Silesian University of Technology
Faculty of Automatic Control,
Electronic and Computer Science
Direction: Macrocourse sem.4

Gliwice
Academic year 2018/2019

Numerical Methods
Laboratory Exercise

Ex.2 Calculation of function value

Group 2, Section 4
Mateusz Warmuz
Szymon Skorupa
Exercise preformed on: 7.03.2019

1. Comparison time needed to calculate function value using Horner scheme and by substitution at given point x.

    - Horner scheme

$$\begin{cases} b_n = a_n \\ b_{n-1} = a_{n-1} + b_n x \\ b_{n-2} = a_{n-2} + b_{n-1} x \\ \cdots \\ b_0 = a_0 + b_1 x = W_n(x) \end{cases}$$

    - Substitution formula

$$W_n(x) = \sum_{i=0}^{n-1} a_i x^i$$

Code:

```
#include <iostream>
#include <math.h>
#include <ctime>

using namespace std;

double calculate_value(double*a, int n, double x){
    double value=0;
    for(int i=n;i>=0;i--){
        value=value + a[i]*pow(x,i);
    }
    return value;
}


double calculate_horner(double* a, double* b, int n, double x){
    double value=0;
    a[n]=b[n];

    for(int i=n-1;i>=0;i--){
        b[i]=a[i]+b[i+1]*x;
    }
    value=b[0];
    return value;
}
int main(){
    int n = 0;
    double *a;              //polynomial coefficient
    double *b;              //horner coefficient
    double x;               //argument in whcich will be calculated
    double limit=10000000;
    time_t beggining, end;

    cout<<"Give order of polynomial: ";
    cin>>n;
    n = n+1;

    a = new double[n];
    b = new double[n];
    for(int i=n-1;i>=0;i--){
        cout<<"Give coefficient a"<<i<<": ";
        cin>>a[i];
    }

    cout<<"Give the value of the argument: ";
    cin>>x;
    cout <<endl<< "Function value using Horner: " <<
calculate_horner(a,b,n,x)<<endl;

    time(&beggining);
    for (int i = 0; i < limit; i++)
    {
        calculate_horner(a,b,n,x);
    }
    time(&end);
    cout << "Horner time: " << difftime(end, beggining) << endl;

    time(&beggining);
    for (int i = 0; i < limit; i++)
    {
        calculate_value(a,n,x);
    }
    time(&end);
    cout<<"Function value using substitution: "<<calculate_value(a,n,x)<<endl;;
    cout << "Substitution time: " << difftime(end, beggining) << endl;
}
```

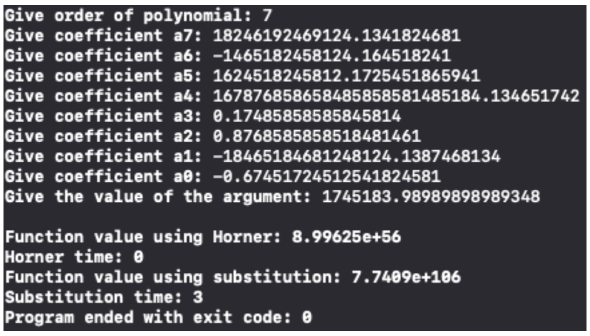
We implement a for loop which repeat 10^7times algorithm for Horner scheme and calculation by substitution. Below we have a table with arithmetic means of times which was requirement to calculate the algorithm

	time in second	
Order	Horner	Substitution
2	0,40	1,30
3	0,30	1,20
4	0,50	1,90
5	0,44	2,00
6	0,78	2,44
7	0,56	2,33
8	0,22	2,67
9	0,67	2,78
10	0,56	3,33
15	1,22	4,78
20	1,11	6,33

Conclusions:
On the table above we can notice that Horner scheme is much more efficiency than calculation by substitution. There were no difference if the coefficient were small or vey big (in millions).The difference grows rather linearly. In higher order of polynomials sometimes compilator throws different results of calculating function value. To verify result I've used to Matlab to compare results, as we see Horner scheme calculate correctly the function value.

