# MATLAB

Lecture 4

# Avoiding Loops

- Example:
- Given x= sin(linspace(0,10*pi,100)), how many of the entries are positive?
- Using a loop and if/else

```
count=0;
for n=1:length(x)
    if x(n)>0
        count=count+1;
    end
end
```

Using build in functions:

```
count=length(find(x>0));
```

# Efficient Code

- **Avoid loops**
  - **This is referred to as vectorization**
- 

**Vectorized code is more efficient for MATLAB**

- **Use indexing and matrix operations to avoid loops**

# Min with matrixes

- Using min with matrices:

```
a=[3 7 5;1 9 10; 30 -1 2];
b=min(a); % returns the min of each column
 m=min(b); % returns min of entire a matrix
m=min(min(a)); % same as above
m=min(a(:));  makes a  vector, then gets min
```

- Common mistake:

```
[m,n]=find(min(a)); % think about what happens
```

# Systems of Linear Equations

Given a system of linear equations

x+2y-3z=5

-3x-y+z=-8

x-y+z=0

Construct matrices so the system is described by Ax=b

```
A=[1 2 -3;-3 -1 1;1 -1 1];
b=[5;-8;0];
```

And solve with a single line of code!

```
x=A\b;
```

x is a 3x1 vector containing the values of x, y, and z

The **** will work with square or rectangular systems.

Gives least squares solution for rectangular systems.

Solution depends on whether the system is over or underdetermined.

# Linear Equations

Calculate the determinant

```
d=det(mat);
```

mat must be square

if determinant is nonzero, matrix is invertible

Get the matrix inverse

```
E=inv(mat);
```

if an equation is of the form A*x=b with A a square matrix,

`x=A\b` is the same as `x=inv(A)*b`

# Matrix Decompositions

MATLAB has built-in matrix decomposition methods

The most common ones are

```
[V,D]=eig(X)
```

Eigenvalue decomposition

```
[U,S,V]=svd(X)
```

Singular value decomposition

```
[Q,R]=qr(X)
```

QR decomposition

# Polynomials

Many functions can be well described by a high-order polynomial

MATLAB represents a polynomials by a vector of coefficients

if vector P describes a polynomial

$ax^2$+b$x$+c$x$+d

P(1) P(2) P(3) P(4)

P=[1 0 -2] represents the polynomial $x^2$-2

P=[2 0 0 0] represents the polynomial $2x^3$

# Polynomial Operations

- **P is a vector of length N+1 describing an N-th order polynomial**

To get the roots of a polynomial

**r=roots(P)**

**r** is a vector of length **N**

Can also get the polynomial from the roots

**P=poly(r)**

**r** is a vector length **N**

To evaluate a polynomial at a point

**y0=polyval(P,x0)**

**x0** is a single value; **y0** is a single value

To evaluate a polynomial at many points

**y=polyval(P,x)**

**x** is a vector; **y** is a vector of the same size

# Polynomial Fitting

- MATLAB makes it very easy to fit polynomials to data
- Given data vectors X=[-1 0 2] and Y=[0 -1 3]

```
p2=polyfit(X,Y,2);
```

finds the best second order polynomial that fits the points (-1,0),(0,-1), and (2,3)

see help polyfit for more information

```
plot(X,Y,'o', 'MarkerSize', 10);
hold on;
x = -3:.01:3;
plot(x,polyval(p2,x), 'r--');
```

# Nonlinear Root Finding

- Many real-world problems require us to solve $f(x)=0$
- Can use **fzero**
- to calculate roots for any arbitrary function
- **fzero** needs a function passed to it.

# Nonlinear Root Finding

- Make a separate function file
- Example: **function y=myfun(x)**

  **y=cos(exp(x))+x.^2-1;**

```
x=fzero('myfun',1)
x=fzero(@myfun,1)
```

**1** specifies a point close to where you think the root is

# Nonlinear Root Finding

- **fzero**
- not always work the way we like
- or not working at all
- How we can solve **problem** **f(x)=0** “by hand”?

