



12. INITIAL- / BOUNDARY VALUE PROBLEMS

12.1.1 Remember: what we saw during the last lesson

Euler and Runge-Kutta methods for solving ODEs

Predator-prey model using a system of ODEs

Anharmonic pendulum writing a 2nd order ODE as a system with two ODEs

12.1.2 Overview: what you will learn today

Boundary value problem

Solution by *shooting* and *finite differences*

50 theory questions for the exam

12.2 Initial- / Boundary value problems (NAM 8.7, H 10.1)

Take Newton's second law of motion for example

$$\begin{pmatrix} \mathbf{x}' \\ \mathbf{v}' \end{pmatrix} = \begin{pmatrix} \mathbf{v} \\ \frac{1}{m}\mathbf{F} \end{pmatrix} \Leftrightarrow \mathbf{x}'' = \mathbf{f}(t, \mathbf{x}, \mathbf{x}')$$

This second order problem is specified with 2 additional constraints.

Initial value problems satisfy 2 *initial conditions (IC)* at the beginning of the interval

$$\mathbf{x}(t_0) = \mathbf{x}_0; \quad \mathbf{v}(t_0) = \mathbf{v}_0$$

Knowing the initial position and velocity of a bullet, where is it going to land?

Boundary value problems satisfy 2 *boundary conditions (BC)* at the boundaries

$$\mathbf{x}(a) = \mathbf{x}_a; \quad \mathbf{x}(b) = \mathbf{x}_b$$

What is the trajectory of a bullet shot from a to b ?

Boundary conditions can be specified as the value $\mathbf{x}(a)$ (Dirichlet), the derivative $\mathbf{x}'(a)$ (Neuman) or a linear combination of the two $\mathbf{x}(a) + \alpha\mathbf{x}'(a) = \beta$ (Robin).

12.3 Shooting method for boundary value problems (NAM 8.7.1, H 10.3)

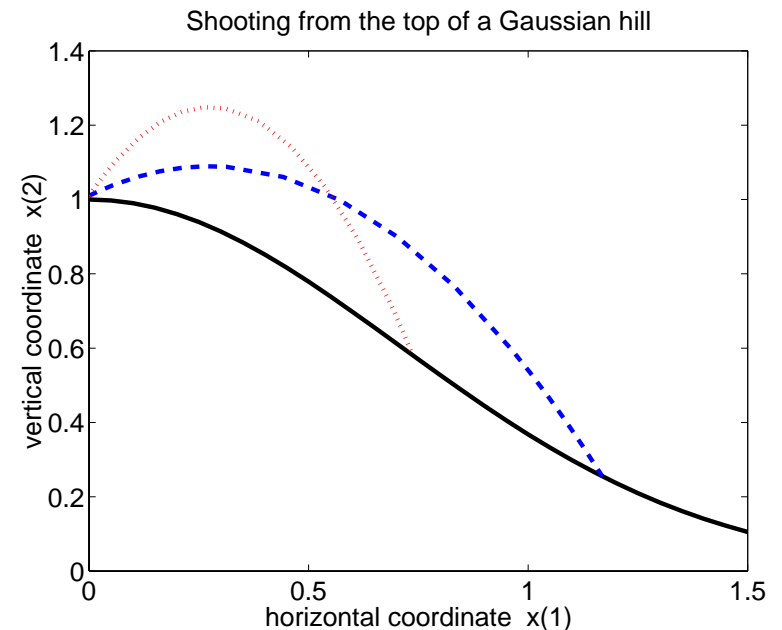
The method solves initial value problems, adjusting the IC to approach the BC.

Shoot from a with a guess velocity; if the bullet falls short, increase the velocity, etc

Example determine the canon angle to shoot from the hill $(0, 1)$ to the target $(1, 1/e)$.

```
function varargout = shoot(t,X,flag);
%
switch flag
  case '',          varargout{1} = dydt(t,X);
  case 'events',   [varargout{1:3}] = events(t,X);
  otherwise,       error(['Unknown flag.']);
end
%
function f=dydt(t,X)
  f=[X(3); X(4); 0; -9.81];    % f=(vx,vy,ax,ay)
%
function [value,isterminal,direction]=events(t,X)
  shapeMountain=exp(-X(1)^2);  % ground elevation
  value = X(2)-shapeMountain;  % present height
  isterminal = 1;              % stop if ground
  direction = -1;              % going down
```

```
>> t=[0 10]; v0=2.5; options=odeset('events','on','outputsel',[2],'refine',5)
>> a=pi/6; X=[0 1.01 v0*cos(a) v0*sin(a)]; [T1,Y1]=ode23(@shoot,t,X,options);
>> a=pi/3; X=[0 1.01 v0*cos(a) v0*sin(a)]; [T2,Y2]=ode23(@shoot,t,X,options);
>> x=0:0.01:1.5; plot(x,exp(x.^2), Y1(:,1),Y1(:,2),'--',Y2(:,1),Y2(:,2),':')
```



Here `events(t,X)` is used to stop the integration when the bullet hits the ground; experimentally, the canon angle is found to be somewhere $\alpha \in [\pi/6; \pi/3]$.

