MATLAB Iterations Exercises

WDRP - Simple Discrete Models in Biology and MATLAB

Contents

1 Printing and For Loops **2**

2 Linear Regression **3**

3 Correlation Coefficients **4**

4 Cauchy Sequences **5**

5 Error Bounds **6**

6 Plotting and Accuracy **7**

7 Direction Fields **8**

8 3D Plotting **9**

9 Monte Carlo Method **10**

10 Monte Carlo Method Cont. **11**

11 Midpoint Approximation **12**

12 Midpoint Approximation Cont. **13**

13 Collatz Conjecture **14**

14 Battle Ship **15**

Exercise 1: Printing and For Loops

In this problem we will print the entries of an array row by row using a for loop. To do this

- Generate an array with 5 rows and 5 columns with random entries between 1 and 10.
- Write a for loop with index i which starts at 1 and goes to 5 (think of this setting which row of the array we are working on).
- Write a for loop with index j, starting at 1 and going to 5 (think of this as setting which column of the matrix we are working on).
- Make sure the second for loop is INSIDE of the first for loop (think of the first loop as saying which row we are in, and the second loop as going throw all the entries of that row).
- print the number in the entry (i, j) using the fprintf command.
- Make sure that whenever a row is done printing, you move to the next line so that the printed array resembled how matlab displays it. To do this use \ n. (But should this be in the first or second for loop?)

Exercise 2: Linear Regression

In this problem we will take a collection of data and make a "line best fit" to average and make predictions. Our line best fit will use one of the simplest methods, called the "least squares regression line". In particular, the line has equation

$$\hat{y} = a + bx$$

where the coefficients a and b are determmined by the following formulas

$$b = \frac{\sum [(x_i - \overline{x})(y_i - \overline{y})]}{\sum [(x_i - \overline{x})^2]} \qquad a = \overline{y} - b\overline{x}.$$

The variables $\overline{x}$ and $\overline{y}$ are the usual averages of the x and y coordinates of the data points. Namely

$$\overline{x} = \frac{1}{n}\sum x_i \qquad \overline{y} = \frac{1}{n}\sum y_i$$

Here is how we will do this

1. Use the provided data points, which are given with both x and y values. Separate the coordinates into two appropriately named arrays.
2. Create a function which takes in a array and computes the average of the values in the array. It should also return the average value computed.
3. Create two functions, one that computes the value of the coefficient b and one which computes the value of the coefficient a.
    - The first should take in the two arrays as well as their averages.
    - It should then compute the value of b using a for loops.
    - It should return the value of b
    - The second should take in the averages of the two arrays and the value b
    - Then it should compute the value of a and return it.
4. Plot the line best fit, as well as the data points.

Exercise 3: Correlation Coefficients

In this problem we will take multiple sets of data and compare them by determining the correlation between them. Recall that for two collections of data we can determine their Pearson Correlation Coefficient using the following formula

$$r_{x,y} = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum(x_i - \overline{x})^2 \cdot \sum(y_i - \overline{y})^2}}$$

where the variables $\overline{x}$ and $\overline{y}$ are the usual averages of the x and y coordinates of the data points. Namely

$$\overline{x} = \frac{1}{n}\sum x_i \qquad \overline{y} = \frac{1}{n}\sum y_i$$

Here's how we'll do it

1. Download the .csv files titled "SeattleTemp", "DiabloTemp", "ForksTemp" from the 'Data' folder. You should put these files in the same folder as your other saved MATLAB files. These files have the average, maximum and minimum daily temperatures of Seatlle, Diablo Lake, and Forks during the last year.
   - Download the .m file at this link and put it in the same folder as the other files you downloaded.
   - Open the file in MATLAB. This is the file you'll work from for this problem.
2. In this problem, the two arrays you will be using are any pair of locations and either their average, maximum, or minimum daily temperatures. For example, you might use "Savg" and "Davg" to see what the correlation between the average seattle temperatures and average Diablo lake temperatures is.
3. Create a function which computes the average of an array. It should take in an array and output the average of it.
4. Create a function which computes the sum in the numerator of the formula. The function should take in two arrays, and their averages while outputting a single value. The value should be the numerator of the formula.
5. Make another function which calculates the sum of the $(x_i - \overline{x})^2$. It should take in an array and it's average and output the value of the sum. You should be able to use this function to compute the sum for both x's and y's.
6. Use the functions previously defined to determine the value of r.

Exercise 4: Cauchy Sequences

In this problem we will determine whether a sequence converges to a particular value, ie. find a stable equilibrium point.