- A number of algorithms exist
- Two basis ones- **bisections** and **Newton** method

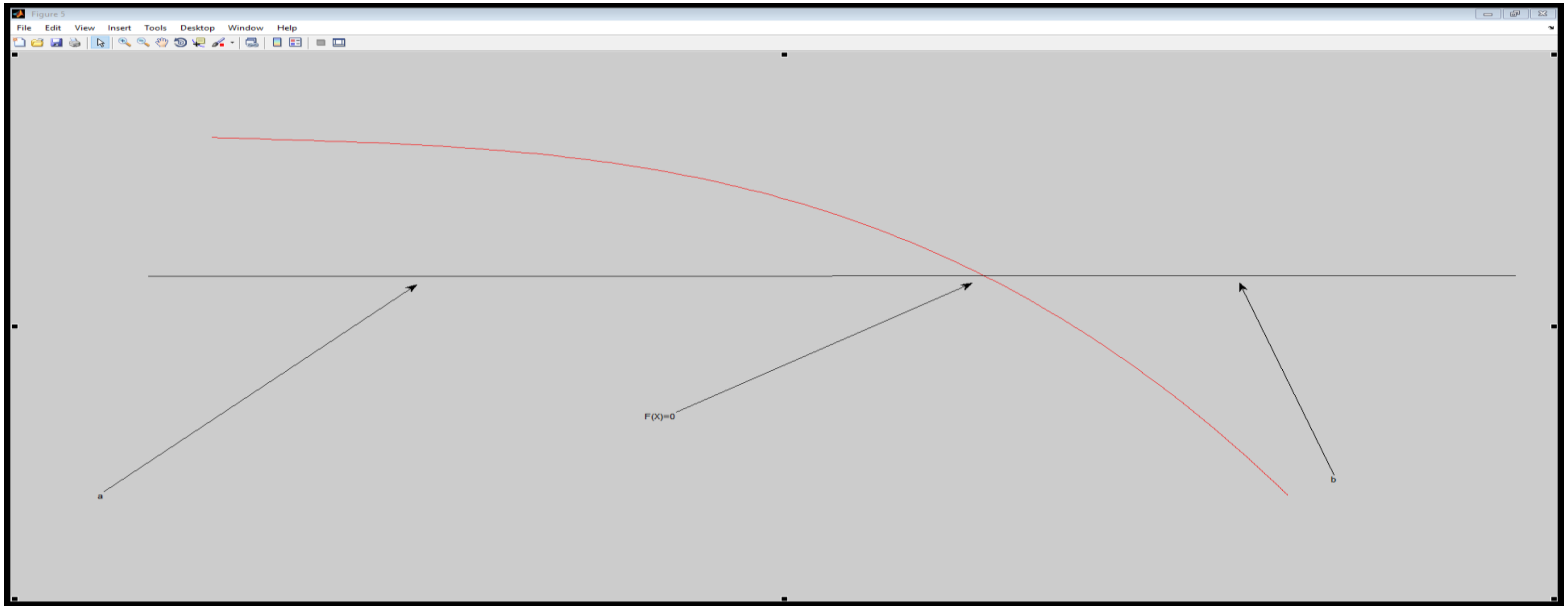
# Nonlinear Root Finding

- Bisection method :
  - Let we consider continuous function y=f(x) such as
  - function y=f(x) cross line y=0 **ones** on interval

