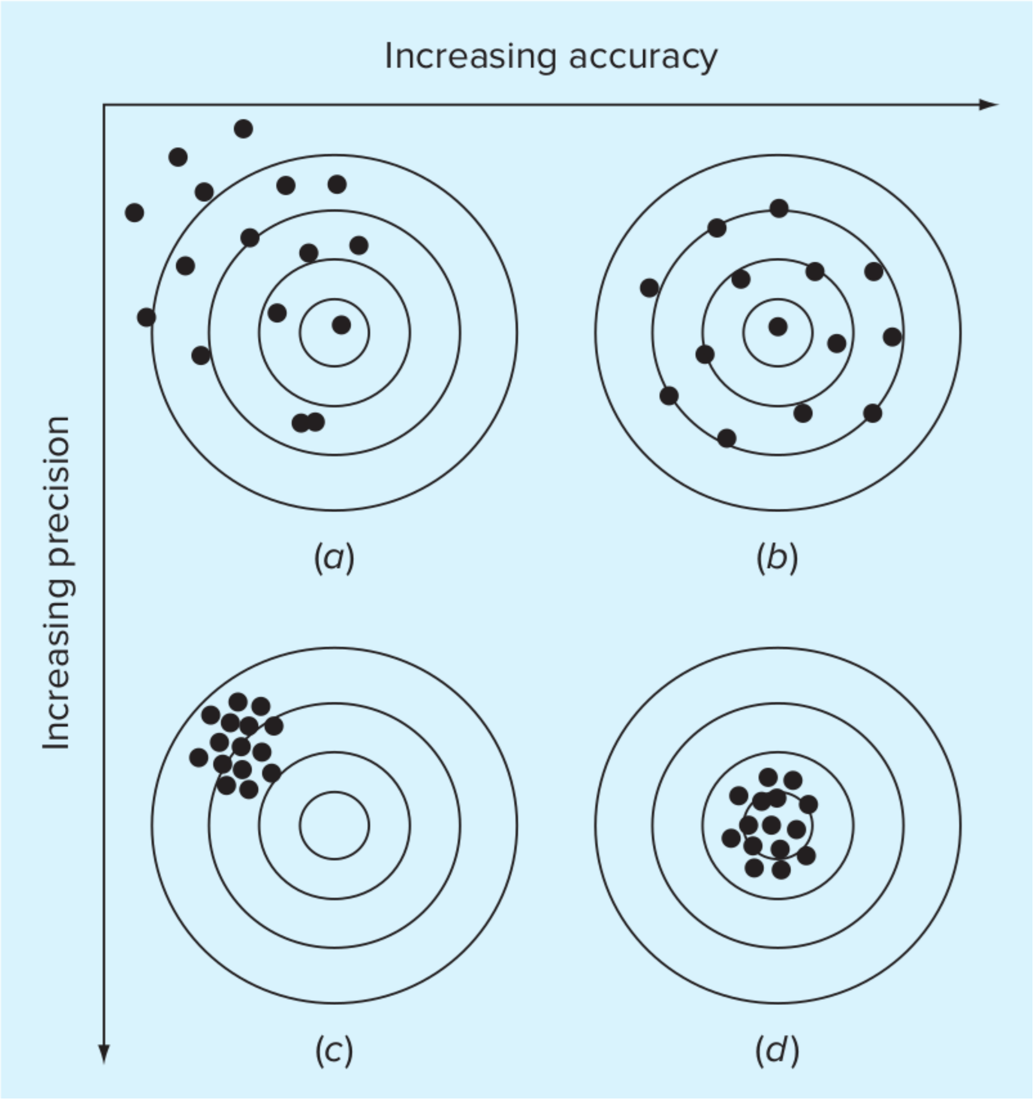
4.1 Computational Error

- Numerical methods always produce computational errors.
- Some are precision errors in computation, but it is often more dangerous when the computer returns a result which meets the criteria but does not have the proper meaning in reality.
- This is quite common with highly non-linear systems which is why we must understand what we are modelling very well in order to interpret the results correctly.

Accuracy vs. Precision



Absolute vs. Relative Error

- **Absolute error** is the difference between the correct value and the estimated value.

- **Relative error** gives an indication of how important that error is.

EXAMPLE 1

We measure a flow rate on a device with a tolerance of 0.5 ml/min.

Absolute error is 0.5 ml/min

Flow rate 1 ml/min

Relative error is
(0.5/1)*100 = 50%

Flow rate 250 ml/min

Relative error is
(0.5/250)*100 = 0.2%

Iterative Measures of Error

- Sometimes a useful **stopping criterion** for a loop is how much a value changes from one iteration to the next (convergence).

$$\boxed{\frac{\text{present approximation} - \text{previous approximation}}{\text{present approximation}}}$$

- This can be used since calculating real absolute error requires knowledge of the solution, which is the thing we don’t know…

- It can be used to make sure we reach a certain number of **decimal places accuracy**, but we must still be careful if the rate of convergence is slow.