Exercise 5: Error Bounds

In this problem we will determine an upper bound on the error of a left riemann sum. Recall that the upper bound on the error can be calculated by

$$\frac{K_1(b-a)^2}{2n}$$

where K_1 is the upper bound on the first derivative on the interval $[a, b]$ and n is the number of subdivisions.

Since we don't want to have to compute derivatives (although we probably could numerically), we will assume some things

- We will use the function $f(x) = \log(x)$
- We will use the interval $[1, e]$
- In this case the constant will be $K_1 = 1$ (check this!)

We want to determine how many subdivisions are needed in order for the error bound to be no bigger than 0.01. Here's how we'll do this

- Initialize the number of subdivisions "n" as 1.
- store the bounds of the interval as "min" and "max".
- Write a function handle for the error called "E" with formula for the error bound given above.
- Make a loop which should continue until the the error is smaller than 0.01
- Each time the loop restarts, you should increase the number of subdivisions by 1.
- Print the number of subdivisions along with an appropriate statement.

Exercise 6: Plotting and Accuracy

In this problem we will plot a function with progressive accuracy. To do this we will use the sin function. Here is how

- Create an array, called "n", with entries 10,20,30,...,100 to use as the different subdivisions.
- Use a loop with index i which takes values from the array "n".
- In the for loop, calculate the size of the partitions on the interval $[-\pi, \pi]$ using the current subdivision i and store it in the variable "Deltax".
- Make the array "x" to be values from $-\pi$ to π with spacing given by "Deltax".
- Make a function named "FuncEval" which takes in the array "x". This function should return an array which has the values of sin at each entry of "x". (Use the arrayfun command).
- Use the function "FuncEval" to find the y coordinates for the graph.
- Use the plot command to plot the "x" and "y" arrays. Make sure you use the command "hold on" so that all the graphs are plotted on the same figure.

Exercise 7: Direction Fields

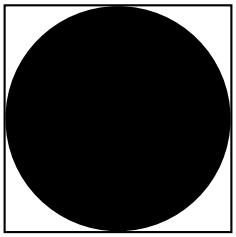
In this problem we will construct a direct field for the logistic model.

Exercise 8: 3D Plotting

In this problem we will plot the evolution of a curve which depends on a parameter.

Exercise 9: Monte Carlo Method

The Monte Carlo method is a probabilistic technique for making approximations. In this problem we will approximate the value of π using a game of darts. Imagine we have a circular dart board inside of a one foot by one foot square frame. Something like the picture below



What is the probability of a dart hitting the dart board? You should convince yourself that it is the ratio of areas

$$P(\text{hit board}) = \frac{\text{Area of Dart Board}}{\text{Area of Entire Frame}}.$$

Remark. Think about the probability of rolling a dice and getting an even number. It is the number of desired outcomes (ie. 2, 4 or 6) divided by the total number of possible outcomes (ie. 1, 2 ,3, 4, 5, 6). So the probability is 1/2.

The area of the frame is 1 and the area of the dart board is πr^2 where the radius is 1/2, and thus is $\pi/4$. Then we can say that

$$\pi = 4 * P(\text{hit board}).$$

The Monte Carlo method we will implement approximates P(hit board). Here's the outline for how we will do it.

- Prompt the user for the number of darts to throw called "DartsNum"
- Create two arrays, called "xTosses" and "yTosses", with random entries between 0 and 1 with up to 3 decimal places (How can you use randi to do this?). The number of entries should be equal to "DartsNum".
- Use a for loop that runs through the entries of the arrays to count how many of the tosses hit the dart board
  - The for loop index should start at 1 and end at the length of your arrays
  - You will have to include a counter that keeps track of every pair that hits the dart board. (This requires a conditional statement since you only want to count a toss <u>if</u> it hits the dart board)
  - (Question) How do we check if a toss hits the dart board mathematically? ie. What equation do the x and y coordinates of the toss need to satisfy to be considered "inside the <u>circle</u>"?
- Just like with the dice example, divide the number of tosses that hit the board by the total number of darts tossed.
- Print a statement describing how many darts were tossed and the approximation of π that you found.

(Bonus) Write the operation that checks if the dart hits the board as a function called "CheckCoord" which takes in the x coordinate and y coordinate of the toss and outputs a truth value.

Exercise 10: Monte Carlo Method Cont.

Use your code from the previous problem to find how many darts are required to have an approximation of π within three decimal places. Here is how

- Put your code from the previous problem into a function called "MonteCarlo" which takes in the number of darts to be tossed and outputs the approximation for π.
- Initialize a variable "Approx" to be 0.
- Initialize a variable "DartsNum" to be 50.
- Create a while loop that continues to run whenever "Approx" is more than 0.001 from the true value of π.
- In the while loop you should call the function "MonteCarlo" to calculate the new value of "Approx". (Don't worry about printing a statement like the last problem unless you want to.)
- Every time the while loop restarts, you should add 10 darts to the number of tosses to, hopefully, get a better approximation.
- Once the desired approximation is reached you should print a statement describing the number of darts tossed and the approximation for π you calculated.

