Some Notes on Neville's Algorithm of Interpolation with Applications to Trigonometric Interpolation

[1]Shpëtim Rexhepi, [2]Egzona Iseni, [3]Bilall I. Shaini, [4]Tetuta Zenku

[1,2,4]Mother Teresa University, Math department, Skopje, Macedonia

[3]State University of Tetovo, Math department, Tetovo, Macedonia.

shpetim.rexhepi@unt.edu.mk, egzona.iseni@unt.edu.mk, bilall.shaini@unite.edu.mk, teuta.zenku@unt.edu.mk

Abstract

In this paper is given a description of Neville's algorithm which is generated from Lagrange interpolation polynomials. Given a summary of the properties of these polynomials with some applications. Then, using the Lagrange polynomials of lower degrees, Neville algorithm allows recursive computation of those of the larger degrees, including the adaption of Neville's method to trigonometric interpolation. Furthermore, using a software application, such as in our case, *Matlab*, we will show the numerical experiments comparisons between the Lagrange interpolation and Neville`s interpolation methods and conclude for their advantages or disadvantages.

Key words: Lagrange interpolation, Trigonometric interpolation, Neville's method, Neville's algorithm.

Subject Classification (2010): 97N50, 65T40

1. Introduction

Interpolation is an important tool in producing computable approximations to commonly used functions. By interpolation can be determined how any size varies within the segment of the measuring points, even outside this interval (in this case we extrapolation). The most basic problem of interpolation arises as follows: In a segment $[a,b]$ are given $n+1$ points $x_i, i=\overline{0,n}$ that are called nodes of interpolation and the respective values $y_i = f(x_i), i=\overline{0,n}$ of a function *f*. Required a simple function, in our case a polynomial P_n of degree less or equal to n that have the same values with function f at interpolation nodes, namely: $P_n(x_i) = f(x_i), i=\overline{1,n}$.

Here we will deal with the problem of polynomial interpolation. Will be examined the interpolation by Neville algorithm which actually is an improved version of the classical polynomial interpolation by Lagrange. Neville's method can be applied in the situation that we want to interpolate $f(x)$ at a given point $x = p$ with increasingly higher order Lagrange interpolation polynomials.

Below is given a description of the interpolation algorithm and examples of execution of this algorithm in programming language. Using undetermined coefficients for the polynomials in Neville's algorithm, one can compute the Mac'Laurin expansion of the final interpolating polynomial, which yields numerical approximations for the derivatives of the function at the origin. While "this process requires more arithmetic operations than is required in finite difference methods", "the choice of points for function evaluation is not restricted in any way". They also show that their method can be applied directly to the solution of linear systems of the Vandermonde type.

On Neville's method

Through any two points can pass only one line. Through any three points can be withdrawn a single(unique) parabola, and so on. Polynom of degree *n*-1 that passes through *n* points $y_i = f(x_i), i = \overline{1,n}$ is given by classical Lagrange as follows: $P(x) = \sum_{i=1}^{n} \left(\frac{\prod_{\substack{j=1 \\ j \neq i}}^{n} (x - x_j)}{\prod_{\substack{j=1 \\ j \neq i}}^{n} (x_i - x_j)} \right) y_i$.

Interpolation algorithm can be implemented directly through the Lagrange formula, but it is not a good solution. Such an algorithm will not have any mechanism for evaluating the error. In this case Neville's algorithm is very appropriate. Description of the algorithm is as follows:

Let P_1 be the value at the point *x* of the unique polynomial of degree zero (constant) which passes through the point (x_1, y_1), therefore $P_1 = y_1$. In the same way can be determined points $P_1, P_2, ..., P_n$. Now if we go a step further and define $P_{1,2}$ as the value at the point x, of the unique polynomial of the first degree that passes throught points (x_1, y_1) and (x_2, y_2). The same thing is valid for points $P_{2,3}, P_{3,4}, ..., P_{n-1,n}$. In a similar way can be defined $P_{1,2,...,n}$ the value of the unique polynomial that passes through all *n*-points. According to these the structure of the Neville algorithm can be displayed in schemes. The Neville Algorithm recursively complements the numbers in this table - the column for columns also from left to right. It is based on the "child's" P links and his two "parents

