
CS1371

Introduction to Computing for Engineers

Vectors and Matrices

1

9/15/03

Vectors and Matrices

Learning Objectives

Understand the
nature of matrices

Understand how to
manipulate
matrices in Matlab

Lecture

- Basic vector & matrix concepts
- Creating arrays and matrices
- Accessing matrix components
- Manipulating matrices
- Matrix functions
- Solving simultaneous equations
- Regression analysis

2

9/15/03

Vectors and Matrices

- We've referred to vectors and matrices frequently... but exactly what are we talking about?
 - *what is a matrix?*
 - *is it different from an array?*
- **ANSWER:**
 - *vectors and matrices are arrays with an “attitude”*
 - *that is, they look just like an array (and they are arrays), but they live by a very different set of rules!*
 - **Vectors:**
 - $f + b = ?$
 - $3f = ?$
 - $f \cdot g = ?$
 - $S \times r = ?$
 - $|h| = ?$
 - $A / b = ?$

Can you explain what, if anything, results from these operations with vectors?

3

9/15/03

Why Matrices?

- A matrix is an array that obeys a different set of rules
 - *addition & subtraction are same as for arrays,*
 - *but multiplication, division, etc. are **DIFFERENT!***
 - *a matrix can be of any dimension but 2D square matrices are the most common by far*
- A large and very useful area of mathematics deals with what is called “linear algebra” and matrices are an integral part of this.
- Many advanced computational methods in engineering make extensive use of linear algebra, and hence of matrices

4

9/15/03

A Simple Example

- A set of simultaneous linear algebraic equations will often arise in engineering applications

$$3x - 2y = 14$$

$$x + 4y = -14$$

- How do you solve these?
 - Solve first for x in terms of y ; substitute in second and solve for y ; use this in first to find x
 - Use “Cramer’s Rule”
 - Other?
- Let’s try a more abstract notation:

$$\begin{bmatrix} 3 & -2 \\ 1 & 4 \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 14 \\ -14 \end{Bmatrix} \quad \text{OR} \quad \mathbf{C} * \mathbf{z} = \mathbf{b}$$

5

9/15/03

A Simple Example-cont’d

- What do we mean by the * for this form?

$$\mathbf{C} * \mathbf{z} = \begin{bmatrix} 3 & -2 \\ 1 & 4 \end{bmatrix} * \begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} 3x - 2y \\ x + 4y \end{Bmatrix}$$

- Note that the column matrix, \mathbf{z} , is multiplied times the first row of \mathbf{C} on an element-by-element basis and the results are summed to get the first row of the answer
- Ditto for the second row...
- This is NOT array multiplication; it is matrix multiplication
- For two 2D matrices in general:

$$\mathbf{A} * \mathbf{B} = \mathbf{C}$$

where :

$$c_{ij} = \sum_{k=1}^N a_{ik} b_{kj}$$

NOTE: the number of columns in A must be equal to the number of rows in B (N in this example)

6

9/15/03

A Few Notes on Matrices

- Matlab handles matrix multiplication with the * symbol (NOTE: this is NOT array multiplication!)
 - From our formula we see that in general: $A*B \neq B*A$
 - In other words, matrix multiplication is NOT commutative
- Matrices behave just like arrays for addition and subtraction
- Matrix division is not strictly defined but a matrix inverse is available to address this situation, among others.
 - suppose: $3y=6$ and you need to find $y...$
 - The usual approach: $y=6/3=2$ (division by 3)
 - Also useful: $y=3^{-1}*6=2$ (multiplication by the inverse of 3)
 - If we don't know how to divide, we can accomplish the same by using the notion of the inverse. Recall definition of inverse:
$$(value)*(inverse\ value) = 1$$
 - Turns out we know how to compute matrix inverses (but it requires a lot of computational effort)

7

9/15/03

Let's Solve Our Problem Using Matlab

```
>> coef=[3 -2; 1 4]
coef =
     3     -2
     1     4
>> inv(coef) % Matlab has the inv() function
ans =
    0.2857    0.1429
   -0.0714    0.2143
>> b=[14 -14]';
b =
    14
   -14
>> z=inv(coef)*b
z =
     2
    -4
>> coef*z % Let's check our answer!
ans =
    14
   -14
```

8

9/15/03

Some More Notes:

- Using the Matlab `inv()` function is not always best
 - It can take a VERY long time for large matrices
 - The inverse may have poor precision for some kinds of matrices
- If you just want to solve the set of equations, there are much quicker and more accurate methods
 - Uses powerful algorithms from linear algebra
 - Notation is tricky because it introduces the concept of a “left” and a “right” matrix division in Matlab

