Lecture 7
Symbolic Computations

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. You should have done some symbolic MATLAB computations in your Calculus courses and in this chapter we review what you should already know.

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 that coefficients 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)
and definite integrals (antiderivatives):
> F = int(f)
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 it 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.

Other useful symbolic operations

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:
\[ X = 2:0.1:4; \]
\[ Y = \text{subs}(\text{polyex}, X); \]
\[ \text{plot}(X, Y) \]

**Exercises**

7.1 Starting from `mynewton` write a well-commented function program `mysymnewton` that takes as its input a symbolic function \( f \) and the ordinary variables \( x_0 \) and \( n \). Let the program take the symbolic derivative \( f' \), and then use `subs` to proceed with Newton’s method. Test it on \( f(x) = x^3 - 4 \) starting with \( x_0 = 2 \). Turn in the program and a brief summary of the results.
Review of Part I

Methods and Formulas

Solving equations numerically:

\( f(x) = 0 \) — an equation we wish to solve.
\( x^* \) — a true solution.
\( x_0 \) — starting approximation.
\( x_n \) — approximation after \( n \) steps.
\( e_n = x_n - x^* \) — error of \( n \)-th step.
\( r_n = y_n = f(x_n) \) — residual at step \( n \). Often \(|r_n|\) is sufficient.

Newton’s method:

\[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \]

Bisection method:

\( f(a) \) and \( f(b) \) must have different signs.
\( x = (a + b)/2 \)
Choose \( a = x \) or \( b = x \), depending on signs.
\( x^* \) is always inside \([a, b]\).
\( e < (b - a)/2 \), current maximum error.

Secant method:

\[ x_{i+1} = x_i - \frac{x_i - x_{i-1}}{y_i - y_{i-1}} y_i \]

Regula Falsi:

\[ x = b - \frac{b - a}{f(b) - f(a)} f(b) \]

Choose \( a = x \) or \( b = x \), depending on signs.

Convergence:

Bisection is very slow.
Newton is very fast.
Secant methods are intermediate in speed.
Bisection and Regula Falsi never fail to converge.
Newton and Secant can fail if \( x_0 \) is not close to \( x^* \).
Locating roots:

Use knowledge of the problem to begin with a reasonable domain. Systematically search for sign changes of $f(x)$. Choose $x_0$ between sign changes using bisection or secant.

Usage:

For Newton’s method one must have formulas for $f(x)$ and $f'(x)$. Secant methods are better for experiments and simulations.

Matlab

Commands:

- $v = [0 \ 1 \ 2 \ 3]$ ................................................................. Make a row vector.
- $u = [0; \ 1; \ 2; \ 3]$ ................................................................. Make a column vector.
- $w = v'$ ................................................................. Transpose: row vector $\leftrightarrow$ column vector.
- $x = \text{linspace}(0,1,11)$ ....................................................... Make an evenly spaced vector of length 11.
- $x = -1:.1:1$ ....................................................... Make an evenly spaced vector, with increments 0.1.
- $y = x.^2$ ................................................................. Square all entries.
- $plot(x,y)$ ....................................................... plot $y$ vs. $x$.
- $f = \text{inline}('2*x.^2 - 3*x + 1','x')$ ....................................................... Make a function.
- $y = f(x)$ ................................................................. A function can act on a vector.
- $plot(x,y,'*','red')$ ....................................................... A plot with options.
- Control-c ....................................................... Stops a computation.

Program structures:

for ... end example:

```
for i=1:20
    S = S + i;
end
```

if ... end example:

```
if y == 0
    disp('An exact solution has been found')
end
```

while ... end example:

```
while i <= 20
    S = S + i;
    i = i + 1;
end
```

if ... else ... end example:

```
if c*y>0
    a = x;
else
```

```
Function Programs:

- Begin with the word `function`.
- There are inputs and outputs.
- The outputs, name of the function and the inputs must appear in the first line.
  i.e. `function x = mynewton(f,x0,n)`
- The body of the program must assign values to the outputs.
- Internal variables are not visible outside the function.

Script Programs:

- There are no inputs and outputs.
- A script program may use and change variables in the current workspace.

Symbolic:

```plaintext
> syms x y
> f = 2*x^2 - sqrt(3*x)
> subs(f,sym(pi))
> double(ans)
> g = log(abs(y))                                  MATLAB uses `log` for natural logarithm.
> h(x) = compose(g,f)
> k(x,y) = f*g
> ezplot(f)
> ezplot(g,-10,10)
> ezsurf(k)
> f1 = diff(f,'x')                                 \text{indefinite integral (antiderivative)
> F = int(f,'x')                                   \text{definite integral}
> int(f,0,2*pi)
> poly = x*(x - 3)*(x-2)*(x-1)*(x+1)
> polyex = expand(poly)
> polysi = simple(polyex)
> solve(f)
> solve(g)
> solve(polyex)
```
Part II
Linear Algebra