MATLAB

for Engineering Applications

Fifth Edition

William J. Palm III

Chapter 07

Statistics, Probability, and Interpolation

Statistics and Histograms

- MATLAB provides the mean(x), mode(x), and median(x) functions to compute the mean, mode, and median of the data values stored in x, if x is a vector.
- If x is a matrix, a row vector is returned containing the mean (or mode or median) value of each column of x.
- A histogram is a plot of the frequency of occurrence of data values versus the values themselves. To plot a histogram, the data must be grouped into subranges, called bins.

Statistics and Histograms

- MATLAB provides the histogram function to generate a histogram.
- The basic form is histogram(y), where y is a vector containing the data. A uniform width is automatically chosen to reveal the underlying distribution.
- The second form is histogram(y, n), where n is a user-specified scalar indicating the number of bins.
- Unshaded rectangles can be obtained by using the syntax histogram(y, 'FaceColor', 'none').

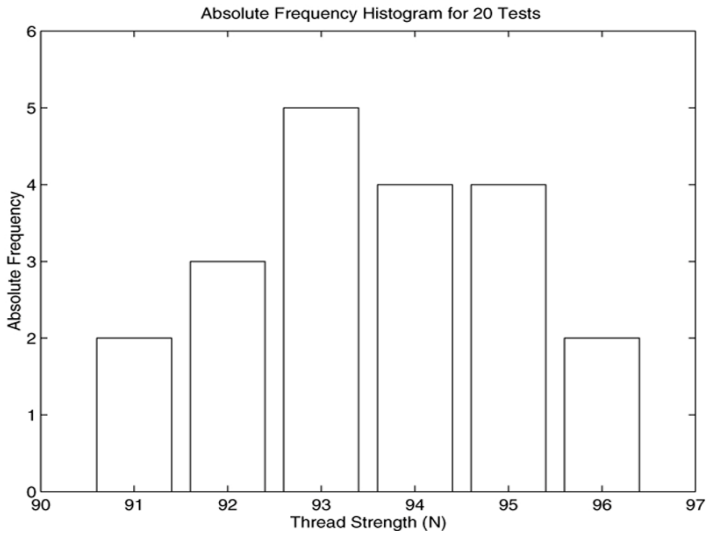
Breaking Strength of Thread

To ensure proper quality control, a thread manufacturer selects samples and tests them for breaking strength. Suppose that 20 thread samples are pulled until they break, and the breaking force is measured in newtons rounded off to integer values. The breaking force values recorded were 92, 94, 93, 96, 93, 94, 95, 96, 91, 93, 95, 95, 95, 92, 93, 94, 91, 94, 92, and 93. Plot the histogram of the data

```
% Thread breaking strength data for 20 tests.
y = [92,94,93,96,93,94,95,96,91,93,...
   95,95,95,92,93,94,91,94,92,93];
histogram(y,'FaceColor','none'),...
   axis([90 97 0 6]),...
   ylabel('Absolute Frequency'),...
   xlabel('Thread Strength (N)'),...
   title('Absolute Frequency Histogram for 20 Tests')
```

This creates the next figure.

Histograms for 20 Tests of Thread Strength



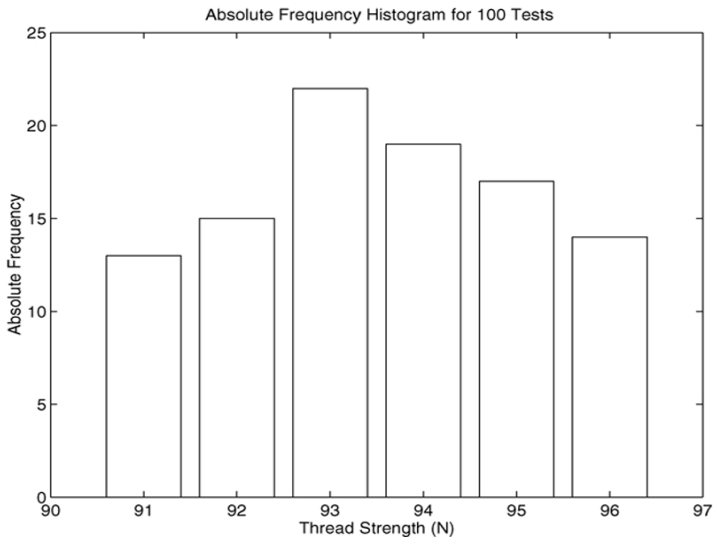
Aggregating the Data

Rather than typing in a lot of data, sometimes we can aggregate them. For example,

```
% Thread strength data for 100 tests.
y = [91*ones(1,13),92*ones(1,15),93*ones(1,22),...
94*ones(1,19),95*ones(1,17),96*ones(1,14)];
histogram(y,'FaceColor','none'),…
 ylabel('Absolute Frequency'),...
 xlabel('Thread Strength (N)'),...
 title('Absolute Frequency Histogram for 100 Tests');
```

See the next slide.

