Silesian University of Technology
Faculty of Automatic Control,
Electronics and Computer Science

Gliwice
Academic year 2007/2008

Direction: Macrocourse sem. 4

Numerical Methods
Laboratory exercises

Ex. 9: Approximate solving of non-linear equations

A report by:
Anna GRZESIK
Tomasz JASTRZĄB

Group 3 Section 2

Exercise performed on:
14.04.2008

Task description

For Newton and regula falsi methods of approximate solving of non-linear equations:

- Compare speed of convergence as a function of differing accuracy ε
- Compare stop criteria
  - $|x_i - x_{i-1}| < \varepsilon$
  - $|f(x_i)| < \varepsilon$
- Analyze influence of numerical differentiation on Newton method

Choose functions differing in slope, i.e. with slow, medium and fast rise.

- Analyze speed of convergence of Bernoulli method as a function of:
  - roots distribution
    - $x_1 \rightarrow x_2$

x_1	x_2	x_3	x_4
± 30	11	-7	2
± 11.5	11	-7	2
± 11.1	11	-7	2
± 11.01	11	-7	2

    - complex roots z_2 z_3:
      x_1 z_2 z_3 x_4; $|z_2| = |z_3| \rightarrow x_1$
    - changing multiplicity of x_1 and x_2 (separately)
  - initial vector (change $[y_0, y_1, \ldots, y_{n-1}]$ for given polynomial)

Results

For the purpose of analysis of Newton and regula falsi method we have chosen following functions:

- $f_1(x) = e^x - 1$
- $f_2(x) = e^{0.5x} - 1$
- $f_3(x) = e^{0.2x} - 1$

The reason for choosing three different functions is connected with checking the influence of the function choice on the computational results of our analysis. As can be expected different types of functions (in this case functions differing in the slope – from fastest to slowest rise) will yield different results as far as convergence is concerned.

Starting from regula falsi method we may present following results of our computations concerning the influence of demanded accuracy ε on the convergence of the process of solving the nonlinear equation given as $f_i(x) = 0$, for i = 1, 2, 3. The analysis conditions taken into account are: **[a, b] = [-2, 2]**, **stop criterion** $|f(x_i)| < \varepsilon$ **.**

	$f_1(x)$		$f_2(x)$		$f_3(x)$	
	x_k	k	x_k	k	x_k	k
ε = 0.01	-0.007666	16	-0.0128835	6	-0.0139471	3
ε = 0.001	-0.000806963	22	-0.0009422	9	-0.002605	4
ε = 0.00001	$-8.91551*10^{-6}$	34	$-1.20278*10^{-5}$	14	$-1.69549*10^{-5}$	7

Table 1 *Regula falsi method*

Note: The table contains the approximate solution obtained x_k as well as number of iterations k necessary to fulfill stop criterion with demanded accuracy.

As we can observe from the table presented above choice of the function have a pretty great influence on the convergence and the approximate solution itself. The best solution was obtained for the function with fastest rise, however, it happened on the cost of bigger number of iterations (compare for instance 16 and 6 or even 3). Apart from particular function choice the demanded accuracy has also quite a significant influence on the solution and number of iterations. Increasing accuracy ε yields better results as far as the value of approximate solution is concerned, which is obvious because we need to be closer to the solution to terminate calculations, but, what is also quite understandable and expected, it happens at a cost of increased number of iterations (see e.g. 16 and 34).

We may also observe that variation of accuracy and the way it affects results of computations strongly depends on the function chosen. Changing ε by 3 orders of magnitude yields the change in number of iterations by 18 in case of f_1 and only by 4 in case of f_3. However, interestingly, the relative change of numbers of iterations seems to be more immune to function changes. We can compare those relative changes using the following table:

ε variation	$f_1(x)$	$f_2(x)$	$f_3(x)$
0.01 → 0.001	1.375	1.500	1.333
0.001 → 0.00001	1.545	1.556	1.75
0.01 → 0.00001	2.125	2.333	2.333

Table 2 *Relative changes of number of iterations with respect to differing accuracy*

As can be observed from the table presented above relative changes of numbers of iterations are similar in case of all functions. Therefore we can conclude that although absolute changes of number of iterations for the three examined functions (with ε as a parameter) may seem to be significant, relative changes are rather immune to this situation.

