Boundary Value Problems (BVP)

Introduction (BVP)

The general form of a BVP is:

$$\frac{dy}{dx} = f(x,y), \quad f:[a,b]\times\mathbf{R}^d \to \mathbf{R}^d,\ d \geq 2$$

along with the boundary conditions (BC) on $[a,b]$:

$$g(y(a), y(b)) = 0$$

where $g:\mathbf{R}^d \times \mathbf{R}^d \to \mathbf{R}^d$ is a function.

Different kind of BC (for $d = 2$):

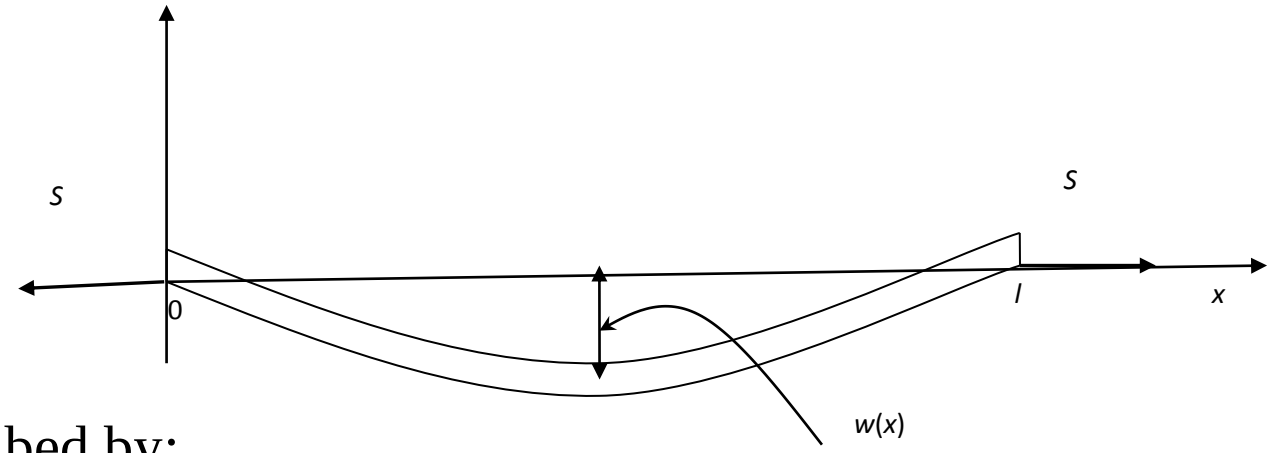
$$y(a) = A \qquad \text{(Dirichlet)}$$

$$\frac{dy}{dx}(a) = A \qquad \text{(Neumann)}$$

$$\frac{dy}{dx}(a) + \alpha\, y(a) = A,\ \ \alpha < 0 \qquad \text{(Robin)}$$

where A is a constant.

Example 1: Bending of a beam due to the gravitational force



is described by:

$$\frac{d^2w}{dx^2}(x) = \frac{S}{EI}w(x) + \frac{qx}{2EI}(x-l)$$

where $w(x)$ is the displacement, l is the beam length, q is the load intensity, E is the elasticity modulus, S is the stress on the two ends, and I is the bending moment of the beam. We have zero displacements on the boundaries, thus:

$$w(0) = 0;\ \ w(l) = 0.$$

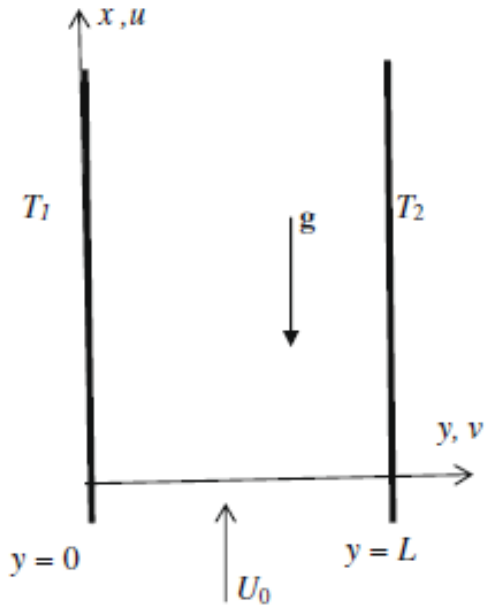
Example 2: Mixed Convection in a Channel with Chemical Reaction

$$U'' + \lambda\theta - \alpha = 0$$

$$\theta'' + K\, e^{\theta} = 0$$

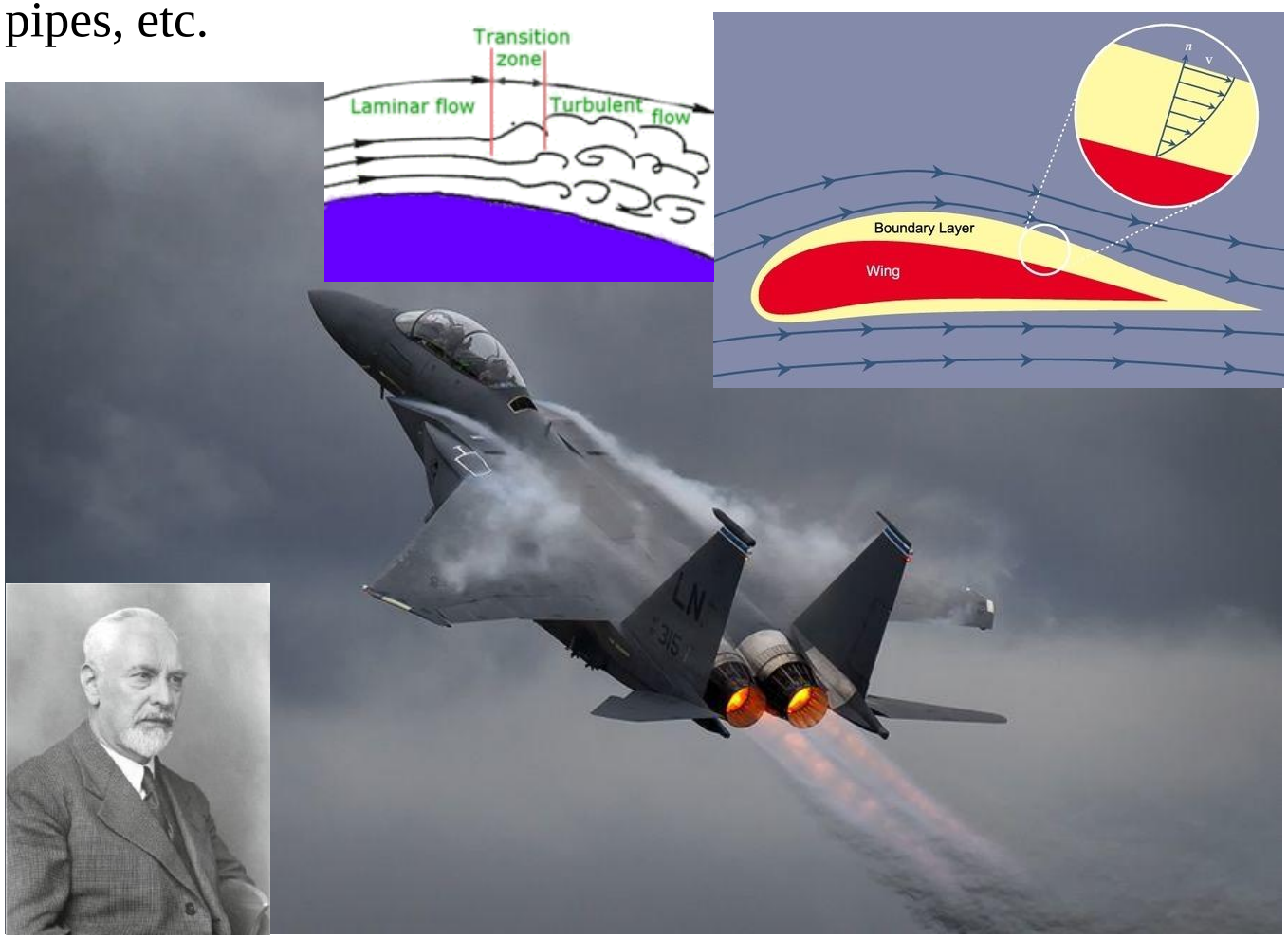
$$U(0) = 0, \quad \theta(0) = r_T, \quad U(1) = 0, \quad \theta(1) = -r_T$$

