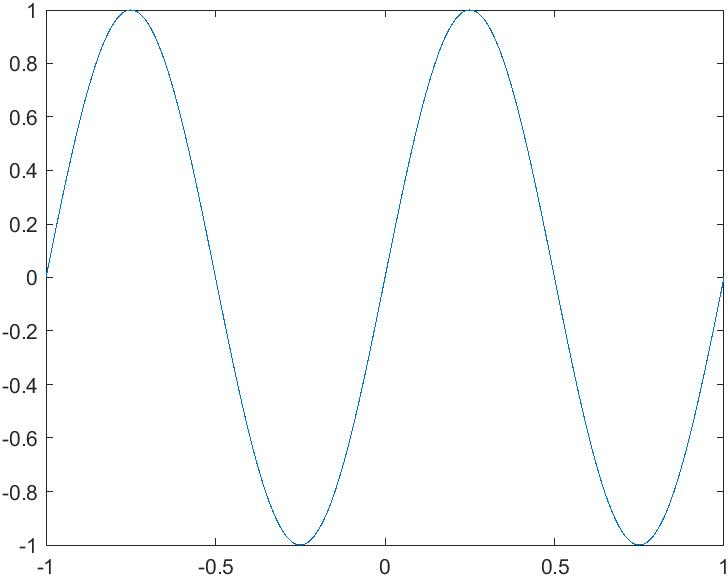
Solutions of Exercise 1

1. For pi, I found 3.1416.
2. For eps, I found 2.2204e-16.
3. For realmax, I found 1.7977e+308.
4. For realmin, I found 2.2251e-308.
5. For pi, I found format long as 3.141592653589793.
6. I found pi-3.1416 = -7.346410206832132e-06
7. I wrote a = 1 and b = 1+eps respectively and calculate them. Then, firstly I compute a – b with a = and b = 1.0000 its result will be a – b = -2.2204e-16. Thus, although a and b seem to be the same, a – b is negative.
8. I firstly check whether b is the same as before format long command. But I saw that it has more decimals than the previous one, i.e. I saw b = 1.000000000000000. Then, difference between a and b a-b will be -2.220446049250313e-16
9. I checked that 2 = 2+eps, then surprisingly we see that they are equal to each other.
10. I chose x = 5.5. Then, we found that square of x is 30.2500 and cube of x is 166.3750
11. I chose = . Then, we found that square of is 0.0086 and is 1.0000. Furthermore, MATLAB uses degree in radians.
12. Firstly, I wrote a1 = ‘sqrt(4)’ is the given as string form, but when we wrote a2 = sqrt(4), MATLAB calculate a2 and gives the result as 2.
13. When we use “eval” for a1, MATLAB again compute a1 and gives the result as the same as a2. Similarly, a3 = 6\*eval(a1) = 12.

Solutions of Exercise 2

1. We wrote it and MATLAB gives us meshPoints = linspace(-1,1,1000);
2. The 95th element of meshPoints is meshPoint(95) = -0.8118
3. Yes, we have seen that its length is 1000 with the help of length(meshPoints).
4. We confirmed that its length is 1000 with the help of numel function.
5. We have a graph using “plot(meshPoints,sin(2\*pi\*meshPoints))“ on the interval [-1,1] and we get the graph located below.