Now we shall check the way different stop criteria affect the approximate solution value and number of iterations necessary to reach the solution with chosen accuracy. The conditions for analysis are as follows: **[a, b] = [-2, 2], ε = 0.001** and here are the results:

Stop criteria	$f_1(x)$		$f_2(x)$		$f_3(x)$	
	x_k	k	x_k	k	x_k	k
$\lvert f(x_i)\rvert < \varepsilon$	-0.000806963	22	-0.0009422	9	-0.002605	4
$\lvert x_i - x_{i-1}\rvert < \varepsilon$	-0.0017096	20	-0.000393883	10	$-9.08124*10^{-5}$	6

Table 3 *Comparison of different stop criteria*

Basing on this small sample of data we can draw a few conclusions concerning influence of stop criteria on the final results of our computations. As previously we can observe at the first glance that the choice of particular function affects the results in a quite significant way. For $f_1(x)$ (function with fastest rise), the second stop criterion seems to be more effective as far as number of iterations is concerned, however the approximate solution itself is worse in the second case. For the functions $f_2(x)$ (middle rise speed) and $f_3(x)$ (slowest rise) we observe that the number of iterations required for achievement of the solution increases with respect to the first stop criterion case, but on the other hand the magnitude of result is improved, especially in case of slowest rising function.

Moreover the absolute change in the number of iterations (and relative change as well) is not really great in comparison to improvement of approximate solution value. For instance let us compare that for function $f_3(x)$ relative change of the number of iterations equals 1.5,

whereas the accuracy of result changed almost 30 times (i.e. relative change of the approximate solution values equals 30).

Therefore we can conclude that differing stop criterion affects final results in quite a significant way, mainly changing the accuracy of the solution, on the cost of slightly increased number of iterations. However, as mentioned previously the way stop criterion affects the magnitude of approximate solution as well as number of iterations is strongly associated with the choice of particular function.

Leaving considerations about regula falsi method, we may proceed to similar analysis of Newton method results, however performing also an additional test concerning the influence of numerical and analytical differentiation. We will again start with analysis of both demanded accuracy ε influence and different stop criteria. We shall gather our results in a form of few tables together with a short comment describing observations we have made. Coming back to the point of our analysis we are going to present the way accuracy ε affects computational process of finding an approximate solution of non-linear equation of the form stated at the beginning.

Analysis conditions are as follows: **[a, b] = [-2, 2], stop criterion** $|f(x_i)| < \varepsilon$

	$f_1(x)$		$f_2(x)$		$f_3(x)$	
	x_k	k	x_k	k	x_k	k
ε = 0.01	0.00393565	4	0.0035384	3	0.0120775	2
ε = 0.001	$7.73473*10^{-6}$	5	$3.12832*10^{-6}$	4	$1.45744*10^{-5}$	3
ε = 0.00001	$7.73473*10^{-6}$	5	$3.12832*10^{-6}$	4	$1.45744*10^{-5}$	3

Table 4 *Newton method*

Applying Newton method for the purpose of solving non-linear equations given we have noticed a few things. First one concerns the influence of the function choice on the approximate solution magnitude. As we can observe the best solution in terms of magnitude is obtained for the 'medium-rise' function $f_2(x)$, while the worst case is the slowest rise function $f_3(x)$. It somehow confirms our expectations concerning the Newton method, but only partially. For the function that is flat in the vicinity of the root the Newton method should yield worse results that for the steep function, which is the case in our examination (slowest rise is the worst case scenario). However the best result is obtained for the 'medium-rise' function, not for the steepest one, which in a way differs from the theoretical assumption. Although absolute differences between solutions we obtained are not very significant, relative values have more importance.

As far as the influence of demanded accuracy is concerned, we observe improvement of results, however only up to a certain point, because then the results remain the same for different accuracies. Moreover, increase in number of iterations is relatively small (only by one iteration at most). Therefore we may conclude that for the particular choice of functions we presented and with application of Newton method, increasing accuracy does not influence the obtained results much.

