Newton, Lagrange and Hermite Interpolation: Convergence and Runge phenomena

Luis E. Tobón

January 19, 2011

There are several purposes for using interpolation (Heat, 2002): plotting a smooth curve based on data or experimental points; extract information from discrete points; accurate differentiation or integration of discrete data; replacing a complicated function by a simple one. In general, for given data (x_i, y_i) with $i = 1 \dots m$, there is a continuous function $f : \mathbb{R} \to \mathbb{R}$ such that $f(x_i) = y_i$. In this case, f is the interpolating function, or interpolant. On this way, the family of funtions used for interpolation is huge: Polynomials, piecewise polynomials, trigonometric functions, exponential functions, rational functions, and so on. On the other hand, the distribution of points x_i can be or not uniform, or may be the number of points can be equal or greater than the number of basis functions used in the interpolation; then, there are a lot of questions related with this topic, however this work is focused in the family of polynomials functions with equal spaced points in x, which coefficients are defined by three different methods: Newton, Lagrange and Hermite. The convergence of these methods and the effect of the Runge phenomena is analyzed in this work. The code with the im[plementation and evaluation of these methods is described in Algorithm 1.

1 Formulations

The interpolating function f is expressed as a linear combination of basis functions $\phi_j(x)$,

$$f(x) = \sum_{j=1}^{n} c_j \phi_j(x) \tag{1}$$

where $c_j$are coefficients to be determinated. The requeriment of the interpolating points are:

Algorithm 1 Interpolation.m

```
f=inline('log(x)','x');
xo=linspace(1/2,3/2,N+1);
x=linspace(1/2,3/2,1001);
% Getting coefficients
        PN=Newton(f,xo);
        PL=Lagrange(f,xo);
        PH=Hermite(f,xo);
% Evaluation
        E=errorN(f,PN,xo,x);
        E=errorL(f,PL,xo,x);
        E=errorH(f,PH,xo,x);
```

Algorithm 2 Newton.m

```
function g=Newton(f,x)
        g=feval(f,x);
        n=length(x);
        for j=1:(n-1)
                for i=n:-1:(j+1)
                        g(i)=(g(i)-g(i-1))/(x(i)-x(i-j));
                end
        end
```

Algorithm 3 Evaluation of Newton interpolation and error.

```
function e=errorN(f,g,xo,x)
        e=feval(f,x);
        n=length(xo);
        p=zeros(1,length(x));
        for j=n:-1:2
                p=(x-xo(j-1)).*(g(j)+p);
        end
        p=p+g(1);
        e=abs(e-p);
```

$$f(x_i) = \sum_{j=1}^{n} c_j \phi_j(x_i) = y_i \tag{2}$$

Which is a system of linear equations that can be solved in several ways.

1.1 Newton Interpolation

The basis functions in Newton interpolation is in the form:

$$\phi_j = \prod_{k=1}^{j} (x - x_k) \tag{3}$$

Using this basis funtion and based on (1) the Newton interpolant can rewritten as follows:

$$p_N^{n-1}(x) = \gamma_1 + (x - x_1)(\gamma_2 + (x - x_2)(\gamma_3 + \cdots (\gamma_{n-1} + \gamma_n (x - x_{n-1})))) \tag{4}$$

where the coefficients γ_k can be found based on the definition of divided differences:

$$\gamma_k = f[x_1, x_2, \ldots, x_k] = \frac{f[x_2, \ldots, x_k] - f[x_1, \ldots, x_{k-1}]}{x_k - x_1} \tag{5}$$

$$f[x_k] = f(x_k)$$

Algorithm 2 and 3 shows the implementation of this method.

