The purpose of this project is to develop and test a MATLAB function which can be used to interpolate a table of values \((x_i, f(x_i))\), where \(f : \mathbb{R} \to \mathbb{R}\) is a continuous function.

You should hand in the following:

1. Your solution of the questions below.
2. Your own implementation of the relevant functions.
3. Documentation describing the algorithm and the call sequence.

You should hand in your solution by noon on Monday, October 17th, 2011.

You are allowed to work together in groups of 2 or 3 people. One report per group will suffice, but remember to state the name of all group members. I expect very member of each group to understand all aspects of the code and the report!
Let $p : \mathbb{R} \to \mathbb{R}$ be the polynomial given by

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n.$$ 

Given an $x \in \mathbb{R}$, Horner’s rule can be used to compute $p(x)$, see Algorithm 1.

**Algorithm 1** Horner’s rule

**Require:** $x \in \mathbb{R}$, $\{a_j\}_{j=0}^n \subset \mathbb{R}$

**Ensure:** $h_n = p(x)$

1. $h_0 = a_n$
2. for $j = 1, 2, \ldots, n$ do
3. $h_j = a_{n-j} + h_{j-1} x$
4. end for

Let $q(d)$ be the statement

“Horner’s rule evaluates all real polynomials of degree $d - 1$ correctly”

and let $V \subseteq \mathbb{N}$ be the set given by

$$V = \{d \in \mathbb{N} : q(d) \text{ is true}\}$$

1. Show that $1 \in V$.
2. Show that if $d \in V$, then $d + 1 \in V$. (Hint: Any for-loop of length $d + 1$ can be broken into two parts, a for-loop of length $d$, followed by one or more statements.)
3. Show that $V = \mathbb{N}$.
4. Write a MATLAB function `horner1.m` which uses Horner’s rule to compute a given polynomial. Call sequence:

```matlab
>> y = horner1(a, x)
```

where $a$ is a vector containing the coefficients $\{a_j\}_{j=0}^n$, $x$ is a vector of arbitrary length and $y$ is a vector such that $y_i = p(x_i)$.

5. Extend Algorithm 1 to handle the more general case of

$$p(z) = \sum_{j=0}^{n} c_j \prod_{i=0}^{j-1} (z - x_j), \quad (1)$$

where the points $\{x_j\}_{j=0}^n$ are part of the input.

6. Write a MATLAB function `horner2.m` which uses your new algorithm to evaluate a polynomial given in the form specified by equation (1). Call sequence:
\[ y = \text{horner2}(c, x, z) \]

where \( c \) is a vector containing the coefficients \( \{c_j\}_{j=0}^{n} \), \( x \) is a vector containing the points \( \{x_j\}_{j=0}^{n} \), \( z \) is a vector of arbitrary length, and \( y \) is a vector such that \( y_i = p(z_i) \).

7. Write a MATLAB function \texttt{nf.m} which computes the coefficients of the Newton form of the interpolating polynomial by solving a suitable lower triangular linear system. Call sequence

\[ >> c = \text{nf}(x, f) \]

where \( x \) and \( f \) are vectors of length \( (n + 1) \) and \( c \) is a vector such that

\[
p_n(x_k) = \sum_{j=0}^{n} c_j \prod_{i=0}^{j-1} (x_k - x_j) = f_k
\]

for \( k = 0, 1, 2 \ldots, n \).

**Caution:** The function must check for user stupidity, i.e. if the \( x_j \) are not distinct then an error message must be displayed and the routine should exit gracefully.

8. Verify experimentally that your new MATLAB functions recover the polynomial \( f : [-1, 1] \rightarrow \mathbb{R} \) given by

\[
f(x) = (x - 1) \left( x - \frac{3}{4} \right) \left( x - \frac{1}{4} \right) x \left( x + \frac{1}{4} \right) \left( x + \frac{3}{4} \right) (x + 1),
\]

with a very small absolute error, provided that you use enough points.

9. Interpolate Runge’s function \( f : [-1, 1] \rightarrow \mathbb{R} \) given by

\[
f(x) = \frac{1}{1 + 25x^2}
\]

using \( k = 5, 10, 15, 20, \) and 25 equidistant points. Determine experimentally a lower bound on the maximal relative error between each of your polynomials and the target. Does high order interpolation necessarily imply a small relative error?

10. Repeat the previous question, but use the roots of the \( k \)th order Chebychev polynomial, rather than equidistant points. Report on the relative error.