Checking the influence of different stop criteria for the following analysis conditions **[a, b] = [-2, 2]; ε = 0.001** we have found out following things gathered in the table below and described later.

Stop criteria	$f_1(x)$		$f_2(x)$		$f_3(x)$	
	x_k	k	x_k	k	x_k	k
$\lvert f(x_i)\rvert < \varepsilon$	$7.73473*10^{-6}$	5	$3.12832*10^{-6}$	4	$1.45744*10^{-5}$	3
$\lvert x_i - x_{i-1}\rvert < \varepsilon$	$2.98128*10^{-11}$	6	$2.39208*10^{-12}$	5	$2.13369*10^{-11}$	4

Table 5 *Comparison of different stop criteria*

Now we may compare the influence of both stop criteria on the final results. At first glance we can see that we gained tremendous improvement as far as magnitude of the results are concerned. The results differ by 5 (or even 6) orders of magnitude for both criteria, and we may say that although approximate solution for the first stop criterion is really close to the actual root, in the second case the closeness is even more visible. It happens on the cost of one more iteration, but comparing the improvement, which, let us emphasize once more, is by five orders of magnitude, change by one iteration is a negligible drawback.

Again the best result is obtained for the second function chosen, which may enable us drawing following conclusion. The overall behaviour (i.e. function choice influence) is not changed by difference in stop criterion decision.

Finally we are going to check the influence of numerical differentiation on obtained results. For this purpose we set following analysis conditions **[a, b] = [-2, 2], stop criterion** $\lvert f(x_i)\rvert < \varepsilon$ **, ε = 0.001**, and parameters of numerical differentiation **h = 0.1, n = 5.**

Differentiation method	$f_1(x)$		$f_2(x)$		$f_3(x)$	
	x_k	k	x_k	k	x_k	k
Analytic	$7.73473*10^{-6}$	5	$3.12832*10^{-6}$	4	$1.45744*10^{-5}$	3
Numerical	$7.7433*10^{-6}$	5	$3.12605*10^{-6}$	4	$1.45611*10^{-5}$	3

Table 6 *Influence of numerical and analytical differentiation*

As we can observe basing on the exemplary data provided in the table above, numerical differentiation in this particular case has little influence on approximate solution of non-linear equation, in some cases it can even slightly improve the value itself. Moreover number of iterations remains the same for both approaches to calculation of first derivatives. Although it may also result from the choice of functions taken into consideration, because it seems quite natural that numerical differentiation, being less accurate that analytic differentiation should affect our computations in negative manner.

As a final step in our examination process we shall compare efficiency of Newton and regula falsi methods. Our first observation concerns number of iterations necessary to terminate the calculations. It can be easily noticed that Newton method depicts much faster convergence than regula falsi method (compare e.g. 5 and 34 iterations). As far as the absolute magnitude of approximate roots is concerned the results obtained with the use of both methods are quite comparable, however best results are obtained for different functions. What is more choice of the function itself can greatly affect the results in case of regula falsi method, while for Newton method it is of minor importance, nevertheless results vary among different functions. One more interesting property was found, namely in regula falsi method the approximate solution approaches the actual one from the negative side, whereas in case of tangent method the situation is opposite (i.e. approximate solution tends to the exact one from the right – positive – side).

For the Bernoulli method we are supposed to choose few polynomials having different roots, because we shall change those roots in order to check the influence of root distribution on the convergence of our method. Therefore we will present those polynomials below, however instead of presenting them in the usual compact form, we will present them as a product of simple expressions, to visualize the roots we are using:

- $W_1(x) = (x-30)(x-11)(x+7)(x-2)$
- $W_2(x) = (x+11.5)(x-11)(x+7)(x-2)$
- $W_3(x) = (x+11.1)(x-11)(x+7)(x-2)$
- $W_4(x) = (x-11.01)(x-11)(x+7)(x-2)$
- $W_5(x) = (x-30)(x-z_2)(x-z_3)(x-2)$, where $z_2 = 25 + j16$ and $z_3 = 25 - j16$, with modulus $|z_2| = |z_3| \approx 29.68$
- $W_6(x) = (x-30)(x-11)^2(x+7)(x-2)$
- $W_7(x) = (x-30)(x-11)^3(x+7)(x-2)$
- $W_8(x) = (x-30)^2(x-11)(x+7)(x-2)$
- $W_9(x) = (x-30)^3(x-11)(x+7)(x-2)$

Note: For all the cases considered in this part we use following **initial vector [1, 0, …, 0]**.

