Root Finding

Lecture 4

Dr. Hao Zhu

Beijing-Dublin International College,
Beijing University of Technology

2019 Autumn

Outline

Outline

Introduction

- In many problems we are required to determine when a function is zero.
- It is our intention to discuss methods which are in some sense robust and so the form of the actual function we are studying is not important, at least in the construction of the method.

Introduction

Here we re-introduce the MATLAB function `feval` which is used within other functions requiring a function name to be passed to them.

This function has the syntax `feval(f,x1,...,xn)` and as you might expect from the name evaluates the function `f` using the arguments `x1,...,xn`. This function can be used as

```
feval('sin',0.3)
feval('mycode',0.2,0.3)
```

In the first example this gives `sin(0.3)` and in the second it returns the value of `mycode(0.2,0.3)`.

Outline

Initial Estimates

In order to determine a root it is usually essential to have an initial estimate of its value or a bracketing interval containing the value. Here we will use the graphical capability in MATLAB.

Firstly, we shall set up a small m-file called `userfn.m`, which will used to specify an example function.

```
function [value] = userfn(x,par1,par2)
value = x-par1*sin(x.^par2);
```

This gives $f(x) = x - \alpha_1 \sin x^{\alpha_2}$, where α_1 and α_2 are parameters, which the user will specify.

Initial Estimates

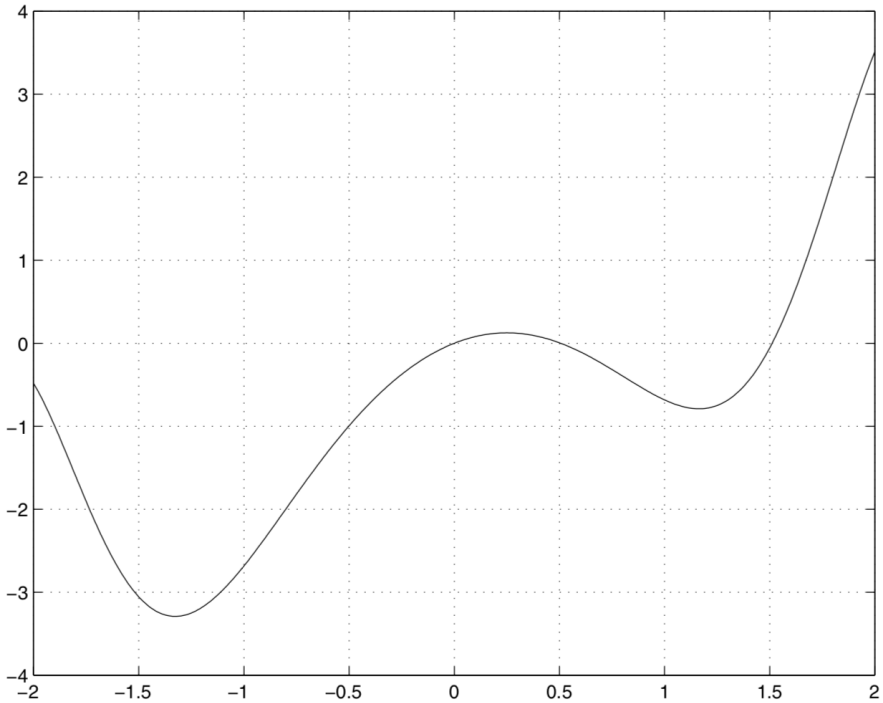
Now in order to plot the function we select a range and use the `plot` command. The commands are

```
x = -2.0:0.01:2.0;
y = feval('userfn',x,2,2);
plot(x,y)
grid on
```

- This first line sets up a vector x running from -2 to 2 in steps of 0.01.
- The next line calculates the value of the function $x - 2\sin x^2$ at these points and puts the answer in y. We could also use the command `y = userfn(x,2,2);`.
- The next two lines plot the graph and add the grid lines.

Initial Estimates

- From this figure, we could find three roots at least. There is also a hint of an extra one to the left of the figure, which could be determined by extending the range leftwards).
- In order to investigate further we use the `zoom on` command, and typing `zoom off` disable this feature.



- However the problem is that if we zoom too close we will be able to see the straight line segments used for the plotting.

Outline

Fixed Point Iteration

Fixed point iteration scheme means a fixed point of an equivalent equation.

