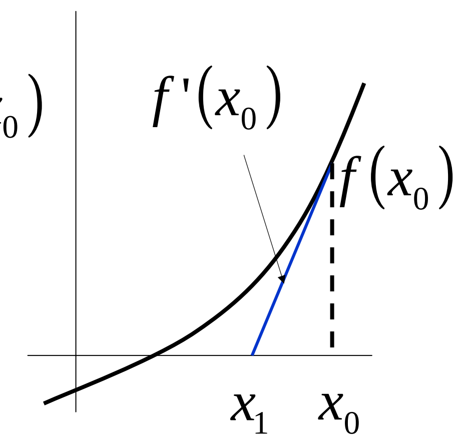
Root Finding

The solution of
nonlinear equations
and systems

Vageli Coutsias, UNM, Fall ‘02

The Newton-Raphson iteration
for locating zeros

$$x_1 = x_0 - f(x_0) / f'(x_0)$$



Example: finding the square root

$$f(x) = x^2 - a$$

$$f'(x) = 2x$$

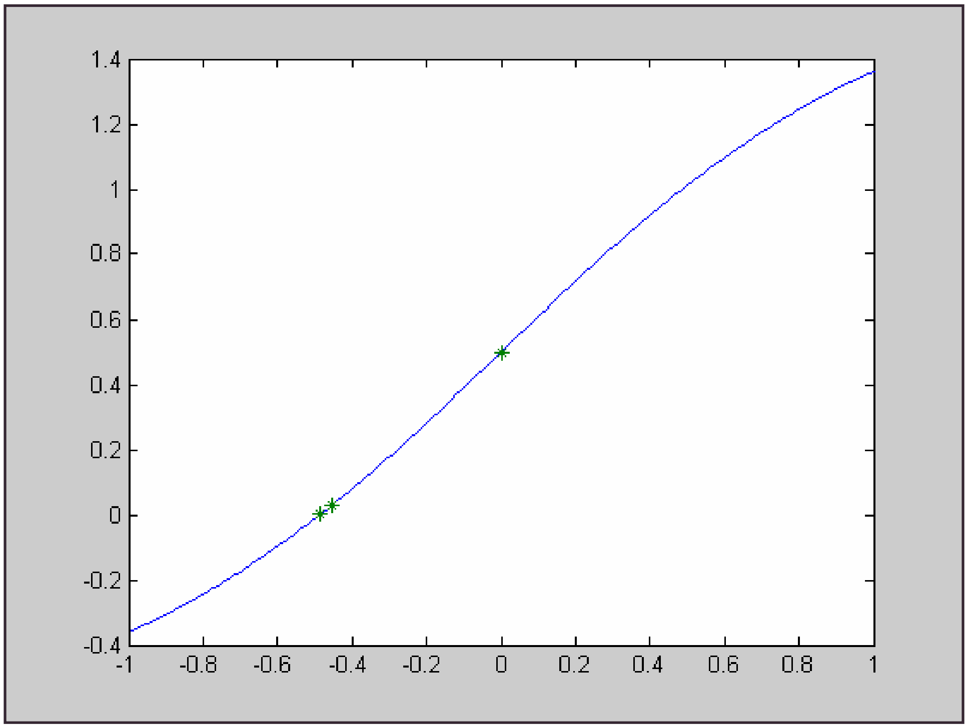
$$x_1 = x_0 - \frac{x_0^{\ 2} - a}{2x_0} = x_0 - \frac{1}{2}\left(x_0 - \frac{a}{x_0}\right)$$