Given :

$$\mathbf{C} * \mathbf{z} = \mathbf{b}$$

Left – divide both sides by \mathbf{C} :

$$\mathbf{C} \setminus \mathbf{C} * \mathbf{z} = \mathbf{C} \setminus \mathbf{b}$$

or :

$$\mathbf{z} = \mathbf{C} \setminus \mathbf{b}$$

NOTE:

$\mathbf{C} \setminus \mathbf{C} = 1$, and
 $1 * \text{anything} = \text{anything}$

9

9/15/03

Let's Try This Out...

```
coef =  
     3     -2  
     1      4  
  
>> b  
b =  
    14  
   -14  
  
>> zz=coef\b  
zz =  
    2.0000  
   -4.0000
```

OK, now what do you think these expressions yield?

```
coef \ eye (2,2)  
coef \ eye (2,2) * coef
```

10

9/15/03

Things Can Get Weird...

- We usually think of the unknown (\mathbf{z}) as a column matrix and the RHS (\mathbf{b}) as a column matrix also
- In some fields, it is more useful if these are ROW matrices
 - *One formulation can easily be converted into the other!*
 - *We can treat either formulation in Matlab*
- First, ON YOUR OWN, prove from our multiplication formula that:

$$(\mathbf{A} * \mathbf{B})^T = \mathbf{B}^T * \mathbf{A}^T$$

- Now, using this, we take the transpose of our equation:

$$\begin{array}{l} (\mathbf{C} * \mathbf{z})^T = \mathbf{b}^T \\ \boxed{\mathbf{z}^T * \mathbf{C}^T = \mathbf{b}^T} \end{array} \quad \text{where} \quad \begin{array}{l} \mathbf{z}^T = [x \quad y] \\ \mathbf{b}^T = [14 \quad -14] \\ \mathbf{C}^T = \begin{bmatrix} 3 & 1 \\ -2 & 4 \end{bmatrix} \end{array}$$

11

9/15/03

Let's Try It Out in Matlab:

```
>> coefT=coef'  
coefT =  
     3     1  
    -2     4  
  
>> bT=b'  
bT =  
    14    -14  
  
>> zT=bT*inv(coefT)  
zT =  
     2     -4  
  
>> % Also we can use Right Divide  
>> zT2=bT/coefT  
zT2 =  
    2.0000   -4.0000
```

12

9/15/03

Other Matlab Matrix Functions

- So far we've only scratched the surface of Matlab's abilities to work with matrices...
- Matrices can contain COMPLEX numbers
- Some of the other matrix functions are:
 - **det(A)**: *determinant of the matrix*
 - **rank(A)**: *rank of the matrix*
 - **trace(A)**: *sum of diagonal terms*
 - **sqrtn(A)**: *matrix square root*
 - (i.e., if $B = \text{sqrtn}(A)$, then $A = B * B$)
 - **norm(A)**: *matrix norm (useful for vector magnitudes)*
 - **eig(A)**: *eigenvalues and eigenvectors of matrix*
 - ...
- Keep in mind that Matlab is using some of the latest and most powerful algorithms to compute these functions.

13

9/15/03

Finally, What About Vectors?

- The matrix and array operations and functions can be used to manipulate vectors, but you'll have to be careful
- Vector dot product:

```
>> f=[1 2]'  
f =  
    1  
    2  
>> g=[4 -3]'  
g =  
    4  
   -3  
>> fdotg=f'*g  
fdotg =  
   -2
```

Column vectors

Row vectors

```
>> f=[1 2]  
f =  
    1    2  
>> g=[4 -3]  
g =  
    4   -3  
>> fdotg=f*g'  
fdotg =  
   -2  
>> gdotf=g*f'  
gdotf =  
   -2
```

Matlab has some functions to do these, too!

- ON YOUR OWN:
 - Vector magnitude?
 - Vector cross product?