The equation $f(x) = 0$ is converted in the form $x = g(x)$, so when x is substituted in the function $g(x)$ it returns the value x, hence the nomenclature fixed point. However, this conversion is not always straightforward and definitely can be done in many ways.

Considering the quadratic $f(x) = x^2 + 2x - 3$, which could be manipulated to give $x = (3 - x^2)/2$, $x = \sqrt{3 - 2x}$ or $x = 3/(x + 2)$ and so on.

Fixed Point Iteration

Let us see how this Iteration is implemented as a numerical scheme. We rewrite the equation as

$$x_{n+1} = g(x_n), \quad n = 0, 1, ...$$

which is a recursion formula starting with an initial guess, namely x_0. The simplest code for this purpose would be

```
x0 = 1;
for j=1:10
    x0 = g(x0);
end
```

which defines the function $g(x)$ and $x_0 = 1$ is a suitable initial guess. This runs through the iterative process ten times (hopefully converged).

Fixed Point Iteration

We need to pause here to think what we mean by convergence.

- We find the value of the function $f(x)$ at the current iterate as a check.
- We require this to be less than a certain tolerance ($\approx$ accuracy).

The code may be successful when the difference between x_{n+1} and x_n is less than a certain tolerance. In this case we have $x_{n+1} \approx x_n \Rightarrow x_n \approx g(x_n) \Rightarrow f(x_n) \approx 0$.

This tolerance reflects how well the answer is and the parameter `maxits` is how many times to perform the iterations. This is to eliminate problems which do not converge and hence cause infinite loops.

The corresponding code we shall use is given below:

Fixed Point Iteration

```
% fixed.m
function [answer,iflag] = fixed(g,xinit)
global tolerance maxits
iflag = 0;
iterations = 0;
xnext = feval(g,xinit);
while (iterations<maxits) & abs(xnext-xinit)>tolerance
        iterations = iterations + 1;
        xinit = xnext;
        xnext = feval(g, xinit);
end
if iterations == maxits
        iflag = -1;
        answer = NaN;
else
        iflag = iterations;
        answer = xnext;
end
```

Fixed Point Iteration

```
% eqn.m
function [g] = eqn(x)
g = 2*sin(x.^2);

% fixed.m (the main function)
global tolerance maxits
tolerance = 1e-4;
maxits = 30;
[root,iflag] = fixed('eqn',0.2);
switch iflag
case -1
        disp('Root finding failed')
otherwise
        disp([' Root = ' num2str(root) ...
            ' found in ' num2str(iflag) ' iterations'])
end
```

Outline

Loops Structures

The basic MATLAB loop command is `for` which repeats an operation for all the elements of a vector. A simple example helps to illustrate this:

```
% looping.m
N = 5;
for ii = 1:N
    disp([int2str(ii) ' squared equals ' int2str(ii^2)])
end
```

This gives the output:

```
1 squared equals 1
2 squared equals 4
3 squared equals 9
4 squared equals 16
5 squared equals 25
```

Loops Structures

```
% looping.m
N = 5;
for ii = 1:N
    disp([int2str(ii) ' squared equals ' int2str(ii^2)])
end
```

- The first line start with % means this is comment (ignored by MATLAB).
- In the third line, the `for` loop will run over the vector `1:N` (default increment is 1), which gives `[1 2 3 4 5]`, setting the variable `ii` to be each of the values in turn.
- The body of the loop is a single line which displays the answer.
- Command `int2str` is to convert the integers to strings so they can be combined with the message `' squared equals '`.
- In the last line, the `end` statement indicates the end of the loop body.

Loops Structures

We pause here to clarify the syntax associated with the `for` command:

```
for ii = 1:N
    commands
end
```

- This repeats the `commands` for each of the values in the vector with `ii` $= 1, 2, ..., N$. If instead we had for `ii` = 1:2:5 then the commands would be repeated with `ii` equal to 1, 3 and 5.
- Command `disp` is to help the reading of the code and is also useful when debugging.
- We have indented the `disp` commands. The spaces are not required by MATLAB. If you use MATLAB built-in editor (try the command `edit`), then this indentation is done automatically.

Example 3.1

The following code writes out the seven times table up to ten seven's.

```
str = ' times seven is ';
for j = 1:10
    x = 7 * j;
    disp([int2str(j) str int2str(x)])
end
```

The first line sets the variable ***str*** *to be the string ' times seven is ' and this phrase (within single quotes) will be used in printing out the answer.*
The second line, the ***for*** *loop, tells us the variable* ***j*** *is to run from 1 to 10, and the commands in the* ***for*** *loop are to be repeated for these values.*
The third line sets the variable ***x*** *to be equal to seven times the current value of* ***j***.
The fourth line constructs a vector consisting of the value of ***j***, *the string* ***str*** *and finally the answer* ***x***.