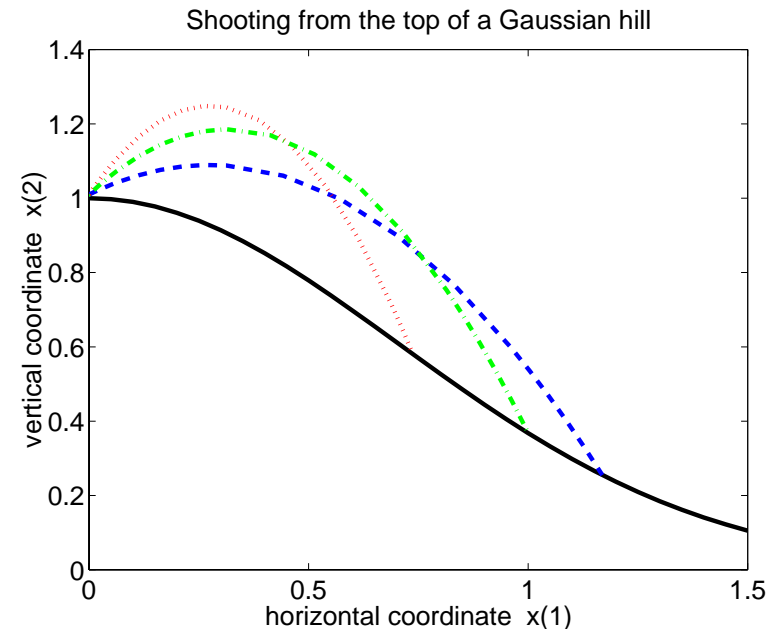
The solution is computed by solving a non-linear equation $f(\alpha) = x^{\text{end}}(\alpha) - x_b = 0$:
the *secant method* is most effective when $f'(\alpha)$ is not known explicitly.

In Matlab use the same driver `shoot()` to search with the secant method:

```
a=pi/6; b=pi/3; tol=pi/180; t=[0 10]; v0=2.5; xtarget=1;
options=odeset('events','on','outputsel',[2],'refine',5)
X=[0 1.01 v0*cos(a) v0*sin(a)]; [T,Y]=ode23(@shoot,t,X,options); fa=Y(end,1)-xtarget;
X=[0 1.01 v0*cos(b) v0*sin(b)]; [T,Y]=ode23(@shoot,t,X,options); fb=Y(end,1)-xtarget;
while abs(b-a)>tol
    x=b-fb*(b-a)/(fb-fa);
    X=[0 1.01 v0*cos(x) v0*sin(x)]; [T,Y]=ode23(@shoot,t,X,options); fx=Y(end,1)-xtarget;
    a=b; fa=fb; b=x; fb=fx;
    sol=180/pi*[a x b]
end
```

a	x	b
60.0000	41.7418	41.7418
41.7418	46.4760	46.4760
46.4760	48.2975	48.2975
48.2975	47.9854	47.9854

==> alpha = 48.0 +/- 0.3 degree



Linear boundary value problems can be solved with a system using only two trials.

12.4 Finite differences for boundary value problems (NAM 8.7.2, H 10.4)

The method divides the interval where the solution is sought into equidistant pieces

$$x_i = x_a + i \cdot h, \quad h = (x_b - x_a)/N$$

using finite difference (FD) to approximate derivatives with unknowns $y_i = y(x_i)$

$$y'_i \approx \frac{y_{i+1} - y_{i-1}}{2h}, \quad y''_i \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

here for example with a precision $\mathcal{O}(h^2)$.

Linear example: solve the boundary value problem in the interval $x \in [0; 4]$

$$y'' + a\sqrt{x}y' - (6 - y)x = 0, \quad y(0) = 1, y(4) = -1$$

for parameter values $a \in \{-3, -2, -1, 0, 1, 2\}$. Discretize using a uniform mesh $x_i = x_a + i \cdot h$, $x_a = 0$, $x_b = 4$, $h = 4/N$ where N is the number of intervals. Insert the FD approximation and evaluate for every mesh point x_i inside

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + a\sqrt{x_i} \frac{y_{i+1} - y_{i-1}}{2h} - (6 - y_i)x_i = 0 \quad \forall i = 2 \dots, N - 1$$