Absolute Frequency Histogram for 100 Thread Tests

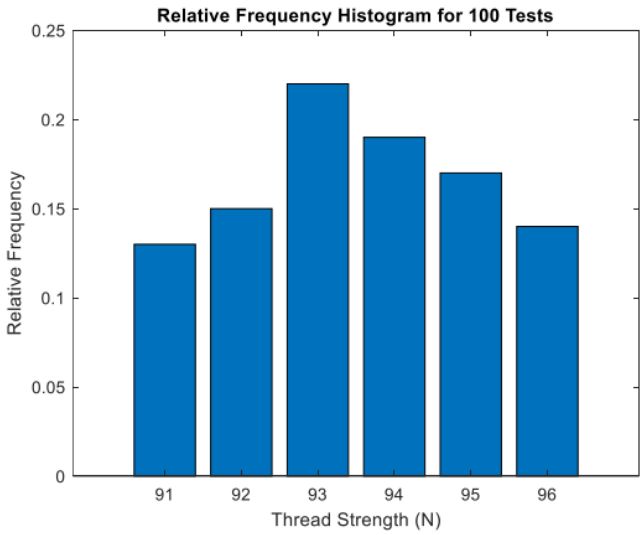


Use of the `bar` Function for *Relative* Frequency Histograms

```
% Relative frequency histogram using ...
 the  bar function.
tests = 100;
y = [13,15,22,19,17,14]/tests;
x = [91:96];
bar(x,y),ylabel('Relative Frequency'),...
 xlabel('Thread Strength (N)'),...
 title('Relative Frequency Histogram ...
 for 100 Tests')
```

This creates the next figure.

Relative frequency histogram for 100 thread tests

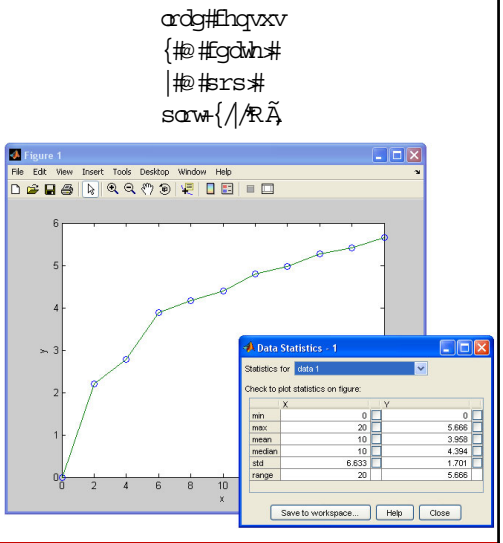


Histogram Functions

Command	Description
bar(x,y)	Creates a bar chart of y versus x using the default color scheme.
bar(x,y,'w')	Creates a bar chart of y versus x using unshaded rectangles.
histogram(y)	Aggregates the data in the vector y into bins of uniform width between the minimum and maximum values in y, using the default color.
histogram(y,n)	Aggregates the data in the vector y into n bins of uniform width between the minimum and maximum values in y.
histogram(y,'FaceColor','w')	Aggregates the data in the vector y into bins of uniform width between the minimum and maximum values in y, using unshaded (white) rectangles.

The Data Statistics Tool

With the Data Statistics tool you can calculate statistics for data and add plots of the statistics to a graph of the data. The tool is accessed from the Figure window after you plot the data. Click on the Tools menu, then select **Data Statistics.**



Scaled Frequency Histogram

Data can be plotted using either the absolute or relative frequencies, or by using data scaled so that the total area under the histogram's rectangles is 1. This *scaled frequency histogram* is the absolute frequency histogram divided by the total area of that histogram.

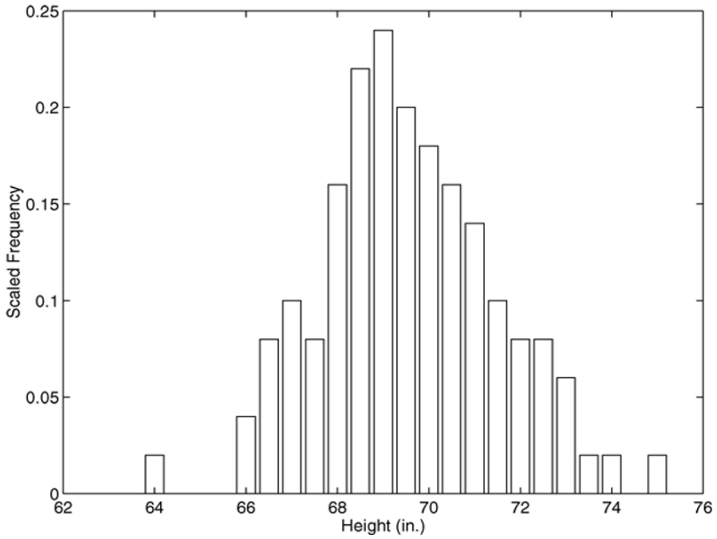
The area of each rectangle on the absolute frequency histogram equals the bin width times the absolute frequency for that bin. Because all the rectangles have the same width, the total area is the bin width times the sum of the absolute frequencies.