Example 3.2

The following code prints out the value of the integers from 1 to 20 (inclusive) and their prime factors.

To calculate the prime factors of an integer, here we use the MATLAB intrinsic command ***factor***

```
for i = 1:20
    disp([i factor(i)])
end
```

This loop runs from ***i*** *equals 1 to 20 (in unit steps) and displays the integer and its prime factors. There is no need to use* ***int2str*** *(or* ***num2str****) here since all of the elements of the vector are integers.*

The following example shows how loops can be used not only to repeat instructions but also to operate on the same quantity.

Example 3.3

Suppose we want to calculate the quantity six factorial ($4! = 4 \times 3 \times 2 \times 1$) *using MATLAB. One possible way is*

```
fact = 1;
for i = 2:4
    fact = fact * i;
end
```

To understand this example it is helpful to unwind the loop and see what code has actually executed.

```
fact = 1;
i=2; fact = fact * i; At this point fact is equal to 2
i=3; fact = fact * i; At this point fact is equal to 6
i=4; fact = fact * i; At this point fact is equal to 24
```

The same calculation can be done by the MATLAB command **factorial(4)**.

Example 3.4

Determine the sum of the geometric progression

$$\sum_{n=1}^{6} 2^n$$

This is accomplished using the code:

```
total = 0
for n = 1:6
    total = total + 2^n;
end
```

which gives the answer 126. The formula for the sum of a geometric progression is

$$S = a\frac{1-r^n}{1-r}$$

Outline

Summing Series

Let us describe summing series in more detail by codes to evaluate

$$\sum_{n=1}^{N} i^2$$

We do not necessarily know N so this will need to be entered by the user. Let us consider $N = 4$, so we wish to determine

$$\sum_{n=1}^{4} i^2$$

On paper we would write down all the terms in the summation and then add them up to evaluate the series

$$\sum_{n=1}^{4} i^2 = 1 + 4 + 9 + 16 = 30$$

Summing Series

Recall the series

$$\sum_{n=1}^{4} i^2 = 1 + 4 + 9 + 16 = 30$$

Logically, we may actually do it like this:

- The first term corresponds to $i = 1$, for which $i^2 = 1$.
- The second term, that is $i = 2$, has $i^2 = 4$ and adding to the previous answer gives $1 + 4 = 5$.
- The third term, that is $i = 3$, has $i^2 = 9$ and adding to our previous answer gives $5 + 9 = 14$.
- The fourth term, that is $i = 4$, has $i^2 = 16$ and adding to our previous answer gives $14 + 16 = 30$.

Summing Series

We can automate this process by using the MATLAB code:

```
N = input('Enter the number of terms required:  ');
s = 0;

for i = 1:N
    s = s + i^2;
end

disp(['Sum of the first ' int2str(N) ...
    ' squares is ' int2str(s)])
```

Summing Series

We can break down our summation code as follows:

- The first line asks the user to enter the value of `N` when prompted with the string contained in the `input` statement.
- The second line initialize a variable `s` to be zero for storing the cumulative sum. The blank line is included to make our code more readable.
- The next three lines form a loop. The loop variable `i` running from `1` to `N` means the command in the loop will be repeated for each of these values. The value of `i^2` will be added to the previous value of `s`.
- For the last two lines, we display our results using the `disp` command.

Summing Series

What about if we want all values during 'Summing Series' ($I_N = I_{N-1} + N^2$)?

```
maxN = input('Enter the maximum value of N required:  ');
I(1) = 1^2;

for N = 2:maxN
    I(N) = I(N-1) + N^2;
end

disp(['Values of I_N'])
disp([1:N; I])
```

Summing Series

This code in last slide uses a vector `I` to store the results of the calculation.

This produces the results:

```
Enter the maximum value of N required:  5
Values of I_N
    1      2      3      4      5
    1      5      14    30      55
```

Up until now we have considered very simple summations. Let us now see how we can sum the series where the terms are defined by some function `f(i)`.

