1 LAB3 (SERIES & SYMBOLIC EXPRESSIONS)

The focus of this course is on numerical computations, i.e. calculations, usually approximations, with floating point numbers. However, MATLAB can also do symbolic computations, which means exact calculations using symbols as in Algebra or Calculus.

Note: To do symbolic computations in MATLAB one must have the Symbolic Toolbox.

1.1 DEFINING FUNCTIONS AND BASIC OPERATIONS

Before doing any symbolic computation, one must declare the variables used to be symbolic:

```
>> syms x y
```

A function is defined by simply typing the formula

```
>> f = cos(x) + 3 * x^2
```

Note the co-efficient must be multiplied using *. To find specific values, you must use the command subs:

```
>> subs(f, pi)
```

This command stands for substitute, it substitutes π for x in the formula for f. If we define another function:

```
>> g = exp(-y^2)
```

Then we can compose the functions:

```
>> h = compose(g, f)
```

i.e. h(x) = g(f(x)). Since, f and g are functions of different variables, their product must be a function of two variables:

```
>> k = f * g
>>subs(k, [x, y], [0, 1])
```

We can do simple calculus operations, like differentiation:

```
>> f1 = diff(f)
>> k1x = diff(k, x)
```

and indefinite integrals (Antiderivatives):

```
>> F = int(f)
```

and definite integrals:

```
>> int(f, 0, 2 * pi)
```

To change a symbolic answer into a numerical answer, use the **double** command which stands for double precision, (not times 2):

```
>> double(ans)
```

Note, that some antiderivatives cannot be found in terms of elementary functions, for some of these the antiderivative can be expressed in terms of special functions:

```
>> G = int(g)
```

and for others MATLAB does the best it can:

```
>> int(h)
```

For definite integrals, that cannot be evaluated exactly, MATLAB does nothing and prints a warning:

```
>> int(h, 0, 1)
```

We will see later that even functions that don't have an antiderivative can be integrated numerically. You can change the last answer to a numerical answer using:

```
>> double(ans)
```

Plotting a symbolic function can be done as follows:

```
>> ezplot(f)
```

or the domain can be specified:

```
>> ezplot(g, -10, 10)
```

```
>> ezplot(g, -2, 2)
```

To plot a symbolic function of two variables use:

```
>> ezsurf(k)
```

It is important to keep in mind that even though we have defined our variables to be symbolic variables, plotting can only plot a finite set of points. For instance:

```
>> ezplot(cos(x^5))
```

will produce the plot in Figure 7.1, which is clearly wrong, because it does not plot enough points.

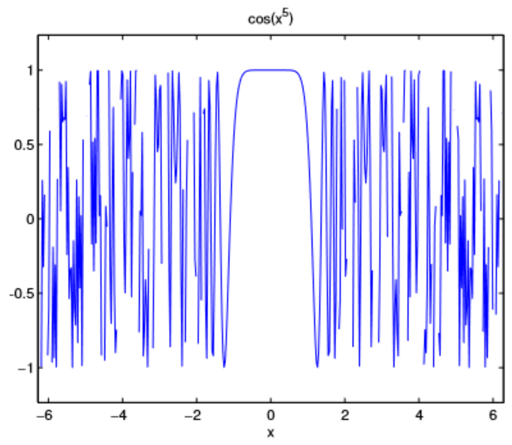

Figure 1: Graph of cos(x^5) produced by ezplot command. It is wrong because cos u should oscillate smoothly between -1 and 1. The problem with the plot is that cos(x^5) oscillates extremely rapidly, and the plot did not consider enough points.