Algorithm 4 Lagrange.m

```
function g=Lagrange(f,x)
        g=feval(f,x);
```

Algorithm 5 Evaluation of Lagrange interpolation and error.

```
function e=errorL(f,g,xo,x)
        e=feval(f,x);
        n=length(xo);
        p=zeros(1,length(x));
        for j=1:n
                q=g(j)*ones(1,length(x));
                for i=1:n
                        if i~=j
                                q=q.*(x-xo(i))/(xo(j)-xo(i));
                        end
                end
                p=p+q;
        end
        e=abs(e-p);
```

1.2 Lagrange Interpolation

The following xpression describe the Lagrange interpolant:

$$p_L^{n-1}(x) = \sum_{j=1}^{n} f(x_j) \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} \tag{6}$$

where the coefficients are the simple evaluation of the function in the interpolating points. The implementation of the method is presented in Algorithms 5 and 6, in which we can note a simple way to found the coefficient but a more difficult way to evaluate the interpolant.

1.3 Hermite Interpolation

This method uses the value of the funtion on the interpolant points and its derivatives. The expression cab defined as follows:

$$p_H^{2n-1}(x) = \gamma_1 + \gamma_2 (x - x_1) + \gamma_3 (x - x_1)^2 + \gamma_4 (x - x_1)^2 (x - x_2) + \cdots \tag{7}$$

The evaluation of the coefficients is similar to Newton method but takin in consideration the derivatives, as we can see in Algortithm 6. The evaluation of the interpolant is similar to the Newton method as we can see in Algorithm 7.

2 Case of analysis

In this work a simple logarithm function is used to evaluate interpolation methods:

$$f(x) = \log(x), \qquad 0.5 \leq x \leq 1.5 \tag{8}$$

Algorithm 6 Hermite.m

```
function g=Hermite(f,x)
        fd=inline('1./x','x');
        gd=feval(fd,x);
        g1=feval(f,x);
        g(1)=g1(1);
        for j=1:(n-1)
          for i=n:-1:(j+1)
            if j==1
              if mod(i,2)==0
                g(i)=gd(i/2);
              else
                g(i)=(g1((i+1)/2)-g1((i-1)/2))/(x(i)-x(i-j));
              end
            else
            g(i)=(g(i)-g(i-1))/(x(i)-x(i-j));
            end
          end
        end
```

Algorithm 7 Evaluation of Hermite interpolation and error.

```
function e=errorH(f,g,xo,x)
        n=length(xo);
        for i=1:n
                x1(2*i-1)=xo(i);
                x1(2*i)=xo(i);
        end
        xo=x1;
        e=feval(f,x);
        n=length(xo);
        p=zeros(1,length(x));
        for j=n:-1:2
                p=(x-xo(j-1)).*(g(j)+p);
        end
        p=p+g(1);
        e=abs(e-p);
```

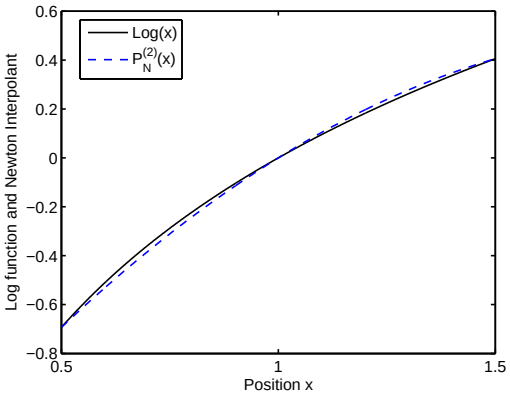

Figure 1: Logarithm function and derivatives

$$\frac{df\left(x\right)}{dx}=\frac{1}{x} \tag{9}$$

$$\frac{d^{(n)}f\left(x\right)}{dx^{(n)}}=\left(-1\right)^{n+1}\left(n-1\right)!\frac{1}{x^{n}} \tag{10}$$

As figure 1 shows, the function is very smooth in all points, and as a simple example an Newton interpolant with order 2 (using three points) can describe the function very well.

3 Results

Figures 2 to 4 show the absolute error of the interpolants for Newton, Lagrange and Hermite methods respectively. On the left hand of this functions low order polynomials are showed, and on the right hand are showed high order. As we can note, the error decrease until valued closed to 10^{-12} when polynomials with order 24 in Newton and Lagrange, and 19 in Hermite are used; however with larger order of polynimials, the error increse again to values similar to low order. This effect is caused by the Runge phenomena, which describe the maximum error close to the end points and an increment with high order polynomials. The position of the maximum error is clearly described in figure 5, in which the maximum error goes close to 0.5 (end point) with large values of N. A clear description of the Runge effect is showed in figure 6.