Summing Series

Let us consider

$$I_N = \sum_{i=1}^{N} i \sin \frac{i\pi}{4}$$

We use and save a code `f.m` returning the coefficients we are going to sum.

```
function [value] = f(inp)
value = inp * sin(inp*pi/4);
```

- The first line tells the computer this program is a function, which should take as input a value `inp` and return the variable `value`.
- The second line works out the required value. Notice that MATLAB already has a variable `pi`.

Summing Series

The code is now modified to:

```
maxN = input('Enter the maximum value of N required:  ');
I(1) = f(1);

for N = 2:maxN
    I(N) = I(N-1) + f(N);
end

disp(['Values of I_N'])
disp([1:N; I])
```

This produces the result:

```
Enter the maximum value of N required:  4
Values of I_N
    1.0000    2.0000    3.0000
    0.7071    2.7071    4.8284
```

Summing Series

Considering the `for` loop (Lines 3, 4 and 5)

```
for N = 2:maxN
    I(N) = I(N-1) + f(N);
end
```

Here we have a good example of the different meanings of the command `y(n)`. In the case of `I(N)`, this refers to the N^{th} entry of the array `I`, whereas for `f(N)`, it refers to the function `f` evaluated at the point `N`.

The reason is `I` is an array, however `f` is a function.

This subtle difference is critical!

Sums of Series of the Form – $\sum_{j=1}^{N} j^p, p \in \mathbb{N}$

Let us consider the simple code which works out the sum of the first N integers raised to the power p: $S_N = \sum_{j=1}^{N} j^p$

```
% Summing series

N = input('Please enter the number of terms required ');
p = input('Please enter the power ');

sums = 0;
for j = 1:N
    sums = sums + j^p;
end

disp(['Sum of the first ' int2str(N) ...
        ' integers raised to the power ' ...
        int2str(p) ' is ' int2str(sums)])
```

Summing Infinite Series and Example 3.5

We now consider examples where we need to truncate the series.

The Taylor series for a function $f(x)$ about a point $x = a$ is given by

$$f(x) = f(a) + \sum_{n=0}^{\infty} \frac{(x-a)^n}{n!} f^{(n)}(a)$$

We can use this to approximate $\sin x$ *by an infinite series in* x *as*

$$\sin x = \lim_{N\to\infty} \sum_{n=0}^{N} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

In using computer we cannot take N to be infinity, but we shall take it to be large in the hope that the error will be small.

Example 3.5(Cont'd)

We can write the above series (taking $N = 10$) and evaluate the series at the points $0, 0.1, 0.2, 0.3, 0.4$ and 0.5. The code for approximating $\sin x$ is given below:

$$\sin x = \lim_{N\to\infty} \sum_{n=0}^{N} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

```
v = 0.0:0.1:0.5;
sinx = zeros(size(v));
N = 10; range = 0:N;
ints = 2*range + 1;

for n = range
    sinx = sinx + (-1)^n*v.^ints(n+1)...
        /(factorial(ints(n+1)));
end
```

Example 3.5(Cont'd)

Let us analyse the code line by line.

`v = 0.0:0.1:0.5;` *sets up the vector size(v) running from zero to a half in steps of a tenth.*

`sinx = zeros(size(v));` *The command size(v) gives the size of the vector v and then the command zero sets up sinx as an array of zeros of the same size.*

`N = 10; range = 0:N;` *sets a variable N to be equal to 10 and then sets up a vector range which runs from zero to N.*

Example 3.5(Cont'd)

`ints = 2*range + 1;` *This gives the mathematical expression* $2n + 1$ *for all the elements of the vector range and puts them in ints.*

`for n = range ..... end` *This loop structure repeats its arguments for* n *equal to all the elements of range.*

`sinx = sinx+(-1)^n*v.^ints(n+1)/(factorial(ints(n+1)));` *This mathematical expression evaluates the subject of the summation as a function of n. Also note the use of* `.^` *when operating on v as this is a vector.*

Example 3.5(Cont'd)

Running this gives:

```
>> sinx
sinx =
    0    0.0998    0.1987    0.2955    0.3894    0.4794

>> sin(v)
ans =
    0    0.0998    0.1987    0.2955    0.3894    0.4794

>> sinx-sin(v)
ans =
  1.0e-16 *
    0    0.1388    0.2776         0   -0.5551         0
```

This results shows that the approximation works very well.

Example 3.6

Calculate the sum

$$\sum_{n=0}^{\infty} e^{-n}$$

For this problem, we can obviously not add up an infinite number of terms, we need to truncate the calculation. We can use the code:

```
N = 10;
total = 0;
for n = 0:N
    total = total + exp(-n);
end
```

Normally, to test for convergence of our results we should compare our answer for different values of N.