MATLAB allows you to do simple algebra. For instance:

```
>> poly = (x – 3)^5
```

```
>> polyex = expand(poly)
```

```
>> polysi = simplify(polyex)
```

To find the symbolic solutions of an equation, f(x) = 0, use:

```
>> solve(f)
```

```
>> solve(g)
>> solve(polyex)
```

Another useful property of symbolic functions is that you can substitute numerical vectors for the variables:

```
>> X = 2:0.1:4;

>> Y = subs(polyex, X);

>> plot(X, Y)
```

1.2 Problems (To Be Completed in Lab)

1.Starting from **mynewton** write a well-commented function program mysymnewton that takes as its input a symbolic function f and the ordinary variables x0 and n. Tolerance should be less than or equal to 10^-4. Let the program take the symbolic derivative f0, and then use subs to proceed with Newton's method. Test it on f(x) = (x^3) −4 starting with x0 = 2. Turn in the program and a summary of the results.

2.Create two multi-variable functions using symbolic expressions in MATLAB.

```
>> z = x^2 + y^2;

>> f = 2*(x-1) + 2*(y-1) + 2;
```

Plot them together. For showing second figure on same figure (figure1) use **hold on** command, as explained below.

```
% Plot function 1 ****

>>hold on

% Plot function 2 ****
```

Second function has more range (y) than function 1. So, to find area between the two functions, make sure to shift f by some value. Your result might be close enough to figure 2. What is your observation?

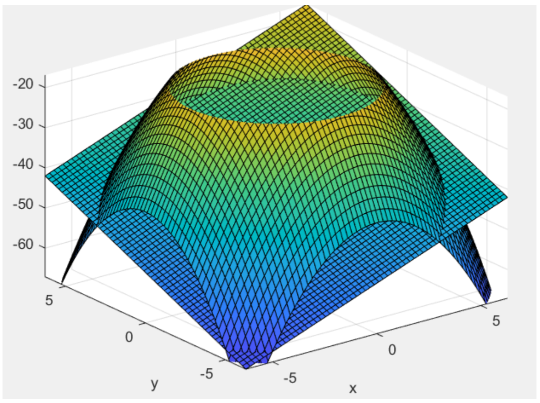

Figure 2: Plot of z and f together

3.Consider following Taylor series

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots \qquad (1)$$

and its partial sum

$$S_N = \sum_{n=0}^{N} x^n \qquad (2)$$

When x ≠ 1, following equation holds

$$S_N = \frac{1 - x^{N+!}}{1 - x} \qquad (3)$$

Observe what happens when abs(x) > 1.

Write MATLAB code for the calculation of approximated value of equation 1 (series expansion) and print actual value of the same equation 1, obtained from f(x). Also, plot two graphs in one figure containing error and Taylor approximation at each iteration. You can create 1x5 array (all zeros) using following instruction.

```
>> j = zeros(1, 5);
```

Following snippet will also work for array with elements of your choice (Currently, 1 to 10).

```
for i = 1: 10
        arr(i) = i;
end
```

1.3 Home Assignment

1. Find a complicated function in an engineering or science textbook or website. Make a well-commented script program that defines a symbolic version of this function and takes its derivative and indefinite integral symbolically. In the comments of the script describe what the function is and properly reference where you got it. Turn in your script and its output (the formula for the derivative and indefinite integral.)

2. Write a MATLAB program, which computes partial sum $S_N\ for\ x = -20$, for equation 4, using various values of *N*. How do you explain that answer is never close to e^(-25), regardless of how you chose *N*?

$$e^x = \sum_{k=0}^{\infty} \frac{x^n}{n!} \qquad (4)$$

3.Write a MATLAB function which calculates maximum and minimum value of functions at certain interval, using extreme value theorem. Test it with following functions.

(a) *f(x)* = sin *x* + cos *x* on [0, 2π].

(b) *f(x)* = $x^4 - 3x^3 - 1$ on [−2,2]

Following function definition will hold for this question.

[min, max] = **calculateExtremeValues**(f, interval)

% where *f* is a function and *interval* is a vector with two values [start end]

4. Write a paragraph on “Numerical analysis in computer science and engineering” in comments. Make sure you do not copy it as it will result in a 0.