4 Code perfoming

Using Profile funtion in Matlab we can check the computational cost of each function and line in the code. The way this function is called is showed in figure 7. The function ErrorL.m shows the most expensive computational cost of this code (in this case just Newton and Lagrange are analized), and this is due to a large number of iteration used to evaluate the interpolant, as figure 9 shows.

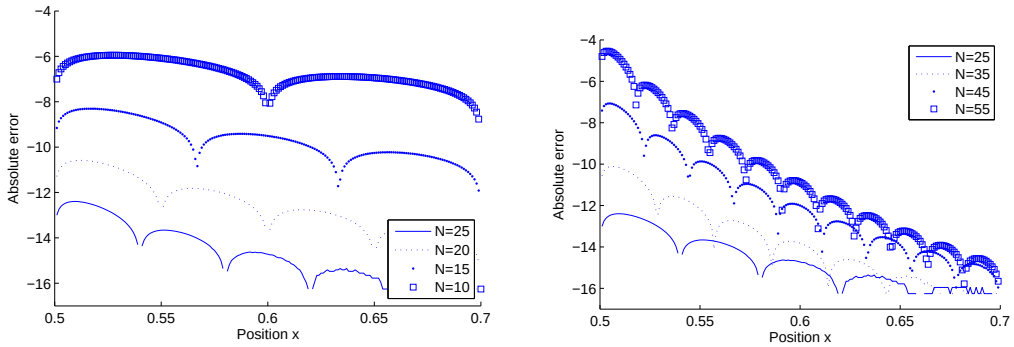

Figure 2: Absolute error using Newton interpolation as a function of the position between 0.5 and 0.7, and eight different number of order of interpolation

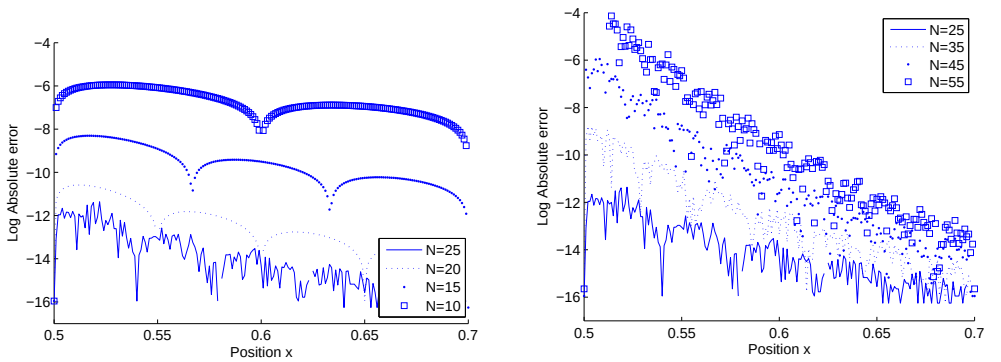

Figure 3: Absolute error using Lagrange interpolation as a function of the position between 0.5 and 0.7, and eight different number of order of interpolation

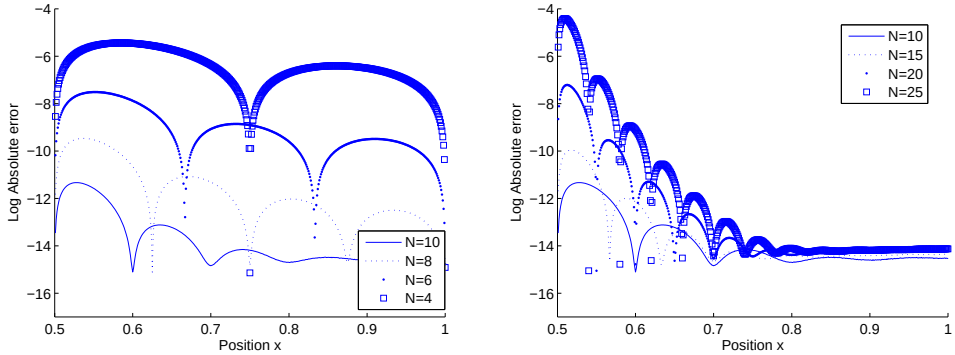

Figure 4: Absolute error using Hermite interpolation as a function of the position between 0.5 and 0.7, and eight different number of order of interpolation