Summing Series Using MATLAB Specific Commands

So far we have used commands which are common to many programming languages (like C language) and have not discovered the power of MATLAB.

Consider the previous example

$$\sum_{i=1}^{10} i^2$$

The full code for this example is

```
i = 1:10;
i_squared = i.^2;
value = sum(i_squared);
```

This can all be contracted on to one line `sum((1:10).^2)`.

Summing Series Using MATLAB Specific Commands

Consider the problem which is to work out the sum of the series, where $f(i) = 1$ when i is odd and $f(i) = 2$ when i is even.

$$\sum_{i=1}^{N} y_i f(i)$$

We can set up these values of f using the commands:

```
N = 11;
iodd = 1:2:N;
ieven = 2:2:(N-1);
f(iodd) = 1;
f(ieven) = 2;
```

This can be achieved in one command, namely `f = 2-mod(1:11,2)`.

Example 3.7

Evaluate the expression for $N = 10$

$$\prod_{n=1}^{N} \left(1 + \frac{2}{n}\right)$$

The symbol $\prod$ *means the product of terms, and the corresponding code is:*

```
n = 1:10;
f = 1+(2)./n;
pr = prod(f)
```

This gives the answer 66. If you want to see what the code does, just leave off the semicolons (;) this will show you the vectors which are generated.

Loops Within Loops (Nested)

Many algorithms require us to use nested loops (loops within loops), as in constructing an array of numbers:

```
for ii = 1:3
    for jj = 1:3
        a(ii,jj) = ii + jj;
    end
end
```

Notice that the inner loop (that is, the one in terms of the variable `jj`) is executed three times with `ii` equal to 1, 2 and then 3. These structures can be extended to have further levels.

Notice each `for` command must be paired within an end.

Example 3.8

Calculate the summations for p equal to 1, 2 and 3 for $N = 6$.

$$\sum_{j=1}^{N} j^p$$

We could perform each of these calculations separately but since they are so similar it is better to perform them within a loop structure:

```
N = 6;
for p = 1:3
    sums(p) = 0.0;
    for j = 1:N
        sums(p) = sums(p)+j^p;
    end
end
disp(sums)
```

Outline

Conditional Statements

We start from the `if` command which takes the form:

```
if (expression)
        commands
        ...
end
```

If the `expression` is true then the commands are executed, otherwise the programme continues with the next command immediately beyond the `end` statement.

Conditional Statements

Let us first discuss the construction of the `expression` from the simple mathematical comparisons.

- `a < b` True if a is less than b
- `a <= b` True if a is less than or equal to b
- `a > b` True if a is greater than or equal to b
- `a >= b` True if a is greater than or equal to b
- `a == b` True if a is equal to b
- `a ~= b` True if a is not equal to b

Conditional Statements

Sometimes we will need to form compound statements, comprising more than one condition. This is done by using logical expressions, these are:

- `and(a,b)` a `&` b Logical AND
- `or(a,b)` a `|` b Logical OR
- `not(a)` `~a` NOT
- `xor(a,b)` Logical exclusive OR

Conditional Statements

The effect of these commands is perhaps best illustrated by using tables

AND	false	true
false	false	false
true	false	true

OR	false	true
false	false	true
true	true	true

XOR	false	true
false	false	true
true	true	false

In many languages you can define Boolean value (`True` or `False`). In MATLAB `0` represents False and any other value (normally `1`) implies True.

Conditional Statements and Example 3.9

Let us consider a command which is only executed if a value x lies between 1 and 2 or it is greater than or equal to 4. We shall try to describe the thought processes involved:

```
((x>1) & (x<2)) | (x>=4)
```

or

```
a = and(x>1,x<2);
b = (x>=4);
c = or(a,b)
```

Notice here we have actually set 'Boolean' variables `a`, `b` *and* `c` *(in fact they are only normal variables which take the values 0 or 1).*

Conditional Statements

For more complex conditional statements, we would use commands `else` and `else if`. The general form of these is given by:

```
if (expression)
        commands ...
elseif (expression)
        commands ...
else
        commands ...
end
```

Each `if` statement must be paired with an `end` statement, but we can use as many `elseif` statements as we like, but only one `else`. Indentation is particularly important to nest `if` statements.