Scaled Frequency Histogram

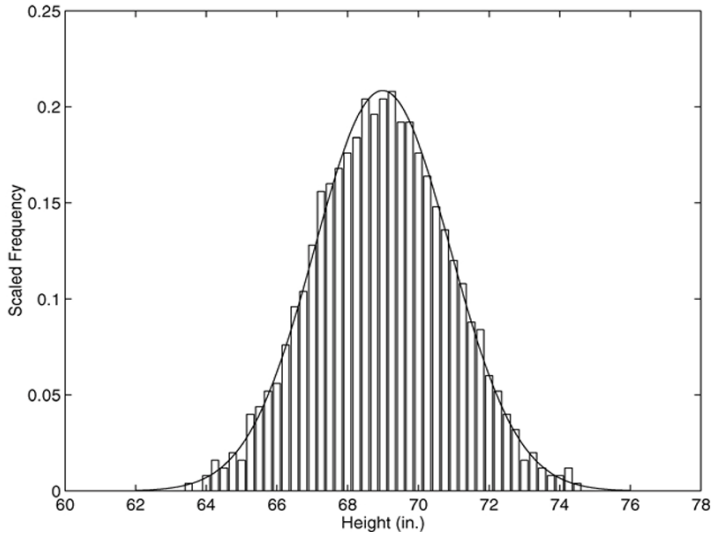
```
% Absolute frequency data.
y_abs=[1,0,0,0,2,4,5,4,8,11,12,10,9,8,…
   7,5,4,4,3,1,1,0,1];
binwidth = 0.5;
% Compute scaled frequency data.
area = binwidth*sum(y_abs);
y_scaled = y_abs/area;
% Define the bins.
bins = [64:binwidth:75];
% Plot the scaled histogram.
bar(bins,y_scaled),...
   ylabel('Scaled Frequency'),…
   xlabel('Height (in.)')
```

This creates the next figure.

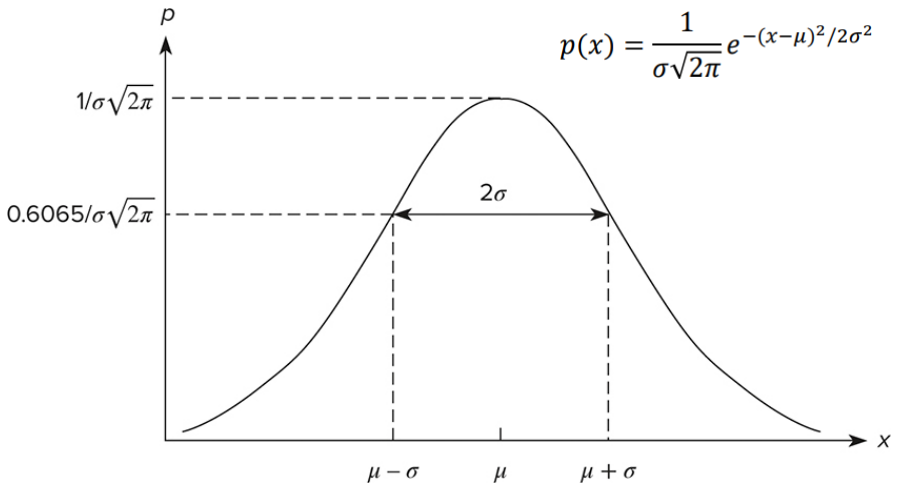
Scaled Histogram of Height Data



Scaled Histogram of Height Data for Very Many Measurements – Normally Distributed