Compilator

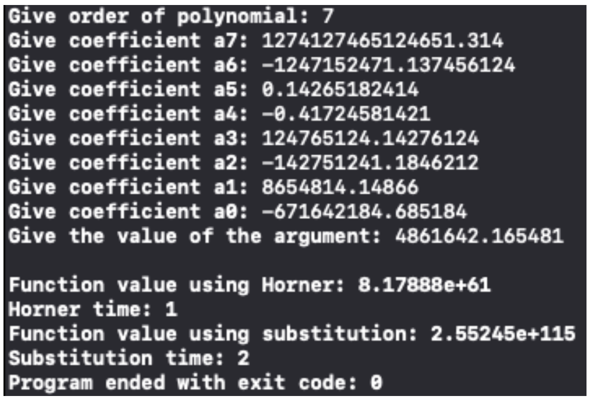
Matlab



```
Give order of polynomial: 7
Give coefficient a7: 18246192469124.1341824681
Give coefficient a6: -1465182458124.164518241
Give coefficient a5: 1624518245812.1725451865941
Give coefficient a4: 16787685865848585858581485184.134651742
Give coefficient a3: 0.17485858585845814
Give coefficient a2: 0.87685858585518481461
Give coefficient a1: -18465184681248124.1387468134
Give coefficient a0: -0.67451724512541824581
Give the value of the argument: 1745183.98989898989348

Function value using Horner: 8.99625e+56
Horner time: 0
Function value using substitution: 7.7409e+106
Substitution time: 3
Program ended with exit code: 0
```

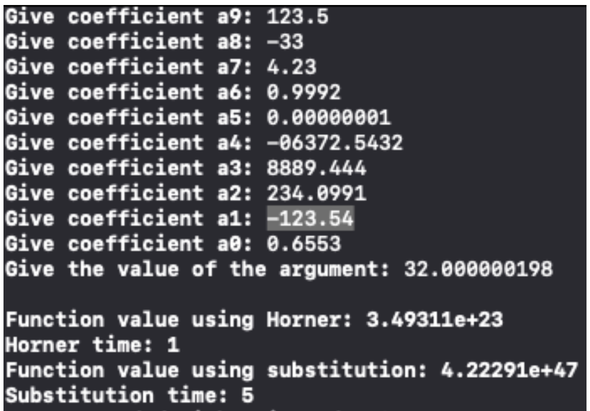
```
8.9963e+56
```



```
Give order of polynomial: 7
Give coefficient a7: 1274127465124651.314
Give coefficient a6: -1247152471.137456124
Give coefficient a5: 0.14265182414
Give coefficient a4: -0.41724581421
Give coefficient a3: 124765124.14276124
Give coefficient a2: -142751241.1846212
Give coefficient a1: 8654814.14866
Give coefficient a0: -671642184.685184
Give the value of the argument: 4861642.165481

Function value using Horner: 8.17888e+61
Horner time: 1
Function value using substitution: 2.55245e+115
Substitution time: 2
Program ended with exit code: 0
```

```
8.1789e+61
```



```
Give coefficient a9: 123.5
Give coefficient a8: -33
Give coefficient a7: 4.23
Give coefficient a6: 0.9992
Give coefficient a5: 0.00000001
Give coefficient a4: -06372.5432
Give coefficient a3: 8889.444
Give coefficient a2: 234.0991
Give coefficient a1: -123.54
Give coefficient a0: 0.6553
Give the value of the argument: 32.000000198

Function value using Horner: 3.49311e+23
Horner time: 1
Function value using substitution: 4.22291e+47
Substitution time: 5
Program ended with exit code: 0
```

```
3.4931e+23
```

2. For polynomials of degree at least 4 (both of even and odd degree) of known roots perform the analysis of accuracy of determining interval (a,b) containing all its real roots. $(x_0 \leq x_1 \leq \ldots \leq x_n - 1 \leq x_n)$Analyze errors $\Delta_1 = a - x_0, \Delta_2 = b - x_n$

Code

```
#include <iostream>
#include <math.h>


using namespace std;

void interval (double* a, int n){

    double x=0;
    double* b = new double[n+1];
    double alfa,beta;
    double epsilon=0.0001;
    bool condition = true;
    bool found;

    double* p= new double[n+1];

    //------------------------- BETA
    b[n]=a[n];
    while(condition){
        x += epsilon;
        found = false;
        for(int i=n-1; i>=0; i--){
            b[i]=a[i]+b[i+1]*x;
            if(b[i]<=0){
                found=true;
                break;
            }
        }
        if(!found)
            condition=false;
    }
    beta = x;

  //-------------------------- ALFA


    for(int i=n; i>=0; i--){
        p[i] = pow(-1,n) * pow(-1,i)*a[i];

        cout<<p[i]<<endl;
    }

    x=0;
    condition=true;

    while(condition){
        x += epsilon;
        found = false;
        for(int i=n-1; i>=0; i--){
            b[i]=p[i]+b[i+1]*x;
            if(b[i]<=0){
                found=true;
                break;
            }
        }
if(!found)
            condition=false;
    }
    alfa = -x;
    cout<<"All real roots are contained in the interval <"<<alfa<<",
"<<beta<<">"<<endl;
}

int main(){
    int n;
    cout<<"Give order of polynomial: ";
    cin>>n;
    cout<<endl;

    double*a =new double[n+1];

    for(int i=n;i>=0;i--){
        cout<<"Give the coefficient a"<<i<<": ";
        cin>>a[i];
    }

 interval(a,n);
}
```