Example 3.10

Consider the following piece of code which determines which numbers between 2 and 9 go into a specified integer exactly:

```
str = 'Divisible by ';
x = input('Number to test:  ');

for j = 2:9
    if rem(x,j) == 0
        disp([str int2str(j)])
    end
end
```

Here we have used the command `rem` *to obtain the remainder.*

Example 3.11

Here we construct a conditional statement which evaluates the function:

$$f(x) = \begin{cases} 0 & x<0 \\ x & 0 \leq x \leq 1 \\ 2-x & 1 < x \leq 2 \\ 0 & x>2 \end{cases}$$

One of the possible solutions to this problem is:

```
if x >= 0 & x <= 1
    f = x;
elseif x > 1 & x <= 2
    f = 2-x;
else
    f = 0;
end
```

Example 3.12 (Nested if statements)

The ideas of nested if statements is made clear by the following example:

```
if raining
    if money_available > 20
        party
    elseif money_available > 10
        cinema
    else
        comedy_night_on_telly
    end
else
    if temperature > 70 & money_available > 40
        beach_bbq
    elseif temperature > 70
        beach
    else
        you_must_be_in_the_UK
    end
end
```

The MATLAB Command – switch

The command switch is for selecting, that takes the form:

```
switch switch_expr
    case case_expr1
        commands ...
    case case_expr2,case_expr3
        commands ...
    otherwise
        commands ...
    end
```

We shall work through the example for this command:

```
switch lower(METHOD)
    case 'linear'
        disp('Method is linear')
    case 'cubic'
        disp('Method is cubic')
    otherwise
        disp('Unknown method.')
    end
```

Outline

Conditional Loops

Suppose we now want to repeat a loop until a certain condition is satisfied. This is achieved by making use of the MATLAB command `while`, which has the syntax

```
while (condition)
    commands...
end
```

This translates into English as: while `condition` holds continue executing the `commands...`.

Example 3.13

Write out the values of x^2 for all positive integer values x such that $x^3 < 2000$. To do this we will use the code

```
x = 1;
while x^3 < 2000
    disp(x^2)
    x = x+1;
end
```

This first sets the variable x to be the smallest positive integer (that is 1) and then checks whether $x^3 < 2000$ (which it is). If the condition is satisfied, it executes the command within the loop.

Example 3.14

Consider the one-dimensional map:

$$x_{n+1} = \frac{x_n}{2} + \frac{3}{2x_n}$$

subject to the initial condition $x_n = 1$. *Let us determine what happens as* n *increases. We note that the fixed points of this map, that is the points where* $x_{n+1} = x_n$, *are given by the solutions of the equation:*

$$x_n = \frac{x_n}{2} + \frac{3}{2x_n}$$

which are $x_n = \pm 3$. *We can use the code with a certain tolerance*

```
xold = 2; xnew = 1;
while abs(xnew-xold) > 1e-5
    xold = xnew;
    xnew = xnew/2+3/(2*xnew);
end
```

The break Command

A command which is important within the context of loops and conditional statements is the break command. This allows loops to stop when certain conditions are met. For instance, consider the loop structure

```
x = 1;
while 1 == 1
    x = x+1;
    if x > 10
        break
    end
end
```

The loop structure while 1==1 ... end is very dangerous since this can give rise to an infinite loop, however the break command allows control to jump out of the entire loop.

Outline

MATLAB Specific Commands, Example 3.15

MATLAB intrinsic function `find` is frequently used when programming. It returns an array of locations at which a certain condition is satisfied.

Find all integers between 1 and 20 for which their sine is negative. Firstly we set up the array of integers, then calculate their sines and subsequently test whether these values are negative.

```
ii = 1:20;
f = sin(ii);
il = find(f<0);
disp(ii(il))
```

In the third line, we use `find` to determine the locations in the vector `f` where the sine is negative. The command returns a list of positions in the vector `f` (and equivalently `ii`) where the condition is true.

MATLAB Specific Commands

Other similar commands are any and all. Examples of their use are:

```
x = 0.0:0.1:1.0; v = sin(x);
if any(v<0)
    disp('Found a negative value')
else
    disp('All values zero or positive')
end
if all(v>0)
    disp('All values positive')
else
    disp('One value is zero or negative')
end
```

There are many other commands which test the properties of a variable, for instance `isempty`, `isreal` etc.

