# Homework no. 6

Given $(n+1)$ distinct points, $x_0$, $x_1$, …, $x_n$ ($x_i \in \mathbf{R}\ \forall i$, $x_i \neq x_j$, $i \neq j$) and the $(n+1)$ values of an unknown function$f$ at these points, $y_0 = f(x_0)$, $y_1 = f(x_1)$, …, $y_n = f(x_n)$:

| $x$ | $x_0$ | $x_1$ | $\cdots$ | $x_n$ |
|---|---|---|---|---|
| $f$ | $y_0$ | $y_1$ | $\cdots$ | $y_n$ |

approximate the value of function $f$ in $\bar{x}$, $f(\bar{x})$, for a given $\bar{x}$, a value which is not in the above table, $\bar{x} \neq x_i$ , $i = 0, \ldots, n$ using:

- Newton forward formula on equispaced points and Aitken's type method for computing the finite differences; display $L_n(\bar{x})$ and $|L_n(\bar{x}) - f(\bar{x})|$;
- polynomial approximation computed with the least squares method. For computing the value of least squares polynomial in $\bar{x}$, use Horner's algorithm; display $P_m(\bar{x})$, $|P_m(\bar{x}) - f(\bar{x})|$ and $\sum_{i=0}^{n} |P_m(x_i) - y_i|$. For $m$ introduce values smaller than 6.

For solving the linear systems involved in solving the above interpolation problems use the library employed in *Homework 2*.

The equispaced interpolation points $\{x_i, i = 0, ..., n\}$ can be generated in the following way: $n$, $x_0$ and $x_n$ are introduced from keyboard or read from a file such that $x_0 < x_n$. Compute $h = (x_n - x_0)/n$ and $x_i = x_0 + ih$ , $i = 1, 2, ..., n-1$ ; the values $\{y_i, i = 0, ..., n\}$ are computed using a given function $f$ implemented in your program (examples of interpolation points $x_0$, $x_n$ and functions $f(x)$ are given at the end of this document), $y_i = f(x_i), i = 0, ..., n$.

**Bonus (15 pt)**: Draw the graphs of function $f$ and of the approximative computed functions $L_n$ and $P_m$.

## Numerical Interpolation

We know the value of a real function in a finite number of points, $x_0, x_1, \dots, x_n$:

$$\begin{array}{c|cccc} x & x_0 & x_1 & \cdots & x_n \\ \hline f & y_0 & y_1 & \cdots & y_n \end{array} \quad , \ x_i \neq x_j \ \ \forall\ i \neq j \ , \quad y_i = f(x_i) \ \ i = \overline{0,n}$$

In order to approximate the value of this function $f$ in $\bar{x}$, $f(\bar{x})$, $\bar{x} \neq x_i$ one computes an elementary function $S(x)$ that satisfies:

$$S(x_i) = y_i \ , \quad i = \overline{0,n}.$$

The approximative value for $f(\bar{x})$ is $S(\bar{x})$:

$$f(\bar{x}) \approx S(\bar{x})$$

One way to build the function $S$ is to use o polynomial of degree $n$, that is, the Lagrange interpolation polynomial.

### Lagrange Interpolation Polynomial

The unique polynomial of degree $n$, $L^{(n)}$, that satisfy the interpolation conditions:

$$L^{(n)}(x_i) = y_i \quad , \quad i = \overline{0,n}$$

can be expressed in many ways. One formula is the following (the definition):

$$L^{(n)}(x) = \sum_{i=0}^{n} \left( y_i \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \right)$$

