"Constantin Brâncuşi" University of Târgu Jiu ENGINEERING FACULTY
SCIENTIFIC CONFERENCE – 13 th edition – with international participation
November 07-08, 2008 – Târgu Jiu

ON THE SPACE AND TIME COMPLEXITIES OF SOME METHODS OF FUNCTION INTERPOLATION

Octavian Dogaru
Professor, "Constantin Brancusi" University, Faculty of Engineering, Tg. Jiu, Romania

Abstract. In this paper we propose calculate and compare the calculation times and memory spaces to determine the value of interpolation polynomial in a point *a* that differs from the interpolation points of the follow methods of function interpolations: Lagrange' method, Aitken's technics and the Newton's method. In general Aitken's technics is light more quickly than Lagrange and Newton formulas. Lagrange method uses the fewest space of memory then Aitken' s technics and then Newton's method.

Keywords. Function interpolation, polynomials, algorithms, C++ programs, time and space complexities.

1.Introduction

Let $f:[c,d]\to\mathbf{R}$, $y=f(x)$ be a real function of real argument and in the interval [c,d] are specified $n+1$ points $c<x_0<x_1<\ldots<x_n<d$ and the values of this function at this points $y_0,y_1,\ldots,y_n$, that is *f* is given as a table

x	x_0	x_1	…	x_n
y	y_0	y_1	…	y_n

and we wish to calculate the value of this function in a point $a\in[x_0,x_n]$ which is not an interpolation point.

The *f(a)* value will be approximate with the value of the interpolation polynomial of degree not higher than *n* built on the *n*+1 points.

We consider the Lagrange interpolation polynomial built on the *n*+1 points, the Aitken representation technics and Newton formula of the same polynomial.

The Lagrange interpolation polynomial. The Lagrange interpolation polynomial of degree not higher than *n* built on the *n*+1 points is

$$L_n(x)=\sum_{i=0}^{n}\frac{(x-x_0)(x-x_1)\ldots(x-x_{i-1})(x-x_{i+1})\ldots(x-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)}y_i$$

One approximates $f(a)$ by $L_n(a)$. Therefore,

$$f(a)\approx L_n(a)=\sum_{i=0}^{n}\frac{(a-x_0)(a-x_1)\ldots(a-x_{i-1})(a-x_{i+1})\ldots(a-x_n)}{(x_i-x_0)(x_i-x_1)\ldots(x_i-x_{i-1})(x_i-x_{i+1})\ldots(x_i-x_n)}y_i=$$

$$=\sum_{i=0}^{n}\left(\frac{a-x_0}{x_i-x_0}\right)\left(\frac{a-x_1}{x_i-x_1}\right)\ldots\left(\frac{a-x_{i-1}}{x_i-x_{i-1}}\right)\left(\frac{a-x_{i+1}}{x_i-x_{i+1}}\right)\ldots\left(\frac{a-x_n}{x_i-x_n}\right)y_i\ . \qquad (1)$$

This last form will be used in the program.

Aitken. interpolation technics. If it is not necessary the Lagrange polynomial expression, but only its value in a point $a\in[x_0,x_n]$, $a\neq x_i$, i=0,1,…,*n* then it may be used Aitken. interpolation technics. In this meaning one constructs the polynomials of first degree $L_{01}(x)$, $L_{12}(x)$,…,$L_{n-1,n}(x)$ where

$$L_{01}(x)=\frac{\begin{vmatrix} y_0 & x_0-x \\ y_1 & x_1-x \end{vmatrix}}{x_1-x_0},\ \ldots,\ L_{n-1,n}(x)=\frac{\begin{vmatrix} y_{n-1} & x_{n-1}-x \\ y_n & x_n-x \end{vmatrix}}{x_n-x_{n-1}}.$$