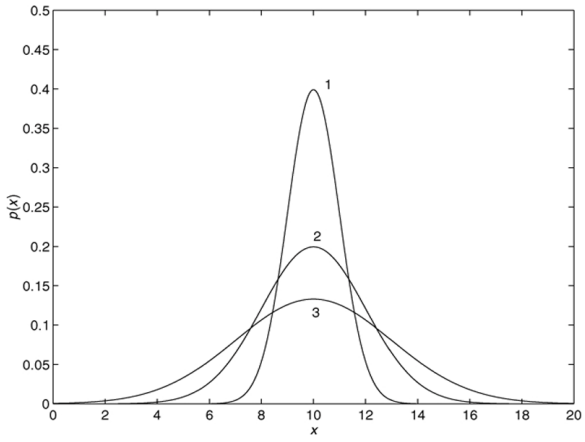


The Basic Shape of the Normal Distribution Curve

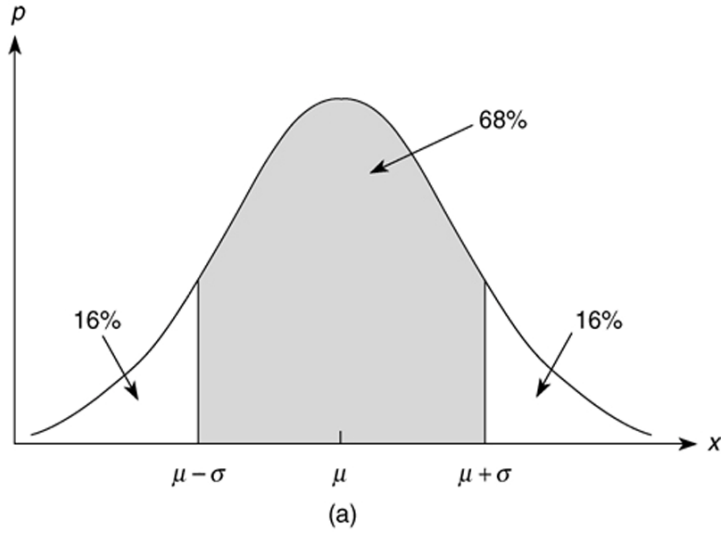


The Effect on the Normal Distribution Curve of Increasing σ

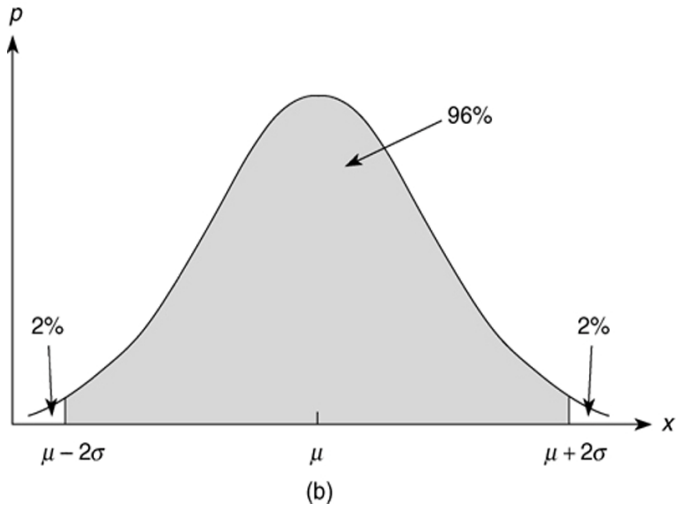
For this case $\mu = 10$, and the three curves correspond to $\sigma = 1$, $\sigma = 2$, and $\sigma = 3$.



Probability Interpretation of the $\mu \pm \sigma$ Limits



Probability Interpretation of the $\mu \pm 2\sigma$ Limits



Mean and Standard Deviation of Heights

Statistical analysis of data on human proportions is required in many engineering applications. For example, designers of submarine crew quarters need to know how small they can make bunk lengths without eliminating a large percentage of prospective crew members. Use MATLAB to estimate the mean and standard deviation for the height data given below.

Height (in.)	Frequency	Height (in.)	Frequency
64	1	70	9
64.5	0	70.5	8
65	0	71	7
65.5	0	71.5	5
66	2	72	4
66.5	4	72.5	4
67	5	73	3
67.5	4	73.5	1
68	8	74	1
68.5	11	74.5	0
69	12	75	1
69.5	10		

Mean and Standard Deviation of Heights

```
% Absolute frequency data.
y_abs = [1,0,0,0,2,4,5,4,8,11,12,10,9,8,7,5,4,4,3,1,1,0,1];
binwidth = 0.5;
%  Define the bins.
bins = [64:binwidth:75];
% Fill the vector y_raw with the raw data.
% Start with an empty vector.
y_raw = [];
for i = 1:length(y_abs)
if y_abs(i)>0
    new = bins(i) *ones(1,y_abs(i));
else
    new = [];
end
y_raw = [y_raw,new];
end
% Compute the mean and standard deviation.
mu = mean(y_raw),sigma = std(y_raw)
```

Results: mu = 69.6 in., sigma = 1.96 in

Probability Calculations with the Error Function `erf`

The probability that the random variable x is less than or equal to b is written as P(x ≤ b) if the outcomes are normally distributed.

$$P(x \le b) = \frac{1}{2}\left[1+\text{erf}\left(\frac{b-\mu}{\sigma\sqrt{2}}\right)\right]$$

The probability that the random variable x is no less than a and no greater than b is written as P(a≤ x ≤ b). It can be computed as follows:

$$P(a \le x \le b) = \frac{1}{2}\left[\text{erf}\left(\frac{b-\mu}{\sigma\sqrt{2}}\right) - \text{erf}\left(\frac{a-\mu}{\sigma\sqrt{2}}\right)\right]$$

Estimation of Height Distribution

Estimate how many 20-year-old men are no taller than 68 inch. How many are within 3 inch of the mean? The mean and standard deviation were found to be $\mu = 69.3$ inch and $\sigma = 1.96$ inch.

```
mu = 69.3;
s = 1.96;
% How many are no taller than 68 inches?
b1 = 68;
P1 = (1+erf((b1-mu)/(s*sqrt(2))))/2
% How many are within 3 inches of the mean?
b2 = mu + 3;
a2 = mu - 3;
P2 = (erf((b2-mu)/(s*sqrt(2)))-erf((a2-mu)/(s*sqrt(2))))/2
```

Answers: P1 = 0.2536 and P2 = 0.8741. Thus, 25 percent are estimated to be 68 inch or less in height, and 87 percent are estimated to be between 66.3 and 72.3 inch tall.