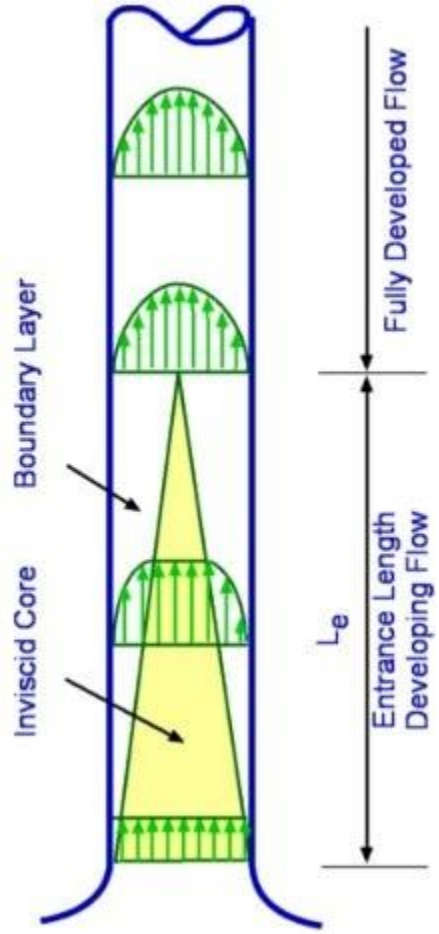
where U is the fluid velocity, θ is the temperature and λ, α, K and r_T are parameters depending on the problem.



Introduction to Computational Fluid Dynamics

In **Fluid mechanics** the mathematical model reduces to BVP in the **boundary layer approximation** (L. Prandtl, 1904) and **fully developed flow** in channels, pipes, etc.





About existence and unicity

See the following examples (Agratini et al., 2002).

Let be the BVP:

$$y''-y=0,\ \ y(0)=0,\ \ \ y(b)=\beta$$

with the solution

$$y(x)=\beta\frac{\sinh x}{\sinh b},\ \ x\in[0,b]$$

We modify the above problem as follow

$$y''+y=0,\ \ y(0)=0,\ \ \ y(b)=\beta$$

Thus, for

$$b\neq k\pi,\ \ \ k\in\mathbf{N}^*\ \text{then}\ \ \ \ y(x)=\beta\frac{\sin x}{\sin b}$$

$$b=k\pi,\ \ \ k\in\mathbf{N}^*\ \text{and}\ \beta\neq 0\ \text{there is no solution}$$

$$b=k\pi,\ \ \ k\in\mathbf{N}^*\ \text{and}\ \beta=0\ \text{then}\ y(x)=c\sin x,\ \ c\in\mathbf{R}$$

However, for the last case, we the number of solution is infinity, but the condition $y(b)=\beta$ cannot be satisfied because $y(b)=y(k\pi)=c\sin(k\pi)=0\neq\beta$. Thus, $b=k\pi,\quad k\in\mathbf{N}^*$ is a critical point.

One can see that a minor change in the problem leads to important changes in the number and the form of the solutions.

Theorem: Let be the BVP

$$y''(x)=f(x,y,y'),\qquad x\in[a,b]$$
$$a_0y(a)-a_1y'(a)=\alpha$$
$$b_0y(b)-b_1y'(b)=\beta$$

If

i) $f(x,u_1,u_2)$ is continuous on $[a,b]\times\mathbf{R}\times\mathbf{R}$

ii) $\partial f/\partial u_1$ and $\partial f/\partial u_2$ are continuous an satisfy $|\partial f/\partial u_1|\le L_1$ and $|\partial f/\partial u_2|\le L_2$ on $[a,b]\times\mathbf{R}\times\mathbf{R}$

iii) $a_0a_1\ge 0,\ b_0b_1\ge 0,\ |a_0|+|b_0|>0$

then the BVP has a unique solution.

Shooting Metohod

BVP = incomplete IVP+other BC

Thus, it is possible to transform a BVP into an IVP with some unknown initial conditions. These unknown conditions must be found using the values on the boundary.