$$P_{j(j+1)...(j+m)}(x) = \frac{1}{x_j - x_{j+m}} \begin{vmatrix} P_{j(j+1)...(j+m-1)}(x) & x_j - x \\ P_{(j+1)(j+2)...(j+m)}(x) & x_{j+m} - x \end{vmatrix}, \quad \text{for} \begin{cases} j = \overline{m+1, n} \\ m = \overline{0, n-1} \end{cases},$$

while it's inverse can be given by

$$P_{j(j+1)...(j+m)}(y) = \frac{1}{y_j - y_{j+m}} \begin{vmatrix} P_{j(j+1)...(j+m-1)}(y) & y_j - y \\ P_{(j+1)(j+2)...(j+m)}(y) & y_{j+m} - y \end{vmatrix}, \text{ for } \begin{cases} j = \overline{m+1, n} \\ m = \overline{0, n-1} \end{cases}$$

The improvement of the above rrecurence can be achieved in such a way as to follow the minor changes between "the child" and "his parents". These differences are equal to:

$$C_{m,j} \equiv P_{j...(j+m)} - P_{j...(j+m-1)}, \qquad D_{m,j} \equiv P_{j...(j+m)} - P_{(j+1)...(j+m)}$$

Respectively,

$$D_{m+1,i} = \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}, \quad C_{m+1,i} = \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}$$

In the above expressions, C and D represent corrections which, for each column of the scheme, increase the interpolation order to 1. For example for $n = 4$ the scheme will be as follows:

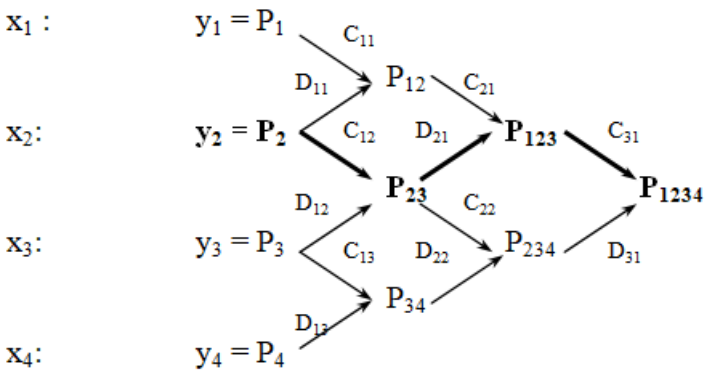


Table 1

filling the scheme from left to right, the correction value C and D will be reduced more and more (in the ordinary case). Thus, the final correction value will show the error indicator. This value actually indicates to what extent the interpolation curve is polished. If the interpolation curve fluctuates (eg outside the entry point range) the curve will not be so smooth, which means that even the correction values C and D will be larger, and as a consequence error estimation.

So Neville's algorithm in recursively way fills the numbers in the scheme column by column from left side to the right. The curves generated by Neville's algorithm are called Lagrange interpolation polynomials.

We will illustrate an example of Lagrange interpolation polynomial in the following figure:

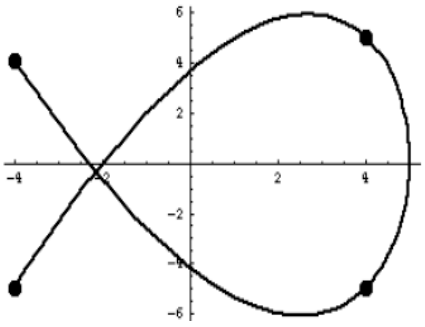


Fig 1. The cubic Lagrange polynomial for the control points: P_1=(-4,4), P_2=(4,-5), P_3=(4,5), P_4=(-4,-5) (dots), interpolated at the nodes $x_k = k,\ (k = 1,2,3,4)$.

Theorem1. Given the affine points $P_1, P_2, ..., P_n$ and distinct parameters $t_1, t_2, ..., t_n$. There is a unique polynomial curve $P_{1,2,...,n}(t)$ of degree $n-1$ that interpolates the given points at the specified parameters. That is $P_{1,2,...,n}(t_k) = P_k, (k = 1,2,..,n)$.