Details: initial iterate must be 'close' to solution for method to deliver its promise of quadratic convergence (the number of correct bits doubling with every step)

```
function x = sqrt1(A)
if A < 0 break;end
if A == 0;  x = 0;
else
   TwoPower = 1;
    m = A;
  while m >=  1, m = m/4; TwoPower = 2*TwoPower ;end
  while m < .25, m = m*4; TwoPower =  TwoPower/2;end
  x = (1+2*m)/3;
  for k = 1:4
      x = (x+ (m/x))/2;
  end
  x = x*TwoPower;
end
```

```
clc; close all; clear all; format long e;      fname = 'func';
a = input('Enter a value:'); b = input('Enter b value:');
xc = input('Enter starting value:'); xmax = xc; xmin = xc;
fc = feval(fname,xc,a,b); delta = .0001;
fpc = (feval(fname,xc+delta,a,b)-fc)/delta;
k=0; disp(sprintf('k       x       fval         fpval   '))
while input('Newton step? (0=no, 1=yes)')
    k=k+1; x(k) = xc; y(k) = fc;
    xnew = xc – fc / fpc;  xc   = xnew;
    fc = feval(fname,xc,a,b);
    fpc = (feval(fname,xc+delta,a,b)-fc)/delta;
disp(sprintf('%2.0f  %20.15f %20.15f  %20.15f',k,xc,fc,fpc))
    if xmax <= x(k);  xmax = x(k); end
   if  xmin  >= x(k);  xmin = x(k);  end
end
x0 = linspace(xmin,xmax,201); y0 = feval(fname,x0,a,b); plot(x0,y0,'r-',x,y,'*')
```

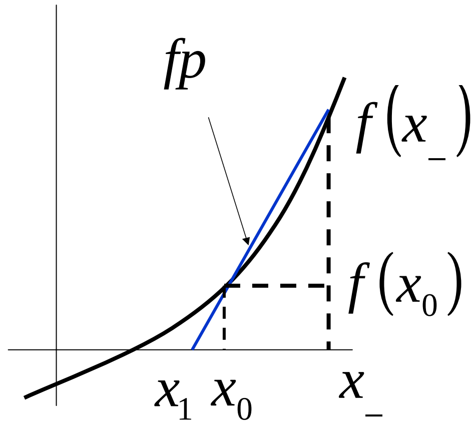
```
Enter a value:.1
Enter b value:.5
Enter starting value:0
k                    x                            fval
                                  Newton step? (0=no, 1=yes)1
1    -0.454545455922915    0.028917256044793
2    -0.486015721951951    0.000349982958035
3    -0.486406070359515    0.000000068775315
4    -0.486406147091071    0.000000000002760
5    -0.486406147094150    0.000000000000000
                                  Newton step? (0=no, 1=yes)0
```




```
% script zeroin
% uses matlab builtin rootfinder "FZERO"
% to find a zero of a function 'fname'
close all; clear all; format long
%fname = 'func'; % user defined function-can also give as @func
%fpname='dfunc'; % user def. derivative-not required by FZERO
del = .0001; % function value limiting tolerance
a = input('Enter a value:');  b = input('Enter b value:');
x0 = linspace(-10,10,201); y0 = feval(@func,x0,a,b);
xc = input('Enter starting value:');
%root = fzero(@func,xc,.0001,1,a,b); % grandfathered format
OPTIONS=optimset('MaxIter',100,'TolFun',del,'TolX',del^2);
root = fzero(@func,xc,OPTIONS,a,b)
y = feval(@func,root,a,b)
plot(x0,y0,root,y,'r*')
```

The Secant iteration for locating zeros

$$x_1 = x_0 - \left(x_{\_} - x_0\right)\frac{f\left(x_0\right)}{f\left(x_{\_}\right) - f\left(x_0\right)}$$




```
% script SEC.M: uses secant method to find zero of FUNC.M
close all;  clear all;   clc;  format long e
fname = 'func'; a = input('Enter a:');b = input('Enter b:');
xc = input('Enter starting value:');   fc = feval(fname,xc,a,b);
del = .0001; k=0; disp(sprintf('k       x       fval       fpval   '))
fpc = (feval(fname,xc+delta,a,b)-fc)/delta;
while input('secant step? (0=no, 1=yes)')
   k=k+1; x(k) = xc; y(k) = fc;  f_ = fc;
   xnew = xc - fc/fpc;
   x_   = xc; xc   = xnew;
   fc = feval(fname,xc,a,b);  fpc= (fc - f_)/(xc-x_);
disp(sprintf('%2.0f    %20.15f %20.15f    %20.15f',k,xc,fc,fpc))
end
    if x(1) <= x(k); xa = floor(x(1)-.5);    xb = ceil(x(k)+.5);
   else;                xb = floor(x(1)-.5);     xa = ceil(x(k)+.5);    end
x0=linspace(xa,xb,201);y0=feval(fname,x0,a,b);plot(x0,y0,x,y,'r*')
```

