Homework (40pts) chapter 7 _ text book (Lent), Numerical Integration

write the program and the outputs.

Q1- (5pts) ……. 7.8

Q2- (5pts) ……. 7.13

Q3

- (20 pts) For the following integral: $\int_{-2}^{4}(1 - x - 4x^3 + 2x^5)dx$

plot the function for the given range using MATLAB. Properly annotate your plot,

a- Write a function called TrapiziodRule (you will have to set up vectors for x and y in the command window based on the value of n and have x, y and n as inputs to the function: the estimate will be your output) and use it to estimate the value of the integral for n=1, n=2, and n=4. For each case, find the error using the built-in function "trapz".

b- Write a function called SimpsonRule (3/8 rule) and implement the integration above in the interval [-2 , 4]. You will have to set up vectors for x and y in the command window based on the value of n and have x, y and n as inputs to the function: the estimate will be your output). Compare the error with the continuous integration value.

Q4 (10 pts) Determine the distance traveled from the velocity data using Trapezoidal and Simpson rules.

Time	1	2	3.25	4.5	6	7	8	8.5	9.3	10
Velocity	5	6	5.5	7	8.5	8	6	7	7	5

Use MATLAB to fit the data above with a cubic equation using polynomial regression. Integrate the cubic equation to determine the distance (use symbolic toolbox or any other method to find the built in function)

2.
```
function Tf=c2f(Tc)
% returns temperature in
  Fahrenheit given
% temperature in Celsius
```

3.
```
function rv=reverse(v)
% reverses vector or string
% example:
% >>reverse([4 5 6])
% >> 6 5 4
```

4.
```
function [x, y]=pol2rect(r, theta);
% given circular coordinates
  r and theta
% returns rectangular
  coordinates x and y
```

5.
```
function vout=shuffle(vin)
% returns vector in random order
% (uses Matlab function randperm)
```

6. Write a program named `test1.m` that uses the function `fymby`, from Problem 1, and a `for` loop to produce a tabular output of your age in the years from 2009 to 2050. Output should look like this:

```
%
...
In 2015 I will turn 22.
In 2016 I will turn 23.
In 2017 I will turn 24.
In 2018 I will turn 25.
In 2019 I will turn 26.
In 2020 I will turn 27.
...
```

7. Write a program named `test2.m` that uses the function `c2f`, from Problem 2, and a `for` loop to produce a tabular output of temperatures in Celsius and Fahrenheit for T_c from 32° to 44° in steps of 2°Celsius. Output should look like this:

```
T(C)      T(F)
0         32
2         35.6
4         39.2
6         42.8
8         46.4
...
```

(Advanced) Try using the MATLAB function `sprintf` instead of `num2str` to make the tabular output look better.

8. Write and test a MATLAB function for simulating a roll of two dice:

```
function [die1, die2, resultstr]=rolldice;
% function [dice1,dice2,resultstr]=rolldice;
%   simulates roll of two fair dice
%  returns
%    die1   an integer in the range [1,6]
%    die2   an integer in the range [1,6]
%    resultstr   string with result (sumd=die1+die2)
%              'snake eyes'  sumd=2
%              'ace-deuce'   sumd=3
%              'yo'          sumd=11
%              'boxcars'     sumd=12
%              'natural'     sumd=7
%              'hard six'    3 3
%              'hard four'   2 2
```

This function should not write anything to the screen. Now write a MATLAB program `playdice` that does the following:

a. Set the number of throws, `N`, and a logical variable `printresults`.

b. Initialize the random number generator using `rng('shuffle')`.

c. (Optional, Advanced) Initialize an array `roll sums=zeros(1,12)` that will hold the number of times each possible outcome (from 2 to 12) occurs.

d. In the "calculate games" section, play `N` throws of the dice using a `for` loop. Play each throw using the `rolldice` function. Print out the results (one roll per line) like this:

```
..
Player rolls a 2 and a 5 : natural
Player rolls a 6 and a 2 :
Player rolls a 2 and a 1 : ace-deuce
Player rolls a 1 and a 4 :
Player rolls a 2 and a 6 :
Player rolls a 2 and a 2 : hard four
Player rolls a 6 and a 3 :
Player rolls a 3 and a 2 :
Player rolls a 3 and a 3 : hard six
Player rolls a 4 and a 1 :
Player rolls a 1 and a 3 :
Player rolls a 6 and a 6 : boxcars
...
```

e. (Optional, Advanced) Turn off the printout (`printresults=false`) and play `N=1e5` times to collect statistics on how many times each result (from 2 to 12) occurs in `rollsums(2:12)`. At the end, make a bar chart of the percentages for each result using the `bar` command.