This yields a tridiagonal linear system $\mathbf{A}y \equiv (\mathbf{E} + \mathbf{D} + \mathbf{F})y = \mathbf{b}$

$$\underbrace{\left(1 - \frac{ha}{2}\sqrt{x_i}\right)}_{e_i} y_{i-1} - \underbrace{(2 - h^2 x_i)}_{d_i} y_i + \underbrace{\left(1 + \frac{ha}{2}\sqrt{x_i}\right)}_{f_i} y_{i+1} = \underbrace{6h^2 x_i}_{b_i} \quad \forall i$$

Boundary conditions are usually imposed by replacing the corresponding equation

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ e_2 & d_2 & f_2 & 0 & \dots & 0 \\ 0 & e_3 & d_3 & f_3 & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & 0 \\ 0 & \dots & 0 & e_{n-1} & d_{n-1} & f_{n-1} \\ 0 & \dots & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} y(a) \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ y(b) \end{pmatrix}$$

and substituting the columns to keep the matrix symmetric if it was in the beginning

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & d_2 & f_2 & 0 & \dots & 0 \\ 0 & e_3 & d_3 & f_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & e_{n-1} & d_{n-1} & 0 \\ 0 & \dots & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} y(a) \\ b_2 - e_2 y(a) \\ b_3 \\ \vdots \\ b_{n-1} - f_{n-1} y(b) \\ y(b) \end{pmatrix}$$

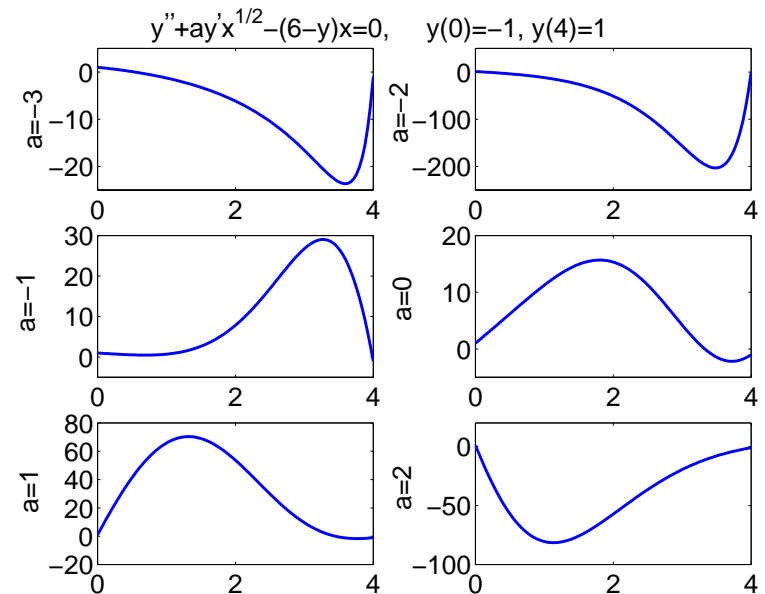
The system is can be solved directly using tridia or solvers for sparse systems.

In Matlab, this yields

```

X=[0; 4]; Y=[1; -1];           % boundaries
for k=1:6
    a=k-4;                       % parameter a
    n=100; h=diff(X)/(n-1);      % n-1 intervals
    x=linspace(X(1),X(2),n)';
    e=1-0.5*h*a*sqrt(x);         % assemble
    d=-(2-h^2*x);               % tridiagonal
    f=1+0.5*h*a*sqrt(x);        % matrix
    A=spdiags([[e(2:n);0] d [0; f(1:n-1)]],-1:1,n,n);
    b=6*h^2*x;                   % assemble RHS
    A(1,1)=1.; A(1,2)=0; b(1)=Y(1); % BC
    A(n,n)=1.; A(n,n-1)=0; b(n)=Y(2);
    y=A\b;                        % solve
    subplot(3,2,k); plot(x,y),
    l=sprintf('a=%i',a), ylabel(1)
end

```



Peer Teaching (2 × 1 minutes to think, explain to your neighbour and vote)

Computational cost. Examine the number of operations required to solve a linear boundary value problem in 1 dimension. Which method is more efficient?

← shooting

↑ about equal

↓ cannot say

→ finite differences

Non-linear example: solve the boundary value problem in the interval $x \in [0; 4]$

$$y'' + cy^2 - (6 - y)x = 0 \quad \text{with} \quad y(0) = 1, y(4) = -1, \quad \text{for} \quad c = 0.1$$

Discretize using finite differences on a uniform mesh

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + cy_i^2 - (6 - y_i)x_i = 0 \quad \forall i = 2 \dots, N - 1$$

$$y_{i-1} + G(y_i) + y_{i+1} - b_i = 0 \quad \text{with} \quad \begin{cases} G(y_i) = -(2 - h^2x_i)y_i + ch^2y_i^2 \\ b_i = -6h^2x_i \end{cases}$$

This non-linear system of the form $F(y) = 0$ is solved after inserting the boundary conditions using the Newton method $y^{\text{new}} = y^{\text{old}} + \delta y$, with $J\delta y = -f(y^{\text{old}})$.