To proof this theorem we will give these auxiliary theorems:

Theorem 2. [*Taylor theorem*] Let $P(t)$ be a polynomial, $\deg P(t) = n-1, r \in \mathbb{R}$ .Then

$$P(t) = P(r) + P'(r)(t-r) + P''(r)\frac{(t-r)^2}{2!} + ... + P^{(n-1)}(r)\frac{(t-r)^{n-1}}{(n-1)!}.$$

Proposition 1. Let $P(t)$ be a polynomial of degree $n-1$. Then r is the root of $P(t)$ it and only if $t-r$ is a factor of $P(t)$.

Proof: Let $P(t)$ be a polynomial of degree n-1. Than by theorem 2

$$P(t)=P(r)+P'(r)(t-r)+P''(r)\frac{(t-r)^2}{2!}+\ldots+P^{(n-1)}(r)\frac{(t-r)^{n-1}}{(n-1)!}.$$

Therefore, by inspection, $P(r)=0$ if and only if if $t-r$ is a factor of $P(t)$.

Proposition 2. Every nonzero polynomial of degree $n-1$ has at most $n-1$ roots.

Proof: From proposition 1 as consequence is obtained that a polynomial of $n-1$ degree can have at most $n-1$ linear factors.

Proposition 3. Let $P(t)$ and $Q(t)$ be two polynomials of degree $n-1$ that agree at *n* parameter values. Than $P(t)=Q(t)$.

Proof: Let $R(t)=Q(t)-P(t)$. Than $R(t)$ is a polynomial of degree *n*-1. Moreover since $P(t)$ and $Q(t)$ agree at n parameter values, $R(t)$ has *n* roots. Therefore by proposition 3, $R(t)$ must be the zero polynomial, respectively $P(t)=Q(t)$.

Now let us proof ***theorem1***.

The proof is by induction on *n*. We have already established this result for $n=1,2,3,4$. Suppose this result is valid for $n-1$. Then by induction there are polynomial curves $P_{1,2,\ldots,n-1}(t)$ and $P_{2,3,\ldots,n}(t)$ of degree $n-2$ that interpolates the points $P_2,\ldots,P_n$ at the parameters $t_2,\ldots,t_n$. Define $P_{1,2,\ldots,n}=\frac{t_n-t}{t_n-t_1}P_{1,2,\ldots,n-2}(t)+\frac{t-t_1}{t_n-t_1}P_{2,3,\ldots,n-1}(t)$.

Easily can be defined that $P_{1,2,\ldots,n}(t_k)=P_k$, $k=1,2,..,n$ and since $P_{1,2,\ldots,n-1}(t)$ and $P_{2,3,\ldots,n}(t)$ of degree $n-2$ from the above equation follows that $P_{1,2,\ldots,n}(t_k)$ is a polynomial of degree $n-1$. Now we will show uniqueness. Suppose that $P(t)$ and $Q(t)$ be two polynomials curves of degree $n-1$ that interpolate the given control points at the specified nodes. Then $P(t)$ and $Q(t)$ are polynomials of degree $n-1$ that agree at the *n* parameter values , so from *proposition 3,* $P(t)=Q(t)$ hence the interpolation polynomial is unique.

The idea of the Neville's method is to use Lagrange polynomials of lower powers recursively in order to compute Lagrange polynomials of higher powers. Neville's method is based on the following theorem:

Theorem 3. Let f be defined at the *k*- points $x_1,x_2,\ldots,x_k$ and let x_i and x_j be two distinct points in this set. Let $P_{1,2,\ldots,i-1,i+1,\ldots,k}(x)$ be the Lagrange polynomial that agrees with f at $x_1,x_2,\ldots,x_{i-1},x_{i+1},\ldots,x_k$. Similarly let

$P_{1,2,\ldots,j-1,j+1,\ldots,k}(x)$ be the Lagrange polynomial that agrees with f at $x_1, x_2, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k$. Than the Lagrange polynomial $P_{1,2,\ldots,k}(x)$ through all the k points $x_1, x_2, \ldots, x_k$ can be computed as follows:

$$P_{1,2,\ldots,k}(x) = \frac{1}{x_i - x_j}\begin{vmatrix} P_{1,2,\ldots,j-1,j+1,\ldots,k}(x) & x - x_i \\ P_{1,2,\ldots,i-1,i+1,\ldots,k}(x) & x - x_j \end{vmatrix}$$