Application: solution of a BVP (Boundary Value Problem)

$$\frac{d^2y}{dx^2} + k^2 y = 0$$

$$y(0) = 0;\; y(\pi) + y'(\pi) = 0$$

Solution

$$y(x) = A\sin(kx)$$

$$\sin(k\pi) + k\cos(k\pi) = 0$$

Problem (HW 5)

Solve previous equation for the smallest suitable nonzero value of k (eigenvalue) using

(a) The Newton method

(script ‘NEWT.M’)

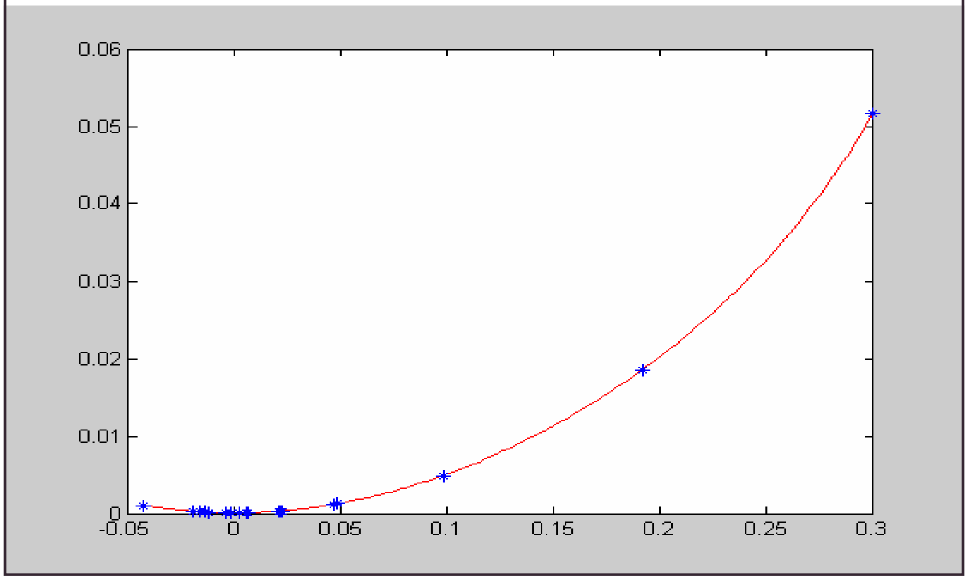
(b) The built-in Newton method

(script ‘ZEROIN.M’)

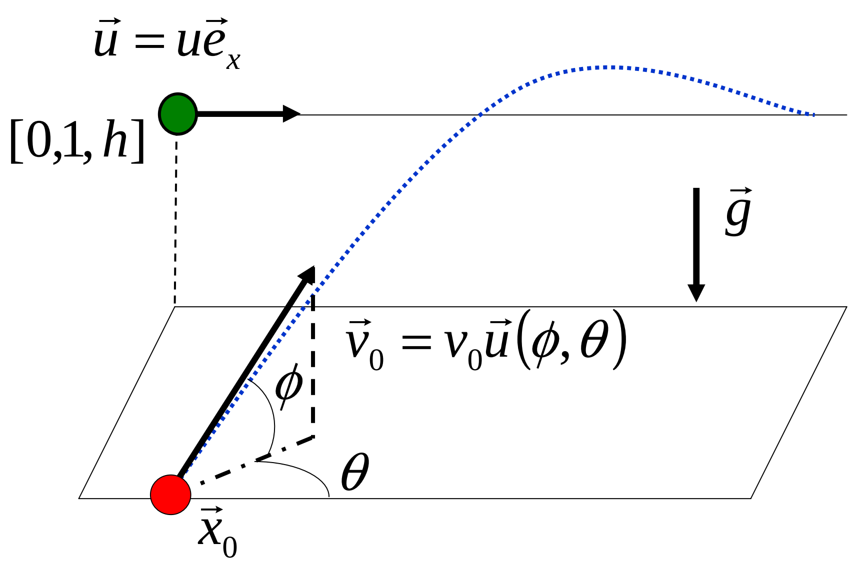
(c) The secant method

(script ‘SEC.M’)

```
function f = func(x,a,b)            % trap near x = 0
% for interesting behavior, use a = .1, b = .0001, x0 = .4
  f = .5*x.^2+ 100*x.^8 - a*x.^16 + b;
```