$f''' + f f'' = 0$ $\Longleftrightarrow$ $f''' + f f'' = 0$

$f(0) = 0, f'(0) = 0, f'(7) = 1$ $f(0) = 0, f'(0) = 0, f''(0)=s$

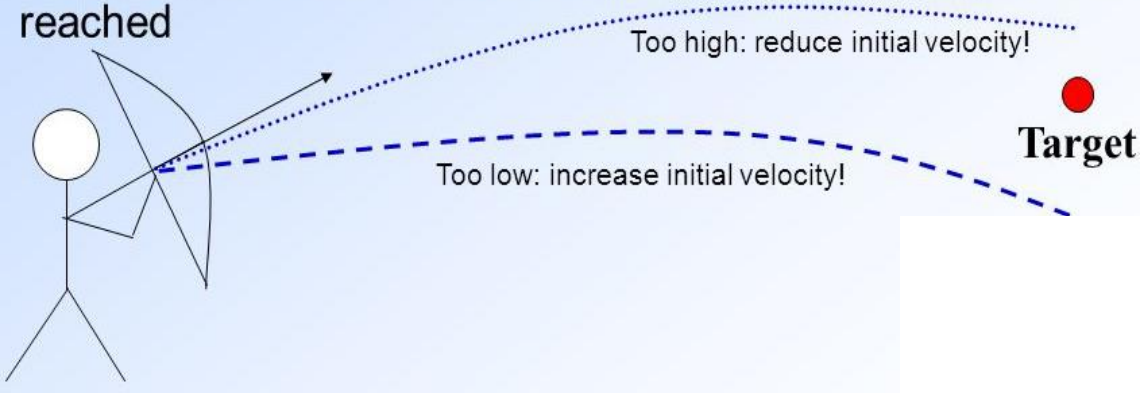
How to find "s" in such a way that $f'(7) = 1$?

This method is called the shooting method because of its resemblance to the problem faced by an artillery officer who is trying to hit a distant target. The right elevation of the gun can be found if two shots are made of which one is short of the target and the other is beyond. That means that an intermediate elevation will come closer.

(Gerald and Wheatley, Applied numerical analysis, California Polytechnic State University, Boston, 2006)

Secant Method

A first approach is to transform the BVP into an initial value problem (IVP), by guessing the missing initial conditions and using the BC to refine the guess, until convergence is reached



$y(a) = 0$
$y(b) = y_b$ BV

↓

$z(a) = Guess$
$y(a) = 0$ IV

↓ Solve with RK4

$y(x)$
$z(x)$
$y(b)$

$m = y_{true}(b) - y_{guess}(b)$ Is m small enough?

No ←

↓ Yes

write out solution

Algorithm:

Consider the boundary conditions on the interval [a,b]:

$$y(a)=\alpha,\ y'(a)=\beta,\ y'(b)=\gamma$$

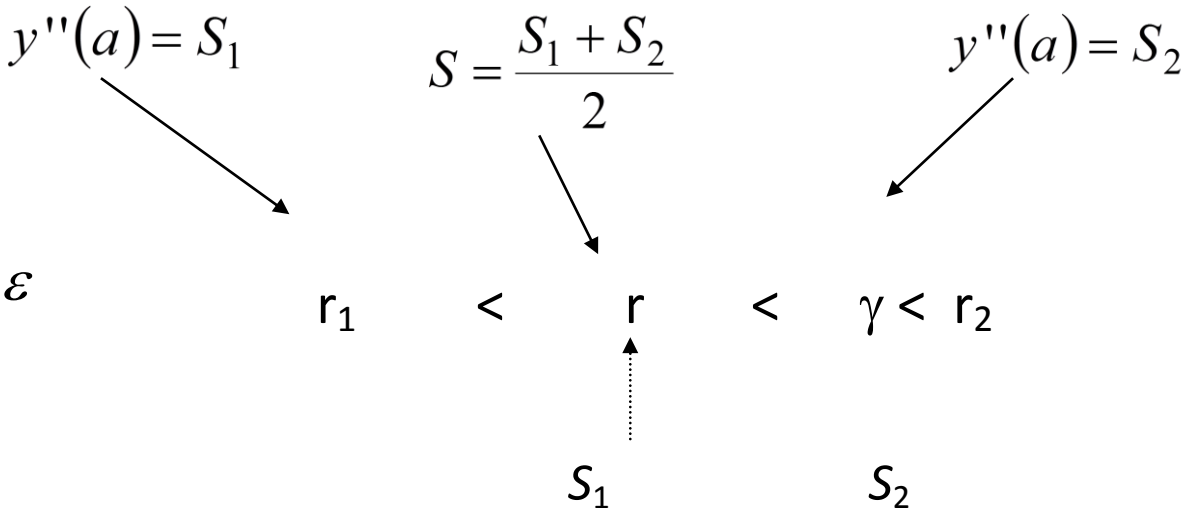
We have to find the value: $y''(a)=\delta$ in such a way that $y'(b)=\gamma$.

Choose two values $y''(a)=S_1$ and $y''(a)=S_2$ such that

$y'(b)=r_1$ and $y'(b)=r_2$ and $r_1 \le \gamma \le r_2$.

Next, we solve the equation for $y''(a)=S$, where $S=(S_1+S_2)/2$

and choose the next S_1 and S_2



Stop when $|y'(b,S)-\gamma|<\varepsilon$

Example: Consider the Blasius equation (see Oroveanu, 1967), that model the boundary layer flow on a plate:

$$f'''(\eta) + f(\eta)\, f''(\eta) = 0$$

$f(0) = 0,\ f'(0) = 0,\ f'(\eta_\infty) = 1$, where η_∞ is large (e.g. $\eta_\infty = 7$)

The equation is transform in the following system:

$$y_1' = y_2,\ \ y_2' = y_3,\ \ \ y_3' = -y_1 y_3$$
$$y_1(0) = 0;\ y_2(0) = 0;\ y_2(\eta_\infty) = 1$$

We consider two starting values for the missing condition $f''(0)$:

$S_1 = 0.1$ (solving using RK one obtain $f'(\eta_\infty, S_1) = 0.356604$)

$S_2 = 0.7$ (solving using RK one obtain $f'(\eta_\infty, S_1) = 1.304993$)

Thus, $f'(\eta_\infty, S_1) = 0.356604) < 1 < f'(\eta_\infty, S_1) = 1.304993$ => "shooting"