Proof:

Obviously polynomials $P_{1,2,\ldots,i-1,i+1,\ldots,k}(x)$ and $P_{1,2,\ldots,j-1,j+1,\ldots,k}(x)$ are polynomials of $k-2$ degree so that the degree of $P_{1,2,\ldots,k}(x)$ is $k-1$. We will check if $P_{1,2,\ldots,k}(x_m) = f(x_m), m = \overline{1,k}$. Since $P_{1,2,\ldots,i-1,i+1,\ldots,k}(x_m) = f(x_m), m \neq i$ and $P_{1,2,\ldots,j-1,j+1,\ldots,k}(x_m) = f(x_m), m \neq j$, than for $m = \overline{1,k} - \{i, j\}$ we get

$$P_{1,2,\ldots,k}(x_m) = \frac{1}{x_i - x_j}\begin{vmatrix} P_{1,2,\ldots,j-1,j+1,\ldots,k}(x_m) & x_m - x_i \\ P_{1,2,\ldots,i-1,i+1,\ldots,k}(x_m) & x_m - x_j \end{vmatrix} =$$

$$= \frac{1}{x_i - x_j}\begin{vmatrix} f(x_m) & x_m - x_i \\ f(x_m) & x_m - x_j \end{vmatrix} = f(x_m)$$

For $m = i$ we have:

$$P_{1,2,\ldots,k}(x_i) = \frac{1}{x_i - x_j}\begin{vmatrix} P_{1,2,\ldots,j-1,j+1,\ldots,k}(x_i) & x_i - x_i \\ P_{1,2,\ldots,i-1,i+1,\ldots,k}(x_i) & x_i - x_j \end{vmatrix} =$$

$$= \frac{1}{x_i - x_j}\begin{vmatrix} f(x_i) & 0 \\ f(x_i) & x_i - x_j \end{vmatrix} = f(x_i)$$

For $m = j$ we have:

$$P_{1,2,\ldots,k}(x_j) = \frac{1}{x_i - x_j}\begin{vmatrix} P_{1,2,\ldots,j-1,j+1,\ldots,k}(x_j) & x_j - x_i \\ P_{1,2,\ldots,i-1,i+1,\ldots,k}(x_j) & x_j - x_j \end{vmatrix} =$$

$$= \frac{1}{x_i - x_j}\begin{vmatrix} f(x_m) & x_j - x_i \\ f(x_m) & 0 \end{vmatrix} = f(x_j)$$

So $P_{1,2,\ldots,k}(x)$ present the $(k-1)-$ degree interpolating polynomial.

The value $P_{1,2,\ldots,k}(x)$ can be calculated using a 1-dimensional data array P[1]···P[k]:

```
for i:=1 to k do begin
P[i]:=f[i];
for n:=i-1 downto 0 do
P[n]:=(P[n+1]*(x-x[n])-P[n]*(x-x[i]) )/(x[n]-x[i]);
```

```
 end;
 f:=P[1];
```

Matlab code

```
function[Table,Px]=neville(FUN,xn,x);
n=length(xn)-1;
Table(:,1)=xn';
ifisstr(FUN)==1;
    Table(:,2)=feval(FUN,xn');
else
Table(:,2)=FUN';
end;
 fori=2:n+1;
  forj=3:i+1;
   Table(i,j)=((x-xn(i))*Table(i-1,j-1)-...
    (x-xn(i-j+2))*Table(i,j-1))/(xn(i-j+2)-xn(i));
end;
  end;
 Px=Table(n+1,n+2);
```

The error of polynomial interpolation is calculated by this formula $R(a)=f(a)-P_n(a)=\prod_{i=0}^{n}\frac{f^{(n+1)}(b)(a-x_i)}{(n+1)!}$ where f has n+1 continuous derivatives on the interpolating interval such that for each a belonging in that interval for which exists b within the smallest interval containing a as well as all the x_i

2. Application of Neville's method on trigonometric interpolation

Next we will give application of Neville method to trigonometric interpolation.