Or go to www.pollev.com/jsands601

Which type or error represents how important an error is?

- Absolute error
- Relative error
- Iterative error

Tc 0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

When poll is active, respond at **pollev.com/jsands601**

Which type of error can be used to get accuracy to a certain number of decimal places (if convergence is fast enough)?

- Absolute error
- Relative error
- Iterative error

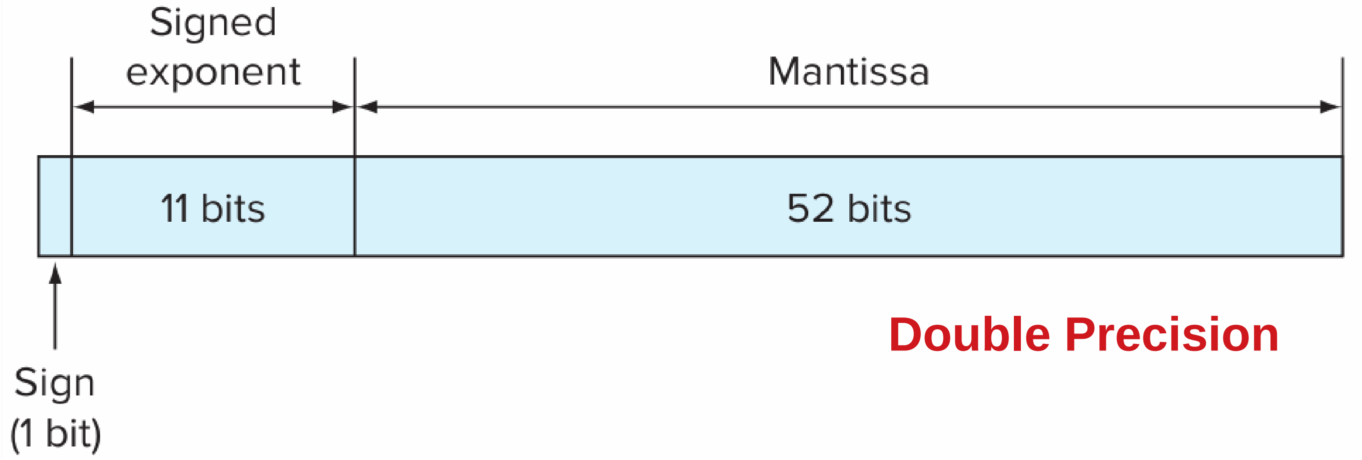
Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Floating Point vs. Fixed Point

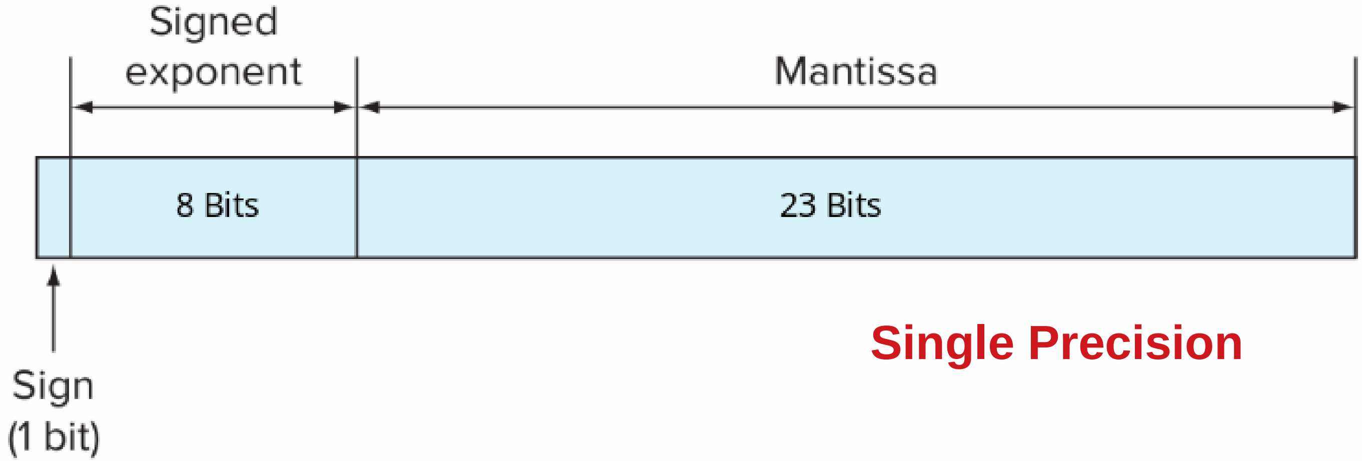
- Floating point numbers are used to compute numbers to a certain number of decimal places. The bytes are used as follows:

64 bit



Double Precision

32 bit



Single Precision

- Just as we would write 132.45 in the base 10 system as:

$$1.3245 \times 10^2 \quad \text{or} \quad 1.3245\text{e}2 \text{ (scientific notation)}$$

we do the same for base 2:

$$0110.001 \longrightarrow 1.10001 \times 2^2$$

- Since the decimal point can move depending on the exponent we call this a **floating point** representation.