Sums and Differences of Random Variables

It can be proved that the mean of the sum (or difference) of two independent normally distributed random variables equals the sum (or difference) of their means, but the variance is always the sum of the two variances. That is, if x and y are normally distributed with means μ_x and μ_y, and variances σ_x and σ_y, and if $u = x + y$ and $v = x - y$, then

$$\mu_u = \mu_x + \mu_y$$
$$\mu_v = \mu_x - \mu_y$$
$$\sigma_u^2 = \sigma_v^2 = \sigma_x^2 + \sigma_y^2$$

Random Number Generation

The MATLAB function rand generates random numbers uniformly distributed over the *open* interval (0,1) using an algorithm called a *random number generator*, which requires a "*seed*" number to start. Type rand to obtain a single random number in the open interval (0,1). Typing rand again generates a different number. For example,

```
>>rand
ans =
   0.7502
>>rand
ans =
   0.5184
```

Random Number Generation

For example, the following script makes a random choice between two equally probable alternatives and computes the statistics for 100 simulated tosses of a fair coin.

```
% Simulates multiple tosses of a fair coin.

heads = 0;
tails = 0;
for k = 1:100
   if rand < 0.5
   heads = heads + 1;
else
   tails = tails + 1;
end
end
heads
tails
```


Random Number Generation

Every time MATLAB starts, the generator is reset to the same state. Therefore, a `rand` command gives an identical result every time it is executed immediately following startup, and you will see the same sequence you saw in a previous startup.

To avoid getting the same random number when MATLAB restarts, use the command `rng('shuffle')` before calling `rand`. This initializes the generator based on the current time given by the computer's CPU clock. To repeat a result obtained at startup without restarting, reset the generator to the startup state by using `rng('default')`. For example,

```
>>rand
ans =
    0.7502
>>rng('default')
>>rand
ans =
    0.7502
```


Extended Syntax of the `rand` Function

Command	Description
`rand`	Generates a single uniformly distributed random number between 0 and 1.
`rand(n)`	Generates an $n \times n$ matrix containing uniformly distributed random numbers between 0 and 1
`rand(m,n)`	Generates an $m \times n$ matrix containing uniformly distributed random numbers between 0 and 1.

Normally Distributed Random Numbers

Command	Description
`randn`	Generates a single normally distributed random number having a mean of 0 and a standard deviation of 1.
`randn(n)`	Generates an $n \times n$ matrix containing normally distributed random numbers having a mean of 0 and a standard deviation of 1.
`randn(m,n)`	Generates an $m \times n$ matrix containing normally distributed random numbers having a mean of 0 and a standard deviation of 1.

Example

Write a script file to play a simple number guessing game as follows. The script should generate a random integer in the range 1, 2, 3, . . . , 14, 15. It should provide for the player to make repeated guesses of the number, and it should indicate if the player has won or give the player a hint after each wrong guess. The responses and hints are as follows:

- "You won" and then stop the game.
- "Very close" if the guess is within 1 of the correct number.
- "Getting close" if the guess is within 2 or 3 of the correct number.
- "Not close" if the guess is not within 3 of the correct number.

Use rng('shuffle') to seed the random number generator based on the current time

Example - Solution

```
clear
rng('shuffle')
x = randperm(15); number = x(1); k = 0;status = 0;
disp('You will have 14 attempts to guess a number from 1 to 15.')
while(k < 14)&(status == 0)
k = k+1; Turn = k
guess = input('Enter your guess (1 to 15)')
diff = abs(guess-number);
if diff == 0
disp('You won!')
status = 1;
elseif diff > 3
disp('Not close.')
status = 0;
Else
status = 0;
if (diff == 2)|(diff == 3)
disp('Getting close.')
Else
disp('Very close.')
End
end
end
if k >= 14
disp('You have no more guesses left.')
Else
disp('The number of guesses you took was')
disp(k)
end
```


Example – Solution Cont'd

```
clear
rng('shuffle')
x = randperm(15); number = x(1); k = 0;status = 0;
disp('You will have 14 attempts to guess a number from 1 to 15.')
   While (k < 14)&(status == 0)
          k = k+1; Turn = k
          guess = input('Enter your guess (1 to 15)')
          diff = abs(guess-number);
                if diff == 0
                      disp('You won!')
                      status = 1;
                elseif diff > 3
                      disp('Not close.')
                      status = 0;
                else
                      status = 0;
                            if (diff == 2)|(diff == 3)
                                  disp('Getting close.')
                            else
                                  disp('Very close.')
                            end
                end
   end
if k >= 14
        disp('You have no more guesses left.')
   else
        disp('The number of guesses you took was')
        disp(k)
end
```


Normally Distributed Random Numbers

You can generate a sequence of normally distributed numbers having a mean μ and standard deviation σ from a normally distributed sequence having a mean of 0 and a standard deviation of 1. You do this by multiplying the values by σ and adding μ to each result.

