Writing Scripts and Functions

Lecture 2

Dr. Hao Zhu

Beijing-Dublin International College,
Beijing University of Technology

2019 Autumn

Outline

Outline

Scripts and Editor

- There are many intrinsic functions in MATLAB that can be typed at the prompt >> directly. However, it is necessary to construct scripts for developing longer codes or running codes many times.
- A *script* is simply a file or a computer program written in the language of MATLAB.
- To invoke the built-in MATLAB editor we type `edit` at the prompt >>. This editor has many advantages:
  - understanding MATLAB syntax
  - producing automatic formatting like indenting
  - colour coding the MATLAB commands and variables
  - setting breaking points when debugging
  - ......

Example 2.1

We begin by entering and running the code:

```
a = input('First number ');
b = input('Second number ');
disp([' Their sum is ' num2str(a+b)])
disp([' Their product is ' num2str(a*b)])
```

This example introduces three new commands, `input,` `num2str` and `disp`.

- Command `input` prompts user within the quotes ' ' and takes user's input; In the first line it stores our response in the variable a.
- Command `num2str` stands for *number-to-string* and instructs MATLAB to convert the argument from a number to a character string.
- Command `disp` is for displaying results.

Example 2.1(Cont'd)

This code above can be entered at the prompt, and we shall also create our first script and save it in a file named ***twonums.m****.*

To do this, first we type `edit` *at the MATLAB prompt to invoke an editor. Since this is our first use, MATLAB will give this code the default name* ***Untitled.m****.*

To proceed we type the above code into the editor and then use the File Menu (sub item ***Save As****) to change the name of the code and* ***Save*** *it as* ***twonums.m****.*

If we now return to the MATLAB window and enter the command ***twonums*** *at the prompt, our code will be executed; we will be asked to enter two numbers and MATLAB will calculate and return their product and sum. The contents of the file can be displayed by typing* ***type twonums****.*

File Names

Important Point: It is very important you give your files a meaningful name and that the files end with `.m`. You should avoid using filenames which are the same as the variables you are using and which coincide with MATLAB commands. Make sure you do not use a dot in the body of the filename and that it does not start with a special character or a number.

Filenames also have the same restrictions which we met earlier for variable names. For instance `myfile.1.m` and `2power.m` are not viable filenames (good alternatives would be `myfile_1.m` and `twopower.m` respectively).

File Paths

When processing a command, MATLAB searches to see if there is a user-defined function of that name by looking at all the files in its search path with a `.m` extension.

Considering creating a directory for your MATLAB work on a Unix / Linux machine the sequence of commands are (at the prompt):

```
mkdir Matlab_Files
cd Matlab_Files
matlab
```

For Windows OS and Macintosh OS, you can click the MATLAB icon and create a new directory.

Command `which file1`. This will tell you the full pathname of the listed files like by typing `pwd`.

Example 2.2

*If we create a MATLAB file called **power.m** using the editor it can be saved in the current directory: however the code will not work. The reason for this can be seen by typing **which power** which produces the output*

```
>> which power
power is a built-in function.
```

So MATLAB will try to run the built-in function.

Functions

We now discuss the important class of codes which actually act as functions. These codes take inputs and return outputs, and save as `xsq.m`.

```
function [output] = xsq(input)
output = input.^2;
```

- The first line of `xsq.m` shows this is a function called `xsq` which takes input called `input`() and returns value called `output`[]. It is encouraged that the function name `xsq` corresponds to the file name `xsq.m`.
- The second line of the function performs the calculation. The function uses dot arithmetic `.^` so that it works with both vector and matrix inputs. A semicolon(`;`) is used to suppress the outputs. In general all communication between function and the main calling program should be done through the input and output.

Functions

Having written our function we can now access it from within MATLAB. Consider the following:

```
function [output] = xsq(input)
output = input.^2;

>> A = [1 2 3 4 5 6];
>> y = xsq(A)

y =
    1    4    9   16   25   36
```

Functions

- It is not possible to call the function just using `xsq` since the code cannot possibly know what the input is. MATLAB will return an error message stating that the `Input argument 'input' is undefined`.
- When the function is run it merely looks at what is given as the argument. It is therefore important the function has the correct input.
- The variables `input` and `output` are local variables that are used by the function; they are not accessible to the general MATLAB workspace.
- Functions can also take multiple inputs and give multiple outputs.

