# Random Numbers and Statistics

Trevor Spiteri
trevor.spiteri@um.edu.mt
http://staff.um.edu.mt/trevor.spiteri

Department of Communications and Computer Engineering
Faculty of Information and Communication Technology
University of Malta

14 March, 2008

# Outline

# Outline

# Uniformly distributed numbers

- A random number generator generates a sequence of numbers that lack any pattern, that is, they appear to be random.
- MATLAB has the `rand` function to generate random numbers.
- The numbers are uniformly distributed from 0 to 1.
- To generate a single random number, call `rand` with no parameters: `>> r = rand`
- To generate a square $n \times n$ matrix of random numbers:
  `>> r = rand(n)`
- To generate an $m \times n$ matrix of random numbers:
  `>> r = rand(m, n)`

# Simulating an event with a known probability

- We can simulate an event with a given probability.
- Suppose that the probability of an event E, $P(E) = 0.3$.
- This can be done as follows:

```
if rand < 0.3
    'Event E'
else
    'No Event E'
end
```

# Generating a random number in a given range

- The `rand` function generates uniformly-distributed numbers in $[0,1]$.
- To generate random numbers in the interval $[a,b]$, we multiply the result by $(b-a)$ and add $a$.
- For example, to generate a number in the interval $[10,50]$:

```
>> a = 10; b = 50;
>> r = rand * (b-a) + a;
```

- To generate a number in the interval $[-10,10]$:

```
>> a = -10; b = 10;
>> r = rand * (b-a) + a;
```

# Generating random integers

- We can also generate an integer $n$ in the range $a \leq n \leq b$, where $a$ and $b$ are integers.
- We first generate a random number in the interval $[a, b+1]$, and then we round down.
- For example, to generate an integer $n$ in the interval$[10, 20]$:

```
>> a = 10; b = 20;
>> n = floor(rand * (b+1 - a) + a);
```

# The state of the random number generator

- The random number generator keeps an internal state.
- Setting the generator to the same state enables you to repeat computations.
- On startup, MATLAB resets the `rand` state; `rand` generates the same sequence in each session unless the state is changed.
- To change the state, use: `>> rand(method, s)`
- `method` can be any of the methods shown below.
- `s` is usually derived from the time to give a different value every time. Usually it is set to `sum(100 * clock)`.

