Problem 1

Using the Horner-scheme, do a polynomial division for $p(x)$ with the linear factor $x - x_0$. Then, from the Horner-scheme, construct $p(x_0)$.

E) $p(x) = 5x^4 + 3x^3 - 30x^2 + 7x + 8, \quad x_0 = 2$

Problem 2 **2 Points**

H) Consider the following interpolation points (x_j, f_j).

x_j	4	2	0	-1
f_j	7	29	27	-73

Let $p(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ be the interpolation polynomial for the given data.

1) Set up the system of linear equations to compute the coefficients a_k. You do not need to solve the linear equation system.

2) Write out the interpolation polynomial p in Lagrange-representation (i.e. as linear combination of the Lagrange-basis polynomials). You do not need to expand the terms of the polynomial.

Problem 3 **4 Points**

Consider the following interpolation points (x_j, f_j).

E a)

x_j	3	1	0	-2
f_j	6	2	15	281

E b)

x_j	3	1	0	-2	5
f_j	6	2	15	281	190

E c)

x_j	-2	0	1	3
f_j	281	15	2	6

H a)

x_j	4	2	0	-1
f_j	7	29	27	-73

H b)

x_j	4	2	0	-1	1
f_j	7	29	27	-73	64

Compute the corresponding divided differences and write out the interpolation polynomial p in Newton-form. (Remark: Although the coefficients for (E a) and (E c) are different, the corresponding interpolation polynomials are the same.)

Programming Assignment 1

Write a function `interpoly(x,f)`, which computes and plots the interpolation polynomial to the given data points (x_j, f_j) in the interval $\min\ x_j \leq t \leq \max\ x_j$. You can use any method, but do not use the Python or Matlab command `polyfit` or the Python command `interp1d`. Test the program with the functions $f(t) = \cos(t)$ and $f(t) = 1/(1 + t^2)$, each in the interval $[-6, 6]$. Use equidistant control points and use also Chebycheff-control points (see lecture notes). Hint: to generate n equidistant control points in an interval $[a, b]$ use the Python or Matlab command `linspace(a,b,n)`.

Note: In all Matlab commands referring to polynomials index the coefficients of the polynomials as follows:

$$p(x) = a_1\,x^n + a_2\,x^{n-1} + \ldots + a_n\,x + a_{n+1},$$

i. e. the coefficient in front of the highest power has index 1, etc.

Programming Assignment 2

Let $f(x) = p(x)/q(x)$, where $p, q : \mathbb{R} \to \mathbb{R}$ are two arbitrary differentiable functions.

The Newton-method to compute a root of f can be written in the form

$$x_{k+1} = x_k - \frac{1}{\frac{p'(x_k)}{p(x_k)} - \frac{q'(x_k)}{q(x_k)}}. \qquad (*)$$

We can use this to find all roots of a polynomial p: Let $p(x) = a_1\,x^n + a_2\,x^{n-1} + \ldots + a_n\,x + a_{n+1}$ be a polynomial with real or complex coefficients a_k and let $z_1, \ldots, z_n \in \mathbb{C}$ be the roots of p. First, find one root by applying the Newton method to p. To find the remaining roots, consider the following: We can write the polynomial p as a product of linear factors

$$p(x) = a_1\,(x - z_1)(x - z_2)\ldots(x - z_n).$$

Assume the roots $z_1, z_2, \ldots, z_\ell$ have been already computed and let

$$q(x) = (x - z_1)(x - z_2)\ldots(x - z_\ell). \qquad (**)$$

Then,

$$f(x) = p(x)/q(x) = a_1\,(x - z_{\ell+1})\ldots(x - z_n)$$

is a polynomial whose roots are the remaining roots of p. Now, we apply the Newton-method in the form $(*)$ to f. Differentiating $(**)$ with the product rule gives

$$\frac{q'(x)}{q(x)} = \frac{1}{x - z_1} + \frac{1}{x - z_2} + \ldots + \frac{1}{x - z_\ell},$$

which we substitute into $(*)$. The roots computed using this method are increasingly less accurate, because we use already computed roots (which are not exact) to find the remaining roots. To improve the results, re-compute the roots z_j by applying the Newton-method to p with the z_j as initial values.

Write a function `z=polyroots(a)`, which computes all roots of a given polynomial p with coefficients $[a_1 \;\; a_2 \;\; \ldots \;\; a_{n+1}]$:

- To be able to find also complex roots, use complex initial values. Use random initial values. The Python command is `random.random`, the Matlab command is `rand` (for complex values `rand+i*rand`).
- To compute $p(x)$, use the Python-command `polyfit` or the Matlab-command `polyval`. To compute $p'(x)$, first compute the coefficients of the derivative with `polyder`, then use the command `polyval`.
- Use $|p(x)|$ to test how close your computed root is to the exact root. The Python or Matlab command to compute the absolute value of a number is `abs`.
- There are at least two ways to test your program:
  (i) Choose roots $z_1, \ldots, z_n \in \mathbb{C}$ and compute the coefficients a_k of the corresponding polynomial $p(x) = (x - z_1)(x - z_2)\ldots(x - z_n)$. Apply the program to these coefficients. This should give the z_k as result.
  (ii) Compare the computed roots with the result of the Python or Matlab command `roots`.