Using first four polynomials we will check the influence the varying distance between the roots of two consecutive absolute values (i.e. the highest modulus and one after the highest). Results of our computations are presented below (**stop criterion** $|W(x_M)| < \varepsilon$**, ε = 0.001**):

Function	Highest modulus root x_M	Number of iterations k
$W_1(x)$	30	19
$W_2(x)$	-11.50008	349
$W_3(x)$	-11.103757	660
$W_4(x)$	11.023167	625

Table 7 *Influence of distance between x_M and root of second highest modulus*

A few remarks concerning the results presented above are necessary. Although all the results seem to be quite close to the actual ones, in fact only in the first case we were able to obtain demanded accuracy of 0.001. In all other cases the result we placed in the table is the one we have observed after very in-deep analysis, but let us emphasize once more that these results in fact do not fulfill the stop criterion given. However waiting for the stop criterion to be fulfilled it may occur and in some of the cases considered actually happened that the calculations will last infinitely long. Therefore we need to stop the computations manually.

An interesting property was noticed concerning two middle polynomials, namely after some initial period of very large variation of values, they started to settle down around the proper value. Moreover we could observe oscillations around the actual value of the root, tending to the exact root but unfortunately not reaching it. As far as the last function is concerned the oscillating property has vanished, but on the other hand we could observe one-sided tendency towards the expected root value. However, as in previous cases the exact solution was not obtained, because after some time the changes became almost unnoticeable.

What regards number of iterations, we can see that the result for the first case was pretty good, especially as compared to other situations. Frankly speaking we should not even try to analyze remaining cases with respect to the number of iterations, because this number is already huge and it would probably grow even more if we didn't apply 'manual stopping'.

Therefore we may conclude, exactly as we expected that the root distribution concerning closeness of roots of successively high absolute values strongly affects the iterative process. What is more, the closer the two mentioned roots are the harder the situation becomes. We could not find the exact value of the highest modulus root for the situations where the second root in terms of magnitude was really close to the highest one. In case of quite a big difference between roots mentioned above we obtain pretty satisfactory results – exact value of x_M and relatively low number of iterations.

Finally we may say that root distribution in this particular case (i.e. if highest modulus root and the second highest are concerned) have a great influence on the computational results. Moreover too closely situated roots make it impossible to reach the exact value, even in a large number of iterations, because they either oscillate around the root or they simply settle in a certain equilibrium points and switch between them or (as in case 4) they tend to the proper value from one side but they cannot achieve it anyway.

Now we shall find how the introduction of complex conjugate roots influences the results of our calculations. Again, confirming our expectations, the iterative process of finding the solution became divergent. Moreover oscillations around the exact value of x_M are of quite a wide range. We observe very large jumps between successive approximate solutions, but in fact the proper one is not reached even after 500 iterations or more. The stop criterion is not fulfilled and the iterative process once more lasts infinitely long, outlining no solution to the problem.

For the purpose of analysis of the way the results are influenced by changing multiplicity of the roots we will use polynomials $W_6(x) – W_9(x)$. First we will change the multiplicity of the second highest root, then of the first one. The results are again gathered in the table below (**stop criterion** $|W(x_M)| < \varepsilon$**, ε = 0.01**):

Function	Highest modulus root x_M	Number of iterations k
$W_6(x)$	29.978279	23
$W_7(x)$	30	22
$W_8(x)$	30.022501	1334
$W_9(x)$	30.008	1739