9. Solve the *birthday problem* by Monte Carlo simulation. The question to be answered is: given a group of N_{people} people, what is the probability that at least two of them will have birthdays on the same day? The goal of this program, `birthday.m` is to calculate this probability for N_{people} in the range [2, `nPeopleMax`].

   Break the problem up into three programs: the main program `birthday.m`, a function `nRowsWith Match(M)`, and another function `vectorHas Match(v)`.

   The function `hasMatch=vectorHasMatch(v)` takes a vector argument and returns a value of true if the vector has as least one pair of identical elements. (Hint: Use the MATLAB function `sort` to first sort the vector, and then search it for *adjoining* elements that are identical.)

   The function `m=nRowsWithMatch(M)` takes a matrix as an argument and returns an integer that is the number of rows of the matrix with at least one pair of identical elements. (Hint: Use the MATLAB function `size` to get the number of rows and columns of the matrix, then loop through each row and call `vectorHasMatch` to determine if that row has a match.)

   The main program `birthday` should use `randi` to set up an `Nsets` × `Npeople` array of integers between 1 and 365. The number of sets (once the program is debugged) should be larger than 5000. For each such array, find `nM`, the number of rows with at least one match. The probability of a birthday match is then just `nM/Nsets`. Tabulate that probability for the various values of `Npeople` in the given range and plot the results. Try plotting using the `bar` command.

10. **Ticker text.** Write a function `tickerText(s,iwidth, dt)` that displays the string `s` scrolling through a text window with width `iwidth`. The parameter `dt` gives the length of the pause after piece of text is displayed. For example, the following would be displayed scrolling across the Command window (here shown at each snapshot in time).

```
>> tickerText('Sic transit gloria mundi!', 8, 0.1)

               S
              Si
             Sic
            Sic 
           Sic t
          Sic tr
         Sic tra
        Sic tran
        ic trans
        c transi
         transit
        transit 
        ransit g
        ansit gl
        nsit glo
        sit glor
        it glori
```

```
        t gloria
         gloria
        gloria m
        loria mu
        oria mun
        ria mund
        ia mundi
        a mundi!
         mundi!
        mundi!
        undi!
        ndi!
        di!
        i!
        !
```

11. Rotation cypher. Characters are represented internally in MATLAB by the standard integer ASCII code. The `char` and `double` functions will convert back and forth between the alphanumeric character and its associated ASCII code.

```
>> v=double('Hello!')
v =
    72  101  108  108  111  33
>> s=char(v)
s =
Hello!
```

The 72 corresponds to "H," 101 to "e," etc. Here we will use the 94 codes for printable ASCII characters from 32 (space) up to 125 ("]") to make a rotation cipher. Think of the numbers from 32 to 125 on a circle:

$$[\ldots 32, 33, 34, \ldots, 124, 125, 32, 33, \ldots] \tag{7.1}$$

The cipher consists of replacing each character with ASCII code N with the character corresponding to the code 47 steps to the right around the circle. Write a function `so=rot47(si)` that returns the input string encrypted in this way. Using the function again should decrypt the string.

```
>> s=rot47('Abort mission.')
               % plaintext
s =
p3@CEO>:DD:@?] % cyphertext
>> rot47(s)
ans =
Abort mission.
               % decoded cyphertext
```

12. **Josephus problem.** The Josephus problem takes its name from a historical incident in the Jewish revolt against the Romans. Flavius Josephus was a Jewish military leader who, with his men, became trapped in a cave by the Romans. Rather than surrender, they were determined to kill themselves and Josephus suggested the following technique. They formed a circle and starting with the first man, selected the third man to be killed by his neighbor. The next two live men were skipped and the third killed, and so on around the circle until only one man is left alive. This turned out to be Josephus. He decided to surrender to and aid the Roman forces, and subsequently went on to write the important book *The Jewish War*, chronicling the destruction of the Jewish state. The Josephus problem is motivated by the suspicion that Josephus could have done the math and selected just the right initial position in the circle to assure his survival.