Newton's method for nonlinear systems:
"shoot down the bogey!"



Equations of motion

invader

$$x_1(t) = ut$$
$$y_1(t) = 1$$
$$z_1(t) = h$$

defender

$$x_2(t) = x_0 + v_0 \cos\phi \cos\theta (t - t_0)$$
$$y_2(t) = v_0 \cos\phi \sin\theta (t - t_0)$$
$$z_2(t) = v_0 \sin\phi (t - t_0) - \frac{1}{2} g (t - t_0)^2$$

Newton's method for 2d or 3d functions

$$\vec{F}(\vec{u}) = 0 \Rightarrow \vec{F}(\vec{u}_n) \xrightarrow{n \to \infty} 0$$

$$\vec{F}(\vec{u}_n + \delta\vec{u}_n) = F(\vec{u}_n) + \delta\vec{u}_n \cdot \frac{\partial \vec{F}}{\partial \vec{u}} = 0$$

$$\Rightarrow \delta\vec{u}_n = -\left(\frac{\partial \vec{F}}{\partial \vec{u}}\right)_{\vec{u}_n}^{-1} F(\vec{u}_n)$$

For ballistic example:

$$\vec{F}(\vec{u}) = \begin{pmatrix} F_1 \\ F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \end{pmatrix} = \begin{pmatrix} x_0 + v_0 \cos\phi \cos\theta (t - t_0) - ut \\ v_0 \cos\phi \sin\theta (t - t_0) - 1 \\ v_0 \sin\phi (t - t_0) - \frac{1}{2} g (t - t_0)^2 - h \end{pmatrix}$$

$$\vec{u} = \begin{pmatrix} t \\ \theta \\ \phi \end{pmatrix} \rightarrow \frac{\partial \vec{F}}{\partial \vec{u}} = \begin{pmatrix} \frac{\partial F_1}{\partial t} & \frac{\partial F_1}{\partial \theta} & \frac{\partial F_1}{\partial \phi} \\ \frac{\partial F_2}{\partial t} & \frac{\partial F_2}{\partial \theta} & \frac{\partial F_2}{\partial \phi} \\ \frac{\partial F_3}{\partial t} & \frac{\partial F_3}{\partial \theta} & \frac{\partial F_3}{\partial \phi} \end{pmatrix}$$

$$\frac{\partial \vec{F}}{\partial \vec{u}} =$$

$$\begin{pmatrix} v_0 \cos\phi \cos\theta - u & -v_0 \cos\phi \sin\theta (t - t_0) & -v_0 \sin\phi \cos\theta (t - t_0) \\ v_0 \cos\phi \sin\theta & v_0 \cos\phi \cos\theta (t - t_0) & -v_0 \sin\phi \sin\theta (t - t_0) \\ v_0 \sin\phi - (t - t_0) & 0 & v_0 \cos\phi (t - t_0) \end{pmatrix}$$

Iterate, starting with a suitable guess, say:

$$\vec{u}_0 = \begin{pmatrix} t_0 \\ \theta_0 \\ \phi_0 \end{pmatrix} = \begin{pmatrix} t_0 \\ \pi / 2 \\ \pi / 4 \end{pmatrix}$$

$$\delta \vec{u}_n = -\left(\frac{\partial \vec{F}}{\partial \vec{u}} \right)_{\vec{u}_n}^{-1} F(\vec{u}_n)$$

Project 5-a

1

Add a small touch of realism: the cannon requires a finite amount of time to be turned to the firing direction. The time depends on the exact rotation. Thus, you must figure out an optimal rotation axis and perform a rotation that is as fast as possible (given the fixed rotational speed of the cannon). The ballistic solution should take account of the time required for the rotation, since the firing angles determine the time required from the moment you decide to fire to the moment the cannon can actually fire to the moment when intercept is achieved.