Table 8 *Influence of root multiplicity*

Basing on the data provided in the table we may draw a few conclusions as well as some comments. First of all let us recall that for the multiplicity one of both the highest modulus root and the second highest the results were proper, i.e. $x_M = 30$, $k = 19$. Now let us examine the cases when either the x_M or the second root in order has odd multiplicity (in our case it was 3). We may notice that the results obtained for both this cases are correct, with the additional remark that for changed multiplicity of x_2 we get exact result, while for multiplicity of x_M the result is not exact but satisfies stop criterion. As far as number of iterations is concerned increased multiplicity of x_2 does not influence the number of iterations much (compare 19 → 22), but greater multiplicity of highest modulus root made the iterative process last much longer (see 19 → 1739). Therefore our first conclusion may be that in case of odd multiplicities of the roots the iterative process is still convergent, but if x_M is e.g. a triple root the number of iterations significantly increases.

What regards two other cases considered, i.e. polynomials $W_6(x)$ and $W_8(x)$ introducing even multiplicities the situation becomes worse, and generally the process is divergent. Although the magnitudes of x_M seem to be pretty close to the actual value they do not fulfill the stop criterion, even in case of very weak accuracy, e.g. $\varepsilon = 1$. Quite an interesting property was found concerning double root of the second highest modulus. From iteration number 23 the value of x_M became settled and we could not observe any other changes or improvements. On the other hand in case of polynomial $W_9(x)$ the magnitude of highest modulus root changes all the time but those differences become smaller and smaller, so that the iterative process cannot finish properly. In fact it last infinitely long, because at some stage the value either become settled as in case of $W_6(x)$ or the changes are so insignificant that they can't be observed. Therefore looking at the data presented in the table we should bear in mind that actual calculations do not stop at the stage presented but they last infinitely long. Consequently we may say that even multiplicities of roots affect the convergence of iterative process in a significant way, namely making the process divergent.

Finally we may conclude that various changes in roots distribution affect the results of our computations, both as far as the approximate solution itself is concerned and the convergence of the iterative process expressed by the number of iterations necessary to reach the solution. We have observed that for the highest modulus root being relatively 'far away' from the second highest solution as well as for the cases of odd multiplicities of both roots mentioned Bernoulli method yield proper results. However for even multiplicities, smaller distance between two roots of successively highest modulus and complex conjugate roots, with modulus tending to x_M we spot divergence of the computational process. What is also worth pointing out that even in cases of very poor demanded accuracy (like $\varepsilon = 0.1$ or even $\varepsilon = 1$) the convergence is not improved and it still do not provide the solution to our problem.

Our next task was to check the influence of initial vector on the results of our calculations. We will not check all the functions one more time, instead we will choose some representatives of all the groups considered (i.e. 'closeness group', complex conjugate roots and 'multiplicity group'). We will especially focus on divergent cases and we will check whether changing of the initial vector may overcome the failure of the procedure or not, however we are going to present proper cases as well. Therefore for our analysis we will choose following polynomials $W_1(x)$, $W_3(x)$, $W_5(x)$, $W_6(x)$, $W_8(x)$ and $W_9(x)$.

Function	Highest modulus root x_M		Number of iterations k	
	First vector	Second vector	First vector	Second vector
$W_1(x)$	30	30	19	18
$W_3(x)$	-11.099894	-11.100295	644	584
$W_5(x)$	No data	No data	No data	No data
$W_6(x)$	29.978279	29.978279	23	24
$W_8(x)$	30.022501	30.019987	1505	1334
$W_9(x)$	30.008	30.008	1691	1739

Table 9 *Influence of initial vector choice*

Note: First vector is [1, 0,…, 0]; second vector is [-0.5, -0.25, 0, …0],
stop criterion $|W(x_M)| < \varepsilon$**, ε = 0.01**.

Few remarks are necessary concerning the results presented above. First of all we shall comment the row containing 'No data' label. We have written so, because the oscillation, similarly as in the case of the first vector were so large and chaotic that we were not able to spot any value which might have been close to the proper one even with a great number of iterations used. Therefore we decided not to present any numerical values but to comment on this phenomenon. In case of two roots that are close to each other changing the initial vector doesn't reduce the divergence, in fact the influence is almost unnoticeable in this particular case.