- We also have **fixed point** representations.

- It's important to know whether your application can/should use one or the other.

Uses of Floating & Fixed Points

- Floating point numbers are more versatile and have a larger range but they always round to a certain number of significant figures which makes error handling more difficult. Scientific fields tend to use floating point representations.

- Fixed point numbers have a smaller range but because their values are well defined the error is always absolute meaning that they can be kept track of and data is not so easily lost during calculation. Financial systems and embedded systems tend to use this data type.

4.3 Overflow/Underflow Error

- Each bit of information (1 or 0) requires an electrical switch (on or off) in a computer circuit board.
- There is a limited number of these so in programming we can assign a fixed number of allowed bits for data.
- Our accuracy is then limited by the number of bits and MATLAB can give an erroneous result.
- The next example demonstrates this for 8-bit numbers but even using double precision (64 bit) it will eventually run out. **So if numbers are too big or too small MATLAB cannot handle it and will return an error.**

EXAMPLE 3

8 bit data type **uint8**

Range from 0 → 2^8 - 1= 255

```
>> x = uint8(0)
x =
  uint8
   0
>> x = uint8(1)
x =
  uint8
   1
```

```
>> x = uint8(255)
x =
  uint8
   255
>> x = uint8(256)
x =
  uint8
   255
```

Wrong answer because MATLAB ran out of bits to represent such a large number

- It’s also possible that the number is too small for the number of bits available.
- Since more decimal places require more bits, there is a lower bound to the smallest number that can be calculated using a computer.
- This value is called the **machine epsilon** and can be found in MATLAB by using the **eps** function.

```
>> eps
ans =
2.220446049250313e - 016
```

4.4 Round-off Error

- Numbers like pi have an infinite number of decimal places.

- We must therefore approximate its value by taking a certain number of decimal places and rounding to the nearest decimal number.

- During calculations if you round too early you are always working with numbers which have too much error and the error gets bigger the more you use them (error propagation). The final answer will be less accurate than if you just round at the end.

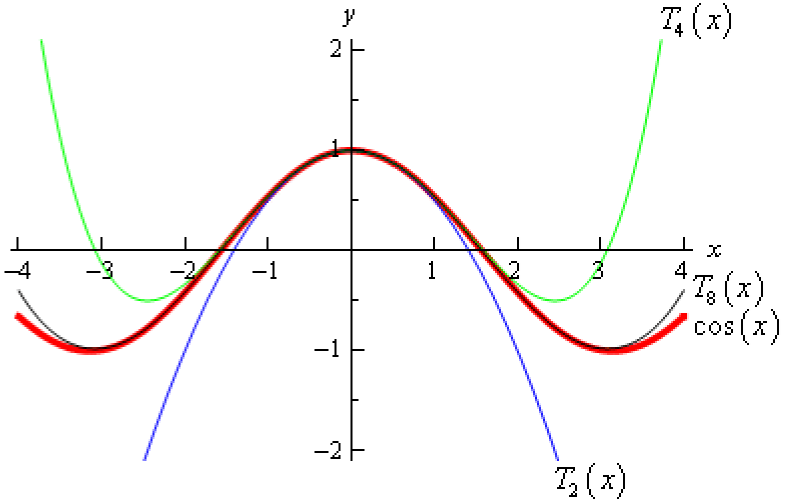
EXAMPLE 4

```
>> x = sin( 1.57 )

x =

0.999999682931835

>> asin( x ) % Inverse function

ans =

1.569999999999999 % Approximately equal to x
```

4.5 Truncation Error

- When we approximate complex functions with a series of terms we can only take a finite number of terms.

- The true value only occurs for infinite terms, which we can never get so we must truncate the terms at some point.



$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots$$

4.6 Human-Induced Error

- Just as it is easy in mathematics to forget a “minus” sign, it is easy in numerical methods to make a typo.

- You may enter a formula wrong and the code produces an output with no warnings from MATLAB.

- This kind of thing is dangerous because it is easy to miss. You must come up with ways of checking every numerical result to make sure this type of error has not occurred.

4.7 Model Errors

- We use equations to describe some physical system.
- These models are approximations of reality themselves.
- If our model equations are not a good description of reality then no matter how good our numerical method is at solving the equations, the answer is meaningless.

4.8 Data Uncertainty

- If we analyse a data set from some experimental observations, even if our maths and programming is correct the equipment may sometimes give false readings, or have been misused during the observation.