    a. Consider the problem of N people arranged in a circle, initially all in the "live" state. Moving around the circle set the state of the kth live person to "dead." Continue until one person, whose initial position was j, survives. Write a MATLAB function `j=Josephus(N, k)`, which returns the survivable position. [Check: `Josephus(10, 2)` should return 5].

    b. Extend the function to return an $N \times N$ array M of ones and zeros, representing alive and dead, encoding the history turn by turn. The first row of M should be all ones, the last row has a single one in the jth column. The syntax should be `[j, M]=Josephus(N, k)`. Write a program that calls the function to get M and devise a graphical display of the deadly history.

13. **Clipping a vector.** Write a function `vout=clipVec(v, vmin, vmax)` that copies the input vector `v` to the output vector `vout`, except if a value is greater than `vmax` or less than `vmin`. Values greater than `vmax` are set to `vmax` and values less than `vmin` are set to `vmin`.

    Check by running this test, which should plot a sine wave clipped level at the tops and bottoms.

```
x=linspace(0, 3, 300);
y=sin(2*pi*x);
plot(x,clipVec(y, -0.9, 0.9));
axis([0, 3, -1, 1]);
```

14. **Swap.** Write a function that returns the values of the two inputs in reversed order: `function [a,b]=swap(x, y)`. To use it to swap the values of two variables simply write `[v1,v2]=swap(v1,v2)`.

15. **Check if sorted.**

    a. Write a function `tf=isSortedAscending(v)` that returns true if each element of the vector `v` is greater than or equal to the preceding element.

    b. Write a function `tf=isSortedDescending(v)` that returns true if each elements of the vector `v` is less than or equal to the preceding element.

    c. Write a function `tf=isSorted(v)` that returns true if `v` is sorted in either ascending or descending order.

16. **Boggle sort.** Write a function `vout=boggleSort(v)` that uses the following terrible algorithm: repeatedly rearrange the elements of `v` randomly until they are sorted. Use the `isSortedAscending` function you've already written, and the MATLAB function

`randperm(n)`, which will generate a vector with a random permutation of the first *n* integers. The expression `v=v(randperm(length(v))` randomly rearranges `v`. Test by sorting nine random integers. (Only try this method on short lists—it's terribly slow.)

```
bogglesort(randi(100, 1, 9))
```

To boggle sort a deck of cards, one would first check to see if the cards are already sorted, then repeatedly throw the cards into the air, randomly pick them back up, and see if they're sorted yet.

17. **Guess sort.** Write a function `vout=guessSort (v)` that sorts the input vector in ascending order by the following algorithm. Randomly pick two elements of the vector, and if the left element (lower index) is greater than the right element, swap the two elements. Repeat until the vector is sorted. This method is better than a boggle sort, but is still very slow.

18. **Bubble sort.** A respectable sorting method is the bubble sort. Examine in turn each element of the vector and it's neighbor to the right. If they're in the wrong order, swap them. Continue going repeatedly through the vector in this way until the whole vector has been traversed once with no swaps.

    a. Write a function `vout=bubbleSort(v)` that sorts a vector by this method.

    b. Write a function `vout=bubbleSortVisualized(v)` that creates a visual representation of the bubble sort in progress. The visualization doesn't need to work for really long vectors.

19. **Vector utilities.**

    a. Write a function `vout=vInsertAfter(v, x, k)` that inserts the value `x` into the vector `v` after the `k`*th* entry.

    b. Write a function `vout=vDelete(v, k)` that deletes the `k`*th* entry of the vector `v`, returning a shorter vector.

20. **Insertion sort.** An insertion sort corresponds to the way people often sort a hand of cards. Given a vector *v* to sort, make a new vector *vout*. Copy each element of *v* into the appropriate position in *vout*. To determine the appropriate position, examine in order each possible insertion point in *vout*. Write a function `vout=insertionSort(v)` that implements this algorithm.

21. **Rotate vector.** Write a function `sout=vrotate(v,k)` that returns a vector (or string) circularly rotated by `k` spaces. If `k` is positive, rotate forward; if `k` is negative rotate backward.

```
>> vrotate('washington', 3)
ans =
tonwashing
>> vrotate('washington', -3)
ans =
hingtonwas
```