(Question) Once you have your script working correctly, try running it a a few times and see if the number of darts required is always the same or not.

(Bonus) Write a script that finds the number of darts required for the approximation ten times and then takes an average of the outputs.

(Bonus Bonus) Use the bonus to make a guess as to if there is an asymptotic average for the number of darts the monte carlo method requires for such approximations of π. You may need to take the average of more than ten calculations.

Exercise 11: Midpoint Approximation

We will compute the midpoint sum approximation for a given function. To do it you should

- Prompt the user for a function to approximate and store it as a variable "f"
- Prompt the user for the number of subdivisions to use, called "n".
- Prompt the user for the lower bound of the interval, called "min".
- Prompt the user for the upper bound of the interval, called "max".
- Calculate the partition sizes and store in a variable called "Deltax".
- Create an array, called "xi", with the x values of the partition starting at "min", ending at "max" and with step size "Deltax"
- Write a function handle which computes the area of a midpoint rectangle. Use something like

$$\text{Mid}(x1, x2) = (f(x1) + f(x2))/2 * \text{Deltax}$$

- Intitialize a variable "Int" to be 0, which we will use to calculate the integral approximation
- Write a for loop with index i going through the "xi" array. (remember we don't need to include the last endpoint in the calculation so you should only use the "xi" values up until the second to last one)
  - Use the for loop to sum up the areas of the midpoint rectangles. You should use the function "Mid(x1,x2)". Do this by writing

$$\text{Int} = \text{Int} + \text{Mid(i,i+1);}.$$

- Print the approximation calculated with an appropriate statement.

Remark. It may be helpful to try and follow along with the example from the notes.

Exercise 12: Midpoint Approximation Cont.

Using the script from the previous problem we will find how many subdivisions are necessary to get within 3 decimal places of the true value.

- In this problem we will not prompt a user. Rather we will use the interval $[0, 1]$ and the function $f(x) = x^3$.
- You should initialize a counter P as 1 which will be the number of subdivisions in the midpoint approximation.
- Initialize a variable "Approx" to be 0.
- Take the script you wrote in the previous problem as turn it into a function called "MidApprox" which takes in the number of subdivisions "P" and outputs the approximation for the given number of subdivisions and stores it in "Approx".
- On your own, using your calculus knowledge, solve

  $$\int_0^1 x^3 dx$$

  and save the number you get as "TrueVal".
- Make a while loop which runs if "Approx" is not within 0.001 of "TrueVal".
- Every time the While loop restarts you should add one to the number of subdivisions P.
- Print the number of subdivisions necessary for the approximation to be within 0.001 of the true value along with an appropriate statement.

Exercise 13: Collatz Conjecture

The Collatz conjecture is one of the most famous unsolved problems in mathematics. Here is the statement:

> Start with a positive integer n and perform one of the following operations,
>
> - If n is even, divide by 2
> - If n is odd, write $3n + 1$
>
> The resulting number is taken as your new n and the process is repeated. The Collatz conjecture states that with any starting n, this process eventually reaches 1.

For example, take $n = 5$, so first step gives $3 * 5 + 1 = 16$. Since 16 is even, divide by 2 to get 8. Even, so divide by 2 again to get 4. Divide again to get 2. Thus, the next step gives us 1.

You will model the process outlined in the Collatz conjecture. Your code should do the following

- Prompt the user for a starting number.
- Write a function "CheckParity" which takes in the current number and checks whether it is even or odd.
- Write a function "EvenCase" which takes in the current (even) number and divides it by 2. The function should return the new number
- Write a function "OddCase" which takes in the current (odd) number and does the odd case operation described before. The function should return the new number
- You should set up a while loop that checks whether the number is even or odd and performs the necessary operation until the number has reached 1. The While loop should use the previous functions defined.
- The while loop should stop once the number becomes 1.
- Initialize a counter called "time" to be 0 and use it to count how many steps it takes for the starting number to become 1. (Hint: You need to add 1 to the counter every time the while loop restarts using time = time + 1 in the while loop).
- Print the number of steps the beginning number took to reach 1 with an appropriate statement.
- (Bonus)
- You should check that the number entered is positive. If it isn't print a message saying that the number entered is not positive and prompt the user for another starting number. This should keep happening until the number entered is positive.

Exercise 14: Battle Ship