Matlab program

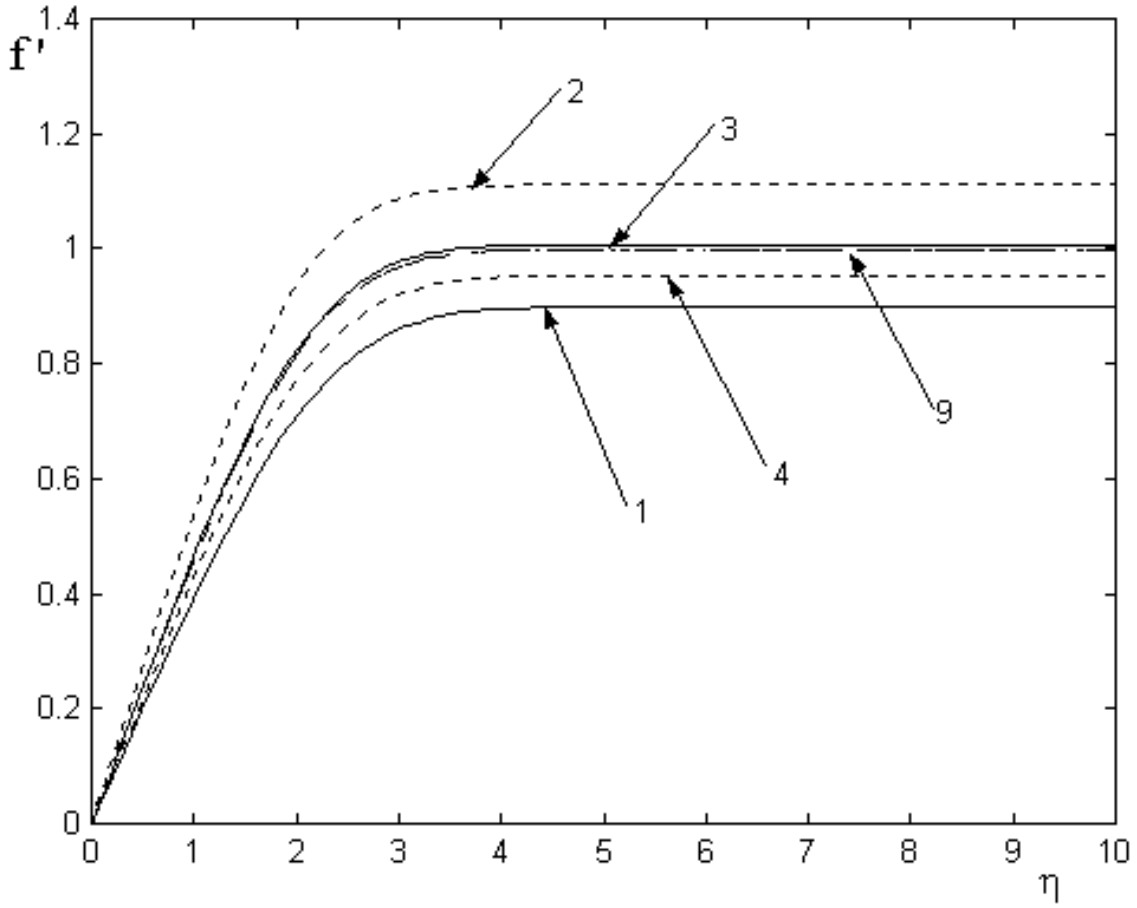
```
a, b   – solution interval;
S1, S2 – aproximations for f''(0);
yp     – value of f ' in ∞;
k      – number of iterations;
A      – matrix of results, A = [η,f , f ', f ''];
```

```
function dy=fBlasius(t,y)
     dy=[y(2),y(3),-y(3)*y(1)];
```

```
function [x,y]=Kutta(f,a,b,N,y0)%ode45 can be used
h=(b-a)/(N-1)% step
y=zeros(N,length(y0)); %solution initialization
y(1,:)=y0;
t=a:h:b;
x=t;
for i=2:N
    K1=feval(f,t(i-1),y(i-1,:));
    K2=feval(f,t(i-1)+0.5*h,y(i-1,:)+0.5*h*K1);
    K3=feval(f,t(i-1)+h,y(i-1,:)-h*K1+2*h*K2);
    y(i,:)=y(i-1,:)+h/6*(K1+4*K2+K3);
end
```

```
%main program
format long;
a=0; b=7;N=701;
S1=0.1; S2=0.7;
yp=2;
k=0; %number of iterations
while abs(yp-1)>0.001
  S=(S1+S2)/2;
  y0=[0,0,S];
  [x,y]=Kutta('fBlasius',a,b,N,y0);
  [m,n]=size(y);
  if y(m,2)<1
    S1=S;
  else
    S2=S;
  end;
  yp=y(m,2)
  k=k+1;
end;
A=[x',y];
disp('  eta     fsec     fprim    f   ')
disp(A);plot(x,y(:,2));
```

Iterations	f '(∞)	f "(0)	Iterations	f '(∞)	f "(0)
1	0.898619238225	0.400000	6	0.994396542243	0.465625
2	1.111165796678	0.550000	7	1.001059313085	0.470312
3	1.007699998861	0.475000	8	0.997730706789	0.467968
4	0.953939187474	0.437500	9	0.999395702387	0.469140
5	0.981003550835	0.456250			



Numerical solution lies between the approximated solutions. (Sandwich rule)

Newton's Method

Consider the equation

$$\frac{d^3 f}{dx^3} = g\left(x, f, \frac{df}{dx}, \frac{d^2 f}{dx^2}\right), \quad f(0) = 0, \quad \frac{df}{dx}(0) = 0, \quad \frac{df}{dx}(L) = f'_L$$

or

$$\begin{cases} \dfrac{df}{dx} = a \\ \dfrac{da}{dx} = b \\ \dfrac{db}{dx} = g(x, f, a, b) \end{cases} \qquad f(0) = 0, \quad \frac{df}{dx}(0) = a(0) = 0, \quad \frac{d^2 f}{dx^2}(0) = b(0) = s \qquad (1)$$

where s is the initial unknown condition.

In order to find s is necessary that

$$a(L, s) - f'_L = \phi(s) = 0$$

Considering s^n the value of s at the n^{th} iteration, one can use a Taylor expansion of ϕ:

$$0 = \phi(s^{n+1}) = \phi(s^n) + \left(s^{n+1} - s^n\right)\frac{d\phi}{ds}(s^n) + \ldots$$

and we get

$$s^{n+1} = s^n - \frac{\phi(s^n)}{\frac{d\phi}{ds}(s^n)} \quad \text{or} \quad s^{n+1} = s^n - \frac{a(L, s^n) - f'_L}{\frac{\partial a}{\partial s}(L, s^n)}$$

Now the unknown is $\frac{\partial a}{\partial s}(L, s^n)$ and in order to find it one can derive with respect to s the initial system:

$$\begin{cases} \dfrac{d}{dx}\left(\dfrac{\partial f}{\partial s}\right)=\dfrac{\partial a}{\partial s} \\ \dfrac{d}{dx}\left(\dfrac{\partial a}{\partial s}\right)=\dfrac{\partial b}{\partial s} \\ \dfrac{d}{dx}\left(\dfrac{\partial b}{\partial s}\right)=\dfrac{\partial g}{\partial f}\dfrac{\partial f}{\partial s}+\dfrac{\partial g}{\partial a}\dfrac{\partial a}{\partial s}+\dfrac{\partial g}{\partial b}\dfrac{\partial b}{\partial s} \end{cases} \qquad \frac{\partial f}{\partial s}(0)=0, \quad \frac{\partial a}{\partial s}(0)=0, \quad \frac{\partial b}{\partial s}(0)=1$$