When the interpolation points are equispaced:

$$x_i = x_0 + ih \quad , \quad i = 0, \dots, n$$

(usually one specifies the number of interpolation points, $x_0$ and $x_n$, the value of $h$ is computed using the formula $h = \dfrac{(x_n - x_0)}{n}$), the above polinyomial can be writen using the formula:

$$\begin{aligned} L_n(x = x_0 + th) = \quad & y_0 + \Delta f(x_0)t + \Delta^2 f(x_0)\frac{t(t-1)}{2!} + \cdots + \\ & + \Delta^k f(x_0)\frac{t(t-1)\cdots(t-k+1)}{k!} + \cdots \\ & + \Delta^n f(x_0)\frac{t(t-1)\cdots(t-n+1)}{n!} \quad , \quad t = \frac{x - x_0}{h}. \end{aligned} \tag{1}$$

This way of writing the Lagrange interpolation polynomial is called Newton forward formula on equispaced points. In the above formula, $\Delta^k f(x)$ are the finite differences of order $k$ for function $f$ and they can be computed recursively by:

$$\Delta f(x) = f(x+h) - f(x) \quad , \quad \Delta^k f(x) = \Delta^{k-1} f(x+h) - \Delta^{k-1} f(x) \ , k > 1$$

The direct definition (without recursion) of the finite difference of order $k$ is:

$$\Delta^k f(x) = \sum_{i=0}^{k} (-1)^{(k-i)} C_k^i f(x+ih)$$

For computing the finite differences $\Delta^k f(x_0)$ that appear in formula (1), use the recursive definition and Aitken's method.

Aitken's method for computing the finite differences

Aitken's method provides a fast algorithm, in $n$ steps, for computing the finite differences necessary for building the Lagrange polynomial using Newton forward formula. The algorithm is presented in the following table:

| | Step 1 | Step 2 | | Step $n$ |
|---|---|---|---|---|
| $y_0$ | | | | |
| $y_1$ | $\Delta f(x_0) = y_1 - y_0$ | | | |
| $y_2$ | $\Delta f(x_1) = y_2 - y_1$ | $\Delta^2 f(x_0) = \Delta f(x_1) - \Delta f(x_0)$ | | |
| $y_3$ | $\Delta f(x_2) = y_3 - y_2$ | $\Delta^2 f(x_1) = \Delta f(x_2) - \Delta f(x_1)$ | | |
| $\vdots$ | | | | |
| $y_{n-1}$ | $\Delta f(x_{n-2}) = y_{n-1} - y_{n-2}$ | $\Delta^2 f(x_{n-3}) = \Delta f(x_{n-2}) - \Delta f(x_{n-1})$ | | |
| $y_n$ | $\Delta f(x_{n-1}) = y_n - y_{n-1}$ | $\Delta^2 f(x_{n-2}) = \Delta f(x_{n-1}) - \Delta f(x_{n-2})$ | $\cdots$ | $\Delta^n f(x_0)$ |

In step $k$ one computes finite differences of order $k$:

$$\Delta^k f(x_0) \ , \ \Delta^k f(x_1) \ , \ \ldots, \Delta^k f(x_{n-k})$$

using only the finite differences computed in the previous step. All the computations can be performed using only one vector $y$. After computing the finite differences or order $k$ (step $k$), the vector $y$ has the following structure:

$$y = (y_0, \Delta f(x_0), \Delta^2 f(x_0), \ldots, \Delta^k f(x_0), \Delta^k f(x_1), \ldots, \Delta^k f(x_{n-k}))$$

After step $n$ the vector $y$ contains all the finite differences needed for computing $L_n$ with formula (1):

$$y = (y_0, \Delta f(x_0), \Delta^2 f(x_0), \ldots, \Delta^{n-1} f(x_0), \Delta^n f(x_0)).$$

The element:

$$s_k = \frac{t(t-1)\cdots(t-k+1)}{k!} \quad , \quad k = 1, 2, \ldots, n$$

can be computed iteratively:

$$s_1 = t \quad , \quad s_k = s_{k-1}\frac{t-k+1}{k} \quad , \quad k = 2, \ldots, n$$

The value of function $f$ in $\bar{x}$ is approximated by $L_n(\bar{x})$.

$$L_n(\bar{x} = x_0 + th) = y_0 + \Delta f(x_0)s_1 + \Delta^2 f(x_0)s_2 + \cdots + \Delta^k f(x_0)s_k + \cdots + \Delta^n f(x_0)s_n$$

### Least Squares Interpolation

Let $a = x_0 < x_1 < \cdots < x_n = b$. Given $\bar{x} \in [a, b]$ approximate $f(\bar{x})$ knowing that the $n+1$ values $y_i$ of function $f$ in the interpolation nodes.