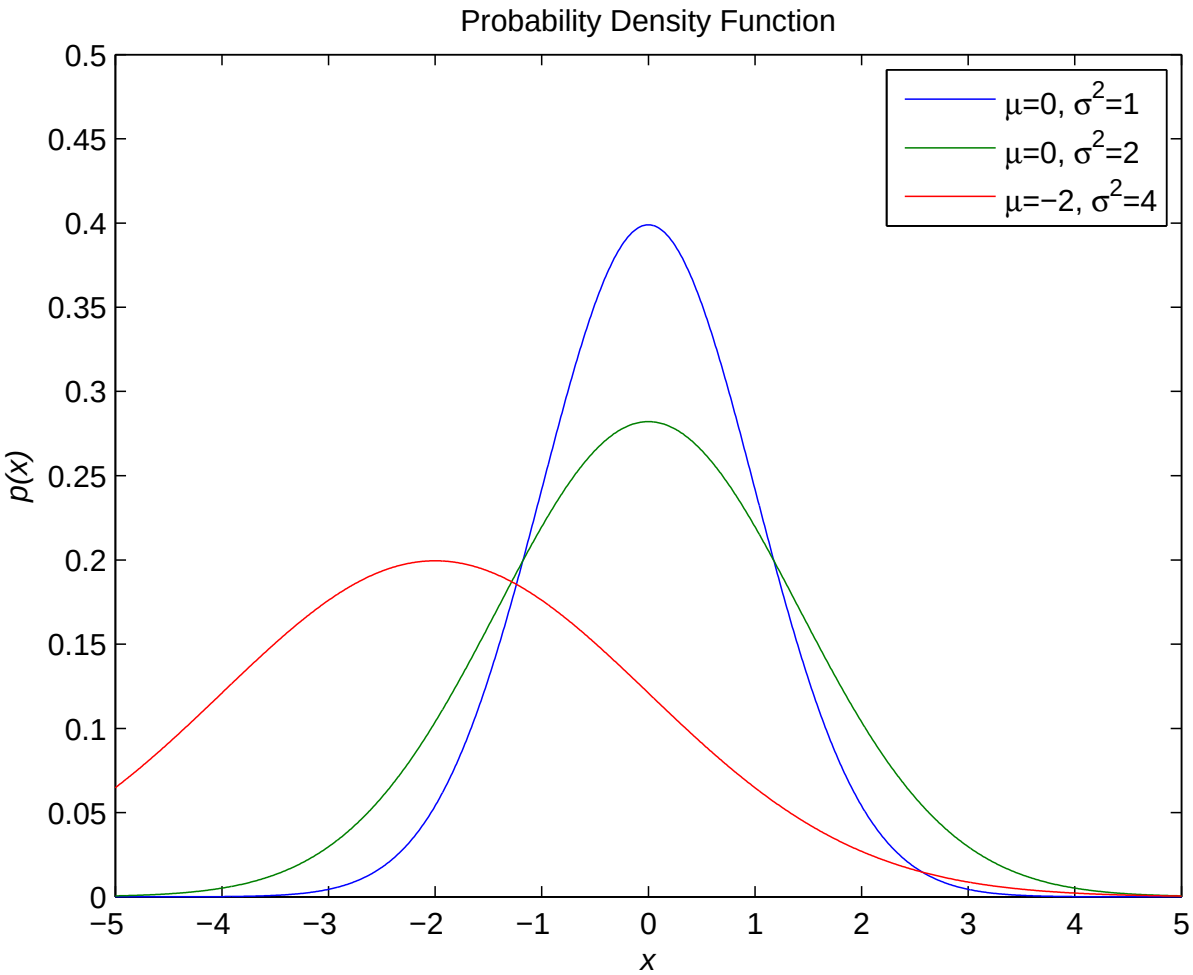
Table : Methods for the generator

| | |
|---|---|
| `'twister'` | Default in MATLAB versions 7.4 and later |
| `'state'` | Default in MATLAB versions 5 through 7.3 |
| `'seed'` | Default in MATLAB versions 4 and earlier |