- One must also consider tolerance of the equipment that is taking the measurements.

4.9 Making It Simple for MATLAB

- The best practice is to do as much maths as you can in a problem first, then only resort to numerical methods when you absolutely have to.

- Try simplifying your equations, reduce the number of variables in a system (scaling & nondimensionalisation), come up with better formulae etc.

EXAMPLE 5

$$\frac{300!}{299!} = ?$$

```
>> factorial(300)
ans =
   Inf
```

Too big for MATLAB (Overflow)

```
>> factorial(300)/factorial(299)
ans =
   NaN
```

MATLAB cannot do calculations with numbers that are too big

Let's simplify with some maths

$$\frac{300!}{299!} = \frac{300 \times 299 \times 298 \times \ldots \times 1}{299 \times 298 \times \ldots \times 1} = 300$$

4.10 Some Error Formulas

- If we know the real solution we can measure the **true relative error** as:

$$\varepsilon_t = \frac{\text{true value} - \text{approximation}}{\text{true value}}$$

- If we don't know the real solution we use **approximate relative error**:

$$\varepsilon_a = \frac{\text{present approximation} - \text{previous approximation}}{\text{present approximation}}$$

- **Absolute errors** are simply the numerators in the above formulae.

- Often we just want to use the **approximate absolute value** of the error as a stopping criterion.

4.11 Root Finding

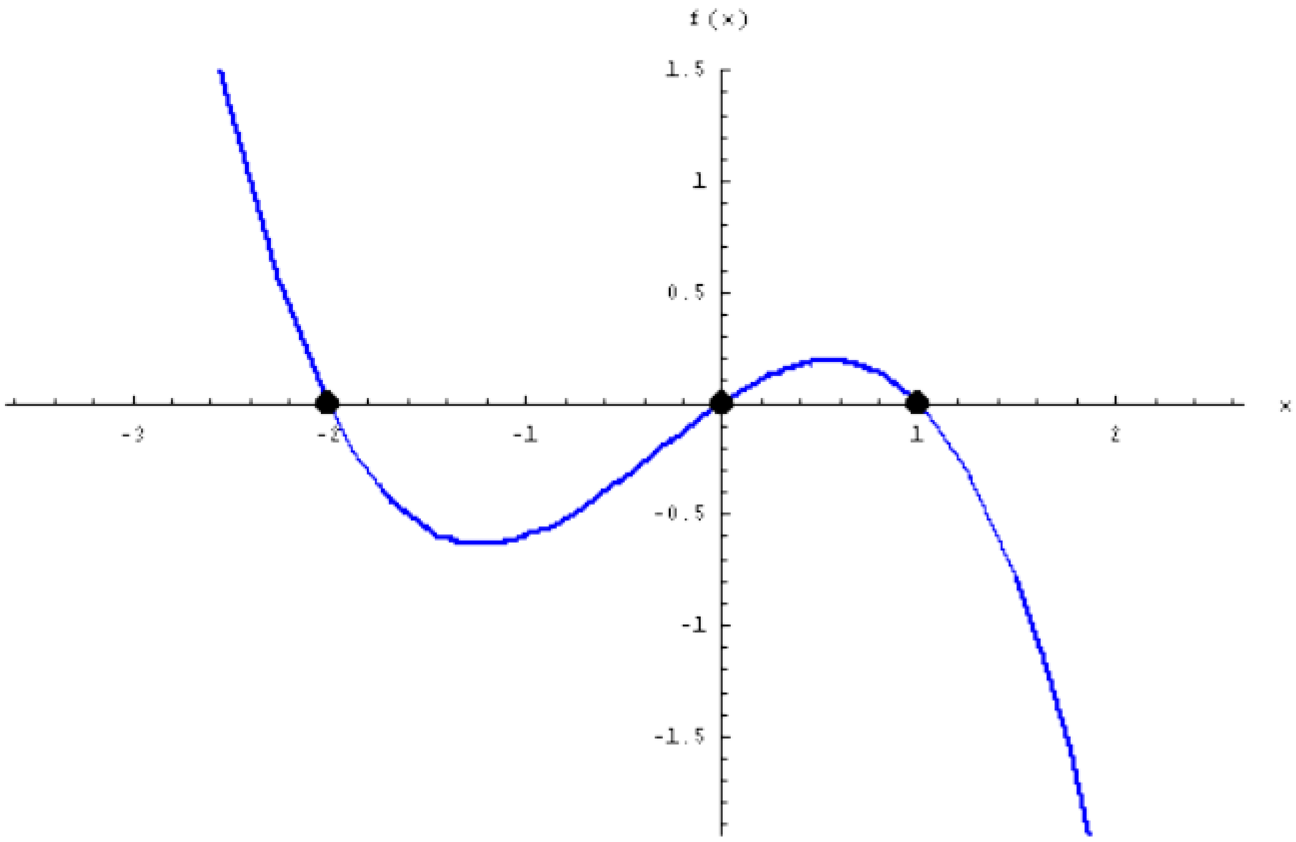
- Often in engineering we have an equation that models our physical system that we must find the roots for.

$$F(x) = 0$$

- To find the value of *x* that makes the equation 0 we can use **bracketed** or **open numerical methods**.