**(a, b)**

**Then ether f(a)>0 and f(b)<0 or f(a)<0 and f(b)>0**

**In both cases f(a)*f(b)<0** (on picture f(a)>0 and f(b)<0 )



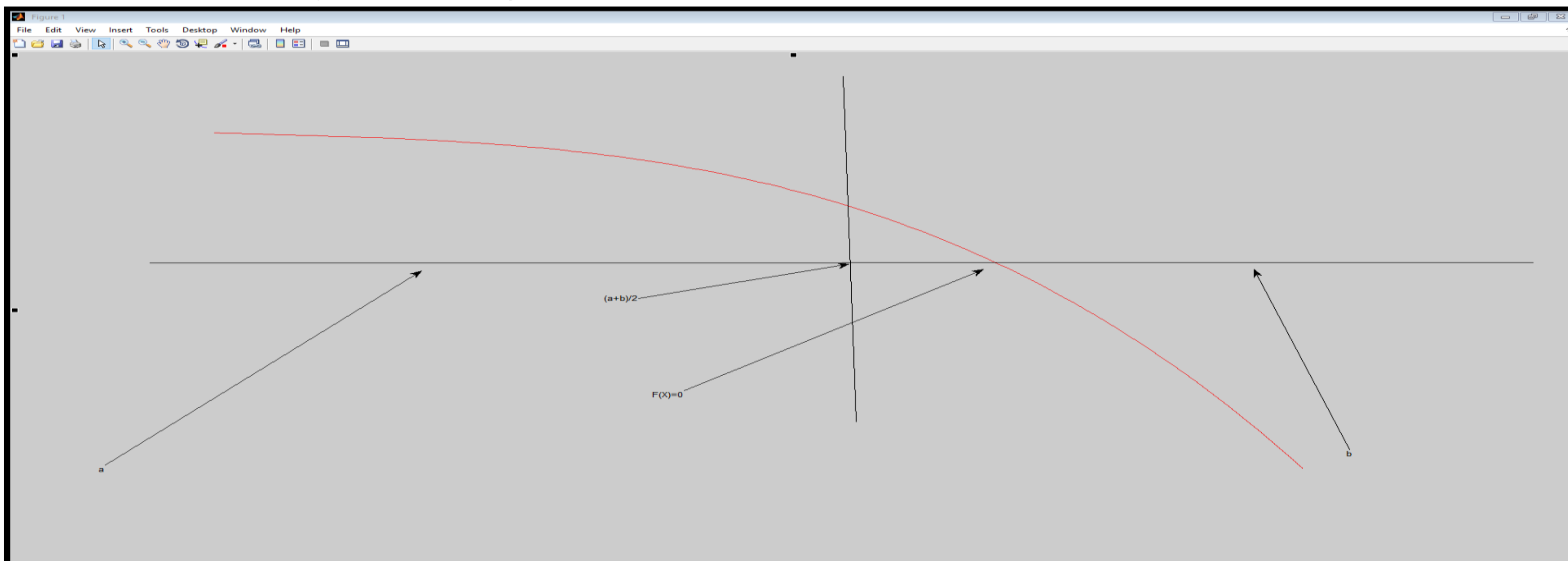
# Nonlinear Root Finding

- Bisection method :

  Let we split interval (a,b) by two

  Then our function f(x)=o ether on left interval (a,(a+b)/2) ether on right ((a+b)/2, b) interval

  (on picture-right)



# Nonlinear Root Finding

- Bisection method :
  - Let we select 'correct' interval and split it again by 2
  - And again…
  - We will do it until interval will be 'very small'
- How to select 'correct' interval ?

**if f(x)=o on interval (a,b)**

**f(a)*f(b)<0 !!**

What 'very small' means?

Depend on particular problem, but in most cases

b-a<$10^8$ is fine

# Nonlinear Root Finding

- **Bisection method example :**

  - function [answer,errr] = bisect(fun,a,b)
  - global tolerance maxits;
  - iterations = 0 ;
  - f_a = feval(fun,a);
  - f_b = feval(fun,b);
  - while ((f_a*f_b<0) && iterations<maxits) && (b-a)>tolerance
  - iterations = iterations + 1 ;
  - c = (b+a)/2;
  - f_c = feval(fun,c);
  - if f_c*f_a<0
  - b=c; f_b = f_c;
  - else
  - a=c; f_a = f_c;
  - end
  - errr = (b-a); answer = c;
  - end;

# Nonlinear Root Finding

- **Bisection method example :**

```
function [output] = myfunc1(x)
global p q;
output = (x - p).*(x-q);
```