Thus, if x is a random number with a mean of 0 and a standard deviation of 1, use the following equation to generate a new random number y having a standard deviation of σ and a mean of μ.

$$y = \sigma x + \mu$$

Normally Distributed Random Numbers

If y and x are linearly related as $y = bx + c$ and if x is normally distributed with a mean μ_x and standard deviation σ_x, it can be shown that the mean and standard deviation of y are given by

$$\mu_y = b\mu_x + c$$
$$\sigma_y = |b|\sigma_x$$

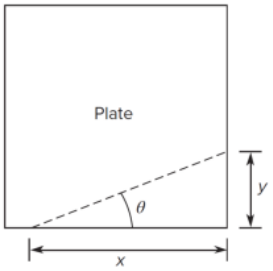
For example, to generate a vector y containing 2000 random numbers normally distributed with a mean of 5 and a standard deviation of 3, you type

```
y = 3*randn(1,2000) + 5.
```


Example to Determine the Statistical Distribution of a Nonlinear Function $\theta = \text{atan}(y/x)$

Suppose you must cut a triangular piece off the corner of a square plate by measuring the distances x and y from the corner. The desired value of x is 10 $in.$, and the desired value of θ is 20°. This requires that y = 3.64 $in.$ We are told that measurements of x and y are normally distributed with means of 10 and 3.64, respectively, with a standard deviation equal to 0.05 $in.$ Determine the standard deviation of θ and plot the relative frequency histogram for θ.

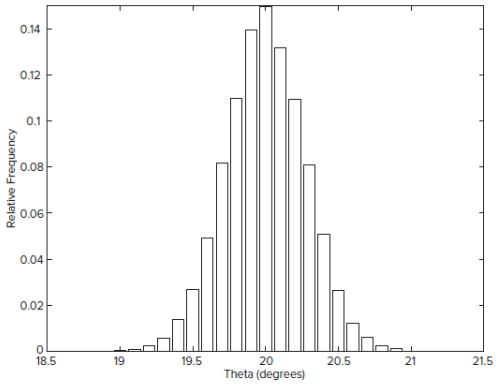
$$\theta = \tan^{-1}(y/x)$$



Determine the Statistical Distribution of the Nonlinear Function $\theta = \text{atan}(y/x)$

```
s = 0.05; % standard deviation of x and y
n = 8000; % number of random simulations
x = 10 + s*randn(1,n);
y = 3.64 + s*randn(1,n);
theta = (180/pi) *atan(y./x);
mean_theta = mean(theta)
sigma_theta = std(theta)
xp = 19:0.1:21;
histogram(theta,xp,'Normalization','probability'),...
   xlabel('Theta (degrees)'),...
   ylabel('Relative Frequency')
```


Determine the Statistical Distribution of the Nonlinear Function $\theta = \text{atan}(y/x)$



μ =19.9993°
σ =0.2730°

From the histogram we can calculate that approximately 65 percent of the values of θ lie between 19.8 and 20.2. This range corresponds to a standard deviation of 0.2°, not 0.273° as calculated from the simulation data.

Generating Random Integers

To simulate some games, such as dice, you must be able to generate only random integers. You can do this with the `randperm(n)` function, which generates a row vector containing a random permutation of the integers from 1 to *n* inclusive.

For example, `randperm(6)` might generate the vector [3 2 6 4 1 5], or some other permutation of the numbers from 1 to 6. Note that `randperm` calls `rand` and therefore changes the state of the generator.

The extended syntax is `randperm(n,k)`. This generates a row vector containing k unique integers selected randomly from 1 to n inclusive.

Generating Random Integers

The function `randi([a,b],[m,n])` returns an *m*-by-*n* matrix containing random integer values between *a* and *b*. Typing `randi(imax)` returns a scalar between 1 and `imax`.

For example,

```
>> randi(20,[1,5])
ans =
     7  3  9  19  16
>>randi([5,20],[1,5])
ans =
     5  12  11  17  17
>> randi(6)
ans =
     3
```

Generating Random Integers

Note that `randperm` returns *unique* integers, whereas the integer values returned by `randi` may be repeated

For example, `randperm(6)` might generate the vector [3 2 6 4 1 5], or some other permutation of the numbers from 1 to 6.

Whereas `randi(6,[1,6])` might generate [1 6 3 5 2 5].

Linear Interpolation with the `interp1` Function

If x_int is a vector containing the value or values of the independent variable at which we wish to estimate the dependent variable y, then typing interp1(x,y,x_int) produces a vector the same size as x_int containing the interpolated values of y that correspond to x_int. The values in x must be in ascending order, and the values in the interpolation vector x_int must lie within the range of the values in x.

For example, estimate the values of y at x = 8 and x = 10.

```
>>x = [7, 9, 11, 12]; y = [49, 57, 71, 75];
>>x_int = [8, 10];
>>y_int = interp1(x,y,x_int)
y_int =
    53
    64
```


Linear Interpolation with the `interp1` Function

The `interp1` function can interpolate in a table of values by defining y to be a matrix instead of a vector. Suppose that we have temperature measurements at three locations and the measurements at 8 and 10 a.m. are missing. The data is:

Time	Loc 1	Loc 2	Loc 3
7 a.m.	49	52	54
9 a.m.	57	60	61
11 a.m.	71	73	75
12 noon	75	79	81

Estimate of the temperatures at 8 and 10 a.m. at each location.

Example: Linear Interpolation

Define the matrix `temp` to be this table.