- As with many numerical methods, they require a **good enough starting guess** otherwise we may not be able to find a solution, or if we do, it might be the wrong one.

- It's often good to plot the function first to get an idea of roughly where it crosses the axes then use those as a starting guess.



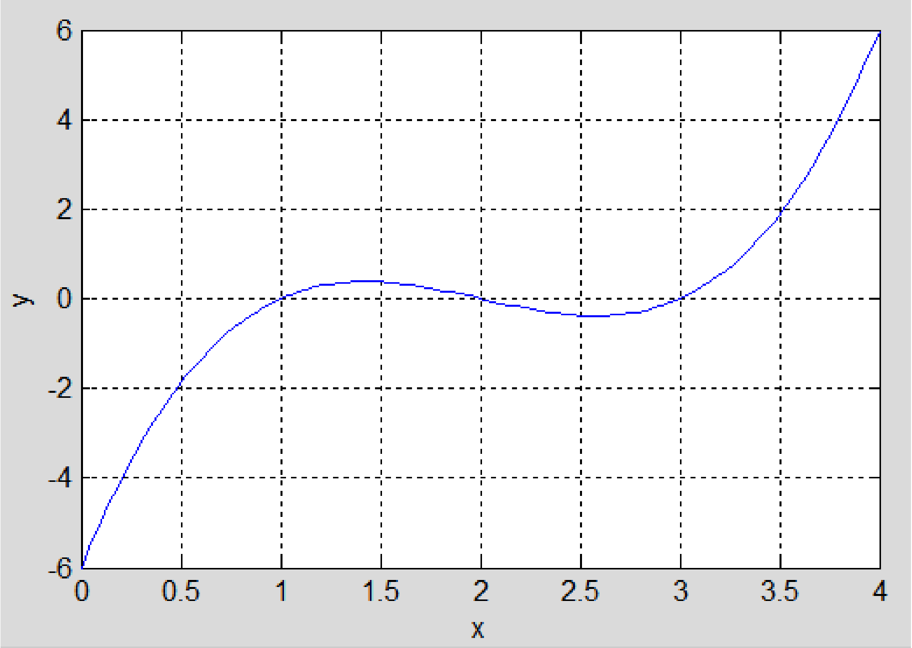
EXAMPLE 6

Find the roots of the following cubic equation.

$$y = x^3 - 6x^2 + 11x - 6$$

```
>> x = linspace(0,4);
>> y = x.^3 - 6*x.^2 + 11*x - 6;
>> plot(x,y)
>> xlabel('x')
>> ylabel('y')
>> grid on
```

Remember element-wise multiplication



- The roots look on the graph to be $x = 1$, $x = 2$, and $x = 3$. Let's **test those values**:

This is how you can check your own solution

```
>> x = [1,2,3];
>> y = x.^3 - 6*x.^2 + 11*x - 6

y =

0        0     0
```

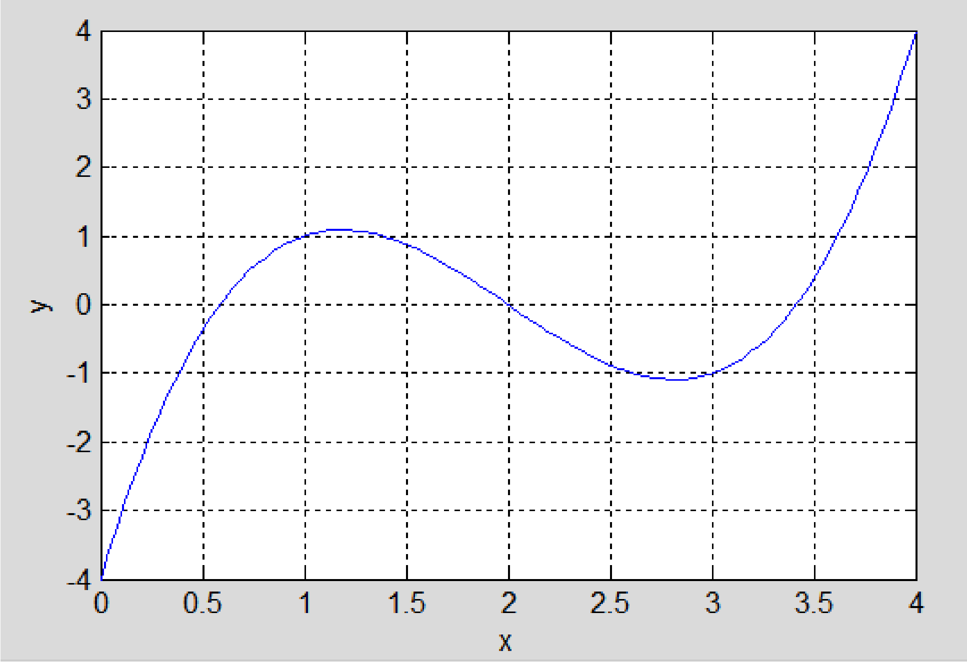
- MATLAB returns 0 for all the values we tried so they are in fact the roots.