**Script:**

```
global p q tolerance maxits delta;
p=-3; q = 2; xmin = -1; xmax = +5;
vec = xmin:0.1:xmax;
y = feval(@myfunc1,vec);
plot(vec,y,vec,zeros(1,size(vec,2)),':');
x0=(p+q)/2 + 0.4;
z = fzero(@myfunc1,x0)
% use bisection method
tolerance = 1e-8; maxits = 2000;
[z_b1,err1] = bisect(@myfunc1,xmin,xmax)
```

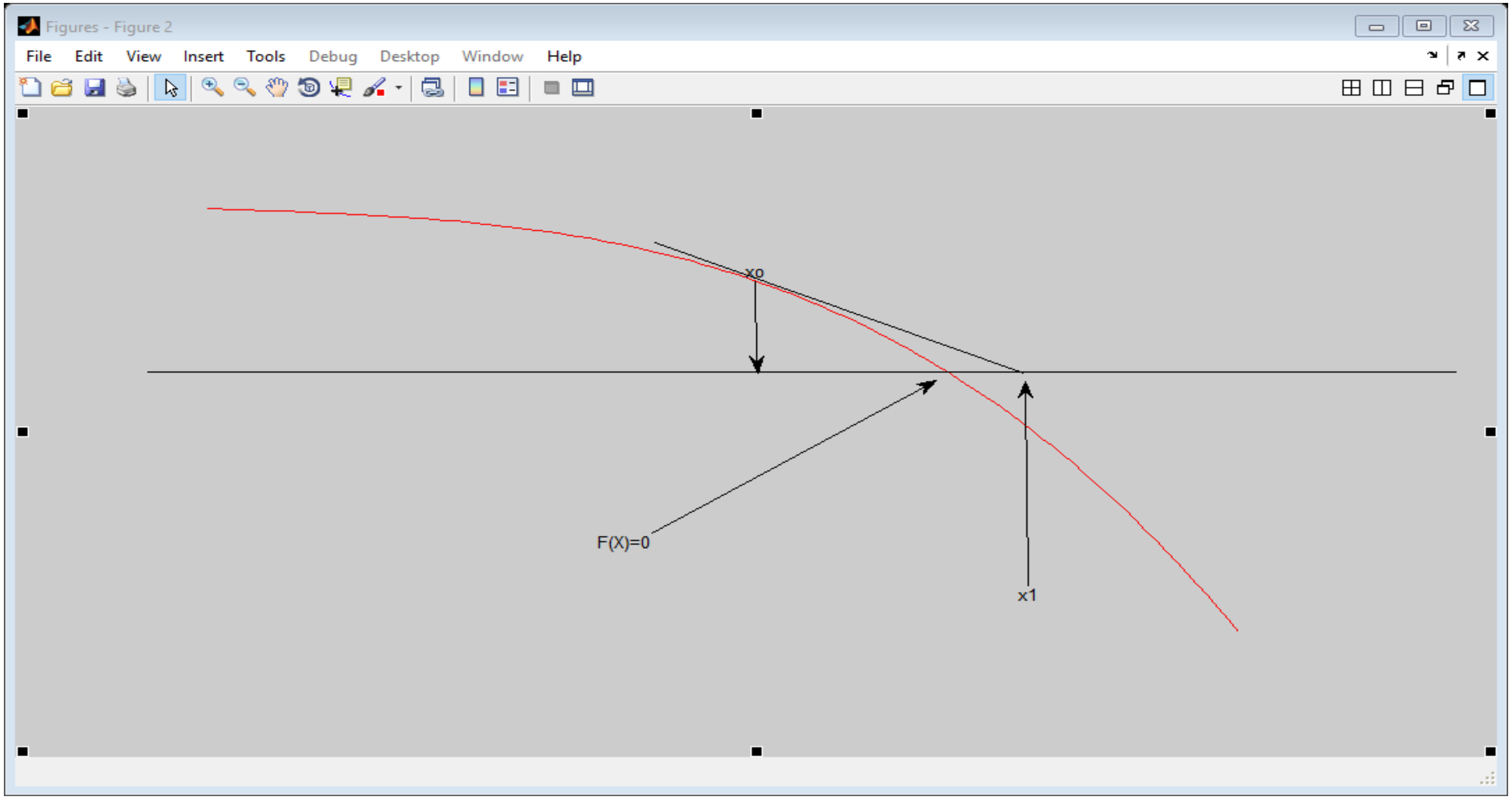
# Nonlinear Root Finding

- **Newton method**

- Newton method is a numerical algorithm to solve equation $f(x)=0$.
- Let we guess that x0 is solution of this equation (initial guess).
- Than the update x1
- $x1 = x0-f(x0)/f'(x0)$
- will be closer to the true solution than x0 (subject to some constrains on f (x)).

# Nonlinear Root Finding

- **Newton method**



# Nonlinear Root Finding

- **Newton method**

```
function [answer, fz, iflag] = NewtonRaphson(func,x0)
global tolerance maxits delta;
iterations = 0 ; x = x0;
while (iterations<maxits) && (abs(func(x))>tolerance)
f0 = func(x);
f1 = func(x+delta);
x = x-f0*delta/(f1-f0);
iterations = iterations + 1;
end;
answer = x; fz = func(x);
if iterations>maxits
iflag = 0;disp('No root found')
else
iflag = 1;disp(['Root = ' num2str(x) 'found in ' num2str(iterations) 'iterations.'])
end;
```

# Nonlinear Root Finding

- **Newton method**
  - Given we run previous programm:

```
delta = 1e-4;
[z_n1, f_z1, iáag1] = NewtonRaphson(@myfunc1,xmin)
```

**Will provede solution f(x)=o**

# Minimizing a Function

- Minimum or maximum function is a point where $f'$(x)=o
  - So, we are already know how to solve equations f(x)=0....

# Minimizing a Function

- **fminbnd**: minimizing a function over a bounded interval