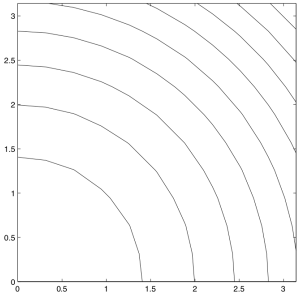
Example 2.3

Suppose we want to plot contours of a function of two variables $z = x^2 + y^2$. We can use the code

```
function [output] = func(x,y)
output = x.^2 + y.^2;
```

To plot the contours (that is the level curves) of the function we would proceed as follows:

```
x = 0.0:pi/10:pi;
y = x;
[X,Y] = meshgrid(x,y);
f = func(X,Y);
contour(X,Y,f)
axis([0 pi 0 pi])
axis equal
```



Example 2.4

Suppose we now want to construct the squares and cubes of the elements of a vector. We can use the code

```
function [sq,cub] = xpowers(input)
sq = input.^2;
cub = input.^3;
```

The first line defines the function ***xpowers****, which has two outputs* ***sq*** *and* ***cub*** *and one input. This function file must be saved as* ***xpowers.m*** *and it can be called as follows:*

```
x = 1:4;
[xsq,xcub] = xpowers(x);
```

Example 2.4(Cont'd)

```
x = 1:4;
[xsq,xcub] = xpowers(x);
```

This gives

```
>> xsq
xsq =
    1    4    9    16

>> xcub
xcub =
    1    8    27    64
```

Example 2.5

As you might expect a function can have multiple inputs and outputs:

```
function [out1,out2] = multi(in1,in2,in3)
out1 = in1 + max(in2,in3);
out2 = (in1 + in2 + in3)/3;
```

*which should be saved as **multi.m**. This function takes three inputs **in1**, **in2** and **in3** and returns two outputs **out1** and **out2**. We can call this function in the following way*

```
x1 = 2; x2 = 3; x3 = 5;
[y1,y2] = multi(x1,x2,x3);
y1, y2
```

For this example we obtain $\boldsymbol{y1 = 7}$ *and* $\boldsymbol{y2 = 3.3333}$.

Example 2.6

Consider a code returning a scalar result from a vector input. For example

```
function [output] = sumsq(x)
output = sum(x.^2);
```

*The function **sumsq** takes a vector as an input and returns the sum of the squares of the elements of the vector. The MATLAB intrinsic function sum calculates the sum of its vector argument. For instance*

```
x = [1 2 4 5 6];
y = sumsq(x)
```

*sets **y** equal to the scalar* $1^2 + 2^2 + 4^2 + 5^2 + 6^2 = 82$.

Outline

Plotting Simple Functions

One of the most powerful elements of MATLAB is its excellent plotting facilities which allow us to easily and rapidly visualise the complex and clear results of calculations.

We start with the simplest command plot and initialise an array `x`.

```
x = 0:0.1:3;
```

Then we calculate the point on a straight line $3x - 1$ and parabola $z = x^2 + 3$ using

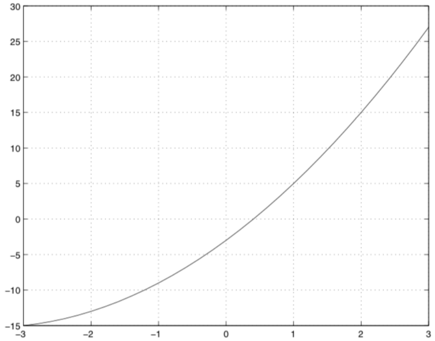
```
y = 3*x-1;
z = x.^2+3;
```

We can plot y versus x using the command plot(x,y) and plot(x,z) to produce a straight line and a parabola (in the default colour blue).

Example 2.7

To plot the quadratic $x^2 + 7x - 3$ *from* x *equals 3 to 3 in steps of 0.2 we use the code*

```
x = -3:0.2:3;
y = x.^2+7*x-3;
grid on
plot(x,y)
```

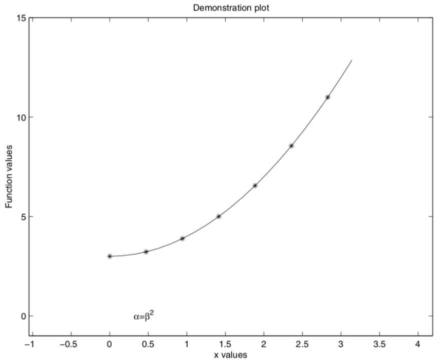


We have given the plot a grid by using the command grid on; this can be removed using the command grid off.