14

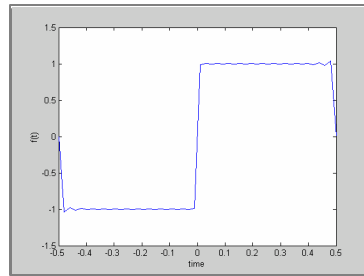
9/15/03

Problem Solving

- Since matrix multiplication involves a summation (recall the definition), we can use this to compute series summations!
- Code will be VERY difficult to understand but VERY fast because Matlab is optimized for matrix and array operations
- Consider the following summation which gives a square wave when computed at points, $-0.5 \leq t_i \leq 0.5$ (we use 50 points)

$$f(t) = \frac{4}{\pi} \sum_{k=odd}^N \frac{1}{k} \sin(2\pi kt) \text{ for } -\frac{1}{2} \leq t \leq \frac{1}{2}$$

```
>> k=1:2:121;  
>> t=linspace(-0.5,0.5,50);  
>> coef=4/pi./k;  
>> sinterm=sin(2*pi*k'*t);  
>> ft=coef*sinterm;  
>> plot(t,ft)  
>> xlabel('time'), ylabel('f(t)')
```

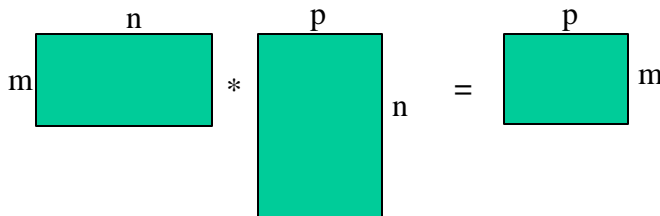


15

9/15/03

Matrix Multiplication

- If I want to multiply $A[5,3]$ by $B[3,4]$, the result will be a $[5,4]$ matrix.
- In general, multiplying $A[m,n]$ by $B[n,p]$ results in a $[m,p]$ matrix.
- The inner dimensions must match.

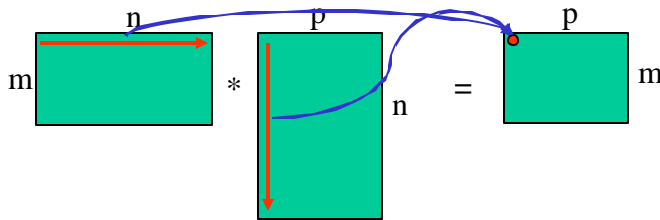


16

9/15/03

Matrix Multiplication

- If I want to multiply $A[5,3]$ by $B[3,4]$, the result will be a $[5,4]$ matrix.
- In general, multiplying $A[m,n]$ by $B[n,p]$ results in a $[m,p]$ matrix.
- The inner dimensions must match.

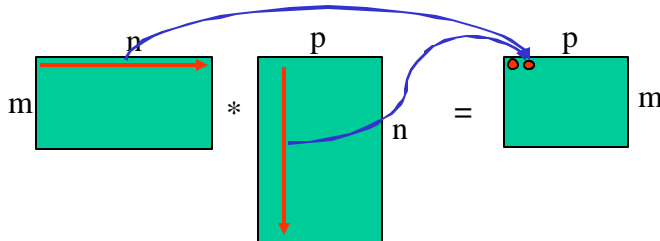


17

9/15/03

Matrix Multiplication

- If I want to multiply $A[5,3]$ by $B[3,4]$, the result will be a $[5,4]$ matrix.
- In general, multiplying $A[m,n]$ by $B[n,p]$ results in a $[m,p]$ matrix.
- The inner dimensions must match.

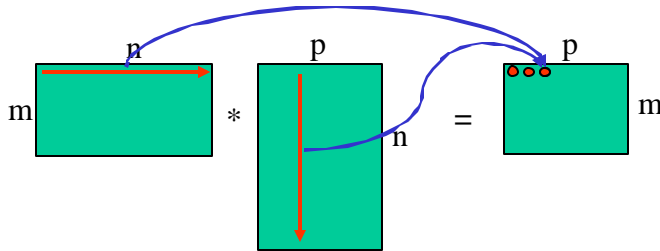


18

9/15/03

Matrix Multiplication

- If I want to multiply $A[5,3]$ by $B[3,4]$, the result will be a $[5,4]$ matrix.
- In general, multiplying $A[m,n]$ by $B[n,p]$ results in a $[m,p]$ matrix.
- The inner dimensions must match.

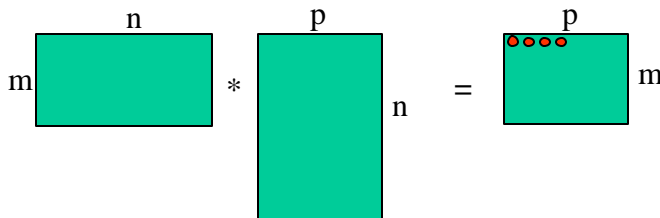


19

9/15/03

Matrix Multiplication

- If I want to multiply $A[5,3]$ by $B[3,4]$, the result will be a $[5,4]$ matrix.
- In general, multiplying $A[m,n]$ by $B[n,p]$ results in a $[m,p]$ matrix.
- The inner dimensions must match.

