Homework 2 Recap

- How long did it take?

- 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 a vector, then gets min
```

- Common mistake:

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

- How to make and run a function: save the file, then call it from the command window like any other function. No need to 'compile' or make it official in any other way

Outline

(1) Linear Algebra

(2) Polynomials

(3) Optimization

(4) Differentiation/Integration

(5) Differential Equations

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.

MATLAB makes linear algebra fun!

More Linear Algebra

- Given a matrix
  » `mat=[1 2 -3;-3 -1 1;1 -1 1];`

- Calculate the rank of a matrix
  » `r=rank(mat);`
    - the number of linearly independent rows or columns

- 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

Exercise: Linear Algebra

- Solve the following systems of equations:

  - System 1:

$$\begin{aligned} x + 4y &= 34 \\ -3x + y &= 2 \end{aligned}$$

  - System 2:

$$\begin{aligned} 2x - 2y &= 4 \\ -x + y &= 3 \\ 3x + 4y &= 2 \end{aligned}$$

Exercise: Linear Algebra

- Solve the following systems of equations:

  - System 1:

$$x + 4y = 34$$
$$-3x + y = 2$$

```
» A=[1 4;-3 1];
» b=[34;2];
» rank(A)
» x=inv(A)*b;
```

  - System 2:

$$2x - 2y = 4$$
$$-x + y = 3$$
$$3x + 4y = 2$$