Example 2.8

Consider the following code:

```
x = 0:pi/20:pi;
n = length(x);
r = 1:n/7:n;
y = x.^2+3;
plot(x,y,'b',x(r),y(r),'r*')
axis([-pi/3 pi+pi/3 -1 15])
xlabel('x values')
ylabel('Function values')
title('Demonstration plot')
text(pi/10,0,'\alpha=\beta^2')
```



More on Plotting

- The `plot(x,y)` command 'simply' draws a line through these points.
- Considering the third argument of the `plot` command, such as in `plot(x,y,'r.')`. The first character `r` is the colour (red) and the second is the symbol (in this case a dot `.`) to be used at each point (as opposed to a line joining the points).
- The are many flag options in colours, symbols and line styles.

More on Plotting

- Colour Options: y (yellow); c (cyan); g (green); w (white); m (magenta); r (red); b (blue); k (black).
- Symbol Options: . (point); o (cycle); x (x-mark); + (plus); * (star); s (square); d (diamond); v (triangle down); X (triangle up); < (triangle left); > (triangle right); p (pentagram); h (hexagram).
- Line Styles:
  - (solid)
  : (dotted)
  -. (dashdot)
  - - (dashed)

Outline

Functions of Functions

Command `feval` provides us with considerable freedom in writing code is the command, which translates as function evaluation. The simplest use for this command is

```
y= feval('sin',x);
```

evaluate the function `sin` at `x`. This is equivalent to `sin(x)`.

The utility of `feval` allows us to use function names as arguments. The function must be either a MATLAB built-in function or a user defined function.

Functions of Functions

Consider the function

$$h(x) = 2\sin^2 x + 3\sin x - 1$$

One way of writing code that would evaluate this function is

```
function [h] = fnc(x)
h = 2*sin(x).*sin(x)+3*sin(x)-1
```

However we can recognise this function as a composition of two functions $g(x) = \sin(x)$ and $f(x) = 2x^2 + 3x - 1$ so that $h(x) = f(g(x))$. This is called function of function.

We can do this with the following modification to our code:

Functions of Functions

Previous code:

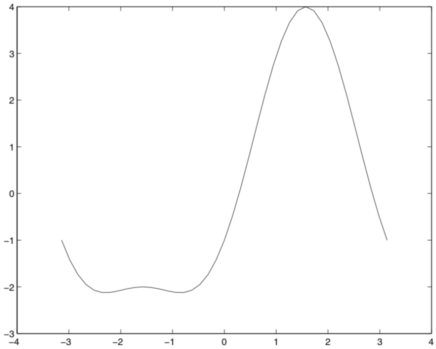
```
function [h] = fnc(x)
h = 2*sin(x).*sin(x)+3*sin(x)-1
```

Modified code:

```
function [y] = f(fname,x)
z = feval(fname,x);
y = 2*z.^2+3*z-1;
```

Calculating the function h(x) is:

```
x = -pi:pi/20:pi;
y = f('sin',x);
plot(x,y)
```



Outline

Numerical Errors

- It is very hard to get computers to perform exact calculations, which means numerical methods are approximate methods($+, -, /$).
- We know that $1/3 = 0.\dot{3}$ does not have a finite representation, so we have no choice but by truncating the sequence. Obviously the more three's we retain the more accurate the answer.
- Almost all numerical schemes are prone to some kind of error. Errors can be expressed as two basic types:
  - Absolute error: This is the difference between the exact answer and the numerical answer.
  - Relative error: This is the absolute error divided by the size of the answer (either the numerical one or the exact one), which is often converted to a percentage.

Example 2.9

Suppose an error of $1 is made in a financial calculation of interest on $5 and on $1,000,000. In each case the absolute error is $1, whereas the relative errors are 20% and 0.0001% respectively.

Example 2.10

Suppose a relative error of 20% is made in the above interest calculations on \$5 and \$1,000,000. The corresponding absolute errors are \$1 and \$200,000.

Example 2.11

Estimate the error associated with taking 1.6 to be a root of the equation $x^2 - x - 1 = 0$.