The above polynomials $L_{i,i+1}(x)$, i=0,…,*n*-1 have the properties $L_{i,i+1}(x_i)=y_i$ and $L_{i,i+1}(x_{i+1})=y_{i+1}$, therefore they solve the problem of interpolation at the points x_i, x_{i+1}. Starting from these polynomials, one constructs the second degree polynomials

$$L_{i,i+1,i+2}(x)=\frac{\begin{vmatrix} L_{i,i+1}(x) & x_i-x \\ L_{i+1,i+2}(x) & x_{i+2}-x \end{vmatrix}}{x_{i+2}-x_i},\ i=0,1,\ldots,n-2$$

which take in the points x_i,x_{i+1},x_{i+2} respectively the values y_i, y_{i+1}, y_{i+2}, i=0,1,…,*n*-2. In the same manner one builts the polynomial of degree not higher *n*

$$L_{012\ldots n}(x)=\frac{\begin{vmatrix} L_{01\ldots n-1}(x) & x_0-x \\ L_{12\ldots n}(x) & x_n-x \end{vmatrix}}{x_n-x_0}. \qquad (2)$$

This polynomial takes the y_i values in the points x_i, therefore it is an interpolation polynomial. Therefore we may take $f(a)\approx L_{012\ldots n}(a)$.

These elements may be arranged as a matrix of (*n*+1) x (*n*+3) type

x	**x-a**	**y**	$\mathbf{L_{i,i+1}}$	$\mathbf{L_{i,i+1,i+2}}$	$\mathbf{L_{i,i+1,i+2,i+3}}$
x_0	x_0-a	y_0	L_{01}	L_{012}	$\underline{L_{0123}}$
x_1	x_1-a	y_1	L_{12}	L_{123}	…
x_2	x_2-a	y_2	L_{23}	…	
x_3	x_3-a	y_3	…		
…	…	…			

Because the calculation of the elements of the column $L_{i,i+1}$ requires the precedent column vector *y* and the column vector *x-a*, the column vector $L_{i,i+1,i+2}$ uses the precedent column vector $L_{i,i+1}$ and the vector *x-a*, the column vector $L_{i,i+1,i+2,i+3}$ uses the precedent column vector $L_{i,i+1,i+2}$ and the vector *x-a* etc, in the program it will be used an auxiliary vector *w* in which one calculates and stores the temporary elements of the column vectors $L_{i,i+1}$, $L_{i,i+1,i+2}$, $L_{i,i+1,i+2,i+3}$ etc.

Finally, the value $L_{012\ldots n}(a)$=w(0).

Using the vector *w* one reduces the storage space from (*n*+1) x (*n*+3) locations to 4(*n*+1).

The new table is following

x	**x-a**	**y**	**w**
x_0	x_0-a	y_0	$\underline{w_0}$
x_1	x_1-a	y_1	w_1
x_2	x_2-a	y_2	w_2
x_3	x_3-a	y_3	w_3
…	…	…	…
x_n	x_n-a	y_n	w_n

The Newton interpolation polynomial. The Newton's interpolation formula for unequally spaced values of the argument is

$$N(x)=y_0+(x-x_0)[x_0,x_1]+\ldots+(x-x_0)(x-x_1)\ldots(x-x_{n-1})[x_0,x_1,\ldots,x_n] \qquad (3)$$

where $[x_0,x_1,\ldots,x_i]$, $i=1,2,\ldots,n$ are the divided differences of order *i* respectively.

To calculate the value of the polynomial N(**a**), $a\neq x_i$, $a\in[x_0,x_n]$, it is necessary to built the table of the divided differences of different orders

x	**y**	D i v i d e d d i f f e r e n c e s			
		1st	2nd	…	*n*nd
x_0	$\mathbf{y_0}$	$\mathbf{[x_0,x_1]}$	**[x0,x1,x2]**	…	$\mathbf{[x_0,x_1,\ldots,x_n]}$
x_1	y_1	$[x_1,x_2]$	[x1,x2,x3]		
x_2	y_2	$[x_2,x_3]$	.		
.	.	.	$[x_{n-2},x_{n-1},x_n]$		
.	.	$[x_{n-1},x_n]$			
x_n	y_n				