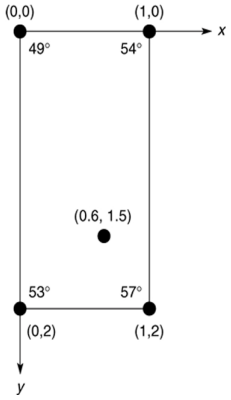
```
≫temp(:,1) = [7, 9, 11, 12]'; temp(:,2) = [49, 57, 71, 75]';
≫ temp(:,3) = [52, 60, 73, 79]'; temp(:,4) = [54, 61, 75, 81]';
≫ x_int = [8, 10]';
≫ y_int = interp1(temp(:,1),temp(:,2:4),×_int);
y_int =
53.0000 56.0000 57.5000
64.0000 65.5000 68.0000
```

Thus, the estimated temperatures at 8 a.m. at each location are 53, 56, and 57.5°F, respectively. At 10 a.m. the estimated temperatures are 64, 65.5, and 68°F.

Two-Dimensional Interpolation

For the function $z = f(x, y)$, to estimate the value of z for $x = x\_i$ and $y = y\_i$, the syntax is `interp2(x,y,z,x_i,y_i)`. For the temperature measurements shown in the figure, estimate the

temperature at the point whose coordinates are (0.6, 1.5).




```
>>x = [0,1];
>>y = [0,2];
>>z = [49,54;53,57]
>>interp2(x,y,z,0.6,1.5)
ans =
   54.5500
```

The estimated temperature is 54.55°F.

Spline Interpolation

- High-order polynomials interpolation can exhibit undesired behavior between the data points, and this can make them unsuitable for interpolation.
- A widely used alternative procedure is to fit the data points using a lower-order polynomial between each pair of adjacent data points. This method is called spline interpolation
- The splines used by illustrators to draw a smooth curve through a set of points.
- Spline interpolation obtains an exact fit that is also smooth. The most common procedure uses cubic polynomials, called cubic splines

Cubic-spline Interpolation

If the data are given as n pairs of (x, y) values, then $n - 1$ cubic polynomials are used. Each has the form

$$y_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

For $x_i \le x \le x_{i+1}$ and $i = 1, 2, \ldots, n - 1$. The coefficients a_i, b_i, c_i and d_i for each polynomial are determined so that the following three conditions are satisfied for each polynomial:

1. The polynomial must pass through the data points at its endpoints at xi and xi+1.
2. The slopes of adjacent polynomials must be equal at their common data point.
3. The curvatures of adjacent polynomials must be equal at their common data point.

Example: Cubic-spline Interpolation

A set of cubic splines for the temperature data given earlier follows. (y represents the temperature values, and x represents the hourly values). The data are repeated here

x	7	9	11	12
y	49	57	71	75

Obtain the following polynomials.

For $7 \le x \le 9$,

$$y_1(x) = -0.35(x - 7)^3 + 2.85(x - 7)^2 - 0.3(x - 7) + 49$$

For $9 \le x \le 11$,

$$y_2(x) = -0.35(x - 9)^3 + 0.75(x - 9)^2 + 6.9(x - 7) + 57$$

For $11 \le x \le 12$,

$$y_3(x) = -0.35(x - 11)^3 - 1.35(x - 11)^2 + 5.7(x - 11) + 71$$

Cubic-Spline Interpolation

Cubic-spline interpolation The following session produces and plots a cubic-spline fit, using an increment of 0.01 in the *x* values.

```
x = [7,9,11,12]; y = [49,57,71,75];
x_int = 7:0.01:12;
y_int = spline(x,y,x_int);
plot(x,y, 'o',x,y,'--',x_int,y_int),...
   xlabel('Time (hr)'), ...
   ylabel('Temperature (deg F)'),...
   title('Temperature Measurements …
   at a Single Location'),...
   axis([7 12 45 80])
```

This produces the figure on the next slide.

Cubic-Spline Interpolation

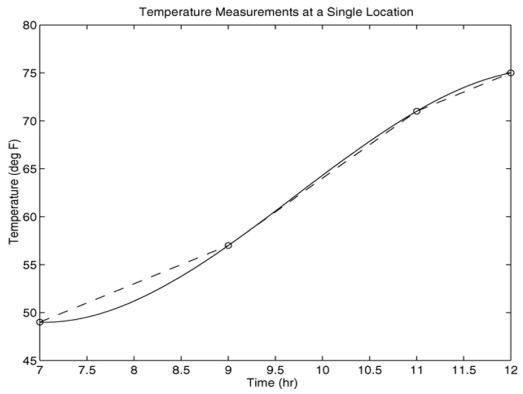
```
x = [7,9,11,12]; y = [49,57,71,75];
x_int = 7:0.01:12;
y_int = spline(x,y,x_int);
plot(x,y,'o',x,y,'--',x_int,y_int),...
xlabel('Time (hr)'),ylabel('Temperature (deg F)'),...
title('Measurements at a Single Location'),...
axis([7 12 45 80])
[breaks, coeffs, m, n] = unmkpp(spline(x,y))

breaks =    7    9   11   12
coeffs =
  -0.3500   2.8500  -0.3000  49.0000
  -0.3500   0.7500   6.9000  57.0000
  -0.3500  -1.3500   5.7000  71.0000
m =     3     n =     4
```

[BREAKS,COEFS,L,K,D] = unmkpp(PP) extracts from the piecewise polynomial PP its breaks, coefficients, number of pieces, order and dimension of its target. PP would have been created by SPLINE or the spline utility MKPP.