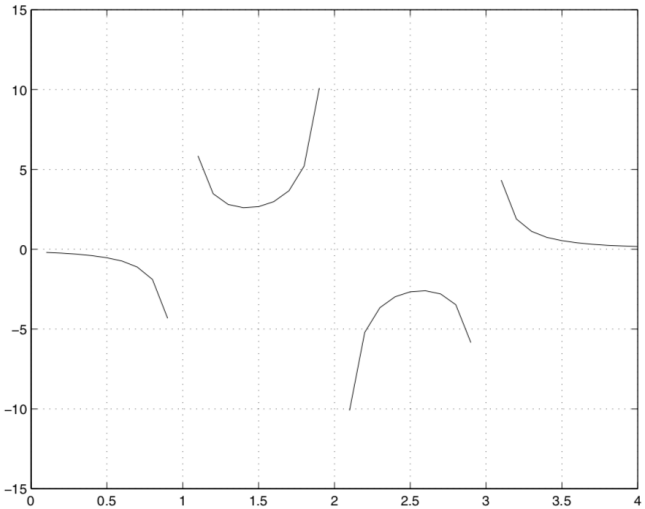
Example 3.16

Now we evaluate the expression

$$f(x) = \frac{1}{(x-1)(x-2)(x-3)}$$

on a grid of points $n/10$ *for* $n = 1$ *to* $n = 40$ *for which the expression is finite, else the function is to be returned as 'NaN'. This is accomplished using the code:*

```
x = (1:40)/10;
g = (x-1).*(x-2).*(x-3);
izero = find(g==0);
ii = find(g~=0);
f(izero) = NaN;
f(ii) = 1./g(ii);
```



This is to plot the function – plot(x,f) whilst missing out the infinite parts.

Outline

Error Checking

It is impossible to assume the data made available to a code is suitable. That means no matter how many different scenarios a programmer comes up with, they can never predict everything a user will do.

Fortunately, MATLAB gives us two very useful commands in this context: `warning` (warn the user of a problem) and `error` (stop the code due to an irretrievable problem).

Both the commands `warning` and `error` are used with an argument, which is displayed when the command is encountered. The typical structure is:

```
if code_fails
    error(' Irretrievable error ')
elseif code_problem
    warning(' Results may be suspect ')
end
```

Example 3.17

Let us now write a code which asks the user for an integer and returns the prime factors of that integer.

```
msg = 'Please enter a strictly positive integer:  ';
msg0 = 'You entered zero';
msg1 = 'You entered a negative integer';
x = input(msg);
if x==0
    error(msg0)
end
if sign(x)==-1
    warning(msg2)
    x = -x;
end
disp(factor(x))
```

When MATLAB encounters these commands it produces a message which indicates on which line they occurred and in which code. This is very helpful when debugging codes.

Example 3.18

Another very useful command in this context is `exist`. This can be used to check whether the requisite variables exist.

Let us consider this header for a code:

```
msg = ['The variable z does not exist ' ...
        'and will be required for this code'];
if ~(exist('z'))
    error(msg)
end
```

We note this code will not work at the prompt, since the command `error` will only run during the execution of an m-file.

Outline

Tasks

Task 3.1 *Calculate the value of the summation*

$$\sum_{i=1}^{100} \frac{1}{i^2}$$

(Notice the answer will probably not be an integer, so you will have to modify the display line to use `num2str`*).*

Tasks

Task 3.2 *Modify the code from Task 3.1 to only sum the values corresponding to odd values of i.*
(Hint: This only involves a very minor change to the `for` *loop. You should check the answer you get is smaller than the one from the previous task.)*

Tasks

Task 3.3 *Calculate the values of:*

$$I_N = \sum_{i=1}^{N} \frac{\sin \frac{i\pi}{2}}{i^2 + 1}$$

for N up to N=20.

Tasks

Task 3.4 *Construct a program to display the values of the function* $f(x) = x^2 + 1$ *for* $x = 0$ *to* $x = \pi$ *in steps of* $\pi/4$. *The key here is to set up an array of points from a to b in steps of h (you can either use the colon command or* `linspace`*). Make sure you remember to use the dot operator.*

Tasks

Task 3.5 *Change the code given in Example 3.5, used to evaluate* $\sin x$*, to one that calculates* $\cos x$ *whose series expansion is given by*

$$\cos x = \lim_{N\to\infty} \sum_{n=0}^{N} (-1)^n \frac{x^{2n}}{(2n)!}$$

Compare the approximations to the values calculated directly from MATLAB for $x = 0, 1/4, 1/2$ *and* $3/4$*. As for Example 3.5 you will need to truncate the summation, choosing* N *to be suitably large.*

Tasks

Task 3.6 *Calculate the sum of the series S_N , where*

$$S_N = \sum_{n=1}^{N} \frac{1}{n^2}$$

for different values of N. Given that as $N \to \infty$, $S_N \to \pi^2/c$ where c is a constant. Determine the value of c. (Here you should modify one of the codes for summation: the best one is probably the solution to Task 3.1.)