Second remark regards those correct situations, which in our case are $W_1(x)$ and $W_9(x)$. Changing the initial vector we mainly affect number of iterations necessary to achieve demanded accuracy. However, as can be viewed from the above table no overall pattern concerning improvement or worsening could be spotted, because for $W_1(x)$ the number of iterations slightly increased, whereas for $W_9(x)$ it has significantly decreased. Nevertheless we may conclude that in convergent cases the change of initial vector influences the iterative process.

What concerns remaining cases, i.e. those which were divergent in case of the first vector, they remained divergent with the use of the second vector. Although the numerical values presented in the table differ among the two vector choices, they are not quite reliable, because the iterative process had to be stopped manually, because of divergence. Therefore we cannot base on the values presented.

To sum up, we may say that the influence of initial vector choice is connected with the polynomial considered, and the other conditions resulting from this fact (e.g. roots distribution). It also depends on the particular choice of the vector, because it may happen that for some vectors the results will be improved, while for other ones they may not represent any new, better features. Generally we expect that the choice of initial vector will not make a divergent case convergent and vice versa, but sometimes we may observe some improvements.

To find the root of the lowest modulus we need to change the polynomial under consideration in the following way $W(x) \to W\left(\frac{1}{x}\right)$, which in fact may be obtained by swapping the a_i coefficients in the following manner $a_0 = a_n$, $a_1 = a_{n-1}$ etc. This way we will obtain new coefficients of our polynomial and we will again search for the highest modulus root of the new function. However, let us observe that the inverse of the highest modulus root for the polynomial $W\left(\frac{1}{x}\right)$ will be the lowest modulus root of the original polynomial W(x). To make those modifications clearer we shall present an example:

Example

Let us assume that $W(x) = x^2 + 9x - 10$, and the roots are -10, 1 ($W(x) = 0$). Then we should provide the polynomial $W\left(\frac{1}{x}\right) = \frac{1}{x^2} + \frac{9}{x} - 10 = \frac{1 + 9x - 10x^2}{x^2} = 0$, so in fact polynomial $1 + 9x - 10x^2 = 0$ and here we can observe swapping of coefficients (and roots become 1 and -0.1 = -1/10). Then we simply apply the standard Bernoulli method and we will find root of the highest modulus $x_M = 1$. Then if we take the inverse of this root we will obtain the root of lowest modulus $x_L = 1$ of the original polynomial.

As far as calculations of all the roots are concerned we have spotted following property, namely this process is very sensitive to the accuracy chosen and it may also happen that it will not converge because of some insignificant (at the first sight) differences between calculated and exact values of the roots. However we will present two exemplary polynomials which yield proper results even with relatively low accuracy i.e. **ε = 0.001**. The polynomials taken into consideration will be as follows:

- $W_1(x) = (x-30)(x-11)(x+7)(x-2)$
- $W_2(x) = (x-10)(x-2)(x+1)$

The results of our computations are gathered in the table below:

Function	$x_1 = x_M$	k	x_2	k	x_3	k	$x_4 = x_L$	k
$W_1(x)$	30	19	11	37	-7	15	2	1
$W_2(x)$	10	22	2.00002	16	×	×	-1.00003	1

Table 10 *All roots of polynomial*

As we can see in the first case the roots are exact, i.e. they really match the expected values, whereas in the second case they are only approximate values, but very close to the exact ones. Moreover the process of finding those roots is relatively quickly convergent, especially as compared to some other cases presented above, requiring over 1000 iterations to reach the solution.

What regards comparison of the x_L (lowest modulus root) values obtained with the use of this approach and with the use of approach described above the results are quite comparable, with the little change in approximate value (e.g. -1.00003 and -1.00002 respectively) or even no change at all (e.g. 2 and 2 in the first case). Therefore we may conclude that both approaches yield similar results however the first method is quicker because to find the lowest modulus root we require 12 or 19 (for the first and second case respectively) iterations while with the second approach mentioned we need 19 + 37 + 15 + 1 = 72 or 22 + 16 + 1 = 39 iterations.