The exact values for the roots are $\left(1 \pm \sqrt{5}\right)/2$ *(let us take the positive root). As such the absolute error is*

$$\left|\frac{1+\sqrt{5}}{2} - 1.6\right| \approx 0.01803398874989$$

and the relative error is the absolute error divided by the value 1.6 (or alternatively the exact root) which is approximately equal to 0.01127124296868 or 1.127%. We could also substitute $x = 1.6$ *into the equation to see how wrong it is:* $1.6^2 - 1.6 - 1 = -0.04$.

This method can be used to determine roots of a function, see next Example.

Example 2.12

Determine a value of x such that

$$f(x) = x^2 + 4x = 40$$

We start by guessing that $x = 6$ is the root we require:

$x = 6$, $f(6) = 60 > 40$ *which is too big, try* $x = 5$.
$x = 5$, $f(5) = 45 > 40$ *which is still too big, try* $x = 4$.
$x = 4$, $f(4) = 32 < 40$ *now this is too small, try* $x = 4.5$.
$x = 4.5$, $f(4.5) = 38.25 < 40$ *a bit too small, try* $x = 4.75$.
$x = 4.75$, $f(4.75) = 41.5625 > 40$ *a bit too big, try* $x = 4.625$.
$x = 4.625$, $f(4.625) = 39.890625 < 40$ *and we can continue this process...*

Here we have just moved around to try to find the value of x such that $f(x) = 40$, but we could have done this in a systematic manner (actually using the size of the errors).

Numerical Errors – eps(smallest positive number)

Usually we focus on errors we are merely trying to work out their size (or magnitude) rather than their actual numerical value.

MATLAB variable `eps` $= 2^{-52} \approx 2.2204 \times 10^{-16}$ is defined as the smallest positive number such that `1+eps` is different from `1`. Consider the calculations:

```
x1 = 1+eps;
y1 = x1-1
x2 = 1+eps/2;
y2 = (x2-1)*2
```

You would expect that `y1` and `y2` are equal to eps; but in fact `y2` is equal to zero. This is because MATLAB cannot distinguish between `1` and `1+eps/2`.

The quantity eps is very useful, especially when it comes to testing routines.

Example 2.13

Calculate the absolute errors associated with the following calculations:

```
sin(15*pi)
(sqrt(2))^2
1000*0.001
1e10*1e-10
```

To calculate the absolute errors we need to know the exact answers which are 0, 2, 1 and 1 respectively. We can use the code:

```
abs(sin(15*pi))
abs((sqrt(2))^2-2)
abs(1000*0.001-1)
abs(1e10*1e-10-1)
```

The errors are $10^{-15}, 10^{-16}$ *and zero (in the last two cases).*

User Error

- User error is avoidable, whereas many numerical errors are intrinsic to the numerical techniques or the computational machinery being employed in the calculation.
- The most severe user errors will often cause MATLAB to generate error messages; these can help us to understand and identify where in our code we have made a mistake.
- If all the syntax errors have been eliminated within a code (that is MATLAB is prepared to run the code), the next level of errors are harder to find. These are usually due to:

User Error – Next Level Error

- *An incorrect use of variable names.*
  This may be due to a typographical error which has resulted in changes during the coding.
- *An incorrect use of operators.*
  The most common instance of this error occurs with dot arithmetic.
- *Syntactical errors which still produce feasible MATLAB code.*
  To evaluate the expression $\cos x$, we should use `cos(x)`: unfortunately the expression `cos x` also yields an answer (which has the cosines of the ASCII values of the letter `x` as element).
- *Mathematical errors incorporated into the numerical scheme.*
  These occur where the requested calculation is viable but incorrect.
- *Logical errors in the algorithm.*
  This is where an error has occurred during the coding and we find we are simply working out what is a wrong answer.

Example 2.14

This code is used to obtain three numbers a, b and c from a user and then produce the results a + b + c, a/((b + c)(c + a)) and a/(bc).

```
a = input(' Please entere a )
b = 1nput(' Please enter b )
a = Input ' Please enter c '

v1 = a+ B+d
v2 = a/((b+c)(c+a)
v3 = a/b*c

disp(' a + b + c= ' int2str(v1)])
disp(['v2 = ' num2str v2 ]
disp(['v3 = num2str(v4) ]);
```

We shall now fix errors through the lines one-by-one.