One computes a polynomial of degree $m$:

$$P_m(x) = P_m(x; a_0, a_1, \ldots, a_m) = a_m x^m + a_{m-1}x^{m-1} + \cdots + a_1 x + a_0 = \sum_{k=0}^{m} a_k\, x^k$$

the coefficients $\{a_i; i = \overline{0, m}\}$ being the solution to the optimization problem:

$$\min \left\{ \sum_{r=0}^{n} \left| P_m(x_r; a_0, a_1, \ldots, a_m) - y_r \right|^2 \; ; \; a_0, a_1, \ldots, a_m \in \mathbb{R} \right\}$$

Solving this problem leads to solving the linear system:

$$Ba = f$$

$$B = (b_{ij})_{i,j=0,\ldots,m} \in \mathbb{R}^{(m+1)\times(m+1)} \quad f = (f_i)_{i=0,\ldots,m} \in \mathbb{R}^{m+1}$$

$$\sum_{j=0}^{m} \Big( \sum_{k=0}^{n} x_k^{i+j} \Big) a_j \;=\; \sum_{k=0}^{n} y_k x_k^i \quad , \quad i = 0, \ldots, m$$

This linear system can be solved with the same numerical library that was used for *Homework 2*.

The value of function $f$ in $\bar{x}$ is approximated by the value of the polynomial $P_m$ in $\bar{x}$:

$$f(\bar{x}) \approx P_m(\bar{x}; a_0, a_1, \dots, a_m)$$

For computing the value of polynomial $P_m(\bar{x})$ use Horner's method described below.

### Horner's method for computing $P(v)$

Let $P$ be a polynomial of degree $p$:

$$P(x) = c_0 x^p + c_1 x^{p-1} + \cdots + c_{p-1} x + c_p \ , \quad (c_0 \neq 0)$$

We can write polynomial $P$ also as:

$$P(x) = ((\cdots(((c_0 x + c_1)x + c_2)x + c_3)x + \cdots)x + c_{p-1})x + c_p$$

Taking into account this grouping of the coefficients, one obtains an efficient way to compute the value of polynomial $P$ in any point $v \in \mathbf{R}$, this procedure is knowns as *Horner's method*:

$$\begin{array}{rcl} d_0 & = & c_0 \ , \\ d_i & = & c_i + d_{i-1} v \ , \quad i = \overline{1, p} \end{array} \tag{2}$$

In the above sequence:

$$P(v) := d_p$$

the rest of the computed elements of the sequence $d_i, i = 1, ..., p-1$, are the coefficients of the quotient polynomial $Q$, obtained in the division:

$$\begin{array}{rcl} P(x) & = & (x - v)Q(x) + r \ , \\ Q(x) & = & d_0 x^{p-1} + d_1 x^{p-2} \cdots + d_{p-2} x + d_{p-1} \ , \\ r & = & d_p = P(v). \end{array}$$

Computing $P(v)$ ($d_p$) with formula (2) can be performed using only one real variable $d \in \mathbf{R}$ instead of using a vector $d \in \mathbf{R}^p$.

Input - examples

1. For table:

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $f$ | 50 | 47 | $-2$ | $-121$ | $-310$ | $-545$ |

   and $\bar{x} = 1.5$ we have $f(1.5) = 30.3125$. The first method should compute exactly (numerically) this value.

2. $x_0 = a = 1 \quad , \quad x_n = b = 5 \quad , \quad f(x) = x^4 - 12x^3 + 30x^2 + 12$