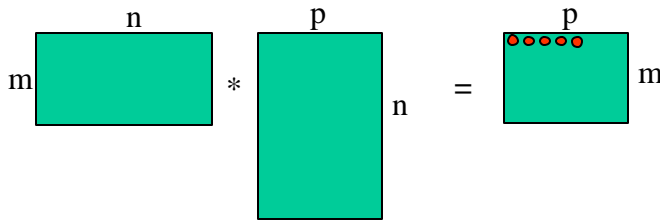


20

9/15/03

Matrix Multiplication

- If I want to multiply $A[5,3]$ by $B[3,4]$, the result will be a $[5,4]$ matrix.
- In general, multiplying $A[m,n]$ by $B[n,p]$ results in a $[m,p]$ matrix.
- The inner dimensions must match.

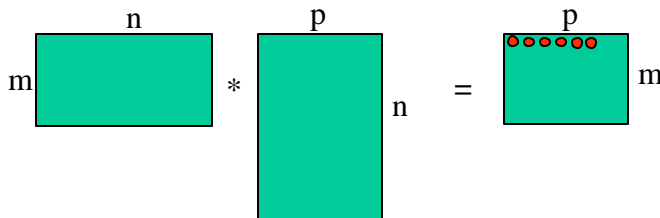


21

9/15/03

Matrix Multiplication

- If I want to multiply $A[5,3]$ by $B[3,4]$, the result will be a $[5,4]$ matrix.
- In general, multiplying $A[m,n]$ by $B[n,p]$ results in a $[m,p]$ matrix.
- The inner dimensions must match.

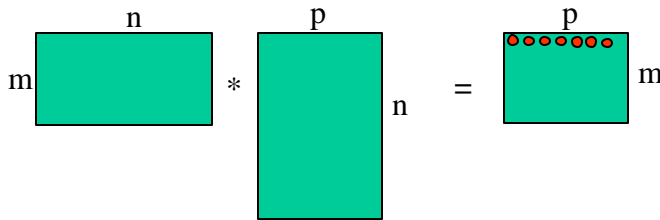


22

9/15/03

Matrix Multiplication

- If I want to multiply $A[5,3]$ by $B[3,4]$, the result will be a $[5,4]$ matrix.
- In general, multiplying $A[m,n]$ by $B[n,p]$ results in a $[m,p]$ matrix.
- The inner dimensions must match.

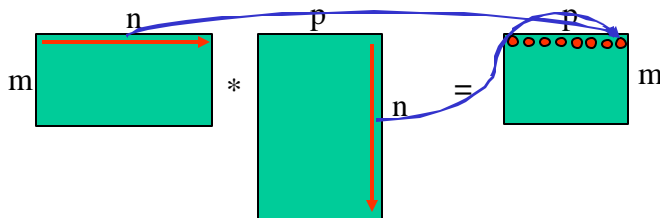


23

9/15/03

Matrix Multiplication

- If I want to multiply $A[5,3]$ by $B[3,4]$, the result will be a $[5,4]$ matrix.
- In general, multiplying $A[m,n]$ by $B[n,p]$ results in a $[m,p]$ matrix.
- The inner dimensions must match.

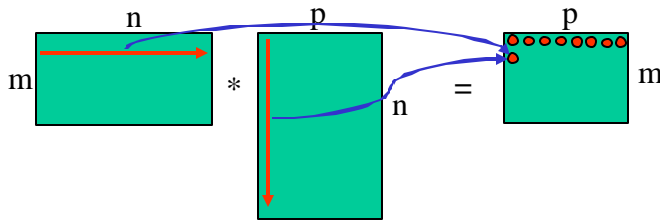


24

9/15/03

Matrix Multiplication

- If I want to multiply $A[5,3]$ by $B[3,4]$, the result will be a $[5,4]$ matrix.
- In general, multiplying $A[m,n]$ by $B[n,p]$ results in a $[m,p]$ matrix.
- The inner dimensions must match.