By using the notations $F=\dfrac{\partial f}{\partial s}$, $A=\dfrac{\partial a}{\partial s}$, $B=\dfrac{\partial b}{\partial s}$ we obtain:

$$\begin{cases} \dfrac{dF}{dx}=A \\ \dfrac{dA}{dx}=B \\ \dfrac{dB}{dx}=\dfrac{\partial g}{\partial f}F+\dfrac{\partial g}{\partial a}A+\dfrac{\partial g}{\partial b}B \end{cases} \qquad F(0)=0, \quad A(0)=0, \quad B(0)=1 \qquad (2)$$

Now, the (Newton) shooting algorithm can be given:

Step 1: $s = s^1$

Step 2: solve (1) with $b(0) = s^1$

Pasul 3: solve (2) and keep the value $A(L,s^1) = A(L)$

Pasul 4: find $s^2 = s^1 - \dfrac{a(L,s^1) - f'_L}{A(L,s^1)}$

Pasul 5: repeat steps 1. – 4. with $s^1 = s^2$ until required accuracy is obtained.

Example: Consider again the Blasius equation (system form)

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ y_3' &= -y_1 y_3 \end{aligned} \quad , \quad y_1(0)=0;\ y_2(0)=0;\ y_2(\eta_\infty)=1$$

By using the above algorithm we have to solve

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ y_3' &= -y_1 y_3 \\ y_1(0)=0;\ y_2(0)=0;\ y_3(0)&=s \end{aligned} \qquad \begin{aligned} y_4' &= y_5 \\ y_5' &= y_6 \\ y_6' &= -y_3 y_4 - y_1 y_5 \\ y_4(0)=0;\ y_5(0)=0;\ y_6(0)&=1 \end{aligned}$$

and in order to find s we use:

$$s^2 = s^1 - \frac{y_2(L, s^1) - 1}{y_5(L, s^1)}$$

Matlab program

```
function dy=fBlasNewt(t,y)
dy=[y(2),y(3),-y(3)*y(1), y(5), y(6), -y(3)*y(4)-y(1)*y(6)];

format long;
a=0; b=5;N=251;
S=0.5;
yp=2;
k=0; %number of iterations
while abs(yp-1)>0.0001
 y0=[0,0,S,0,0,1];
 [x,y]=Kutta('fBlasNewt',a,b,N,y0);
 S=S-(y(N,2)-1)/y(N,5);
 yp=y(N,2);
 k=k+1;
end;

disp([x',y]);
plot(x,y(:,2));
```

Iterations	f '(∞)	S = f "(0)
1	1.655190	0.406241
2	0.907751	0.468075
3	0.997769	0.469644
4	0.999998	0.469645

There are necessary less iterations, but the ODE system have 6 equations .

Finite Difference Method

In order to solve BVP by using finite differences one have to follow the steps:

- Choose a grid (mesh)

$$g : a = x_1 < x_2 < x_3 < \ldots < x_N < x_{N+1} = b$$
$$h_i = x_{i+1} - x_i, \quad i = 1,\ 2,\ldots,N$$

- Approximate the derivatives and the boundary conditions using finite difference and form an algebraic system of equations

- Solve the system and obtain the numerical approximations

  - $y_i \approx \tilde{y}(x_i), \quad i = 1,\ 2,\ldots,N+1$

A scheme for a second order BVP

Consider the linear BVP

$$\tilde{y}''(x) - p(x)\tilde{y}'(x) - r(x)\tilde{y}(x) = q(x), \quad x \in [0,1]$$
$$\tilde{y}(0) = \alpha, \ \tilde{y}(1) = \omega$$

We idefine the following uniform grid:

$$h = \frac{1}{N}$$
$$x_i = (i-1)h, \ i = i = 1, \ 2, \ldots, N+1$$

then evaluate the 1st and 2nd order derivatives using the Taylor expansion:

$$y(x_{i+1}) = y(x_i + h) = y(x_i) + hy'(x_i) + \frac{h^2}{2}y''(x_i) + \frac{h^3}{6}y'''(x_i) + O(h^4)$$

$$y(x_{i-1}) = y(x_i - h) = y(x_i) - hy'(x_i) + \frac{h^2}{2}y''(x_i) - \frac{h^3}{6}y'''(x_i) + O(h^4)$$

By adding and subtracting the above expansions we get:

$$y'(x_i) = \frac{y(x_{i+1}) - y(x_{i-1})}{2h} + O(h^2)$$

$$y''(x_i) = \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} + O(h^2)$$

and using the notation $y_i = y(x_i)$ the differential equation is discretized as follow:

$$y_1 = \alpha$$

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} - p(x_i)\frac{y_{i+1} - y_{i-1}}{2h} - r(x_i)y_i = q(x_i), \quad i = 2, 3, \ldots, N$$

$$y_{N+1} = \omega$$

or

$$y_1 = \alpha$$

$$y_{i-1}\left[-1 - \frac{h}{2}p(x_i)\right] + y_i\left[2 + r(x_i)h^2\right] + y_{i+1}\left[-1 + \frac{h}{2}p(x_i)\right] = q(x_i)h^2, \quad i = 2, 3, \ldots, N$$

$$y_{N+1} = \omega$$

which form a tri-diagonal system of algebraic equations:

$$\begin{bmatrix} a_1 & c_1 & & & & \\ b_2 & a_2 & c_2 & & & \\ & b_3 & a_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & b_N & a_N & c_N \\ & & & & b_{N+1} & a_{N+1} \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \\ y_{N+1} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_N \\ d_{N+1} \end{bmatrix} \qquad (***)$$

where

$$a_1 = a_{N+1} = 1,\ c_1 = 0,\ b_{N+1} = 0,\ d_1 = \alpha,\ d_{N+1} = \omega$$

$$b_i = -1 - \frac{h}{2} p(x_i),\quad a_i = 2 + r(x_i)h^2,\quad c_i = -1 + \frac{h}{2} p(x_i),\quad d_i = q(x_i)h^2$$

$$i = 2,\ 3,\ldots,N$$

The tri-diagonal system of equations can be solve efficiently by using different methods, e.g. Thomas algorithm. Faires and Burden (2002) stated:

Theorem: If $p, q, r \in C[0,1]$ and $r(x) \geq 0$ on [0,1] then the system (***) has a unique solution for $h < \frac{L}{2}$ where $L = \max_{0 \leq x \leq 1} |p(x)|$.