Generally, if one builts the divided differences of order *k*, then the divided differences of order *k+1* one builts by the formula

$$[x_{i-1},x_i,\ldots x_{i+k}]=\frac{[x_i,x_{i+1},\ldots,x_{i+k}]-[x_{i-1},x_i,\ldots,x_{i+k-1}]}{x_{i+k}-x_{i-1}}$$

Because the elements of first row of the table excepting x_0 enter in the Newton's formula, the entire table excepting the column *x* must be kept in a inferior triangular matrix *d* of order *n+1*. Now, the value of N(a) is

$$N(a)=y_0+(a-x_0)[x_0,x_1]+(a-x_0)(a-x_1)[x_0,x_1,x_2]+\ldots+(a-x_0)(a-x_1)\ldots(a-x_{n-1})\,[x_0,x_1,\ldots,x_n].$$

The matrix *d* of order n+1 will be built by the following C++ function *difdiv*

```
float x[20],y[20],d[20][20];
int n;
…
void divdif()
{
      int i,j,k,nj;
      for(i=0;i<=n;i++)d[i][0]=y[i];
      for(j=1;j<=n;j++){
            nj=n-j; k=j;
            for(i=0;i<=nj;i++){
               d[i][j]=(d[i+1][j-1]-d[i][j-1])/(x[k]-x[i]);
               k++;
            }
      }
}
```

2.The C++ implementation

One will write the C++ program for the calculations of $L_n(a)$, $L_{012\ldots n}(a)$ and N(a), simultaneous calculating and their running times. Using the vector *xa*(0:n), where $xa_i=x_i-a$ in Lagrange and Newton polynomials, in the last form presented above, these may be written as

$$L_n(a)=\sum_{i=0}^{n}\left(\frac{-xa_0}{x_i-x_0}\right)\left(\frac{-xa_1}{x_i-x_1}\right)\ldots\left(\frac{-xa_{i-1}}{x_i-x_{i-1}}\right)\left(\frac{-xa_{i+1}}{x_i-x_{i+1}}\right)\ldots\left(\frac{-xa_n}{x_i-x_n}\right)y_i \qquad (4)$$

and

$$N(a)=y_0+(-xa_0)[x_0,x_1]+\ldots+(-xa_0)(-xa_1)\ldots(-xa_{n-1})[x_0,x_1,\ldots,x_n] \qquad (5)$$

respectively.

The program, written in C++, will use these expressions and it will determine and the calculus times of every method.

```
#include<iostream.h>
#include<time.h>
#include<dos.h>
double x[20],y[20],xa[20],w[20],a;
int n;

void citire(double z[])
{
      for(int i=0;i<=n;i++)cin>>z[i];
}

void Lagrange(double &LL)
{
      double L=0,t1,t2;  int j;
      for(int i=0;i<=n;i++){
            t1=t2=1;
            for(j=0;j<=i-1;j++)t1*=-xa[j]/(x[i]-x[j]);
            for(j=i+1;j<=n;j++)t2*=-xa[j]/(x[i]-x[j]);
            L+=t1*t2*y[i];
      }
      LL=L;
}

void Aitken(double &AA)
{
      int k=1,nj,i;
      for(int j=1;j<=n;j++){
            nj=n-j;
```

```
           for(i=0;i<=nj;i++)w[i]=(w[i]*xa[i+k]-
                w[i+1]*xa[i])/(x[i+k]-x[i]);
           k++;
      }
      AA=w[0];
}
void Newton(double &NN)
{

      int i,j,k,nj; float d[20][20];
      for(i=0;i<=n;i++)d[i][0]=y[i];
      for(j=1;j<=n;j++){
           nj=n-j;
           k=j;
           for(i=0;i<=nj;i++){
              d[i][j]=(d[i+1][j-1]-d[i][j-1])/(x[k]-x[i]);
              k++;
           }
      }

      float N1=0;
      for(i=n;i>=0;i--)N1=N1*(-xa[i])+d[0][i];
      NN=N1;
}
void main()
{
      cout<<"n="; cin>>n; cout<<"a="; cin>>a;
      cout<<"vector x: "; citire(x);
      cout<<"vector y: "; citire(y);
      for(int i=0;i<=n;i++)xa[i]=x[i]-a;
      //vector w is initialized with y
      for(i=0;i<=n;i++)w[i]=y[i];
      double LL,AA,NN;
      clock_t  s,t;

      s=clock();Lagrange(LL);t=clock();
      cout<<" value Lagrange="<<LL<<
            "  time Lagrange="<<t-s<<endl;
      s=clock();Aitken(AA);t=clock();
      cout<<" value Aitken=  "<<AA<<
            "   time Aitken="<<t-s<<endl;
      s=clock();Newton(NN);t=clock();
      cout<<" value Newton=  "<<NN<<
            "   time Newton="<<t-s<<endl;
}
```