Solutions of Exercise 3

Firstly, I define rowVec1 = , colVec1 = and mat1 =

1. I multiply colVec1 by and named it by colVec2 such that colVec2 =
2. Then I applied cos function to colVec2 and I get colVec2 = colVec2 at 1. and colVec2 at 2. are different from each other clearly because we overwrite it.
3. Matrix addition are defined if dimensions are the same for two matrices. So, we can add colVec1 and colVec2 and get colVec3 =
4. But we cannot add a column vector and a row vector. We try that adding colVec1 and rowVec1 and we get illegal = , which is WRONG.
5. We can compute norm of a matrix with the help of “norm” function. Thus, norm(colVec3) = 13.3876
6. We can multiply two matrices A and B if number of rows of A = number of columns of B. So, we can multiply mat1 and colVec1 matrices and we get colVec4 =
7. ‘ denotes the transposition. So, mat1Transpose = mat1‘ = and rowVec2 = colVec3‘ =
8. Now, we multiply mat1 and mat1Transpose and we get mat2 = Clearly, mat2 is a symmetric matrix.
9. If we multiply rowVec1 and mat1, which is possible because our condition is satisfied, we get rowVec3 =
10. We can compute dot product also in MATLAB by using “dot” command. So, dot product of colVec3’ and colVec1 will be 163.3640.
11. Then euclideanNorm = sqrt(colVec2’ \* colVec2) = 1.2247
12. We can compute determinant of a matrix and its trace with the help of “det” and “tr” commands respectively. Then, det(mat1) = -22.0000 and tr(mat1) = 16
13. To find minimum element of rowVec1, we should use “ min(rowVec1) “ command; and thus, min(rowVec1) = -9
14. When I compute min of mat1 matrix and max of mat1 matrix, I get min(mat1) = and max(mat1) =
15. max norm of the matrix mat1 will be max(abs(rowVec1)) = 9
16. We found single element because we compute row vector. But if we calculate a 3x3 matrix as like mat1, we will get 1x3 matrix like in the example of 14. Thus, we get single element.
17. I called magic square matrix A whose dimension is 201x201 and I put “ ; “so that it does not print all such matrices. That is A = magic(201);
18. We can find column sum of the matrix A by column\_sum = sum(A). Clearly, all column sum is 4060401. Now, let’s look row sum. We can find row sum of the matrix A by row\_sum = sum(A,2). Thus, clearly, row sum will be the same as column sum. To find diagonal sum, we should use trace(A) command. Then, it gives us also 4060401. Since the matrix A is a magic square, we can verify that it is a magic square with this method. Instead of this long process, we can do also define again column\_sums = sum(A); and row\_sums = sum(A,2); which gives us all column and row sums without printing all values and compute their min and max values. Thus,

min(column\_sums) = 4060401

max(column\_sums) = 4060401

min(row\_sums) = 4060401

max(row\_sums) = 4060401

as expected. Furthermore, since diagonal sum is the same as we find above, we can briefly prove that A is a magic square.

1. We firstly formed a sequence of integers by integers = 0:10 (0 1 2 3 4 5 6 7 8 9 10) As expected, multiplication of integers \* integers give us an error. So, when we apply correct version of this operation, we get squareIntegers = integers .\* integers give us squareIntegers = 0 1 4 9 16 25 36 49 64 81 100

Then, similarly, cubeIntegers =

Columns 1 through 10

0 1 8 27 64 125 216 343 512 729

Column 11

1000

and similarly, fourthIntegers = Columns 1 through 10

0 1 16 81 256 625 1296 2401 4096 6561

Column 11

10000

Finally, to give all of them in a table, we should use command tableOfPowers = [integers', squareIntegers', cubeIntegers', fourthIntegers'] and this gives us the table below,

tableOfPowers =

0 0 0 0

1 1 1 1

2 4 8 16

3 9 27 81

4 16 64 256

5 25 125 625

6 36 216 1296

7 49 343 2401

8 64 512 4096

9 81 729 6561

10 100 1000 10000

1. We firstly evaluate square of the integers from 0 to 10 by sqIntegers = integers .^ 2 and we wonder whether these values are the same as numbers we found before. To see this, we use norm(sqIntegers - squareIntegers) command and we see the result as 0. Thus, we conclude that these two integers are the same as to each other.

For the command tableOfCubes = tableOfPowers(:,[1,3]) we get

tableOfCubes =

0 0

1 1

2 8

3 27

4 64

5 125

6 216

7 343

8 512

9 729

10 1000

because this says that from the Table of Powers table, you should take first and third columns with all entries and call this as tableOfCubes.

For the command tableOfEvenCubes = tableOfPowers(1:2:end,[1,3]) we get

tableOfEvenCubes =

0 0

2 8

4 64

6 216

8 512

10 1000

because this says that from the Table of Powers table, you should take first and third columns and begin with first row and go to the end incremented by 2. So, we took even integers.

For the command tableOfOddFourths = tableOfPowers(2:2:end,1:3:4) we get

tableOfOddFourths =

1 1

3 81

5 625

7 2401

9 6561

because this says that from the Table of Powers table, you should take first column and end it fourth column incremented by 3, i.e. it says us just take first and fourth columns, and begin with second row and go to the end incremented by 2. So, we took odd integers.

