Faculty of Engineering
MM1CPM

Worksheet 7 – Algorithms

Sorting

Q1: This example will guide you through the process of writing a sorting algorithm from scratch. There is an example in the lecture notes too, but this should give you some practical experience about building up algorithms step-by-step.

a) Define a 1D random array of numbers b of length 5. It is best for this example if you choose the random numbers by hand rather than use the rand command.

b) Set the variable 'len' to five.

c) Write a for loop that counts from 1 to len-1 using the variable 'n'.

d) Inset an ***if*** statement into the ***for*** loop to check if the n^{th} value is bigger than the $n+1^{th}$ value in the array. If the condition is met the program should print the text 'I need to swap the values screen'.

e) If the ***if*** statement is true to part d, swap the values at the n and n+1 position. There is an example of how to do this in the lecture notes. What happens when you run your script? Why does it not sort the numbers?

d) Enclose the first ***for*** loop your wrote with another for loop that counts from 1 to len using the variable nn. You have now written your first sort algorithm! What happens if you change the '>' in the if statement to a '<'.

e) Modify your code so it will sort any list of numbers.

Finding the maximum and minimum

Q2: Although MATLAB provides a lot of built in functions such as ***max*** and ***min***. It is often a good idea to at least have an idea how the built in functions work. This example demonstrates how the min and max functions work. Make a new script file called q2.m

a) Define an array ten of numbers by hand between 0 and 100, call this array 'b'

b) Define a new variable my_max by setting it equal to the 1st element of array 'b'

c) Write a for loop that loops from 1 to the length of array 'b'

d) Write an if statement within the for loop that checks if the i^{th} value of array b is bigger than my_max. If this condition is met set my_max to the ith value of the array b.

e) Add an ***sprintf*** command at the bottom of the script to print 'The maximum of the array is ??', where ?? is value of my_max.

f) The built in functions in MATLAB such as max and min, are good but often don't do exactly what you want them to do. Now we are going to produce a modified function which only works on the first half of the data in the array. Do this by either inserting an ***if*** statement into the ***for*** loop or by changing the for range of the loop.

Q3: In this example you will change the script q2.m to find the minimum value of an array instead of the maximum value. First, copy the script file q2.m to q3.m. Change the variable name my_max to my_min, now change the if statement to check if the i^{th} element of array 'b' is smaller than my_min. Update the sprintf statement accordingly. Your script should now find the minimum value in the array.

Integrating

Q4: Integrating is just finding the area under a curve. In this question we are going to do a numerical integration of the equation $y=x^2$, between 0 and 2 by hand:

a) On the piece paper sketch the curve $y=x^2$. Do this by first calculating the value of y at x=0, x=1 and x=2. Using a pen or a pencil, draw a smooth curve joining the dots. Now using a pencil, draw two rectangular boxes which approximate the area under the curve between x=0 and x=2. What is the total area of the boxes using this method?

b) Now using what you learnt in mathematics integrate the equation $y=x^2$ between the limits x=0 and x=2. What's the area under the curve using this method?

c) Why do the answers for a and b not match? What could you do to make the answers match?

Q5: Numerical integration on a computer works just like the example in question 4a. In this question we will be integrating the equation $y=x^2$ between x=0 and x=2 using MATLAB.

a) Make a new script file called q5.m in this script file make a while loop that will continue running as long as x is smaller or equal to 2. You should initialize x to 0 at the start of the script. The loop should count up in steps of 0.1, the step size should be initialized as a variable dx at the start of the program.

b) Inside the while loop, calculate the value of 'y' for each value of 'x'. Multiply y by the step size to calculate the area of each integration box store this value in the variable 'box_area'.

c) Initialize a variable at the top of the script called 'sum', this will hold the sum of all boxes. Add the area of each box to 'sum' each time the loop runs.

d) Using the analytical expression for the integral between 0 and 2 you calculated in question 4b, define a variable error which will hold the difference between the numerical and analytical expressions. Print out the error at the end of the script using **sprintf**.

e) Congratulations you have just numerically integrated your first function! Try changing the step size dx from 0.5 to 0.0001. What happens to the error? Why is this?

Q6: Copy the script you write in q5.m to a new file called q6.m. Remove the lines which calculate the error.

a) The script only integrates between 0 and 2. Change the script so that it asks the user for the two limits of integration. If the second limit is smaller than the first limit the program should swap the first and second limit so the program continues to run. If the limits are both the same then the program should print the warning message 'Limits of integration are the same'

Now integrate the following functions by using your script:

b) $\sin(x)$ between 1 and 2

c) $\cos(x)$ between 1 and 2

d) x^3 between 1 and 2

e) $\cos(\cos(x))$ between 2 and 4

f) e^{-x} between 0 and 1

g) $\sin(x)e^{-x}$ between 0 and 1

Congratulations you can now integrate any function you want to between any limits!

Differentiating

Q7: In this question you will practice numerical differentiation of functions. Differentiation, is just finding the gradient of a line. The simplest way to differentiate a function is to calculate it's y value at one place on a curve, calculate it's y value at another place on the curve. Then calculate the difference between the y values and the x values at the two points. This was explained during the lecture.

a) Make a script to ask the user for two points on the x-axis, x1 and x2.

b) For the equation $y=x^2$ calculate the value of y at each point, and store the values in y1 and y2

c) Calculate the difference between x2 and x1 and store this in a new variable dx.

d) Calculate the difference between y2 and y1 and store this in a new variable dy.

e) Divide dy by dx to calculate the gradient of $y=x^2$ between the two points, store the value in numerical_dydx.

f) Differentiate $y=x^2$ by hand and evaluate it mid way between x2 and x1, store the result in analytical_dydx.

g) Subtract analytical_dydx and numerical_dydx. Why is there a difference between the values. Try choosing points of x1 and x2 closer together. What happens to the error?

Q8: Copy the script you made in q7, to q8.m. Remove the line which calculate the error and replace the function $y=x^2$ with the following functions and find the differential at x=1:

b) $\sin(x)$

c) $\cos(x)$

d) x^3

e) $\cos(\cos(x))$

f) e^{-x}

g) $\sin(x)e^{-x}$

Q9: Write a script to plot sin(x) between 0 and 4pi using 100 points. Now edit the script to calculate the numerical derivative of sin(x) at every point on the x-axis.