Suppose that $f(x)$ be a 2π-periodic function of *x*, specified by its values $y_i=f(x_i), i=\overline{0,n}$ where $x_i\in[-\pi,\pi], i=\overline{0,n}$ than its trigonometric interpolation polynomial takes form $\overline{f}=\frac{1}{2}a_0+\sum_{k=1}^{m}(a_k\cos kx+b_k\sin kx)$ where m depends on the form of required fit. By $f_{i,j}(x), i\le j$ we

denote trigonometric polynomials which takes values $y_i, y_{i+1}, ..., y_j$ for $x_i, x_{i+1}, ..., x_j$, therefore $\overline{f}(x) = f_{0,n}(x)$. Let us deal the cases:

The algorithm occurs when data points lie in the range $[0, \pi]$ and depending on functions if it is odd function the trigonometric polynomial will contain sine terms and if it is even cosine terms

i) Neville's method for even functions

Let $x_i \in [0, \pi]$, $i = 0,1,...,n$. In this case $m = n$ and the unique trigonometric interpolation polynomial will be $\overline{f}(x) = \sum_{k=0}^{n} a_k \cos kx$ and it can be generated as follows: let us set $f_{i,i}(x) = y_i \ (i = 0,1,...,n)$ than

$$f_{i,j}(x) = \frac{1}{\cos x_j - \cos x_i} \begin{vmatrix} f_{i,j-1}(x) & \cos x_i - \cos x \\ f_{i+1,j}(x) & \cos x_j - \cos x \end{vmatrix}.$$

ii) Neville's method for odd functions

Let $x_i \in [0, \pi]$, $i = 0,1,\ldots,n$. In this case $m = n+1$ and the unique trigonometric interpolation polynomial will be $\overline{f}(x) = \sum_{k=0}^{n+1} a_k \sin kx$ and it can be generated as follows: let us set $f_{i,i}(x) = y_i \frac{\sin x}{\sin x_i} \ (i = 0,...,n+1)$ than

$$f_{i,j}(x) = \frac{1}{\cos x_j - \cos x_i} \begin{vmatrix} f_{i,j-1}(x) & \cos x_i - \cos x \\ f_{i+1,j}(x) & \cos x_j - \cos x \end{vmatrix}.$$

iii) *Neville's method if the function is not even or odd*

Let $g(x)$ be a 2π-periodic function of *x*, specified by its values $y_i = g(x_i), i = \overline{0,n}$ where $x_i \in [-\pi, \pi], i = \overline{0,n}$. In this case $n = 2m$ and the unique trigonometric interpolation polynomial will be $\overline{g} = \frac{1}{2}a_0 + \sum_{r=1}^{m} (a_r \cos rx + b_r \sin rx)$ which is provided by formula $\overline{g} = \sum_{i=0}^{n} \left(\frac{\prod_{\substack{j=0 \\ j \neq i}}^{n} \sin\left(\frac{x - x_j}{2}\right)}{\prod_{\substack{j=0 \\ j \neq i}}^{n} \sin\left(\frac{x_i - x_j}{2}\right)} \right) y_i$.

Equivalent to this result is obtained from method similar to Neville's method with the initial approximation $g_{i,i}(x) = y_i \ (i = 0,1,...,n)$ and further approximations are obtained from the relation

$$g_{i-1,j+1}(x) = \frac{\begin{vmatrix} \varphi_{j,j+1} & g_{i,j}(x) \\ 0 & g_{i-1,j-1}(x) \end{vmatrix} - \begin{vmatrix} \varphi_{i-1,j+1} & g_{i+1,j+1}(x) \\ \varphi_{i-1,i} & g_{i,j}(x) \end{vmatrix}}{\sin\left(\frac{x_j - x_{i-1}}{2}\right) \sin\left(\frac{x_{j+1} - x_{i-1}}{2}\right) \sin\left(\frac{x_{j+1} - x_i}{2}\right)}$$

Where

$$\varphi_{r,s} = \sin\left(\frac{x_s - x_r + x_j - x_i}{2}\right)\sin\left(\frac{x - x_r}{2}\right)\sin\left(\frac{x - x_s}{2}\right).$$

The following table illustrates the case for $n = 6$.