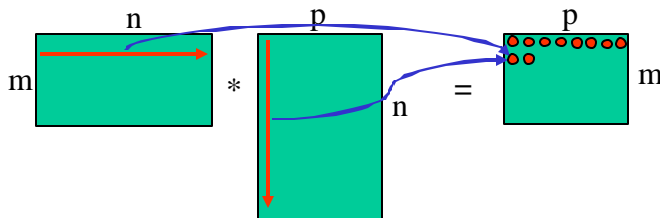


25

9/15/03

Matrix Multiplication

- If I want to multiply $A[5,3]$ by $B[3,4]$, the result will be a $[5,4]$ matrix.
- In general, multiplying $A[m,n]$ by $B[n,p]$ results in a $[m,p]$ matrix.
- The inner dimensions must match.

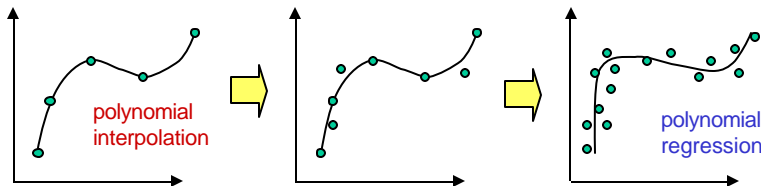


26

9/15/03

Fitting Polynomials to Data

- Fitting analytical curves to experimental data is a common task in engineering.
- The simplest is **linear interpolation** (straight lines between data points). Polynomials are often used to define curves, and there are several options:
 - Fit curve through all data points (**polynomial interpolation**),
 - Fit curve through some data points,
 - Fit curve as close to points as possible (**polynomial regression, also called “least square fitting”**)



27

9/15/03

Fitting Polynomial Through N Points

- A polynomial of degree N-1 contains N coefficients:

$$p(x) = p_{N-1}x^{N-1} + p_{N-2}x^{N-2} + \dots + p_1x + p_0$$

- To define these requires N constraints and these can be the requirement that the curve passes through N points:

$$y_1 = p(x_1) = p_{N-1}x_1^{N-1} + p_{N-2}x_1^{N-2} + \dots + p_1x_1 + p_0$$

$$y_2 = p(x_2) = p_{N-1}x_2^{N-1} + p_{N-2}x_2^{N-2} + \dots + p_1x_2 + p_0$$

⋮

$$y_N = p(x_N) = p_{N-1}x_N^{N-1} + p_{N-2}x_N^{N-2} + \dots + p_1x_N + p_0$$

NOTE: This is called **polynomial interpolation**

- These can be written in matrix notation:

$$\begin{Bmatrix} y_1 \\ \vdots \\ y_N \end{Bmatrix} = \begin{bmatrix} x_1^{N-1} & x_1^{N-2} & \dots & 1 \\ \dots & \dots & \dots & \dots \\ x_N^{N-1} & x_N^{N-2} & \dots & 1 \end{bmatrix} \begin{Bmatrix} p_{N-1} \\ \vdots \\ p_0 \end{Bmatrix} \quad \text{or:} \quad \mathbf{y} = \mathbf{X} * \mathbf{p}$$

28

9/15/03

Fitting Polynomial Through N Points...

- We now need to solve this equation for the unknown polynomial coefficients, p_i , which can be done with matrix operations in Matlab and expressed as:

$$p = X \setminus y$$

- *Note that we are using the Matlab “left divide” operator, \setminus , to compute a solution (the `inv()` function could also be used).*
- *The number of points available (N) determines the degree (N-1) of the resulting polynomial curve.*
- It is an easy task to do all this in Matlab and you can try to build an m-function (e.g., `mypolyfit()`) to accomplish this

NOTE: Matlab actually has a powerful m-function called `polyfit()` that will do exactly this and more... see `help polyfit` for details.

29

9/15/03

Testing Your `mypolyfit()`

- Try out your new m-function by computing a polynomial interpolation for the curve, $y=3\sin(2\pi x)$ from $x=-\pi$ to π . Assume that you know the “value” of this function at only 5 points (you can specify them).
 - *What happens when you increase the number of points to 8? to 10? higher?*
 - *The function: $f(x)=1/(1+25x^2)$ over $-1 \leq x \leq 1$ presents major problems for simple polynomial interpolation. Try polynomial interpolations through an odd number (1,3,...) of points up to 11 and make plots to compare the interpolation to the actual function. Surprised? This is sometimes called the “Runge” problem after a mathematician who used it to illustrate problems.*

30

9/15/03

Problem Solving

- Frequently in engineering, we need to determine the best polynomial expression to match a set of data values.
- Unfortunately, the data are frequently measured with some noise, and the actual expression may not be polynomial.
- However, the method of Least Squares allows us to determine the “best” fit of any set of data to a polynomial equation of a given order (1st, 2nd, ...)
- For example, consider determining the equation that best fits the following data:

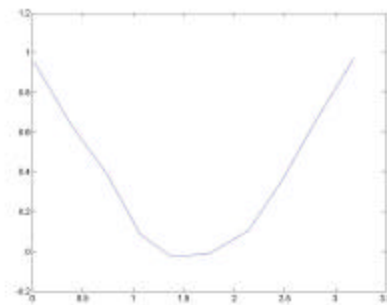
31

9/15/03

Experimental Data

```
x = [0.011 0.378 0.740 1.071 1.364 1.735 2.138 2.485 2.784 3.181]
```

```
y = [0.956 0.643 0.388 0.085 -0.021 -0.014 0.104 0.368 0.635 0.970]
```



32

9/15/03

Analysis

- This looks a bit like a parabola, so perhaps a quadratic equation might fit well:

$$y = A*x^2 + B*x + C$$

- How do we find the values of A, B and C that “best” fit the data?
- The method of Least Squares determines the values of A, B and C for which the sum of the squared differences between the exact curve and the data points $[x_i, y_i]$ is minimized.
- Formally, if y_i' is the value $A*x_i^2 + B*x_i + C$
- Then we need to find the values of A, B and C that minimize the value

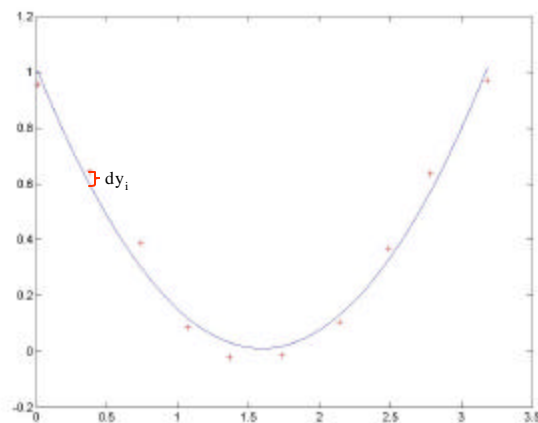
$$E^2 = \sum (y_i - y_i')^2$$

33

9/15/03

Graphically

- We choose A, B and C so that the sum of the squares of the errors dy_i is a minimum.



34

9/15/03

[You can skip this math if you want]

$$E^2 = \sum (y_i - y_i')^2$$

$$= \sum y_i^2 - 2 \sum (y_i y_i') - \sum y_i'^2$$

and $y_i = A \cdot x_i^2 + B \cdot x_i + C$

Hence, (eventually)

$$S = E^2 = \sum y_i^2 - 2A \sum (x_i^2 y_i) - 2B \sum (x_i y_i) - 2C \sum y_i$$

$$+ A^2 \sum x_i^4 + B^2 \sum x_i^2 + C^2 \sum 1$$

$$+ 2AB \sum x_i^3 + 2AC \sum x_i^2 + 2BC \sum x_i$$

To minimize this, we take the partial derivatives of this equation with respect to A, B and C, and set the result to zero:

$$\delta S / \delta A = \delta S / \delta B = \delta S / \delta C = 0$$

Which leads us to the following set of linear equations:

35

9/15/03

The End Result

$$[\delta S / \delta A =] -2 \sum (x_i^2 y_i) + 2A \sum x_i^4 + 2B \sum x_i^3 + 2C \sum x_i^2 = 0$$

$$[\delta S / \delta B =] -2 \sum (x_i \cdot y_i) + 2A \sum x_i^3 + 2B \sum x_i^2 + 2C \sum x_i = 0$$

$$[\delta S / \delta C =] -2 \sum y_i + 2A \sum x_i^2 + 2B \sum x_i + 2C \sum 1 = 0$$

From which we derive the following simultaneous equations:

$$\begin{bmatrix} \sum x_i^4 & \sum x_i^3 & \sum x_i^2 \\ \sum x_i^3 & \sum x_i^2 & \sum x_i \\ \sum x_i^2 & \sum x_i & \sum 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} \sum x_i^2 y_i \\ \sum x_i \cdot y_i \\ \sum y_i \end{bmatrix}$$

Which is of the form $[M][V] = [K]$, M and K being constants for any given data set, and V the vector of coefficients we need to find.