Example 2.14(Cont'd, Lines 1 & 2)

- `a = input(' Please entere a )`
  Firstly the single left-hand quote mark should be balanced by a corresponding right-hand quote mark.
  The command should also end with a semicolon.
  The misspelling of the word enter is annoying but will not actually affect the running of the code.

- `b = 1nput(' Please enter b )`
  We have an imbalance in the single quote mark and the lack of a semicolon. We also have the fact that `input` has become `1nput`.

Example 2.14(Cont'd, Line 3)

- `a = Input ' Please enter c '`
  Here we have a capitalised command, which MATLAB will actually point out in its error message.
  We are missing the semicolon, but the more severe errors here are the lack of brackets around the message and the fact that we are resetting the variable a.

Example 2.14(Cont'd, Line 4)

- `v1 = a+ B+d`
  The spacing is unimportant and is merely a matter of style.
  Here again we should add a semicolon.
  The variable `b` has been capitalized to B.
  New variable `d` has also slipped in. This means MATLAB will either complain that the variable has not been initialised or will use a previous value which has nothing to do with this calculation.
  To avoid this it is a good idea to start the code with Command `clear all` to clear all variables from memory.

Example 2.14(Cont'd, Lines 5 & 6)

- `v2 = a/((b+c)(c+a)`
  This command presents a number of errors in the evaluation of the denominator of the fraction.
  The first of these is the missing asterisk between the two factors and secondly we have an unbalanced bracket.

- `v3 = a/b*c`
  We need to force the calculation bc to be performed by using brackets.
  We might still want to add semicolon to stop extra output.

Example 2.14(Cont'd, Line 7)

- `disp(' a + b + c= ' int2str(v1)])`
  This command is missing a left square bracket to show that we are going to display a string.
  Variables `a`, `b` and `c` are probably not integers, and command `int2str` which takes an integer and returns a character string. We should instead use the command `num2str` which converts a general number to a character string.

Example 2.14(Cont'd, Lines 8 & 9)

- `disp(['v2 = ' num2str v2 ]`
  This command is missing a round bracket from the end of the expression and a pair of brackets to show that `num2str` is being applied to the variable `v2`.

- `disp(['v3 = num2str(v4) ]);`
  We are missing a single quote to show that the string has ended.
  We have also introduced a new variable `v4` which should be `v3`.
  The semicolon on the end is unnecessary since the `disp` command displays the result.

Example 2.14(Cont'd)

The corrected code looks like:

```
clear all
a = input(' Please enter a ');
b = input(' Please enter b ');
c = input(' Please enter c ');

v1 = a+b+c;
v2 = a/((b+c)*(c+a));
v3 = a/(b*c);

disp([' a + b + c= ' num2str(v1)])
disp([' v2 = ' num2str(v2)])
disp([' v3 = ' num2str(v3)])
```

Outline

Tasks

Task 1.1 *Enter and run the code*

```
a = input(Enter a :  );
b = input(Enter b :  );
res = mod(a,b);
str1 = The remainder is ;
str2= when;
str3 =  is divided by ;
disp([str1 num2str(res) str2 ...
  num2str(a) str3 num2str(b)]);
```

*which should be saved as **remainders.m**. Experiment with this code by running it with various values for **a** and **b**. Make sure you understand how this code works, in particular the **mod** command and the **disp** command.*

Tasks

Task 2.2 *Modify the code in Task 2.1 so it returns the answer to* a^b *and change the character strings* $\boldsymbol{str}\mathbf{1}$, $\boldsymbol{str}\mathbf{2}$ *and* $\boldsymbol{str}\mathbf{3}$ *so that the format of the answer is 'The answer is* a^b *when a is raised to the power b'.*

Tasks

Task 2.3 *Write a code which takes a variable x and returns the value of 2^x. Make sure that your code works for variables which are scalars, vectors and matrices.*

Tasks

Task 2.4 *By modifying the function* ***func.m*** *(given earlier on Example 2.3) repeat the example for the functions* $x^2 - y^2$ *and then* $\sin(x + y)$ *(extending the range to* $[0, 2\pi]$ *in the latter case).*

Tasks

Task 2.5 *Modify the code **xpowers.m** (given earlier on Example 2.4) to simultaneously give the values of the functions* $\sin x$, $\cos x$ *and* $\sin^2 x + \cos^2 x$.