Calculation for full-range series(*function is not even or odd)*

g_{00}			
g_{11}	g_{02}		
g_{22}	g_{13}	g_{04}	
g_{33}	g_{24}	g_{15}	g_{06}
g_{44}	g_{35}	g_{26}	
g_{55}	g_{46}		
g_{66}			

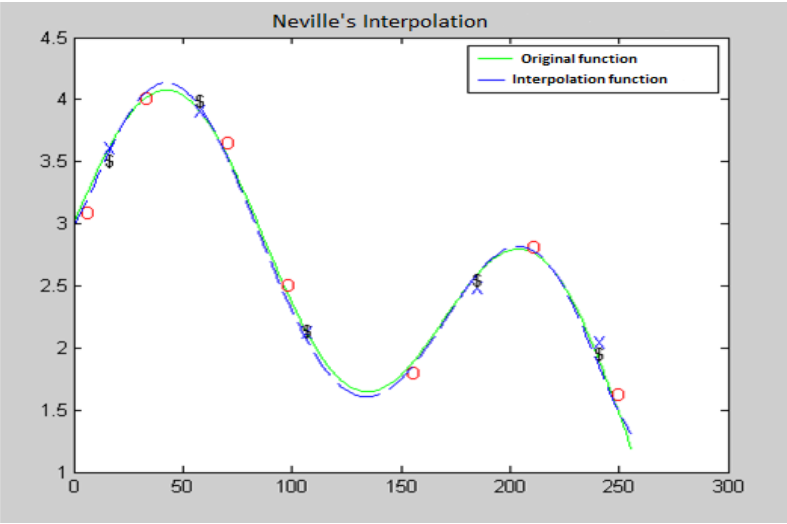
Table2.

3. Illustrations and discussions

In the *Matlab* software package we have created a program that can enter the original function that later will be interpolated. The user initially gives assigned number of points on the original function that are believed to be known as interpolation points for Neville's algorithm. Here is the advantage is the appearance of the original function (plotted in green), then given an arbitrary number of known points of the function (marked with red circles), which are then given to Neville's algorithm. Finally, the user sets the second set of points (red or blue crosshairs) asking for the values obtained by interpolation of initial given points. As a result of the program, displayed are the values of *x* (that are initially given) and *y* (interpolated) coordinates of the second set marked with "$", and the interpolated function and the graph of interpolated function plotted by a dashed blue line (from x_{min} to x_{max}).The results are shown in the figures below:

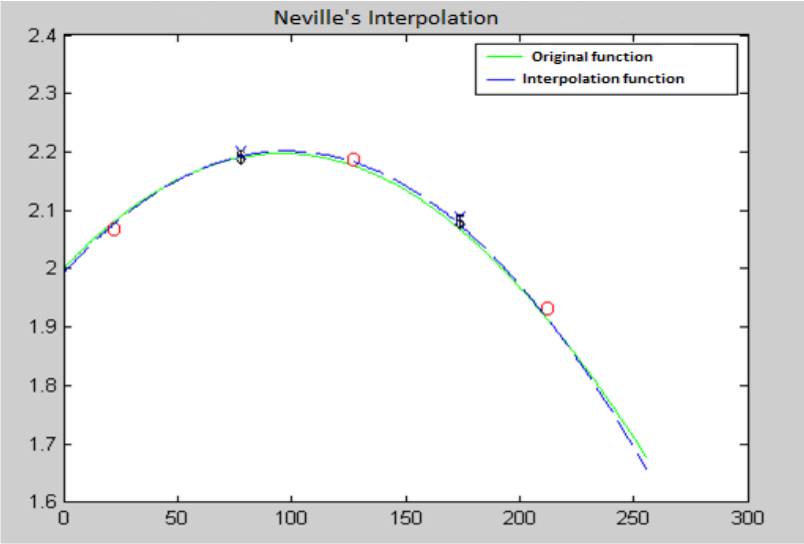
Examples of interpolated functions

1. $y = \sin(1.5x) + \cos(0,3x) + \cos(0,2x) + \cos(0,16x) + \sin(0,16x)$



(fig.1)