2

Specifically, the simulation universe is a cubic box with an edge of length of 10 miles. At time t=0 a bogey enters at the point (x,y,z) = (0,10,h) moving in the positive x-direction with speed u.

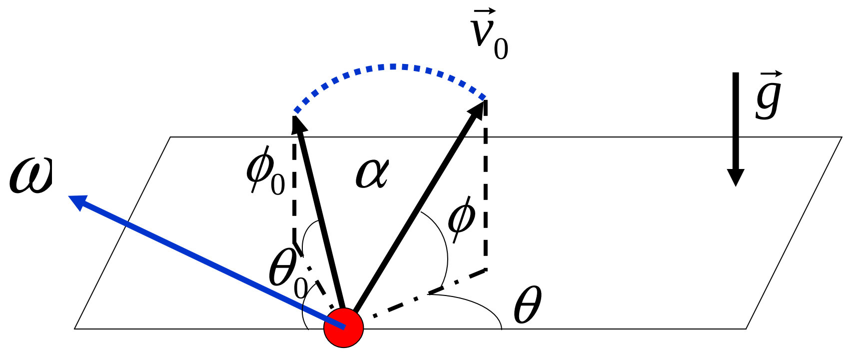
The canon is situated at the point (x0, 0, 0) and points initially at a direction characterized by angles (ϕ_0, θ_0). As the bogey progresses your script should ask you whether you want to shoot it down. If you answer ``no'' (0), it should advance the bogey by 1 'click' and ask again. If you answer ``yes'' (1) the shoot-down process should get started. The cannon should advance from its idle position to its firing position by the best rotation (which you must compute!). Its motion should be shown by a succession of images, where care should be taken to produce the sense of motion (e.g. exhibit current position as an arrow and previous position as a broken line). When the cannon reaches its firing position it should fire a projectile. From that point on the cannon stops and the program should display the projectile and bogey by appropriate symbols, advancing along their respective trajectories each at its own speed. When contact is achieved, an appropriate symbol should be displayed and statistics printed (e.g. time and location of intercept, firing angles etc.). 3

If an intercept solution is impossible at the point you decide to fire, the program ought to output a failure comment (such as: ``intercept impossible-abort launch'' etc.) and the bogey should complete its trajectory exiting the domain (or hitting a target, if you feel like

adding further graphic realism to your simulation. 4
In that case, the target is the entire plane x=10. (Optional: the target is located at the point (x,y,z) = (10,5,.1) and the bogey enters as before but it has a straight line motion heading for the target. Once you specify the trajectory of the bogey, all else works as in the trivial case of x-parallel motion! For this case, the entry point is a random point on the plane x=0, and the speed is directed along the straight line joining the entry point and the target.)

You should turn in a “cartoon” of your simulation, with a succession of images in a “comic book” format. Your output should give a complete and clear picture of what is happening as time advances.

5



Aiming initially at angles: (ϕ_0, θ_0)
Must turn to angles: (ϕ, θ)
Total rotation by angle: α
The time required equals: $T_{turn} = \alpha / \omega$
Given angular speed: ω

Initial conditions and other constants

6

$$u_0 = (1 + 4 * rand(1)) * .03 (mi / \sec)$$

$$h = .1 + 9.9 * rand(1)$$

$$v_0 = .4 (mi / \sec)$$

$$\omega = \frac{2\pi}{60} \sec^{-1}$$

$$g = 32.2\, ft / \sec^2$$

Fixed-value parameters 7

x_0, ϕ_0, θ_0 are built-in constants

Success condition

$$\left\|\vec{F}(\vec{u})\right\| = \sqrt{F_1^2 + F_2^2 + F_3^2} \le 5\, ft$$

Discussion: modifying the equations to account for the turning time 8

$$t_0 = T_0 + T_{turn}(\phi, \theta)$$

$$T_{turn}(\phi, \theta) = \frac{\alpha(\phi, \theta)}{\omega}$$

$$\cos(\alpha) = u(\phi, \theta) \cdot u(\phi_0, \theta_0)$$

$$u(\phi, \theta) = [\cos\phi \cos\theta, \cos\phi \sin\theta, \sin\phi]$$

Since the launch time now depends on the angles, the partial derivatives with respect to the angles must now reflect that dependence. For example:

$$\frac{\partial F_1}{\partial \theta}$$

$$= \frac{\partial\left(x_0 + v_0 \cos\phi \cos\theta\left(t - t_0\right) - ut\right)}{\partial \theta}$$

$$= -v_0 \cos\phi \sin\theta\left(t - t_0\right) - v_0 \cos\phi \cos\theta \frac{\partial t_0}{\partial \theta}$$

9

$$\frac{\partial t_0}{\partial \theta} = \frac{1}{\omega}\frac{\partial \alpha}{\partial \theta}$$

10

$$\cos\alpha$$

$$= \cos\phi\cos\phi_0\left(\cos\theta\cos\theta_0 + \sin\theta\sin\theta_0\right) + \sin\phi\sin\phi_0$$

$$\frac{\partial \cos\alpha}{\partial \theta} = -\sin\alpha \frac{\partial \alpha}{\partial \theta}$$

$$\sin\alpha = \sqrt{1 - \cos^2\alpha}$$

11

$$\frac{\partial(\cos\alpha)}{\partial\theta}$$

$$= \frac{\partial(\cos\phi\cos\phi_0(\cos\theta\cos\theta_0+\sin\theta\sin\theta_0))}{\partial\theta}$$

$$= \cos\phi\cos\phi_0\frac{\partial(\cos\theta\cos\theta_0+\sin\theta\sin\theta_0)}{\partial\theta}$$

$$= \cos\phi\cos\phi_0(-\cos\theta_0\sin\theta+\sin\theta_0\cos\theta)$$

FMINBND Scalar bounded nonlinear function minimization.
X = FMINBND(FUN,x1,x2) starts at X0 and finds a local minimizer X of the
function FUN in the interval x1 < X < x2. FUN accepts scalar input X and returns
a scalar function value F evaluated at X.

SPLINE Cubic spline data interpolation.

YY = SPLINE(X,Y,XX) uses cubic spline interpolation to find YY, the values of the underlying function Y at the points in the vector XX. The vector X specifies the points at which the data Y is given. If Y is a matrix, then the data is taken to be vector-valued and interpolation is performed for each column of Y and YY will be length(XX)-by-size(Y,2).

PP = SPLINE(X,Y) returns the piecewise polynomial form of the cubic spline interpolant for later use with PPVAL and the spline utility UNMKPP.

Ordinarily, the not-a-knot end conditions are used. However, if Y contains two more values than X has entries, then the first and last value in Y are used as the endslopes for the cubic spline. Namely:

```
f(X) = Y(:,2:end-1),   df(min(X)) = Y(:,1),   df(max(X)) = Y(:,end)
```

Example:
This generates a sine curve, then samples the spline over a finer mesh:

```
x = 0:10;  y = sin(x);
xx = 0:.25:10;
yy = spline(x,y,xx);
plot(x,y,'o',xx,yy)
```

PPVAL Evaluate piecewise polynomial.

V = PPVAL(PP,XX) returns the value at the points XX of the piecewise polynomial contained in PP, as constructed by SPLINE or the spline utility MKPP.

V = PPVAL(XX,PP) is also acceptable, and of use in conjunction with FMINBND, FZERO, QUAD, and other function functions.

Example:
Compare the results of integrating the function cos and this spline:

```
a = 0; b = 10;
int1 = quad(@cos,a,b,[],[]);
x = a : b; y = cos(x); pp = spline(x,y);
int2 = quad(@ppval,a,b,[],[],pp);
```

int1 provides the integral of the cosine function over the interval [a,b] while int2 provides the integral over the same interval of the piecewise polynomial pp which approximates the cosine function by interpolating the computed x,y values.