36

9/15/03

Something We Can use:

So the vector of coefficients [A B C]' is found from:

$$[V] = [M] \setminus [K]$$

Note that this set of equations has some symmetry that would allow us to generalize the solutions to any polynomial order. For example, a linear fit to the equation

$$y = A*x + B$$

Would result in the following:

$$\begin{bmatrix} Sx_i^2 & Sx_i \\ Sx_i & S1 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} S x_i * y_i \\ S y_i \end{bmatrix}$$

This is usually referred to as 'Linear Regression'

37

9/15/03

Linear Regression

- The general development for a polynomial of degree R is very tedious, and we won't attempt it!
- Rather, for R=1, we have a linear (straight line) fit

$$y(x) = c_0 + c_1x$$

- Using our previous results

$$\begin{bmatrix} \sum_{i=1}^N x_i & N \\ \sum_{i=1}^N x_i^2 & \sum_{i=1}^N x_i \end{bmatrix} \begin{Bmatrix} c_1 \\ c_0 \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^N y_i \\ \sum_{i=1}^N x_i y_i \end{Bmatrix} \quad \text{or} \quad \mathbf{M} * \mathbf{c} = \mathbf{d}$$

- And we can now solve for the coefficients, c:

$$\mathbf{c} = \mathbf{M} \setminus \mathbf{d}$$

- and use the `polyval(c, x)` function to evaluate this polynomial for any value(s) of x

38

9/15/03

Linear Regression Example

- We are given measurements of the deflection of a beam as a function of the applied load. The ratio of load to deflection is the lateral stiffness (lbs/inch)
 - A .mat file called `stiffness_data.mat` is provided and it contains the measured load and deflection.
1. Construct a plot of the data (load vs deflection) using circle symbols to denote each data point
 2. Compute a linear regression for the data
 3. Add a plot of the resulting regression line to your plot constructed in Step #1
 4. Print out the computed slope which is the estimated stiffness (note that we ignore the constant term which represents a nonzero measured deflection for zero load)

39

9/15/03

Quadratic Regression Code

```
% given the data items:
x = [0.011 0.378 0.740 1.071 1.364 1.735 2.138 2.485 2.784 3.181]
y = [0.956 0.643 0.388 0.085 -0.021 -0.014 0.104 0.368 0.635 0.970]

plot(x,y,'r+')
hold on

% use the method of least squares to find the coefficients of a
% parabola (2nd order polynomial) that fits these points
s1 = length(x)
sxi = sum(x)
sxi2 = sum(x.^2)
sxi3 = sum(x.^3)
sxi4 = sum(x.^4)
sxi2yi = sum(x.^2.*y)
sxiyi = sum(x.*y)
syi = sum(y)
```

40

9/15/03

Code used:

```
C = [sxi4 sxi3 sxi2
     sxi3 sxi2 sxi
     sxi2 sxi s1]

K = [sxi2yi sxiyi syi]'

V = C \ K;
A = V(1)
B = V(2)
C = V(3)

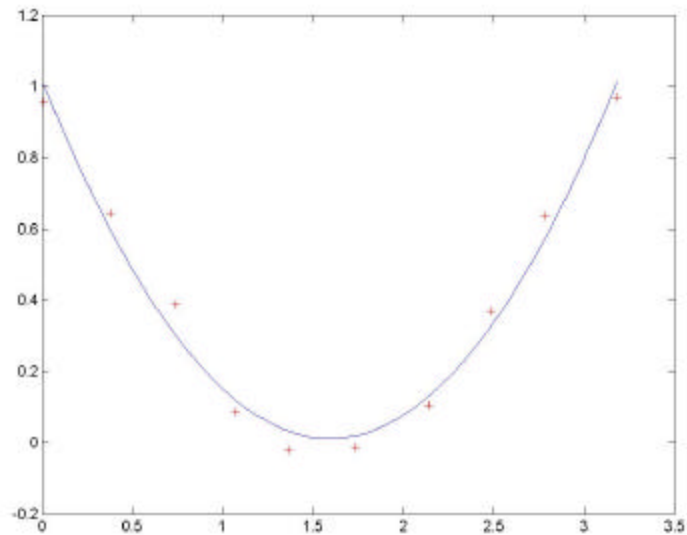
% notice that while the data are defined only at the points xi, the real
% parabola is defined "continuously" - hence, we can plot it at higher
% resolution
xp = linspace(x(1),x(length(x)),100)
yp = A.*xp.^2 + B.*xp + C

plot(xp, yp)
```

41

9/15/03

Results



42

9/15/03

Interpolation & Regression Summary

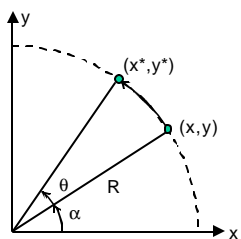
- We have introduced basic concepts only; much more advanced theory exists for these topics...
- The Matlab `polyfit()` function does both regression and interpolation, and `interp1()` and `interp2()` are also useful for interpolation; use the `help` function or consult our book. You should use these built-in functions whenever possible rather than using your own. They are faster and more accurate, especially for tricky problems.
- Spline curves, b-spline curves and their more powerful “cousin” called Nonuniform Rational B-Splines (NURBS) provide very powerful ways to interpolate or approximate data and are used extensively in CAD software.