Linear and Cubic-Spline Interpolation

The dashed lines represent linear interpolation, and the solid curve is the cubic spline. Evaluating the spline at $x = 8$, we obtain $y(8) = 51°F$. This estimate is different from the 53°F estimate from linear interpolation.



Alternate Form

```
y_est = interp1(x,y,x_est, method)
```

Returns a column vector *y_est* that contains the estimated values of *y* that correspond to the x values specified in the vector *x_est*, using interpolation specified by method. The choices for method are 'nearest', 'linear', 'next', previous','spline', and 'pchip'.

```
y_int = spline(x,y,x_int)
```

Computes a cubic-spline interpolation where `x` and `y` are vectors containing the data and `x_int` is a vector containing the values of the independent variable *x* at which we wish to estimate the dependent variable *y*. The result `y_int` is a vector the same size as `x_int` containing the interpolated values of *y* that correspond to `x_int`.

Alternate Form

The `pchip` Function

```
y_int = pchip(x,y,x_int)
```

Similar to `spline` but uses piecewise cubic Hermite polynomials for interpolation to preserve shape and respect monotonicity.

The `unmkpp` Function

```
[breaks, coeffs, m, n] = unmkpp(spline(x,y))
```

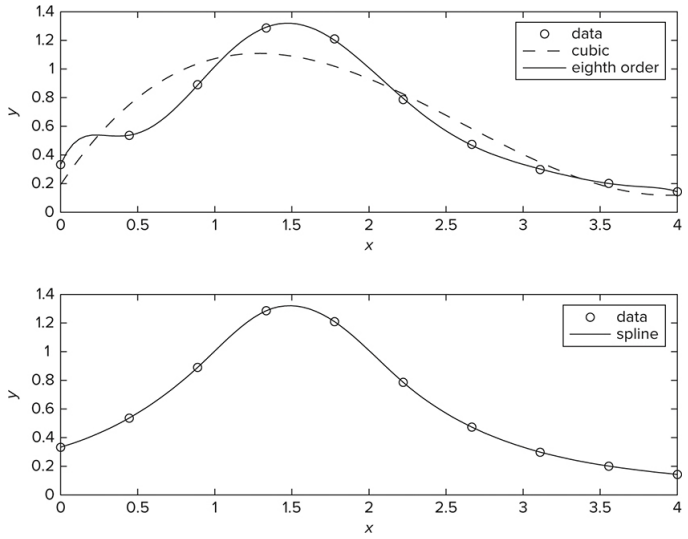
Computes the coefficients of the cubic-spline polynomials for the data in `x` and `y`. The vector `breaks` contains the x values, and the matrix `coeffs` is an $m \times n$ matrix containing the polynomial coefficients. The scalars `m` and `n` give the dimensions of the matrix `coeffs; m` is the number of polynomials, and `n` is the number of coefficients for each polynomial.

Comparison of Cubic Spline and Eighth-Order Polynomial Interpolation

The next slide illustrates interpolation using a cubic polynomial and an eighth-order polynomial (top graph). The cubic is not satisfactory in this case, and the eighth order polynomial is not suitable for interpolation over the interval $0 < x < 0.5$.

The cubic spline does a better job in this case (bottom graph).

Top: Cubic and eighth-order polynomial interpolation. Bottom: Cubic spline.

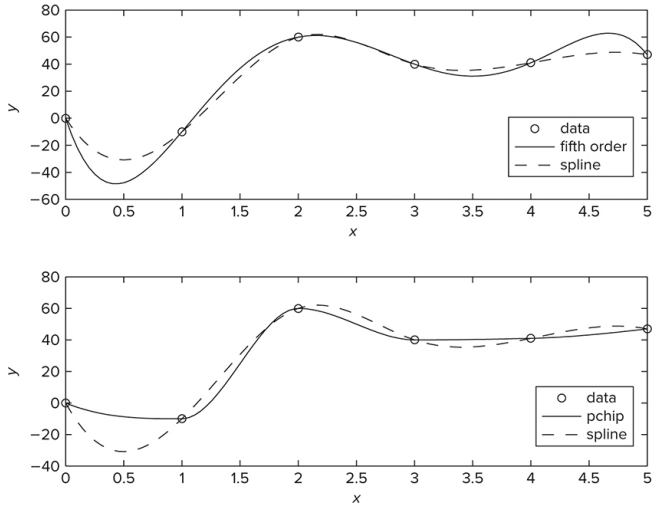


Fifth-Order Polynomial versus Cubic Spline

The next slide illustrates interpolation using a fifth order polynomial and a cubic spline (top graph). The cubic spline is better because the fifth order polynomial displays wide variations between the data points.

The pchip polynomial does a better job than the cubic spline in this case (bottom graph).

Top: Fifth order polynomial and cubic spline interpolation. Bottom: pchip and cubic spline interpolation.



Because learning changes everything.®

www.mheducation.com