```
» A=[2 -2;-1 1;3 4];
» b=[4;3;2];
» rank(A)
```

  - rectangular matrix

```
» x1=A\b;
```

  - gives least squares solution

```
» error=abs(A*x1-b)
```

Outline

(1) Linear Algebra

(2) Polynomials

(3) Optimization

(4) Differentiation/Integration

(5) Differential Equations

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

$$\underset{P(1)}{a}x^3+\underset{P(2)}{b}x^2+\underset{P(3)}{c}x+\underset{P(4)}{d}$$

- 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--');`

Exercise: Polynomial Fitting

- Evaluate $y = x^2$ for x=-4:0.1:4.

- Add random noise to these samples. Use **randn**. Plot the noisy signal with **.** markers

- Fit a 2nd degree polynomial to the noisy data

- Plot the fitted polynomial on the same plot, using the same x values and a red line

Exercise: Polynomial Fitting

- Evaluate $y = x^2$ for x=-4:0.1:4.
  » **x=-4:0.1:4;**
  » **y=x.^2;**

- Add random noise to these samples. Use **randn**. Plot the noisy signal with **.** markers
  » **y=y+randn(size(y));**
  » **plot(x,y,'.');**
- Fit a 2nd degree polynomial to the noisy data
  » **p=polyfit(x,y,2);**
- Plot the fitted polynomial on the same plot, using the same x values and a red line
  » **hold on;**
  » **plot(x,polyval(p,x),'r')**

Outline

(1) Linear Algebra

(2) Polynomials

(3) Optimization

(4) Differentiation/Integration

(5) Differential Equations

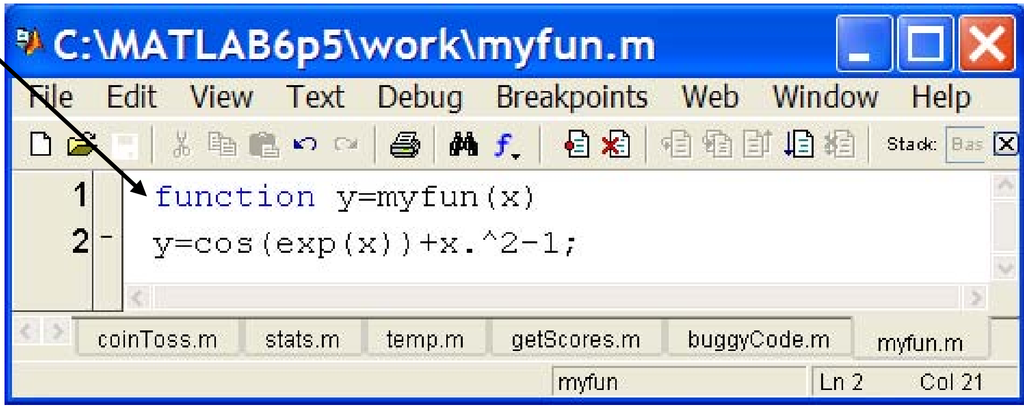
Nonlinear Root Finding

- 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.
- We will see this more and more as we delve into solving equations.
- Make a separate function file

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

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



Courtesy of The MathWorks, Inc. Used with permission.

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 to evaluate

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

Optimization Toolbox

- If you are familiar with optimization methods, use the optimization toolbox

- Useful for larger, more structured optimization problems

- Sample functions (see **`help`** for more info)
  - » **`linprog`**
    - ➢ linear programming using interior point methods
  - » **`quadprog`**
    - ➢ quadratic programming solver
  - » **`fmincon`**
    - ➢ constrained nonlinear optimization

Exercise: Min-Finding

- Find the minimum of the function $f(x) = \cos(4x)\sin(10x)e^{-|x|}$ over the range −π to π. Use **`fminbnd`**.
- Plot the function on this range to check that this is the minimum.

Exercise: Min-Finding

- Find the minimum of the function $f(x)=\cos(4x)\sin(10x)e^{-|x|}$ over the range −π to π. Use **fminbnd**.
- Plot the function on this range to check that this is the minimum.

- Make the following function:
  - » `function y=myFun(x)`
  - » `y=cos(4*x).*sin(10*x).*exp(-abs(x));`

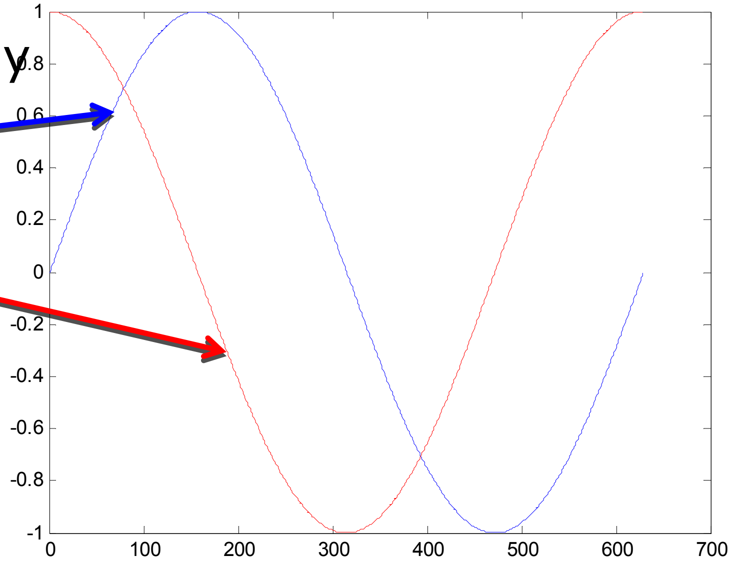
- Find the minimum in the command window:
  - » `x0=fminbnd('myFun',-pi,pi);`
- Plot to check if it's right
  - » `figure; x=-pi:.01:pi; plot(x,myFun(x));`

Outline

(1) Linear Algebra
(2) Polynomials
(3) Optimization
(4) Differentiation/Integration
(5) Differential Equations

Numerical Differentiation

- MATLAB can 'differentiate' numerically
  - » `x=0:0.01:2*pi;`
  - » `y=sin(x);`
  - » `dydx=diff(y)./diff(x);`
    - ➢ diff computes the first difference



- Can also operate on matrices
  - » `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`

- 2D gradient
  - » `[dx,dy]=gradient(mat);`

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)`
    - z2 is the integral of $\sqrt{e^x}/x$ from 0 to pi

End of Lecture 3

(1) Linear Algebra
(2) Polynomials
(3) Optimization
(4) Differentiation/Integration
(5) Differential Equations