- Roots symmetrical with respect to 0

    -small step (accuracy=0.000001)

$f(x) = (x+2)(x+1)(x-1)(x-2) = x^4 - 5x^2 + 4$

$result = (-2.23607{,}2.23607)$ $\quad solution = (-2{,}2)$ $\quad error = 0.23607$

$f(x) = x(x+5)(x+1)(x-1)(x-5) = x^5 - 26x^3 + 25x$

$result = (-5.09902{,}5.09902)$ $\quad solution = (-5{,}5)$ $\quad error = 0.09902$

-big step(accuracy=0.1)

$f(x) = (x+2)(x+1)(x-1)(x-2) = x^4 - 5x^2 + 4$

$result = (-2.3{,}2.3)$ $\quad solution = (-2{,}2)$ $\quad error = 0.3$

$f(x) = x(x+5)(x+1)(x-1)(x-5) = x^5 - 26x^3 + 25x$

$result = (-5.1{,}5.1)$ $\quad solution = (-5{,}5)$ $\quad error = 0.1$

- Roots asymmetrical with respect to 0 which roots considerably different in values

    -small step (accuracy=0.000001)

$f(x) = (x+10)(x+2)(x-2)(x-5) = x^4 + 5x^3 - 54x^2 - 20x + 200$

$result = (-10.2621{,}5.49318)$ $\quad solution = (-10{,}5)$ $\quad error = 0.49318$

$f(x) = x(x+23)(x+2)(x-1)(x-7) = x^5 + 17x^4 - 147x^3 - 193x^2 + 322x$

$result = (-23.3071{,}7.18763)$ $\quad solution = (-23{,}7)$ $\quad error = 0.3071$

-big step(accuracy=0)

$f(x) = (x+10)(x+2)(x-2)(x-5) = x^4 + 5x^3 - 54x^2 - 20x + 200$

$result = (-10.3, 5.5)$ $solution = (-10, 5)$ $error = 0.5$

$f(x) = x(x+23)(x+2)(x-1)(x-7) = x^5 + 17x^4 - 147x^3 - 193x^2 + 322x$

$result = (-23.4, 7.2)$ $solution = (-23, 7)$ $error = 0.4$

- Multiple roots

    -small step (accuracy=0.000001)

$f(x) = x^2(x-5)^2(x+2) = x^5 - 8x^4 + 5x^3 + 50x^2$

$result = (-2, 8)$ $solution = (-2, 5)$ $error = 3$

$f(x) = (x+1)^2(x-7)^2 = x^4 - 12x^3 + 22x^2 + 84x + 49$

$result = (-1.79583, 12)$ $solution = (-1, 7)$ $error = 5$

-big step(accuracy=0)

$f(x) = x^2(x-5)^2(x+2) = x^5 - 8x^4 + 5x^3 + 50x^2$

$result = (-2, 8.1)$ $solution = (-2, 5)$ $error = 3.1$

$f(x) = (x+1)^2(x-7)^2 = x^4 - 12x^3 + 22x^2 + 84x + 49$

$result = (-1.8, 12.1)$ $solution = (-1, 7)$ $error = 5.1$

- Roots of the same sign

    -small step (accuracy=0.000001)

$f(x) = (x+5)(x+3)(x+1)^3 = x^4 + 10x^3 + 32x^2 + 38x + 15$

$result = (-10, 1 \cdot 10^{-6})$ $solution = (-5, -1)$ $error = 5$

$f(x) = (x-2)(x-5)(x-10)(x-11)(x-20) = x^5 - 48x^4 + 827x^3 - 6320x^2 + 20700x - 22000$

$result = (-1 \cdot 10^{-6}, 48)$ $solution = (2, 20)$ $error = 28$

-big step(accuracy=0)

$f(x) = (x+5)(x+3)(x+1)^3 = x^4 + 10x^3 + 32x^2 + 38x + 15$

$result = (-10.1, 0.1)$ $solution = (-5, -1)$ $error = 5.1$

$f(x) = (x-2)(x-5)(x-10)(x-11)(x-20) = x^5 - 48x^4 + 827x^3 - 6320x^2 + 20700x - 22000$

$result = (-0.1, 48)$ $solution = (2, 20)$ $error = 28$

Conclusions

As we see algorithm works correctly. In every result roots of the function are contained in the interval. For symmetrical roots with respect to 0 the accuracy is very small, in case of asymmetric roots the error is only a bit bigger. The error becomes much bigger when we have k-fold roots. The least efficiency of the algorithm was observed when the polynomial with the same roots of the same sign was considered. The result become from assumption that the upper limit is positive and lower limit is negative. Therefore when all roots are positive than the lower limit is the last positive number (0-step) and reverse (0+step)