & i \text{ odd} \end{cases}$$

Use Richardson Extrapolation to estimate errors in the usual way. Expect O(h^2). Error can be estimated for points used with the largest h-value.

Example with N=4 for the largest h:





$$\begin{bmatrix} b_0 & c_0 & 0 & \cdots & & & \\ a_1 & b_1 & c_1 & \cdots & & & \\ & & & \cdots & & & \\ & & & \cdots & a_{N-2} & b_{N-2} & c_{N-2} \\ & & & \cdots & 0 & a_{N-1} & b_{N-1} \end{bmatrix} \cdot \begin{bmatrix} u_0 \\ u_1 \\ \cdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ \cdots \\ r_{N-2} \\ r_{N-1} \end{bmatrix} \qquad (2.4.1)$$

Notice that a_0 and c_{N-1} are undefined and are not referenced by the routine that follows.

```
void tridag(VecDoub_I &a, VecDoub_I &b, VecDoub_I &c, VecDoub_I &r, VecDoub_O &u)
```
Solves for a vector `u[0..n-1]` the tridiagonal linear set given by equation (2.4.1). `a[0..n-1]`, `b[0..n-1]`, `c[0..n-1]`, and `r[0..n-1]` are input vectors and are not modified.

Routine "tridag" page 56-57 in NR
Usable ONLY for tridiagonal matrices
(It is O(N) instead of O(N^3) !!!!!)

For the first equation, we get nonzero Jacobian elements

$$\begin{aligned} J_{1,1} &= 2 + h^2 F_y(\frac{y_2 - \alpha}{2h}, y_1, x_1) \\ J_{1,2} &= -1 + \frac{h}{2} F_{y'}(\frac{y_2 - \alpha}{2h}, y_1, x_1) \end{aligned}$$

For the last equation, we get nonzero Jacobian elements

$$\begin{aligned} J_{N-1,N-2} &= -1 - \frac{h}{2} F_{y'}(\frac{\beta - y_{N-2}}{2h}, y_{N-1}, x_{N-1}) \\ J_{N-1,N-1} &= 2 + h^2 F_y(\frac{\beta - y_{N-2}}{2h}, y_{N-1}, x_{N-1}) \end{aligned}$$

and for the internal equations $i = 2, \ldots, N-2$, we get nonzero Jacobian elements

$$\begin{aligned} J_{i,i-1} &= -1 - \frac{h}{2} F_{y'}(\frac{y_{i+1} - y_{i-1}}{2h}, y_i, x_i) \\ J_{i,i} &= 2 + h^2 F_y(\frac{y_{i+1} - y_{i-1}}{2h}, y_i, x_i) \\ J_{i,i+1} &= -1 + \frac{h}{2} F_{y'}(\frac{y_{i+1} - y_{i-1}}{2h}, y_i, x_i) \end{aligned}$$

Directly usable with tridag except indexing from 1 to N-1 instead of 0 to N-1.

IMPLEMENTATION: Notice that for a new problem, you ONLY need to define a, b, alpha, beta, and functions that you call for F(y',y,x) and the derivatives F_y'(y',y,x) and F_y(y',y,x).

Shooting vs Finite Difference method

Shooting method:

- Shooting is good if very high accuracy is required.
- Boundary value problems are often of a nature leading to stability problems.
- Easy to implement.

Finite Difference method:

- Finite Difference method is ultimately stable as it does not start from one end.
- Requires very small h if desired accuracy is very high.
- A bit more complicated to program, but generally preferable.

Notice: As mentioned, the found result will have an accuracy |y(x_i)-y_i| which is O(h^2). Use Richardson to estimate the error.

Notice: As mentioned, J(y) is ”tri-diagonal” (only nonzero elements on the diagonal and the two neighbour elements in each row). Use routine ”tridag” page 56-57 (It is O(N) instead of O(N^3) !!!!!)

Exercise:

$$\begin{cases} y''(x) = 2x + \sin[y'(x)] - \cos[y(x)] & 0 < x < 2 \\ y(0) = 0; \quad y(2) = 1 \end{cases}$$

We now want to apply the Finite Difference method. Assume we discretize using N steps. Hence, we have $h = (b-a)/N$. Use as usual the notation $x_i = a + ih$; $y_i \simeq y(x_i)$.

We are now interested in estimating $y(1)$ with a proven accuracy. Use the Finite Difference method to estimate $y(1)$ with an accuracy better than 10^{-4}. Start with $h = 1$, then $h = 0.5, h = 0.25, ...$

HINT: Use the general formulation above with a, b, α, β as parameters. Define and implement functions $F(y', y, x)$, $F_y(y', y, x)$ and $F_{y'}(y', y, x)$. These functions are then just called from the general program which implements the system of non-linear equations and its solution using ”tridag”.