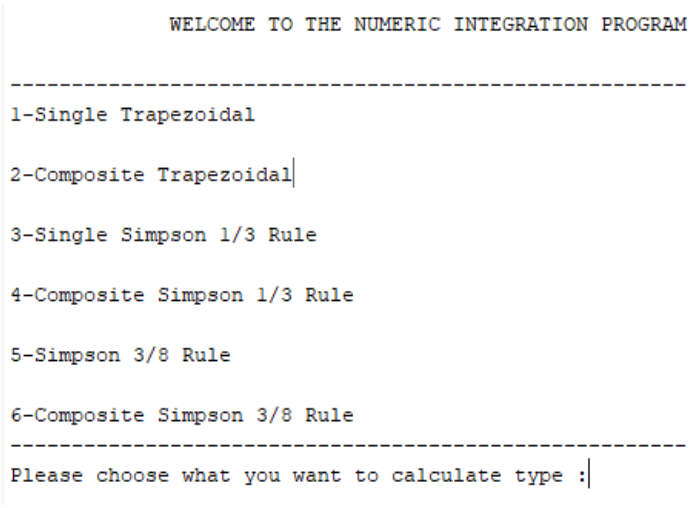
Tevfik ERKİN 16030021014

HOMEWORK

In numerical analysis, numerical integration constitutes a broad family of algorithms for calculating the numerical value of a definite integral, and by extension, the term is also sometimes used to describe the numerical solution of differential equations. The basic problem considered by numerical integration is to compute an approximate solution to a definite integral. It is different from analytical integration in two ways: first it is an approximation and will not yield an exact answer; Error analysis is a very important aspect in numerical integration. Second it does not produce an elementary function with which to determine the area given any arbitrary bounds; it only produces a numerical value representing an approximation of area.

To obtain a result, we have some type of numerical integration methods. These are:

- Single Application Of The Trapezoidal Rule
- Composite Trapezoidal Rule
- Single Application Of Simpson's 1/3 Rule
- Composite Simpson's 1/3 Rule
- Simpson's 3/8 Rule



These would be very complicated equations so that we can not solve those by hand, instead of that we can code and computer can solve for us.

My code starts with some assignments and wants to value from the user. But, I assigned specific values for this homework and this functions not working.

Only you must enter which calculation that you want and it starts to calculate.

```
cont='y';
while cont=='y';
    clc;
    clear all;
    fprintf('\t\t\t WELCOME TO THE NUMERIC INTEGRATION PROGRAM \n\n');
    fprintf('-------------------------------------------------------\n');
    fprintf('1-Single Trapezoidal\n\n2-Composite Trapezoidal\n\n3-Single Simpson 1/3 I
    fprintf('-------------------------------------------------------\n');
    wish=input('Please choose what you want to calculate type :');
    a=0; %input('Please enter the integral lower value: ');%alt limit
    b=4; %input('Please enter the integral upper value: ');%üst limit
    fun=@(x)(1 -exp(-14.*x));%fonksiyonu tanımladım
    %      fun=@(x)(0.2 + 25.*x - 200*x.^2 + 675.*x.^3 - 900.*x.^4 + 400.*x.^5);
    %      fun=@(x)(1 -exp(-2.*x));
    result = integral(fun,a,b);%sonucu bir değere atadım
```

Firstly, I determined the upper value and lower value of integral. After that, I made the function what I want to calculate. The final process for beginning, I obtained the real result of integral to determine the error of rules.

```
if wish==2 || wish==4 ||wish==6
    n=input('Please enter a value ''n'': ');
end
%% Composite Trapezoidal Rule
switch wish
    case 2
        choosing='Composite Trapezoidal';
        al=a;
        sum=0;%fonksiyonların aldıkları değerlerin toplamı

        h=(b-a)/n;%aralıkların kaç birim olması gerekiyor onu hesaplıyoruz

        for j=1:n+1
            if j==1 || j==n+1
                sum=fun(al)+sum;
            else
                sum=2.*fun(al)+sum;
            end
            al=al+h;%hangi x değerleri aldığını göstermek için bu matrise kaydediyorum.
```

The first screen that the user sees firstly, the user must indicate the process number. Namely, if the user pushes the number 1 button, that comes to mean he chose **‘ SINGLE TRAPEZOIDAL RULE ’** and computer makes calculation relative this choosing. So that the most important section is that section to obtain the true result. However, there is a problem that if the user chooses one of **‘COMPOSITE APPLICATIONS’**, the computer needs one more value for calculation. So, if the user chose any composite applications he must enter one more value as **‘n’**. In the background of the program, I built that condition with **‘if’ command**. Because if the user wants to make the calculation with **‘SINGLE APPLICATIONS’** this value **‘n’** is unnecessary and it **decelerates the program**. That’s why I built this section.

```
%% Composite Trapezoidal Rule
switch wish
    case 2
        choosing='Composite Trapezoidal';
        a1=a;
        sum=0;%fonksiyonların aldıkları değerlerin toplamı

        h=(b-a)/n;%aralıkların kaç birim olması gerekiyor onu hesaplıyoruz

        for j=1:n+1
            if j==1 || j==n+1
                sum=fun(a1)+sum;
            else
                sum=2.*fun(a1)+sum;
            end
            a1=a1+h;%hangi x değerleri aldığını göstermek için bu matrise kaydediyorum.
        end
        res=(h/2)*(sum);
        error=abs(res-result);
        perc_error=(error./result)*100;
        j=1;i=1;
```