- x=**fminbnd**('myfun',-1,2);

    myfun takes a scalar input and returns a scalar output

    myfun(x) will be the minimum of myfun for $-1 \leq x \leq 2$

**fminsearch**: unconstrained interval

    x=**fminsearch**('myfun',.5)

    finds the local minimum of myfun starting at x=0.5

# Anonymous Functions

- You do not have to make a separate function file

```
x=fzero(@myfun,1)
```

What if myfun is really simple?

Instead, you can make an anonymous function

```
x=fzero(@(x)  (cos(exp(x))+x^2-1),  1  );
```

input function

**x=fminbnd(@(x) (cos(exp(x))+x^2-1),-1,2);**

# Numerical Differentiation

- Big topic
- Simplest way :
- Lets y=f(x),
- $y' = \lim_{\Delta x \to 0} \left( \frac{\Delta f}{\Delta x} \right)$
- **diff** computes the first difference
- Example:

```
x=0:0.01:2*pi;
y=sin(x);
dydx=diff(y)./diff(x);
```

# Numerical Differentiation

- **diff** can work with matrixes too:

```
mat=[1 3 5;4 8 6];
dm=diff(mat,1,2)
```

- first difference of mat along the 2nd dimension, **dm=[2 2;4 -2]**
- see help for more details
- The opposite of **diff** is the cumulative sum **cumsum**

# Numerical Integration

- **MATLAB contains common integration methods**

**Adaptive Simpson's quadrature** (input is a function)

```
q=quad('myFun',0,10);
```

q is the integral of the function myFun from 0 to 10

```
q2=quad(@(x) sin(x)*x,0,pi)
```

q2 is the integral of sin(x)*x from 0 to pi

**Trapezoidal rule (input is a vector)**

```
x=0:0.01:pi;
z=trapz(x,sin(x));
```

z is the integral of sin(x) from 0 to pi

```
z2=trapz(x,sqrt(exp(x))./x)
```