EXAMPLE 7

Find the roots of the following cubic equation.

$$y = x^3 - 6x^2 + 10x - 4$$

```
>> x = linspace(0,4);
>> y = x.^3 - 6*x.^2 + 10*x - 4;
>> plot(x,y)
>> xlabel('x')
>> ylabel('y')
>> grid on
```

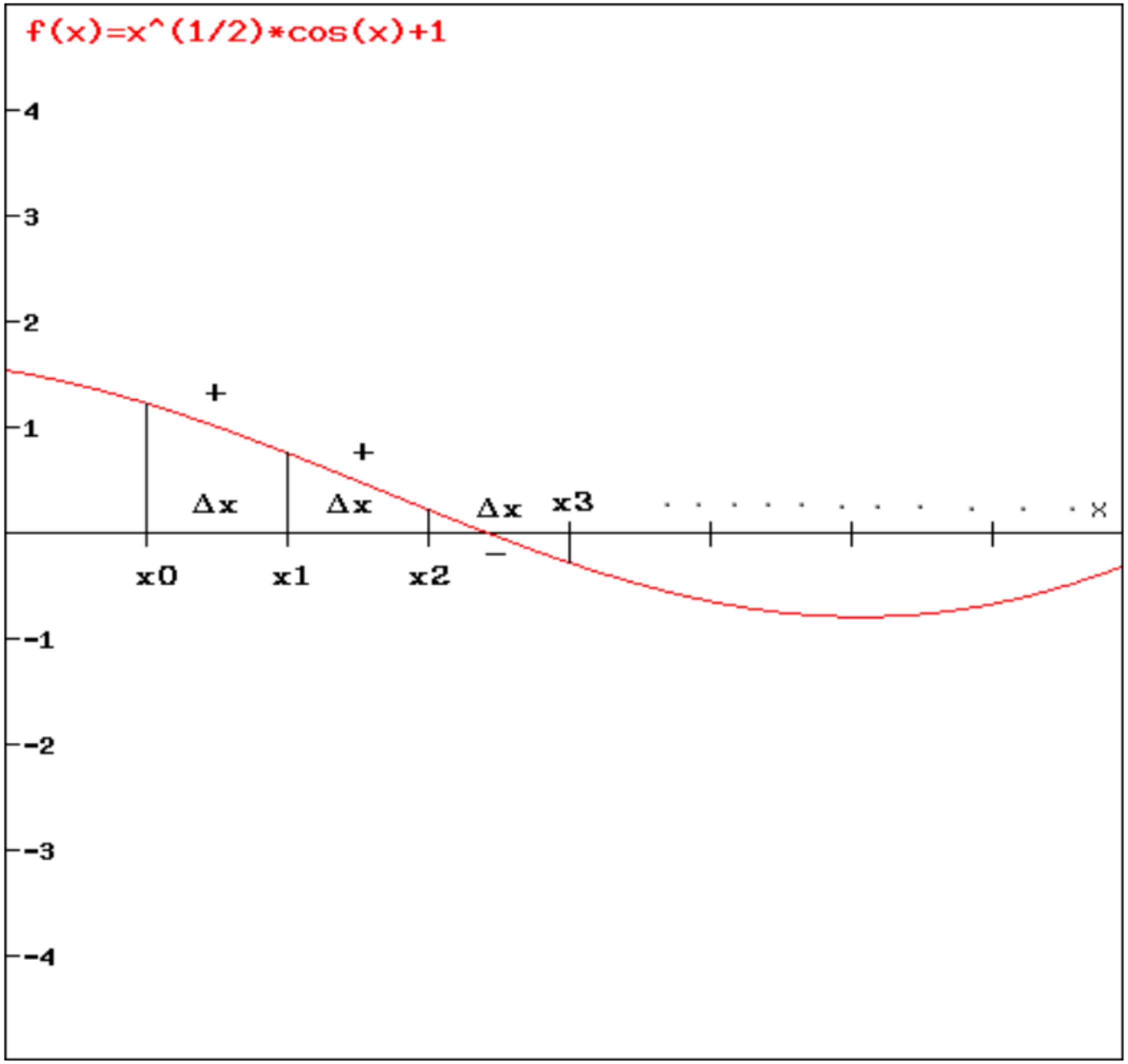


- It is not easy to see exactly where all the roots occur. They seem to be around $x = 0.5$, $x = 2$, and $x = 3.5$. Let’s test those values:

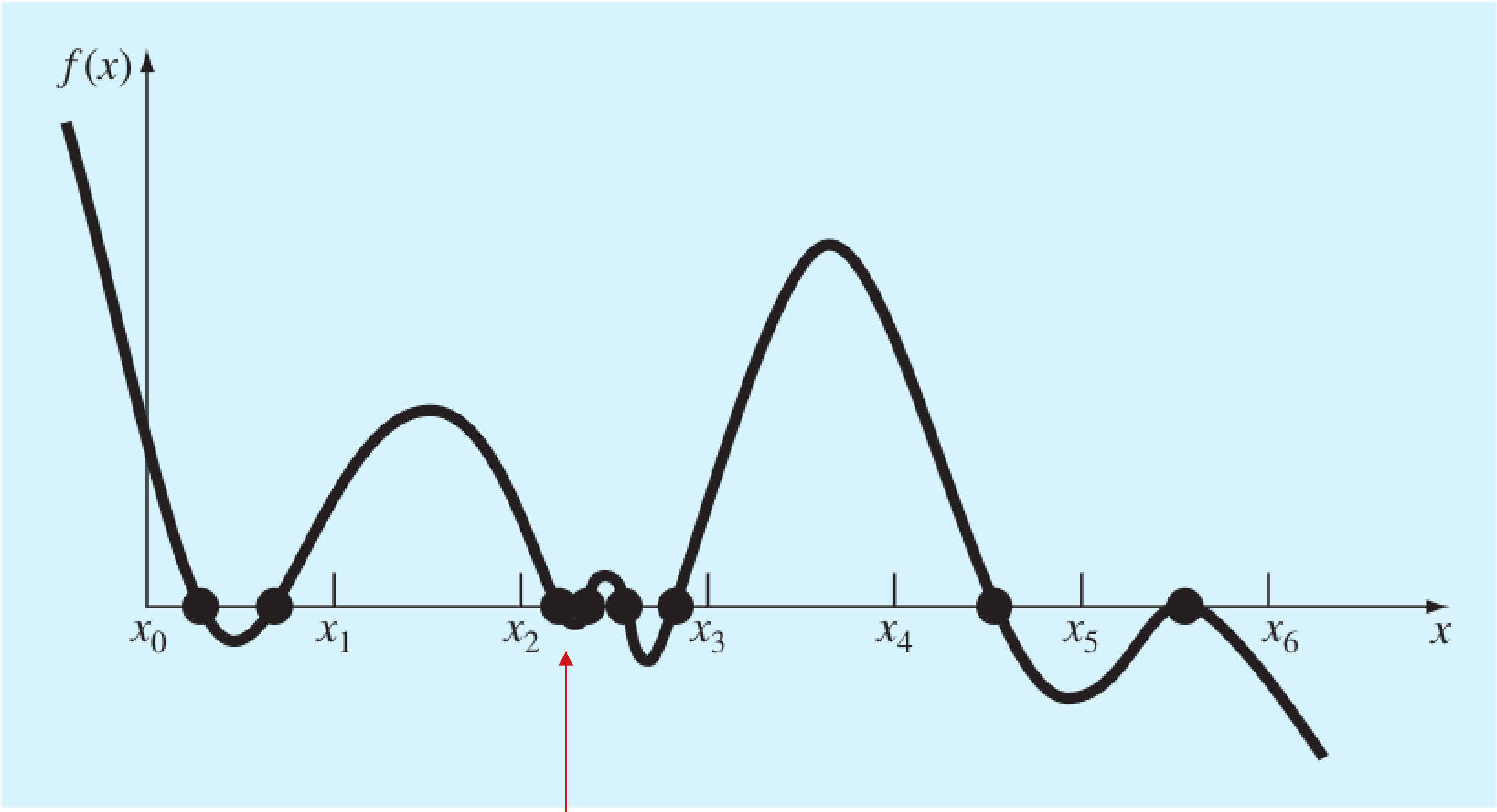
```
>> x = [0.5,2,3.5];
>> y = x.^3 - 6*x.^2 + 11*x - 6

y =

-0.3750         0    0.3750
```

- The first and third roots are not very accurate.
- How can we get closer to the roots?

4.12 Incremental Search Method



- Keep stepping up the *x*-axis until we find the root.
- The root is between where f(x) changes sign.
- Can be slow if step size is small.
- Can miss roots if step size is too big.



