MA2501: Numerical Methods
Spring 2017

Solutions to exercise set 0

Norwegian University of Science
and Technology
Department of Mathematics

This set of exercises was meant to give a short introduction into the usage of Matlab.

1 Linear algebra and plotting:

The solution of the linear system is $(a_0, a_1, a_2, a_3) = (-3, 3/2, -1/2, 1)$ and thus

$$p(x) = x^3 - \frac{1}{2}x^2 + \frac{3}{2}x - 3.$$

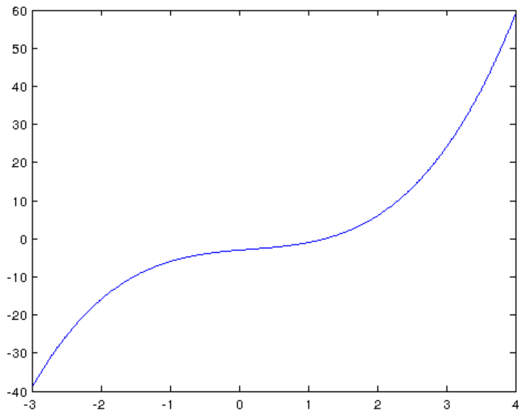
It can be obtained in Matlab with:

`A = [1,-2,4,-8;1,0,0,0;1,1,1,1;1,3,9,27];`	define the matrix
`b = [-16;-3;-1;24];`	define the vector
`a = A\b`	solve the equation, store it as the variable `a`, and show it

Note that it is important to keep track of the correct dimensions: The variable `b` above is a 4×1 vector. Also note that the semicolon (`;`) at the end of a line surpresses the visual output of the result of a calculation.

The function p can (in the possibly interesting interval $[-3, 4]$) be plotted with:

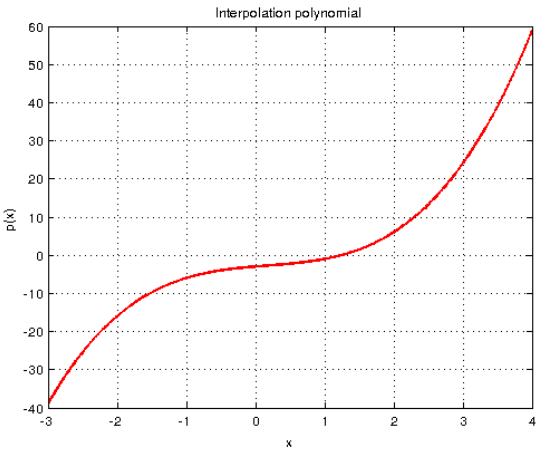
`x = [-3:0.01:4];`	discretise the interval $[-3, 4]$
`p = -3 + 1.5*t - 0.5*t.^2 + t.^3;`	evaluate the function at the discretisation points
`plot(t,p)`	a simple plot



Now it is possible to play around with the result a bit. For instance:

`plot(t,p,'Color','red','LineWidth','2');`	change color and line width
`xlabel('x');`	add a label to the x-axis
`ylabel('p(x)');`	add a label to the y-axis
`title('Interpolation polynomial');`	add a title
`grid on;`	add a grid

yields



2 Some simple programming:

a) A program for the first method can for instance be:

```
function a = myeuler1(n)
a = (1+1/n)^n;
```

A possibility for the (slightly more complicated) second method is:

```
function b = myeuler2(m)
c = 1;
b = 1;
for k = 1:m
  c = c/k;
  b = b + c;
end
```

A different possibility that takes advantage of the capabilities of MATLAB of working with vectors and the inbuilt function `factorial` is:

```
function b = myeuler3(m)
b = sum(1./factorial(0:m));
```

b) Testing the second program, we see[1] that the result does not change for $m \geq 17$ and in fact coincides with the result of the computation `exp(1)`.

In contrast, the first program requires a fairly large number n to yield a reasonable result. For $n = 100$, the error is about 10^{-2}, for $n = 10^4$, it is about 10^{-4}, finally, for $n = 10^8$ it is of the order of 10^{-8}. Increasing n further, however, tends to decrease the accuracy: If we choose $n = 10^{12}$, then the error increases to about 10^{-4}.

This behaviour can be explained by understanding that the total error of the program can be decomposed into two parts: first, the approximation error, which comes from the fact that the formula is only exact for "$n = \infty$", and, second, computational (i.e., rounding) errors, which come mainly from the fact that the division $1/n$ is, in general, inexact. Now note that the division $1/n$ can be performed exactly, if n is some power of 2. Indeed, choosing $n = 2^{40}$ (which is about the same as 10^{12}) yields an error of about 10^{-12}. Choosing $n = 2^{52}$, we basically obtain an exact result. If, however, we choose $n = 2^{53}$, then $1 + 1/n$ is indistinguishable from 1 in double precision. Thus the result of the algorithm for the input $n = 2^{53}$ is simply 1.

[1] Usually MATLAB only shows 5 significant digits. Using the command `format long`, one can increase this to 15 digits for double precision.