Example: Consider the linear BVP

$$y''(x) - y(x) = e^x, \quad y(0) = 1/2, \quad y(1) = e$$

The exact solution $y(x) = \frac{1}{2} e^x (1 + x)$ can be obtained using Mathematica.

```
In[25]:= DSolve[{y''[x] == y[x] + Exp[x], y[0] == 1 / 2,
    y[1] == Exp[1]}, y[x], x]
```

Out[25]= $\left\{\left\{y[x] \to \frac{1}{2} e^x (1 + x)\right\}\right\}$

Further, we solve the problem using finite differences on an equidistant grid with N nodes:

$$y_1 = 1/2$$

$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} - y_i = e^{x_i}, \quad i = 2, 3, \ldots, N-1$$

$$y_N = e$$

The system of algebraic equations is given by:

$$\begin{bmatrix} 1 & & & & \\ -1 & 2+h^2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2+h^2 & -1 \\ & & & & 1 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix} = \begin{bmatrix} 1/2 \\ -h^2 e^h \\ \vdots \\ -h^2 e^{(N-2)h} \\ e \end{bmatrix}$$

Matlab program:

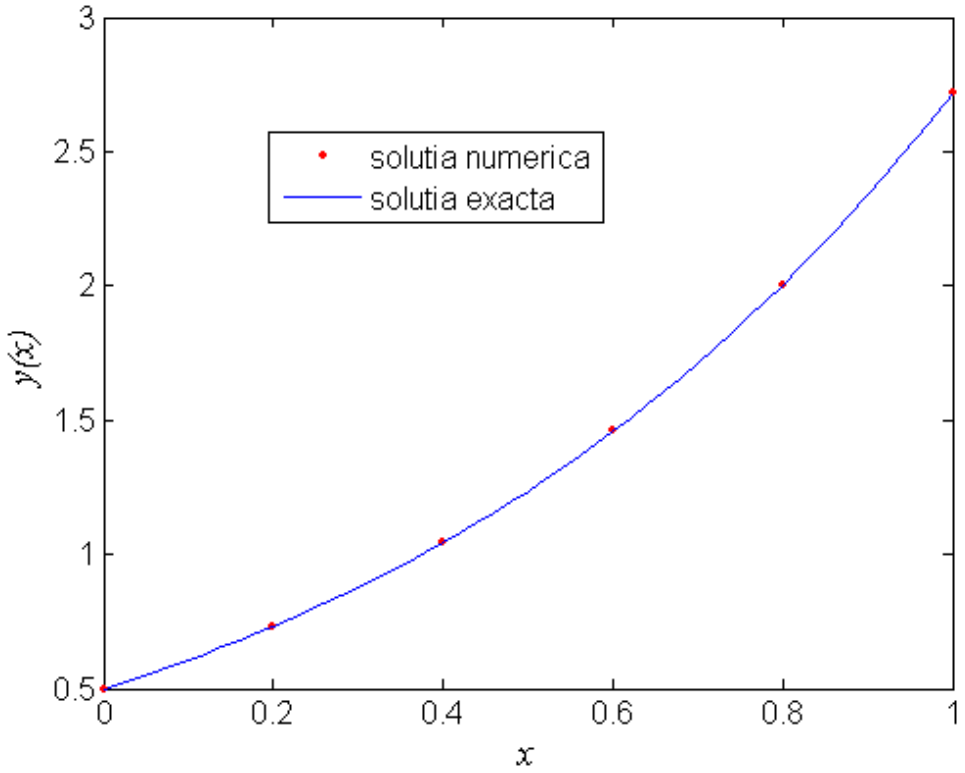
```
%Linear BVP
a=0;b=1;
N=6;
h=(b-a)/(N-1);
A=zeros(N,N);y=zeros(N,1);r=zeros(N,1);

A(1,1)=1;r(1)=0.5; %boundary value in the point a
for i=2:N-1
    A(i,i-1)=-1;A(i,i)=2+h*h;A(i,i+1)=-1;
    r(i)=-h*h*exp((i-1)*h);
end

A(N,N)=1;r(N)=exp(1); %boundary value in the point b

y=A\r;%solve the system

x=a:h:b;
plot(x,y,'.r');%plot the numerical solution
hold on
x=a:0.1*h:b;
plot(x,0.5*(1+x).*exp(x),'b'); %plot the exact solution
```



h	y	y_{exact}	$\lvert y - y_{\text{exact}} \rvert$
0.0	0.500000	0.500000	0.000000
0.2	0.733839	0.732841	0.000998
0.4	1.045889	1.044277	0.001612
0.6	1.459447	1.457695	0.001752
0.8	2.004268	2.002986	0.001281
1.0	2.718281	2.718281	0.000000

Consider the nonlinear BVP (it is supposed to have a unique solution)

$$\tilde{y}'' = f(x, \tilde{y}, \tilde{y}'),\ x \in [a,b],\ \tilde{y}(a) = y_a,\ \tilde{y}(b) = y_b$$

We discretize the problem on an equidistant grid having the step h with $N+2$ nodes:

$$y_0 = y_a$$

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = f(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h}),\ i = 1, 2, ..., N$$

$$y_{N+1} = y_b$$

or

$$2y_1 - y_2 + h^2 f\left(x_1, y_1, \frac{y_2 - y_a}{2h}\right) - y_a = 0$$

$$-y_{i-1} + 2y_i - y_{i+1} + h^2 f(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h}),\ i = 2, ..., N-1$$

$$-y_{N-1} + 2y_N + h^2 f\left(x_N, y_N, \frac{y_b - y_{N-1}}{2h}\right) - y_b = 0$$

This system has a general form of a nonlinear system

$$f_1(y_1, y_2, \ldots, y_N) = 0$$
$$\ldots$$
$$f_N(y_1, y_2, \ldots, y_N) = 0$$

The Newton method can be used:

$$\mathbf{y}^{(k)} = \mathbf{y}^{(k-1)} - \mathbf{J}\left(\mathbf{y}^{(k-1)}\right)^{-1}\mathbf{F}\left(\mathbf{y}^{(k-1)}\right)$$

or

$$\mathbf{y}^{(k)} = \mathbf{y}^{(k-1)} + \boldsymbol{\varepsilon}^{(k-1)}, \quad \mathbf{J}\left(\mathbf{y}^{(k-1)}\right)\boldsymbol{\varepsilon}^{(k-1)} = -\mathbf{F}\left(\mathbf{y}^{(k-1)}\right)$$

where $\mathbf{F} = (f_1, \ldots, f_N)^T$, and the components of the Jacobian $\mathbf{J} = \dfrac{\partial(f_1, \ldots, f_n)}{\partial(y_1, \ldots, y_N)}$, in this case form a tri-diagonal matrix:

$$J(y_1, y_2, \ldots, y_N)_{ij} = \begin{cases} -1 + \frac{h}{2} f_{\tilde{y}'}\left(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h} \right), & i = j-1, \;\; j = 2, \ldots, N \\ 2 + h^2 f_{\tilde{y}}\left(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h} \right), & i = j, \;\; j = 1, \ldots, N \\ -1 + \frac{h}{2} f_{\tilde{y}'}\left(x_i, y_i, \frac{y_{i+1} - y_{i-1}}{2h} \right), & i = j+1, \;\; j = 1, \ldots, N-1 \end{cases}$$

Keller – Box Method

More general then the previous method, the Keller-Box method is also based on the Newton method (Keller (1970) and Cebeci and Bradshaw (1984)) used for boundary layer applications.