Therefore, we can say that in MATLAB, ordering is beginning with 1 and goes to end.

1. Let B = magic(10); i.e. B be a magic square matrix by 10 x 10. Then the matrix will be

B =

92 99 1 8 15 67 74 51 58 40

98 80 7 14 16 73 55 57 64 41

4 81 88 20 22 54 56 63 70 47

85 87 19 21 3 60 62 69 71 28

86 93 25 2 9 61 68 75 52 34

17 24 76 83 90 42 49 26 33 65

23 5 82 89 91 48 30 32 39 66

79 6 13 95 97 29 31 38 45 72

10 12 94 96 78 35 37 44 46 53

11 18 100 77 84 36 43 50 27 59

Then, Upper Left Corner of B (BUL) will be

BUL =

92 99 1 8 15

98 80 7 14 16

4 81 88 20 22

85 87 19 21 3

86 93 25 2 9

Similarly, Upper Right Corner of B (BUR) will be

BUR =

67 74 51 58 40

73 55 57 64 41

54 56 63 70 47

60 62 69 71 28

61 68 75 52 34

Similarly, Lower Left Corner of B (BLL) will be

BLL =

17 24 76 83 90

23 5 82 89 91

79 6 13 95 97

10 12 94 96 78

11 18 100 77 84

and finally, Lower Right Corner of B (BLR) will be

BLR =

42 49 26 33 65

48 30 32 39 66

29 31 38 45 72

35 37 44 46 53

36 43 50 27 59

Then, we write C = [BUL BUR;BLL BLR] and check that are C and B same by norm(B-C) command. Then, we find this norm as 0. Therefore, we conclude that B and C are the same. So, we learn that we can write a matrix as like C.

Solutions of Exercise 4

1. First of all, we have evaluated the true value of the = 1.7183. Then we have used the exer4.m file in order to obtain an approxIntegral, which is equal to 1.7184. The result is quietly near to the true value of the .
2. We have done this.
3. The complete sequence of all values taken on by the variable x is the following

-0.025641025641026  
 0

0.025641025641026

0.051282051282051

0.076923076923077

0.102564102564103

0.128205128205128

0.153846153846154

0.179487179487179

0.205128205128205

0.230769230769231

0.256410256410256

0.282051282051282

0.307692307692308

0.333333333333333

0.358974358974359

0.384615384615385

0.410256410256410

0.435897435897436

0.461538461538462

0.487179487179487

0.512820512820513

0.538461538461539

0.564102564102564

0.589743589743590

0.615384615384616

0.641025641025641

0.666666666666667

0.692307692307693

0.717948717948718

0.743589743589744

0.769230769230770

0.794871794871795

0.820512820512821

0.846153846153847

0.871794871794872

0.897435897435898

0.923076923076924

0.948717948717949

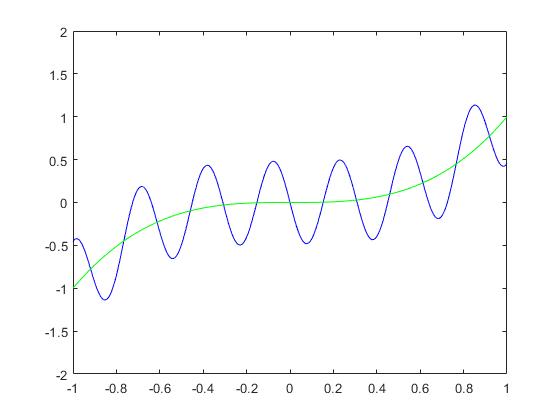
0.974358974358975

1.000000000000000

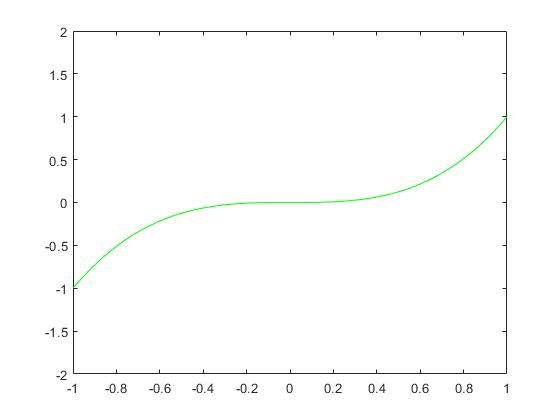
1. 2 times.
2. 38 times.