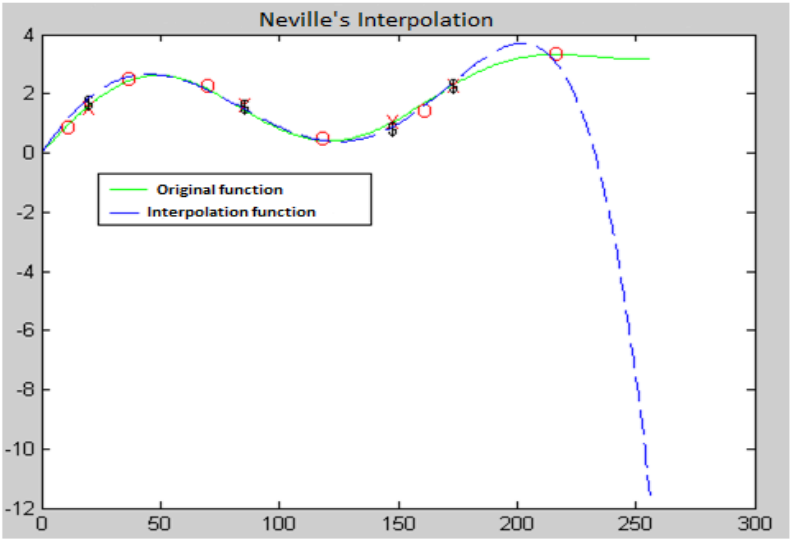
2. $y = \cos(0,2x) + \cos(0,16x) + \sin(0,16x)$



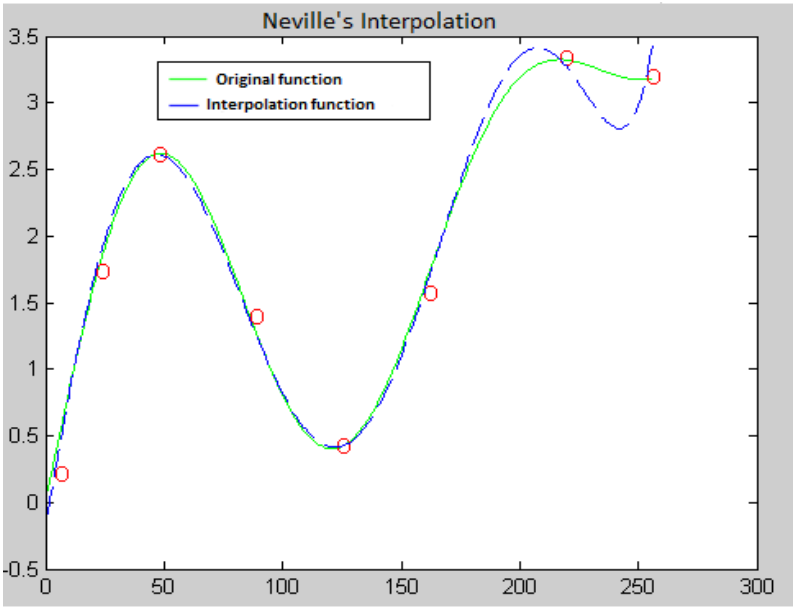
(fig.2)

In the following example can be seen that the interpolation accuracy is highly dependent on the fact that chosen points (known given points) are found in crossings or out crossings of the functions, and the fact if the boundary points are taken or not. Compare figure 3 and 4.

3. $y = \sin(1.2x) + \sin(1.7x) + \sin(0,2x) + \sin(0,16x) + \sin(0,16x) + \sin(0,083x)$



(fig.3)



(fig.4)

4. An example to find the error bound consider the sample points

I	x_i	$\sin x_i$
1	0	0
2	$\frac{\pi}{2}$	1
3	π	0
4	$\frac{3\pi}{2}$	-1
5	2π	0

Table3.