Example. Let $y=f(x)$, $x=\{0, 0.2, 0.3, 0.4, 0.7, 0.9\}$, $y=\{132.651, 148.877, 157.464, 166.375, 195.112, 216.000\}$. Compute the value $f(0.6)$.By the above program we have

RUN

```
n=5
a=0.6
vector x:  0        0.2      0.3      0.4      0.7      0.9
vector y:  132.651  148.877  157.464  166.375  195.112  216.000
value Lagrange=185.193   time Lagrange=37
value Aitken=  185.193   time Aitken=36
value Newton=  185.193   time Newton=37
```

3.Calculus of the number of operations

We shall take into consideration only the operations realized on floating point data.

Lagrange's method. The number of operations realized by the C++ function *Lagrange* for the determination of value of Lagrange's polynomial $L_n(a)$, using for $L_n(a)$ the formula (3) it is of $2(n+1)^2$ multiplications and divisions, $n(n+1)$ additions and subtractions and $(n+1)(n+3)$ assignments of floating points numbers.

Aitken's technics. To determine the number of operations realized by Aitken technics implemented by C++ function *Aitken* one may observe that for each j=1, 2,…, n there are respectively n, n-1,…,1 values for i. For each value of i one realizes 3 multiplications and divisions plus two subtractions. Alltogether there are $1.5n(n+1)$ multiplications and divisions, $n(n+1)$ subtractions and $0.5n(n+1)$ assignments of floating point numbers.

Newton's method. To determine the number of real operations for Newton's method implemented by C++ function *Newton* described by formula (4) one must observe that there are $0.5n(n+1)$ divisions, $n(n+1)$ subtractions and $0.5n(n+1)$ assignments for the building of the divided differences table and $n+1$ multiplications, $n+1$ additions and $n+3$ assignments for the calculus of N(a). In whole, there are $0.5(n+1)(n+2)$ multiplications and divisions, $(n+1)^2$ additions and subtractions and $0.5(n^2+3n+6)$ assignments. Moreover the method uses and a supplementary matrix.

The three methods are of $O(n^2)$ time complexity.

4.Memory space

It will taken into consideration only the space used for the vectors and matrices.

The space of memory used by the Lagrange's polynomial is the space used for the two vectors x(0;n), y(0;n) and the vector xa(0;n) that is 3(n+1) memory locations.

The space of memory used by the Aitken's technics it is the space for the two vectors x, y and xa like above and the space for the vector w(0;n) used for the calculus of values of $L_{01...}$(a), that is, 4(n+1) memory locations.

The space of memory used by the Newton's polynomial is the space used for the vectors x(0;n), y(0;n), xa(0:n) and moreover the matrix d of order n+1.

5.Conclusions

The number of multiplication and division in Lagrange's method is light greater than in Aitken's and Newton's methods. The number of additions and subtractions are the same in the Lagrange and Aitken methods but lesser that in Newton method. The number of assignments is the least in Aitken's method. The Aitken's technics seems to be faster.

The running times of the three methods are sensible equals with a light advantage for Aitken and Newton technics which may be preferable Lagrange formula.

The Lagrange's method used the fewest memory locations and the most the Newton's method.

BIBLIOGRAPHY

[1].Berezin, I., Jidkov, N., *Metodie vicislenii*, Editura Mir, Moscova, 1971
[2].Demidovich, B.P., Maron, I.A., *Computational Mathematics*, Mir Publisher, Moskow, 1973