The Jacobian matrix $J_{i,j} = \partial F_i / \partial y_j$ is tridiagonal

$$\mathbf{J} = \begin{pmatrix} G'(y_1) & 1 & 0 & \dots & 0 \\ 1 & G'(y_2) & 1 & \dots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \dots & 1 & G'(y_{n-1}) & 1 \\ 0 & \dots & 0 & 1 & G'(y_n) \end{pmatrix}$$

with $G'(y_i) = -(2 - h^2x_i) + 2ch^2y_i$. What should we take for the initial guess?

Method: gradually increase the non-linearity

With strongly non-linear equations, **first switch-off the non-linear term** and calculate a linear solution for the initial guess. Then **gradually switch-on the non-linearity**, using increasingly non-linear solutions to start the Newton iterations.

In Matlab this can be implemented as

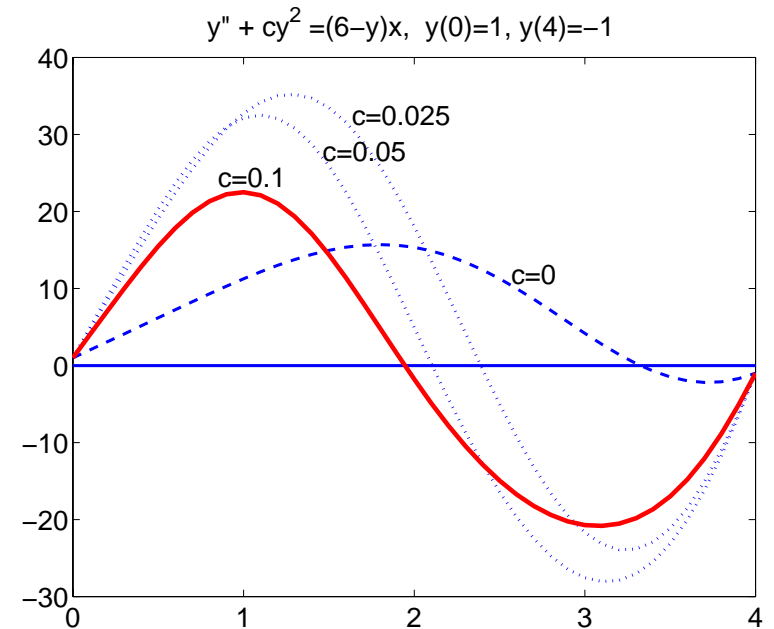
```
% randolinj,  $y''+c*y^2=(6-y)x$ ;  $y(0)=1$ ;  $y(4)=-1$ 
% bandmatrix, non-linear system with Newton's method
clear, clf
x0=0; xend= 4; x=[0 xend];
y0=1; yend=-1; y=[0 0 ]; plot(x,y); hold on
title('y" + cy^2 =(6-y)x,  y(0)=1, y(4)=-1 ')
ntot=40;
h=xend/ntot;  n=ntot-1; x=h*(1:n)';

% Solve linear problem with c=0 for an initial guess
c=0
dia=-(2-h^2*x); one=ones(n-1,1); b=6*h^2*x;
b(1)=b(1)-y0;  b(n)=b(n)-yend;
y=tridia(dia,one,one,b);
X=[0; x; xend]; Y=[y0; y; yend]; plot(X,Y,'--')
[ymax,ix]=max(y); text(x(ix),ymax,'c=0'), drawnow
```

```

% Gradually increase c using Newton's method
% with previous solutions as initial guess
cvec=[0.025 0.05 0.1]; % experimentally
for nr=1:3
    c=cvec(nr), err=1; iter=0;
    while err>1e-4 & iter<15
        G=diag(y+c*h^2*y.^2); Gprim=diag+2*c*h^2*y;
        F=[y0; y(1:n-1)]+G+[y(2:n);yend]-6*h^2*x;
        dy=-tridia(Gprim,one,one,F); % J dy=-f(y)
        err=norm(dy,inf)/norm(y,inf)
        y=y+dy; iter=iter+1;
    end
    if iter==15, disp('No convergence'),break,end
    Y=[y0; y; yend]; plot(X,Y,':')
    [ymax,ix]=max(y); ymax
    text(x(ix),ymax,['c=' num2str(c)]), drawnow
end
plot(X,Y,'r')

```



The non-linearity is very strong as can be seen from the plot.