Tasks

Task 3.7 *Calculate the summations*

$$\sum_{j=1}^{p+1} j^p$$

for p equal to 1, 2, 3 and 4 (using a nested loop structure).

Tasks

Task 3.8 *Show that, to within the accuracy permitted by MATLAB,*

$$\lim_{N\to\infty} \sum_{n=1}^{N} \frac{(-1)^n}{n} = -\ln 2$$

and

$$\lim_{N\to\infty} \sum_{n=1}^{N} \frac{1}{n(n+1)} = 2$$

Tasks

Task 3.9 *Write down (on paper) a logic expression which is true for values of x such that $2 < x < 4$. Make sure your expression works for $x = 1, 3$ and 5. Repeat the exercise for the union of the sets $x > 3$ and $x < -1$. (You need to write the answers in the form of the combination of x is greater than something and/or x is less than something else).*

Tasks

Task 3.10 *Write down a logical expression which is true for even values of n. Extend this to only be true for even values of n greater than 20. (Hint: Try using the MATLAB command* `mod`*: for information on the command type* `help mod`*.*

Tasks

```
x = 1;
if tan(73*pi*x/4) >= 0
    x = 2;
else
    x = pi;
end
if floor(x) == x
    x = 10;
else
    x = 7;
end
if isprime(x)
    x = 'True';
else
x = 'False';
end
```

Task 3.11 *Try to work out what value of x the following code returns (initially without running it).*

Is this result true whichever value of x we start with? (You can find out about the command `isprime` *by typing* `help isprime`*.)*

Tasks

Task 3.12(D) *Write a loop structure which iterates* $x_{n+1} = 5x_n|1$ *for* $x_n = 1/7$ *until it returns to the same value (use* `while`*).*

Tasks

Task 3.13(D) *Determine all integers between 1 and 50 for which* $n^3 - n^2 + 40$ *is greater than 1000 and n is not divisible by 3. Are any integers between 1 and 50 perfect (that is, are they equal to the sum of their factors)?*

Tasks

Task 3.14 *Write a code which only allows the user to input an integer n between 1 and 10 (inclusive) and then prints out a string of the first n letters of the alphabet. (Hint: You could start with a string of the form "abcdefghij".)*

Tasks

Task 3.15(D) *Write a code which allows the user to input a two character string, the first being a letter and the second being a digit. (Note that to check if a character is a letter we can use inequalities on strings.)*

Tasks

Task 3.16(D) *By modifying the code in Example 3.11 use the* `find` *command to plot the function*

$$f(x) = \begin{cases} 0 & x < -1 \\ x^2 & -1 \leq x \leq 1 \\ 1 & 1 < x < 4 \\ 0 & x \geq 4 \end{cases}$$

for the range $x \in [-3, 5]$ *using one hundred points (try using the command* `linspace` *to set up the array of points).*

Tasks

Task 3.17(D) *Using the command* `find`*, and modifying the code in Example 3.16, plot the function*

$$f(x) = \frac{1}{\cos \pi x}$$

for $x \in [-3, 3]$*. You will need to change the code to determine when the denominator becomes small, rather than when it is identically zero.*

Tasks

Task 3.18(D) *The following code is supposed to evaluate the function*

$$f(x) = \begin{cases} 0 & x<0 \\ x & 0 \leq x \leq 1 \\ 2-x & 1 < x \leq 2 \\ 0 & x>2 \end{cases}$$

Correct the code so that it accomplishes this. This can be checked by using the command `plot(x,f)` *and comparing the figure to what you expect the function to look like.*

Tasks

Task 3.18(D)(Cont'd) *Try to correct the code:*

```
x=lnspace(-4,4);
N = length x

for j = 1;N

if x(j)>=0 and x(i)<=1
    f(j) = x(j);
elseif x(j)>1 or x(i)<2
    f(j) = 2 - x
else
    f(j) = zero;
end
```