To make faster the program, I used **'switch-case'** struct. Every rules/process assigned a number value that range is **one** to **six**. After, I calculated the result by the numerical method that the user choice and assigned a variable that. Secondly, the program determined the **'relative error'** and **'percent relative error'**.

```
%% question 2
multi_matris=[-8 -8 -6  4;
    -4 -3  1  7;
    -2 -1  4  10];
upper_x=12;
lower_x=0;
upper_y=4;
lower_y=0;
del_x=(upper_x-lower_x);
leng_x=size(multi_matris,2)-1; % n değerini bulmak için kullandım.
leng_y=size(multi_matris,1)-1; % tüm sonuçlar elde edildikten sonraki y
%deperindeki n değerini bulmak için kullandım
sum_multi_1=0;
for p=1:size(multi_matris,1)
    sum_multi_1=0;
    for o=1:size(multi_matris,2)
        multi_matris(p,o)
        if o==1 || o==size(multi_matris,2)
            sum_multi_1=multi_matris(p,o)+sum_multi_1;
        else
            sum_multi_1=2.*multi_matris(p,o)+sum_multi_1;
```

The homework contains two questions. So that I built a code for two of them. Firstly, question-1 is calculated by the program after question-2 is calculated.

I assigned an array for question-2 table values and indicated the level of the table for every x-axis and y-axis. At the same time, assigned some specific values to make a calculation.

```
for p=1:size(multi_matris,1)
    sum_multi_1=0;
    for o=1:size(multi_matris,2)
        multi_matris(p,o)
        if o==1 || o==size(multi_matris,2)
            sum_multi_1=multi_matris(p,o)+sum_multi_1;
        else
            sum_multi_1=2.*multi_matris(p,o)+sum_multi_1;
        end
    end
    sum_3(p)=((upper_x-lower_x)./(leng_x.*2)).*sum_multi_1;
    %     res=(del_x./4).*sum_2;
end

result_multi = ((upper_y-lower_y)./(leng_y.*2)).*(sum_3(1)+2.*sum_3(2)+sum_3(3))
result_multi = result_multi./((upper_x-lower_x).*(upper_y-lower_y));
```

I used nested for loops and landed up with **‘trapezoidal rule application’**.

```
fprintf('**************--------RESULTS Of QUESTION-1--------**************\n\n');
fprintf('Your Choosing: %s\nError: %1.6f\nPercent Relative Error: %%%1.6f\nReal Result: %1.6f\nNumerical
fprintf('\n**************--------RESULTS Of QUESTION-2--------**************\n\n');
tab=array2table(multi_matris,'VariableNames',{'Column_1','Column_2','Column_3','Column_4'});
disp(tab);
fprintf('Avarage Value of Table: %1.6f\n\n',result_multi);
fprintf('//////////////////////---END OF CALCULATIONS--////////////////////// \n\n');
cont=input('Do you want to make a new calculation ? Y/N: ','s');
cont=lower(cont);
if cont=='n'
    clc;
    disp('Thanks for using program, bye bye...');
end
```

I used **'fprintf'** command to display the results. Finally, the user must value that whether he wants to make a new calculation without exit program and enter a string value and the user can **capital or lower case** to **avoid the error** relative this value in the program I used **'lower' command**.

```
**************--------RESULTS Of QUESTION-1--------**************

Your Choosing: Single Trapezoidal
Error: 1.928571
Percent Relative Error: %49.090909
Real Result: 3.928571
Numerical Result: 2.000000


**************--------RESULTS Of QUESTION-2--------**************

    Column_1    Column_2    Column_3    Column_4
    ________    ________    ________    ________

    -8          -8          -6           4
    -4          -3           1           7
    -2          -1           4          10

Avarage Value of Table: -0.833333

----------------------END OF CALCULATIONS------------------------

Do you want to make a new calculation ? Y/N:
```

Screenshot from program. I show off the matrix as a table for user.

Referances:

http://www.csun.edu/~ajp42955/Math382Paper.html

Numerical Methods for Engineers. SEVENTH EDITION. Steven C. **Chapra**

https://en.wikipedia.org/wiki/Simpson%27s_rule