The maximum interpolation error is estimated by $(\left|f^{(n+1)}(b)\right| \le 1)\left|R(a)\right| \le \left|\prod_{i=0}^{n}\frac{(a-x_i)}{(n+1)!}\right| \le \frac{35}{120} \approx 0,3$ whereas the error increases rapidly outside the interval $[0, 2\pi]$ as can be seen from the figure 5, below:

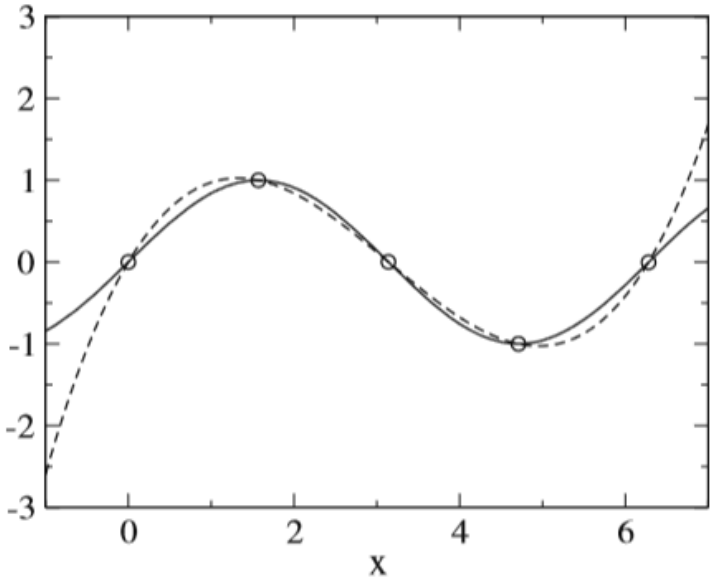


Fig5. (Interpolating polynomial) The interpolated function (solid curve) and the interpolating polynomial (broken curve)

4. Conclusion

The advantages of interpolated function: are used only operations +, -, *, /, in contrast to the original function, which can contain a variety of mathematical functions (sinh, ln, arc, exp…). This property is very useful in cases when the execution speed can be very important.

Using the program for the famous original function it is possible to vary the number (and coordinates) known points, and for a given problem to detect the optimal configuration of the known values, which satisfies the required accuracy interpolation. An important property Neville`s method is that successive interpolations get everything precise and more accurate values, with the exception of the last, which - diverges.

The advantage of Neville's method over a Lagrange interpolating polynomial, if the data are arranged in order of closeness to the interpolated point, is that none of the work performed to obtain a specific degree result must be redone to evaluate the next higher degree result. Neville's method has a couple of minor disadvantages. All of the work must be redone for each new value of x. The amount of work is essentially the same as for a Lagrange polynomial. The divided difference polynomial minimizes these disadvantages.

References

1. W. Gautschi, *Numerical Analysis*, Birkhäuser, Boston, 1997.
2. D. Levy, *Numerical Analysis* , University of Maryland, March 4, 2008.
3. N. J. Higham, *Accuracy and Stability of Numerical Algorithm*, SIAM, Philadelphia, 1996.
4. I.S Berezin and N.P. Zhidkov, *Computing methods,*Oxford, Pergamon Press
5. L. N. Trefethen, *Approximation Theory and Approximation Practice*, SIAM, 2012.

6. B. Shaini,Sh.Rexhepi , E.Iseni ,, *On advantages of Neville Method and its adapting to trigonometric series*, Conference SPNA Tirana, 2014

7. M. J. D. Powell, *Approximation Theory and Methods*, Cambridge University Press, 2001.

8. J. Mason, D. Hands comb, *Chebyshev Polynomials*, CHAPMAN & HALL/CRC , 2003.

9. T. J. Rivlin, *Chebyshev Polynomials-from appromaxion theory to algebra and number theory*, John Wiley, 1990.

10. B. Hunter, ,, *Nevilles method for trigonometric interpolation*,, the Computer journal, 1968

11. John MasonDavid Handscomb, "*Chebyshev Polynomials*", April 2002

12. Doron Levy ,,*Numerical Analysis* ,,University of Maryland March 4, 2008

13. Xiaozhe Hu,, *Introduction to Numerical Analysis*,, The Pennsylvania State University

14. Z.Drmac, M.Marusic,S.Singer,V.Hari,M.Rogina,S.Singer, "*Numericka Analiza*", predavanje i vezbe, Zagreb, 2003