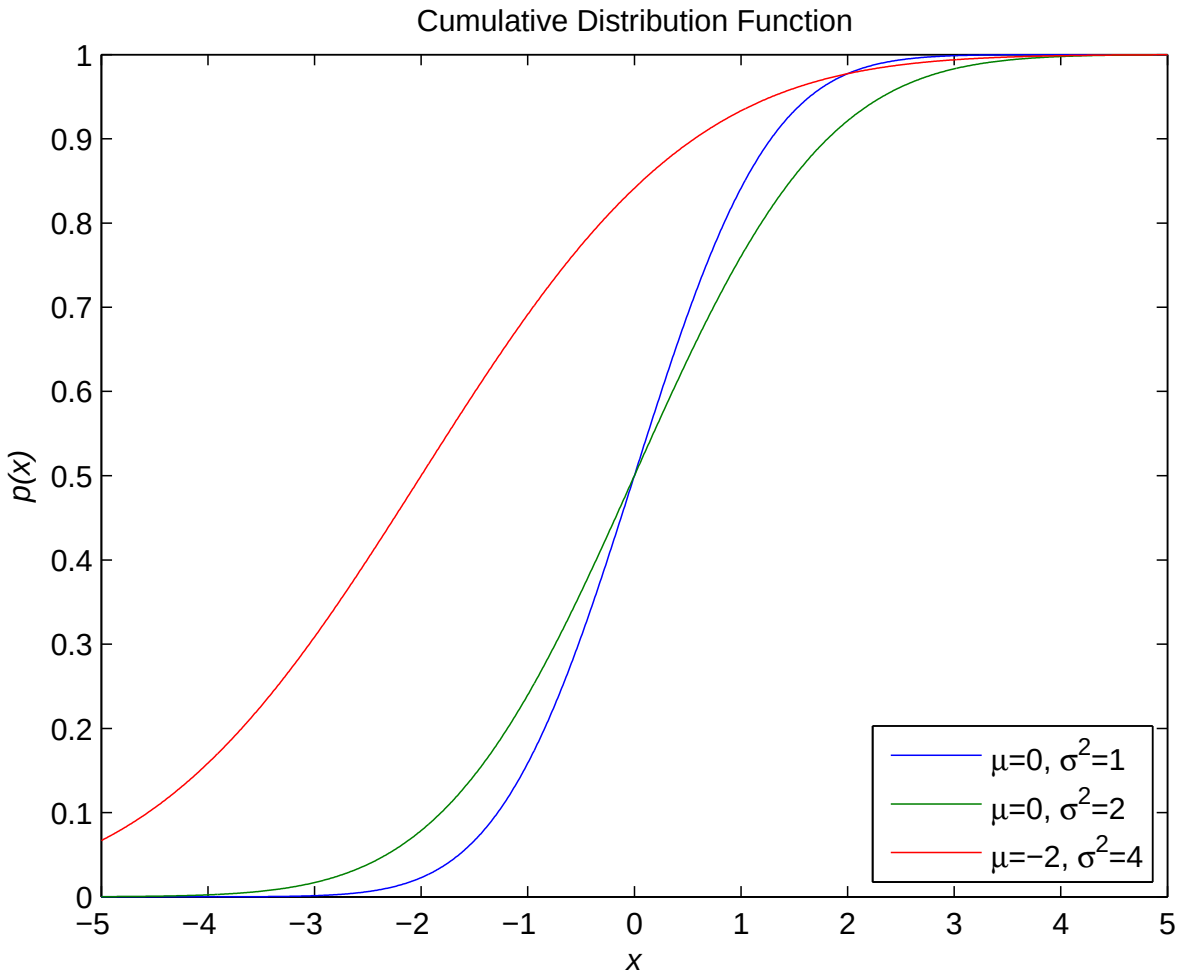
# The normal distribution

- The normal distribution is also called the Gaussian distribution.
- It is a family of continuous probability distributions.
- The normal distribution has a mean $\mu$ and a variance $\sigma^2$, which is the square of the standard deviation $\sigma$.
- When many small, independant, random numbers are added, the result approaches the normal distribution.
- Many physical phenomena can be approximated well by the normal distribution.

# The probability density function



# The cumulative distribution function



# Generating random numbers with a normal distribution

- The `randn` function generates random numbers with a normal distribution, with $\mu = 0$ and $\sigma = 1$.
- The `randn` function has a separate internal state from the `rand` function, and should be seeded using:
  `>> randn('method', s)`
- To generate a random number `r` from a normal distribution with mean `mu` and standard deviation `sigma`, type:
  `>> r = mu + sigma * randn`
- `randn` can generate a whole matrix in the same way as `rand`.

# Outline

# Drawing histograms using the `hist` command

- `>> hist(y)` distributes the elements of `y` into 10 bins, and draws the bins on a figure.
- If `y` is a matrix, the histogram of each column is created.
- `>> hist(y, m)`, where $m$ is a scalar, distributes the element into `m` bins instead of 10.
- `>> hist(y, x)`, where `x` is a vector, uses bins with centers specified by `x`.
- The $y$-axis for the `hist` command is the count of elements in the bin.
- To scale the $y$-axis, the `hist` command cannot be used on its own.

# Using `hist` together with `bar`

- If `hist` is called with output arguments, the histogram is not automatically drawn in a figure.
- To do this, type: `>> [n, x] = hist(...);`
- `n` contains the number of elements in each bin.
- `x` contains the center of each bin.
- If there is only one output argument `n`, only the number of elements in each bin is returned.
- To draw the returned data, type: `>> bar(x, n)`
- Using this method, you can scale the frequency.

# Scaling the frequency of a histogram

- Suppose we have a vector `score` containing scores from a test.
- To draw a histogram of the scores:

```
>> hist(score, 5 : 10 : 95)
```

- The $y$-axis is the absolute frequency, that is, the number of times each particular outcome occurs.
- To scale the $y$-axis such that it shows a percentage of the students instead of the absolute frequency:

```
>> [n, x] = hist(score, 5 : 10 : 95);
>> n = n / length(score);
>> bar(x, n)
```

# Outline

# Interpolating in one dimension

- To interpolate in one dimension, use the `interp1` command.
- `>> yi = interp1(x, y, xi)`
- `x` is a monotonic vector containing $n$ values for $x$.
- `y` is usually a vector containing $n$ values of $y$.
- `xi` is a vector containing $m$ values of $x$ for which $y$ is required.
- `yi` is the answer vector, containing $m$ results.
- If `y` is a matrix, each column describes a different function.
- In this case, `yi` is a matrix containg one column for each function.
- Each column of `yi` will have $m$ elements.

# Methods of interpolation

- To specify the method of interpolation, use:
  `>> yi = interp1(x, y, xi, method)`

Table : Some possible values for `method`

| | |
|---|---|
| `'nearest'` | nearest neighbour interpolation |
| `'linear'` | linear interpolation (the default) |
| `'spline'` | piecewise cubic spline interpolation |
| `'cubic'` | shape-preserving piecewise cubic interpolation |

# Extrapolation

- `>> yi = interp1(x, y, xi, method)`
- Suppose some values of `xi` lie outside the range of `x`.
- This is extrapolation, not interpolation.
- To use `method` for extrapolation as well as interpolation, type:
  `>> yi = interp1(x, y, xi, method, 'extrap')`
- To replace the values outside the range with a number *n*, type:
  `>> yi = interp1(x, y, xi, method, n)`
- Common values for *n* are `0` and `NaN`.

# Interpolating in two dimensions

- Sometimes we have a function of two variables.
- `x` contains $n$ monotonic values for $x$.
- `y` contains $m$ monotonic values for $x$.
- `z` is an $m \times n$ matrix containing the results of the function.
- To interpolate, use: `>> zi = interp2(x, y, z, xi, yi)`
- `xi` and `yi` can be matrices of the same size.
- The size of `zi` will be the same.
- Other ways of calling `interp2`:

```
zi = interp2(x,y,z, xi,yi, method)
zi = interp2(x,y,z, xi,yi, method, 'extrap')
zi = interp2(x,y,z, xi,yi, method, n)
```