Tasks

Task 2.6 *Modify the code* ***multi.m*** *in Example 2.5 to work out the values of the map:*

$$\begin{aligned} x \mapsto & \quad (x+y)|1 \\ x \mapsto & \quad (x+2y)|1 \end{aligned}$$

where $(a|b)$ is the remainder when a is divided by b and can be calculated using the MATLAB function ***mod****. If b is unity this is the fractional part. This new function will take two inputs and return two outputs.*

Tasks

Task 2.7 *The code*

```
clear x y
x = -2:0.1:2;
y = 9-x.^2;
plot(x,y)
```

plots the function $y = 9 - x^2$ for $x \in [2, 2]$ in steps of 1/10. Modify the code so the function $y = x^3 + 3x$ is plotted between the same limits and then for $x \in [4, 6]$ in steps of 1/4. (If you can't see the current figure, type ***figure*****(1)** *which should bring it to the foreground). This code can be run from the prompt or can be entered using* ***edit*** *and then saved.*

Tasks

Task 2.8 *Consider the quartic* $y = x^4 + x^2 + a$. *For which values of* a *does the equation have two real roots?*

Tasks

Task 2.9 *Plot on the same graph the functions* $f(x) = x+3$, $g(x) = x^2+1$, $f(x)g(x)$ *and* $f(x)/g(x)$ *for the range* $x \in [1, 1]$. *(You need to decide how many points to use to get a smooth curve, and whether to set up the vector* ***x*** *either using the colon construction or* ***linspace****). You also need to remember to use dot arithmetic since you are operating on vectors.*

Tasks

Task 2.10(D) *This task contains codes which are written with deliberate mistakes. You should try to debug the codes so that they actually perform the calculations they are supposed to:*

1. Perform the calculations

$$x = 4$$

$$x + 2 = y$$

$$z = \frac{1}{y^2\pi}$$

```
x=4
x+2=y
z=1/y^2Pi;
```

Tasks

Task 2.10(D)(Cont'd) *2. Calculate the sum*

$$\sum_{i=1}^{N} \frac{1}{i} + \frac{1}{(i+2)(i+3)}$$

where the user inputs N.

```
N=input('Enter N )

for i=1:n
    sum = 1/j + 1/(j+2)*(j+3)
end

disp( ' The answer is ' s])
```

Make sure the code gives the correct answer, for instance for $N = 1$.

Tasks

Task 2.10(D)(Cont'd) *3. Calculate the function*

$$f(x) = \frac{x\cos x}{(x^2+1)(x+2)}$$

for x from 0 to 1 in steps of 0.1.

```
x==0.0:0.1:1.0;
f=xcos x/*([x^2+1]*(x+2)
```

Tasks

Task 2.10(D)(Cont'd) *4. Set up the vector 1 3 3 3 5 3 7 3 9*

```
w = ones(9);

w(1) = 1;

for j = 1:4
    w(2j) = 3:
    w(2j+1) = 2j+1:
```

Make sure that the code returns the correct values of the entries of w.

Tasks

Task 2.11 *The following codes should be written to produce conversion between speeds in different units.*

(***a***) *Construct a code which converts a speed in miles per hour to kilometres per hour.*
(***b***) *Write a code which converts metres per second to miles per hour and use it to determine how fast a sprinter who runs the 100 metres in 10 seconds is travelling in miles per hour (on average).*
(***c***) *Rewrite your code from part* (***a***) *so that it is now a function that takes a single input, the speed in miles per hour, and returns a single output, the speed in kilometres per hour.*
(***d***) *Now modify the code you wrote in part* (***b***) *to determine the sprinter's speed in kilometres per hour by calling the function from the previous part.*

*(***NB***: 1 mile = 1760 yards; 1 yard = 36 inches; 1 inch = 2.54 cm; 1 m = 100 cm)*

Tasks

Task 2.12 *The functions f(x) and g(x) are defined by*

$$f(x) = \frac{x}{1+x^2} \quad \text{and} \quad g(x) = \tan x$$

Write MATLAB codes to calculate these functions and plot them on the interval $(\pi/2, \pi/2)$*. Also plot the functions* $f(g(x))$ *and* $g(f(x))$ *on this interval.*

Tasks

Task 2.13 *Write a code which enables a user to input the coefficients of a quadratic* $q(x) = ax^2 + bx + c$ *and plot the function* $q(x)$ *for* $x = \sin y$ *where* $y \in [0, \pi]$.