Solution for Exercise 5

1. We have written our names and the date into the comment section.
2. The variable x in Matlab can be used to determine the result that the Equation (2) gives us, however here x does not have to be a specific real number, for instance x given as a matrix (linspace), in this case the dummy variable will be correspondence nothing but each entry of this matrix x here. In this way, the result of the equation can be evaluated for each point in the linspace (for each entry in the matrix).
3. Inside the “for” loop, the statement begins with “y = y …” actually means that, the variable y recursively obtains sum operations in the loop over and over, which correspondences to ∑ operation in the equation.
4. The statement “y = zeros(size(x))”, creates a square matrix with all entries are 0 and the dimension of this matrix is the size of the variable x. Here x is assigned to be a “linspace”, therefore the size of x would simply mean the number of distinct points in the linspace’s interval. Also, this newly created matrix assigned to the variable y.
5. In exercise set, 5. question is not given, so we skipped.
6. We have executed the script and obtain the two graphs in one image just as follows;



1. If we omitted these two lines from the script, then we do not obtain the same graph with the original script. We can observe that the command “hold” helps us to combine multiple graphs on one picture. So, these two lines are crucial in order to obtain the both graph at the same time.



Solutions of Exercise 6

1. We have done this part.
2. We have also done this.
3. This part of the code makes the term always bigger than tolerance, with this way we can start the while loop in the code. Otherwise, the loop code may not be executed at the start.
4. The purpose of this code line is that in order to iterate the Fourier Series sum.
5. 349 iterations are needed.

Solutions of Exercise 7

1. We have done this part.
2. We have replaced the assignment for TOLERANCE into lower-case version.
3. We have made the name of the function exer7 to agree with the name of the file.
4. Since the graphs of the functions are not needed anymore, we have deleted the display code lines.
5. We have called the function with the code: exer7(0.05)
6. We have called the function with the code: numItsRequired=exer7(0.05)
7. We have done this; we make sure that it is really a function.
8. 349 iterations.
9. For the tolerance 0.1 we obtain 198 iterations.

For the tolerance 0.05 we obtain 349 iterations.

For the tolerance 0.025 we obtain 784 iterations.

For the tolerance 0.0125 we obtain 1580 iterations.

Solutions of Exercise 8

1. We have copied the “exer7.m” file from the last exercise, and save as “exer8.m”. In this updated script file, we have changed some structures of the function especially the returning part of the function. In this version, our function returns two different values, first one is the converged value of the sum, which is a vector and, the second one, the number of iterations required for the given tolerance.
2. In order to obtain the only vector value as an output from the function, we have used the following code;

|  |
| --- |
| x\_0 = exer8(0.03) |

|  |
| --- |
| norm(x\_0) |

After using this code, the system created new variable namely x\_0 and assigned it to the output sum vector. After this part in order to obtain the norm of this vector (in Euclidean Metric space, usual metric space), we have used the following code;  
  
  
The workspace gave the output as “11.995259787700071”, which is the norm of the vector.

1. In order to obtain the both vector value and the number of iterations as an output from the function, we have used the following code;

|  |
| --- |
| [x\_1, y\_1] = exer8(0.02) |

|  |
| --- |
| norm(x\_1) |

After using this code, the system created new variable namely x\_1 and y\_1, which are assigned into the sum vector and the number of iterations respectively. Also, from the workspace output we can see that ” y\_1 = 987” , so 987 iterations exactly needed for 0.02 tolerance. After this part in order to obtain the norm of this vector (in Euclidean Metric space, usual metric space), we have used the following code;  
  
The workspace gave the output as “11.990957936147460”, which is the norm of the x\_1. We can see that, the differences between x\_0 vector’s norm is actually very close to this vector’s norm

1. In order to obtain the only the number of iterations as an output from the function, we have used the following code;

|  |
| --- |
| [~, y] = exer8(tolerance) |