43

9/15/03

Coordinate Transformations

- Quite often it is necessary to compute the (x,y) coordinates of an object that is to be rotated by some amount, ?.
- Rotation about the origin is defined below:



We can write:

$$x = R \cos a$$

$$y = R \sin a$$

and when rotated to $\alpha + \theta$ we get:

$$x^* = R \cos(a + \theta) = R \cos a \cos \theta - R \sin a \sin \theta$$

$$y^* = R \sin(a + \theta) = R \sin a \cos \theta + R \cos a \sin \theta$$

Or using the first equations:

$$x^* = x \cos \theta - y \sin \theta$$

$$y^* = y \cos \theta + x \sin \theta$$

In matrix form:

$$\begin{Bmatrix} x^* \\ y^* \end{Bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} \quad \text{Or:} \quad \mathbf{p}^* = \mathbf{T} * \mathbf{p}$$

44

9/15/03

Coordinate Transformations...

- Using the column vectors (matrices) for the points can be awkward, especially when there are multiple points that must be rotated.
- We can rewrite the equation by taking the matrix transpose of both sides (i.e., switching rows and columns for each matrix) to yield:

$$[x^* \quad y^*] = [x \quad y] \begin{bmatrix} \cos q & \sin q \\ -\sin q & \cos q \end{bmatrix}$$

- If we have multiple points to rotate, we can simply add them to the p matrix, one row at a time:

$$\begin{bmatrix} x_1^* & y_1^* \\ x_2^* & y_2^* \\ \vdots & \vdots \\ x_n^* & y_n^* \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} \begin{bmatrix} \cos q & \sin q \\ -\sin q & \cos q \end{bmatrix}$$

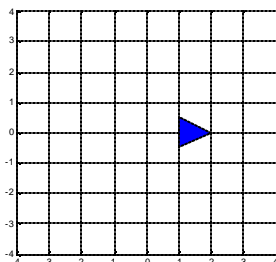
NOTE: Verify on your own that this matrix multiply works!

45

9/15/03

Example: Coordinate Transformations

- Let's construct a simple blue triangle whose vertices are defined by points: $(x,y) = (1,-0.5), (1,0.5), (2,0)$
- We can create a point matrix to define this: $p = \begin{bmatrix} 1 & -0.5 \\ 1 & 0.5 \\ 2 & 0 \end{bmatrix}$
- The Matlab `fill(x,y,color)` can be used to draw the shape.



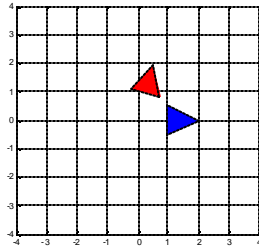
```
% Use this script to create a new figure window and
% plot the initial position of a figure defined by a
% p matrix and a color string which must be defined in
% the command window.
% Use testfill2 to add other rotated figures without
% erasing previous ones.
clf
axis([-4 4 -4 4])
grid on
axis square
hold on
fill(p(:,1),p(:,2),color)
```

46

9/15/03

Example: Coordinate Transformations...

- Now let's rotate this object by 75 degrees and draw it in red color (we must define `angle=75` and `color='r'` first):



```
% Use this script to create another object  
% without erasing the figure.  
% The angle and color must be specified from  
% the command line first.
```

```
T=rot2d(angle);  
p1=p*T; % matrix multiply  
fill(p1(:,1),p1(:,2),color)
```

```
function T = rot2d(angle)  
% Compute rotation matrix  
% T = rot2D(angle)  
% T = 2 x 2 transformation matrix  
% angle = angle in degrees
```

```
ang = angle*pi/180;  
ca = cos(ang); sa = sin(ang);  
T = [ca sa; -sa ca];
```

NOTE: We are introducing the Matlab `fill()` function and are creating our own rotate function (`rot2d`)

47

9/15/03

Questions?

48

9/15/03