Easy to miss roots here

Algorithm

- Choose initial guess, x_0, and step size, h.
- Calculate value of function at initial point, $f(x_0)$.
- Calculate value of function at next point, $f(x_1) = f(x_0+h)$.
- Calculate the value $f(x_0)f(x_1)$. If positive then take another step and calculate $f(x_1)f(x_2)$, etc.
- Repeat until it becomes negative. Somewhere between the last two x points is the root. Choose a value or go back to the last point and take a smaller step size.

- To get accurate values of the roots using this method takes a long time because the step size must be small.
- However we can use it with reasonably sized steps to locate intervals for which the roots are in.
- This is called **locating the brackets** and can be used in conjunction with bracketed methods such as the Bisection Method.
- Once a bracket has been found, a bracketed method will always converge to the root, and usually much faster than incremental search.

EXAMPLE 8

Use incremental search to locate a root of the following function on the interval [0,4].

$$f(x) = 9x^2 + 45x - 154$$

Choose step, h = 0.5 (8 intervals):

- Start at x = 0: f(0) = -154
- Next is x = 0.5: f(0.5) = -129.25
- Next is x = 1: f(1) = -100
- Next is x = 1.5: f(1.5) = -66.25
- Next is x = 2: f(2) = -28
- Next is x = 2.5: f(2.5) = 14.75

f(2.25) = -7.19 so not very accurate

→ The root is somewhere between 2 and 2.5.

For now let's just take the average: $\frac{2+2.5}{2} = \boxed{2.25}$

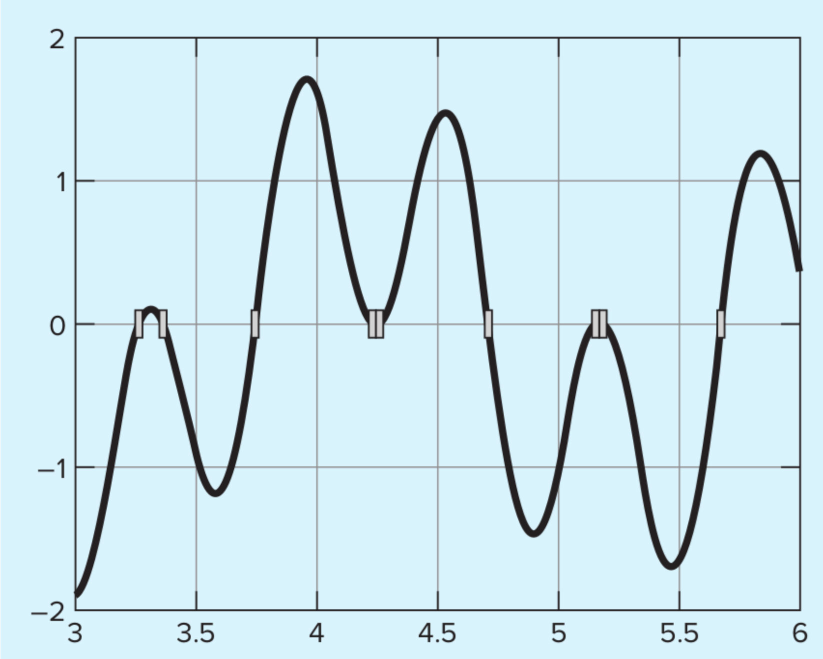
EXAMPLE 9

Using incremental search we can locate brackets for the roots of the following function on the interval [3, 6].

$$f(x) = \sin(10x) + \cos(3x)$$

- Splitting up the interval into 50 pieces and doing this method reveals 5 brackets that roots occur between:

```
number of brackets:
     5

ans =

    3.2449    3.3061
    3.3061    3.3673
    3.7347    3.7959
    4.6531    4.7143
    5.6327    5.6939
```



- The problem is that there are more than 5 roots looking at the graph… So we should divide the interval into more than 50.

- Using 100 we get 9 brackets:

- So make sure to plot the graph and decide if you have found all the brackets or not before moving on.

```
number of brackets:
     9

ans =
    3.2424    3.2727
    3.3636    3.3939
    3.7273    3.7576
    4.2121    4.2424
    4.2424    4.2727
    4.6970    4.7273
    5.1515    5.1818
    5.1818    5.2121
    5.6667    5.6970
```

What is relative true error?

$$\frac{\text{True value} - \text{Approximation}}{\text{True Value}}$$

$$\frac{\text{Current value} - \text{Previous value}}{\text{Current Value}}$$

$$\text{True value} - \text{Approximation}$$

$$\text{Current value} - \text{Previous Value}$$

To 0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

What is absolute iterative error?

$$\frac{\text{True value} - \text{Approximation}}{\text{True Value}}$$

$$\frac{\text{Current value} - \text{Previous value}}{\text{Current Value}}$$

$$\text{True value} - \text{Approximation}$$

$$\text{Current value} - \text{Previous Value}$$

Tc 0

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**