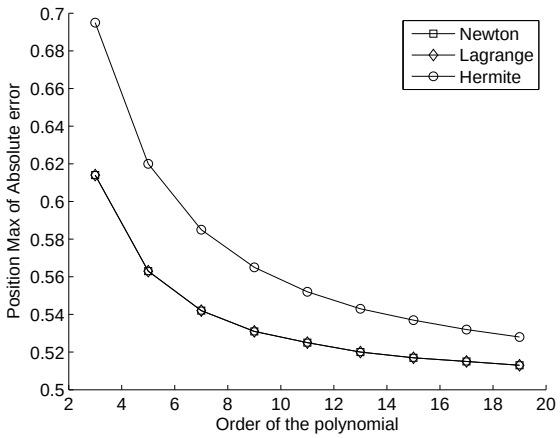

Figure 5: Position of the maximum absolute error for Newton interpolation as a function of the order of the polynomial.

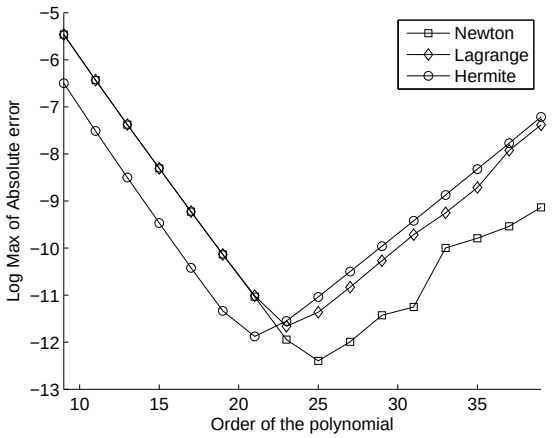

Figure 6: Maximum absolute error for Newton, Lagrange and Hermite interpolation as a function of the order of the polynomial.

```
19 -        profile on;
20      % Getting coefficients
21 -        PN=Newton(f,xo);
22 -        PL=Lagrange(f,xo);
23      % Evaluation
24 -        E=errorN(f,PN,xo,x);
25 -        E=errorL(f,PL,xo,x);
26 -        profile viewer;
```

Figure 7: Calling to Profile function in Matlab.

Profile Summary

Generated 18-Jan-2011 16:29:39 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
errorL	1	0.021 s	0.020 s	
Newton	1	0.006 s	0.003 s	
inline.feval	4	0.005 s	0.003 s	
inlineeval	4	0.002 s	0.002 s	
errorN	1	0.002 s	0.001 s	
Lagrange	1	0 s	0.000 s	

Figure 8: Main view of Profile function in Matlab.

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
10	`q=q.*(x-xo(i))/(xo(j)-xo(i));`	1640	0.018 s	85.7%	
3	`e=feval(f,x);`	1	0.002 s	9.5%	
9	`if i~=j`	1681	0.001 s	4.8%	
17	`e=abs(e-p);`	1	0 s	0%	
14	`end`	41	0 s	0%	
All other lines			0 s	0%	
Totals			0.021 s	100%	

Function listing

Color highlight code according to time

```
  time    calls  line
                    1 function e=errorL(f,g,xo,x)
                    2
< 0.01          1   3 e=feval(f,x);
                1   4 n=length(xo);
                1   5 p=zeros(1,length(x));
                1   6 for j=1:n
               41   7     q=g(j)*ones(1,length(x));
               41   8     for i=1:n
< 0.01       1681   9         if i~=j
  0.02       1640  10             q=q.*(x-xo(i))/(xo(j)-xo(i));
             1640  11         end
             1681  12     end
               41  13     p=p+q;
               41  14 end
                   15
                   16 %plot(x,e,'k',x,p,'--b')
                1  17 e=abs(e-p);
```

Figure 9: Evaluation of function errorL.m, and highlights in code.