For one nonlinear equation having, the form f(x) = 0, the iteration for the root finding is:

$$x_{n+1} = x_n + \varepsilon_n$$

where x_n is the root approximation at the step n, and ε_n is the error at the same step. In order to find the value of ε_n, the usual Taylor expansion of f(x) in x_{n+1} is usedf:

$$0 = f(x_{n+1}) = f(x_n + \varepsilon_n) = f(x_n) + \varepsilon_n f'(x_n) + \ldots$$

so that,

$$\varepsilon_n \approx -\frac{f(x_n)}{f'(x_n)}$$

Thus,

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

For a system of N equations we have:

$$(x_1, x_2, \ldots, x_N)^{(n+1)} = (x_1, x_2, \ldots, x_N)^{(n)} + (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N)^{(n)}$$

$$0 = f_j\left(x_1^{(n+1)}, x_2^{(n+1)}, \ldots, x_N^{(n+1)}\right) = f_j\left(x_1^{(n)}, x_2^{(n)}, \ldots, x_N^{(n)}\right) + \sum_{i=1}^{N}\left(\frac{\partial f_j}{\partial x_i}\right)^{(n)} \varepsilon_i^{(n)}$$

$$j = \overline{1, N}$$

In order to find the unknowns $\varepsilon_i^{(n)}$, i = 1, 2, …, N, one have to solve:

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_N} \\ \cdots & & & \\ \frac{\partial f_N}{\partial x_1} & \frac{\partial f_N}{\partial x_2} & \cdots & \frac{\partial f_N}{\partial x_N} \end{bmatrix}^{(n)} \cdot \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \cdots \\ \varepsilon_N \end{bmatrix}^{(n)} = -\begin{bmatrix} f_1 \\ f_2 \\ \cdots \\ f_N \end{bmatrix}^{(n)}$$

where the matrix of the system is the Jacobian $\mathbf{J} = \frac{\partial(f_1,\ldots,f_n)}{\partial(y_1,\ldots,y_N)}$.

The derivatives in the Jacobian can be numerically evaluated for complicated functions (for a small δ (e.g. $\approx 10^{-5}$)):

$$\frac{\partial f_j}{\partial x_i} \approx \frac{f_j(x_1,\ldots,x_{j-1},x_j+\delta,x_{j+1},\ldots,x_N) - f_j(x_1,\ldots,x_{j-1},x_j,x_{j+1},\ldots,x_N)}{\delta}$$

Usually, a relaxation factor, ω, $(0 < \omega \leq 1)$ is used to avoid possible divergences:

$$(x_1, x_2, \ldots, x_N)^{(n+1)} = (x_1, x_2, \ldots, x_N)^{(n)} + \omega(\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N)^{(n)}$$

Example:

Consider the equation of the free convection from a permeable vertical plate with heat generation placed in a porous medium.(Postelnicu et al., 2000):

$$f''' + \frac{\lambda + 1}{2} f f'' - \lambda f'^2 + e^{-\eta} = 0$$

$$f(0) = -f_w \,,\; f'(0) = 1,\; f'(\infty) = 0$$

where f_w is the mass flux parameter (suction or injection), and λ = const.

The equation is transformed into a first order system of equation:

We note: $a = f,\ b = f',\ c = f''$

$$\begin{cases} a' - b = 0 \\ b' - c = 0 \\ c' + \dfrac{\lambda + 1}{2} ac - \lambda b^2 + e^{-\eta} = 0 \end{cases} \qquad \text{BC:} \quad \begin{cases} a(0) + f_w = 0 \\ b(0) - 1 = 0 \\ b(\infty) = 0 \end{cases}$$

Consider the grid:

$$\eta_1, \eta_2, \ldots, \eta_N \ (\eta_1 = 0,\ \eta_N = \eta_\infty),$$

$$h_i = \eta_{i+1} - \eta_i.$$

The unknowns are approximated in the points $\eta = \eta_{i+1/2} = \dfrac{\eta_{i+1} + \eta_i}{2}$

$$a'_{i+1/2} \approx \frac{a_{i+1} - a_i}{h_i} \qquad a_{i+1/2} \approx \frac{a_{i+1} + a_i}{2}$$

.

Thus, the system becomes:

$$\begin{cases} \dfrac{a_{i+1}-a_i}{h}-\dfrac{b_{i+1}+b_i}{2}=0 \\ \dfrac{b_{i+1}-b_i}{h}-\dfrac{c_{i+1}+c_i}{2}=0 \\ \dfrac{c_{i+1}-c_i}{h}+\dfrac{\lambda+1}{8}(a_{i+1}+a_i)(c_{i+1}+c_i)-\dfrac{\lambda}{4}(b_{i+1}+b_i)^2+e^{-\eta_{i+1/2}}=0 \end{cases}$$

for i = 1, 2,…, N-1, 3N – 3 equations and 3N unknowns

BC complete the system: $a_1 + f_w = 0$, $b_1 - 1 = 0$, $b_N = 0$

To solve the system the Newton iteration is used:

$$a_i^{(n+1)} = a_i^{(n)} + \delta a_i^{(n)}$$

keeping only the first order terms in δ:

$$\frac{\delta a_{i+1}-\delta a_i}{h}-\frac{\delta b_{i+1}+\delta b_i}{2}=-\left[\frac{a_{i+1}-a_i}{h}-\frac{b_{i+1}+b_i}{2}\right]^{(n)}$$

$$\frac{\delta b_{i+1}-\delta b_i}{h}-\frac{\delta c_{i+1}+\delta c_i}{2}=-\left[\frac{b_{i+1}-b_i}{h}-\frac{c_{i+1}+c_i}{2}\right]^{(n)}$$

$$\frac{\delta c_{i+1}-\delta c_i}{h}+\frac{\lambda+1}{8}(a_{i+1}+a_i)(\delta c_{i+1}+\delta c_i)+\frac{\lambda+1}{8}(c_{i+1}+c_i)(\delta a_{i+1}+\delta a_i)-$$

$$-\frac{\lambda}{2}(b_{i+1}+b_i)(\delta b_{i+1}+\delta b_i)=-\left[\frac{c_{i+1}-c_i}{h}+\frac{\lambda+1}{8}(a_{i+1}+a_i)(c_{i+1}+c_i)-\frac{\lambda}{4}(b_{i+1}+b_i)^2+e^{-\eta_{i+1/2}}\right]^{(n)}$$

