Math 340, Lab 1, SPRING 2016

DUE Tuesday 01-26-16

This worksheet serves as a basic introduction to some of the core concepts that we will be using throughout the semester. Example programs will be given, and the exercises are very similar to the examples.

1. Flow control statements: the "for" loop and the "if-else" conditional statement

Consider the task of adding up the first n=100 integer numbers: $S = \sum_{k=1}^{n} k$, where *n*=100. The following code accomplishes this task:

```
n = 100;                % How far we're summing up to
S = 0;                  % The sum variable S starts out at 0
for k  = 1 : n          % Loop over values of k from 1 to n
     S = S + k;         % Adds integer "k" to sum "S" at each step...
     disp( ['k=', num2str(k),  ':  partial sum equals ', num2str(S) ]);  % Display the result
end
```

Note that the command "num2str" converts a numerical value to a string, in order to combine it with other text strings inside the "disp" statement, to display a carefully formatted output (type "help num2str" or "help disp" to find more information about these commands).

Now consider a more complex sum, for instance$S = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \ldots - \frac{1}{100} = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k}$.This sum represent a Taylor approximation for a value of an elementary function (which one?). One possible implementation of this summation involves the use of conditional command "**if**" and the division remainder (modulus) operation "mod" (to test whether the integer is odd or even):

```
n=100;                   % How far we're summing up to
S=0;                     % The sum variable S starts out at 0
for k = 1 : n            % Loop over values of k from 1 to n
       if mod(k,2) == 0  % if k is even, the next line is executed
         S = S – 1 / k;
       else              % otherwise, k is odd, the following line is executed:
         S = S + 1 / k;
      end                % end of "if" statement
      disp( ['k=', num2str(k),  ':  partial sum equals ', num2str(S) ]);  % Show the result
end                      % end of loop
```

The "if" statement ensures that 1/k is added to S if k is odd, but is subtracted from S if k is even. This can be also accomplished in several different other ways, without the "if..else" statement (How?)

Exercise 1: Write a MATLAB program calculating S= $\sum_{k=0}^{100} \frac{(-1)^k}{k!}$

Hint: MATLAB has a built-in function to calcuate the factorial of any number k, it is simply factorial(n)

2. Controlling accuracy; the "while" loop

Let's again consider the sum $S = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k}$, but instead of adding a certain fixed number of terms, let us specify the accuracy (error tolerance) for this infinite series calculation, and add as many elements as needed to satisfy this accuracy. For the sake of simplicity let's think of the size of the next term as a rough estimate of the error tolerance (as we saw in class, this often agrees with the bound on the remainder of the Taylor sum). This time it is more convenient to use the **while** loop instead of the **for** loop:

```
tolerance = 1e-4;      % Set the error tolerance to 10^(-4)
S = 0;                  % The sum variable S starts out at 0
term  = 1;             % Set to high enough value (this initial value won't be used)
k = 1;                  % first term corresponds to k=1
sgn = 1;                % Figure out the meaning of this variable on your own
while abs(term)  >  tolerance   % repeat until the next term is less than the error tolerance
        term = sgn / k;   % calculate the next (k-th) term
        S = S + term;      % add k-th term to the sum variable S
        disp( ['k=', num2str(k),  ':  partial sum equals ', num2str(S) ]);  % Show the result
        k = k + 1;            % add k-th term to the sum
        sgn = –sgn;
 end
```

Note that we got rid of the conditional "if-else" statement, by introducing an extra variable called "sgn" . What is the purpose of this variable in this code?

Exercise 2: Write a MATLAB program calculating $S = \sum_{k=0}^{\infty} \frac{(-1)^k}{k!}$ with an error tolerance of 10^{-6}

3. Making plots

We are going to be plotting our results throughout the semester. MATLAB plots functions the way you would do manually: it puts a set of (x, y) points and connects them with a line. Therefore, If you wanted to plot x^2 on the interval (−1, 1), you would do the following:

```
x = –1:0.01:1;   % Create a set of x values between -1 and 1, with a step of 0.01
y=x.^2;           % The period in ".^" ensures that the operation is performed on each element
plot(x,y);        % Plot y against x
xlabel('x');      % label the plot
ylabel('y=x^2');
legend('Plot of f(x)=x^2');
title('Plot of function f(x) = x^2');
```

Note that there is another method of creating a set of values (a.k.a. an array, or a vector) between two given bounds; you could replace the first line in above with the following command:

```
x = linspace(-1, 1, 200);   % Create a set (an array, a vector) of 200 values between -1 and 1
```

Exercise 3: plot the function $f(x) = \sin(x)/(x^2+1)$ on the interval [–π; π]

Exercise 4: Calculate $f(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^k}{k!}$ with an error tolerance of 10^{-6}, and plot the values of the function for a set of 80 values of *x* on the interval [–1; 1]

In the last exercise, you will need an extra outside loop to calculate the sum for each value of x:

```
N = 80; % Number of points in the plot

X = linspace(-1, 1, N); % Create a set of N values between -1 and 1

Y = zeros(N, 1); % Create an array of corresponding y-values, and initialize them to zero

for k = 1 : N % Main part of the program: calculate the sum for each of the N x-values

x = X(k); % Assign the next x-value to the variable x

% put here all commands needed to calculate the sum “S” for each value of x

Y(k) = S; %; store the result in the appropriate element of the y-array

end

plot(X, Y, ‘r-‘); % plot; don’t forget to add labels and title
```