BC:

$$\delta a_1 = -a_1^{(n)} + f_w\ ,\quad \delta b_1 = 1 - b_1^{(n)},\quad \delta b_N = -\,b_N^{(n)}$$

In a matrix form the above system is given by:

$$
\begin{bmatrix}
1 & 0 & 0 & & & & & & \\
0 & 1 & 0 & & & & & & \\
-\frac{1}{h} & -\frac{1}{2} & 0 & \frac{1}{h} & -\frac{1}{2} & 0 & & & \\
0 & -\frac{1}{h} & -\frac{1}{2} & 0 & \frac{1}{h} & -\frac{1}{2} & & & \\
\frac{\lambda+1}{8}(c_2+c_1) & -\frac{\lambda}{2}(b_2+b_1) & \frac{\lambda+1}{8}(a_2+a_1)-\frac{1}{h} & \frac{\lambda+1}{8}(c_2+c_1) & -\frac{\lambda}{2}(b_2+b_1) & \frac{\lambda+1}{8}(a_2+a_1)+\frac{1}{h} & & & \\
 & & & & & & & & \\
\dots & \dots & \dots & \dots & \dots & \dots & & & \\
 & & -\frac{1}{h} & -\frac{1}{2} & 0 & \frac{1}{h} & -\frac{1}{2} & 0 & \\
 & & 0 & \frac{1}{h} & -\frac{1}{2} & 0 & \frac{1}{h} & -\frac{1}{2} & \\
 & & \frac{\lambda+1}{8}(c_N+c_{N-1}) & -\frac{\lambda}{2}(b_N+b_{N-1}) & \frac{\lambda+1}{8}(a_N+a_{N-1})-\frac{1}{h} & \frac{\lambda+1}{8}(c_N+c_{N-1}) & -\frac{\lambda}{2}(b_N+b_{N-1}) & \frac{\lambda+1}{8}(a_N+a_{N-1})+\frac{1}{h} & \\
 & & & 0 & 1 & 0 & & &
\end{bmatrix}
\times
$$

$$
\begin{bmatrix}
\delta a_1 \\ \delta b_1 \\ \delta c_1 \\ \dots \\ \dots . \\ \dots \\ \\ .. \\ \\ \dots \\ \delta a_N \\ \delta b_N \\ \delta c_N
\end{bmatrix}^{(n)}
=
\begin{bmatrix}
-a_1 + f_w \\ 1 - b_1 \\ (r_1)_1 \\ (r_2)_1 \\ (r_3)_1 \\ \\ \\ \\ \\ (r_1)_{N-1} \\ (r_2)_{N-1} \\ (r_3)_{N-1} \\ -b_N
\end{bmatrix}^{(n)}
$$

$$(r_1)_i = -\left[\frac{a_{i+1}-a_i}{h} - \frac{b_{i+1}+b_i}{2}\right]$$

$$(r_2)_i = -\left[\frac{b_{i+1}-b_i}{h} - \frac{c_{i+1}+c_i}{2}\right]$$

$$(r_3)_i = -\left[\frac{c_{i+1}-c_i}{h} + \frac{\lambda+1}{8}(a_{i+1}+a_i)(c_{i+1}+c_i) - \frac{\lambda}{4}(b_{i+1}+b_i)^2 + e^{-\eta_{i+1/2}}\right]$$

By solving the system the unknowns

$$\delta a_i^{(n)}, \delta b_i^{(n)}, \delta c_i^{(n)}, i = 1, 2, \ldots, N$$

are obtained and thus, one can calculate the next iterations:

$$a_i^{(n+1)}, b_i^{(n+1)}, c_i^{(n+1)}$$

The procedure continues until the desired accuracy is obtained

Different methods for solving the system can be used.

The routine bv4c from Matlab

The `bvp4c` solver deals with BVP problems defined as

$$\frac{dy}{dx} = f(x, y), \quad f : [a, b] \times \mathbf{R}^d \rightarrow \mathbf{R}^d, \; d \geq 2$$

along with the BC:

$$g(y(a), y(b), p) = 0$$

where p is a parameter that have to be determined.

The syntax of the `bvp4c` solver has the syntax:

```
sol = bvp4c(odefun,bcfun,solinit)
sol = bvp4c(odefun,bcfun,solinit,options)
sol = bvp4c(odefun,bcfun,solinit,options,p1,p2...)
```

where

`odefun` – specify the right hand term of the system

`bcfun` – funcţia $g(y(a), y(b), p)$ specifying the BC

```
dydx = odefun(x,y,p,p1,p2,...)
res = bcfun(ya,yb,p,p1,p2,...)
```

where `p` is the unknown parameter specified in BC and `p1,p2,...,` are parameters transmitted to `odefun` and `bcfun,` and `ya,`yb are the vectors $y(a)$ and $y(b)$.

`solinit` – is a structure with the fields

`solinit.x` (`a= solinit.x(1)` and `b = solinit.x(end)`)

and

`solinit.y` where `solinit.y(:,i)` is an initial approximation of the solution in the nodes `solinit.x(i)`.

`solinit` can be initialized using the function `bvpinit`.

`options` –permits the setting of some of the solver parameters using `(bvpset)`. After `options` values of the parameters `p1, p2, ...` sent to `odefun` and `bcfun` can be specified.

Example: Solve the equations of the heat transfer generated by catalytic reactions in a channel.

$$\theta'' + K\, e^{\theta} = 0$$

$$\theta(0) = r_T\,, \theta(1) = -r_T$$

where K is the Frank-Kamenetskii's number.

The Matlab program:

```
function [X,T,Tp]=ChimChannelBVP
a=0;  %left limit of the channel
b=1;  %right limit of the channel
K=3;  %Frank-Kamenetskii's number
rT=1; %value of the temperature in a(= 0);
```

```
solinit.x=linspace(0,1,101);
solinit.y=[solinit.x*rT;solinit.x*(-rT)];%solution initialization

options=bvpset('AbsTol', 1e-12);

sol = bvp4c(@chim_ode,@chim_bc,solinit,options,K,rT);

X=sol.x;T=sol.y(1,:);Tp=sol.y(2,:);
plot(X,T)

function dydx = chim_ode(x,y,K,rT)
dydx = [  y(2)
         -K*exp(y(1)) ];

function res = chim_bc(ya,yb,K,rT)
res = [ ya(1) - rT
        yb(1)+rT];
```