Programme listing:

- ***Newton & regula falsi methods***

```
const int N = 100;
float x[N];
float table[N][N];
int main(int argc, _TCHAR* argv[])
{
	float a=0, b=0, epsilon=0;
	char choice =0;
	for(int i =0; i < N; i++)
		for(int j=0; j < N; j++)
			table[i][j]=0;
	cout << "Choose the method: " << endl;
	cout << "1 - Regula falsi method" << endl;
	cout << "2 - Newton method" << endl;
	choice = getch();
	while((choice != '1') && (choice != '2'))
		choice = getch();
	putch(choice);
	cout << endl;
	switch(choice)
	{
	case '1': cout << "Type in a (lower bound of interval): ";
			cin >> a;
			cout << "Type in b (upper limit of interval): ";
			cin >> b;
			while(a>=b)
			{
			    cout << "Type in value of b greater than a: ";
			      cin >> b;
			}
			cout << "Type in the accuracy epsilon: ";
			cin >> epsilon;
			reg_falsi(a, b, epsilon);
			break;
	case '2': cout << "Type in a (lower bound of interval): ";
			cin >> a;
			cout << "Type in b (upper limit of interval): ";
			cin >> b;
			while(a>=b)
			{
			    cout << "Type in value of b greater than a: ";
			      cin >> b;
			}
			cout << "Type in the accuracy epsilon: ";
			cin >> epsilon;
			Newton(a, b, epsilon);
			break;
	}
	cout << setprecision(8) << fixed;
	system("PAUSE");
	return 0;
}
float f(float x)
{
	//return exp(x)-1;
	//return exp(0.5*x)-1;
	return exp(0.2*x)-1;
}
```

```
float first_der(float x)
{
      //return exp(x);
      //return 0.5*exp(0.5*x);
      return 0.2*exp(0.2*x);
}
float second_der(float x)
{
      //return exp(x);
      //return 0.25*exp(0.5*x);
      return 0.04*exp(0.2*x);
}
float* forward(float table[N][N], int n, float x0, float h)
{//forward difference operators calculation
  for(int i=0; i<=n; i++)
    table[i][0]=f(x0+i*h);
  for(int i=1; i<=n; i++)
    for(int k=n-i; k>=0; k--)
            table[k][i] = table[k+1][i-1]-table[k][i-1];
  return *table;
}
float first_delta(float table[N][N], float h, int n)
{//numerical 1st derivative
  float f1 = 0;
  for(int i=1; i<=n; i++)
    f1 += (1/h)*(powf(-1, i-1)*(1.0/i)*table[0][i]);
  return f1;
}
void reg_falsi(float a, float b, float epsilon)
{
  int i = 1;
  cout << "Subsequent approximations: " << endl;
      x[0] = a - f(a)*(b-a)/(f(b)-f(a));
  cout << x[0] << endl;
  if(fabs(f(x[0])) <= epsilon)
  {
        cout << "The approximated root equals: " << x[0] << endl;
      cout << "The number of iterations equals: " << 1 << endl;
  }
  if(((f(a) < 0) && (second_der(a) < 0)) || ((f(a) > 0) && (second_der(a) >
0)))
  {
     do
       {
            x[i] = x[i-1] - f(x[i-1])*(a - x[i-1])/(f(a)-f(x[i-1]));
            cout << x[i] << endl;
            i++;
       //}while(fabs(f(x[i-1])) > epsilon);
       }while(fabs(x[i-1]-x[i-2]) > epsilon);
  }
  else if(((f(b) < 0) && (second_der(b) < 0)) || ((f(b) > 0) &&
(second_der(b) > 0)))
  {
     do
       {
            x[i] = x[i-1] - f(x[i-1])*(b - x[i-1])/(f(b)-f(x[i-1]));
            cout << x[i] << endl;
            i++;
       //}while(fabs(f(x[i-1])) > epsilon);
       }while(fabs(x[i-1]-x[i-2]) > epsilon);
  }
```

```
  else
    cout << "No constant point could be found!" << endl;
  cout << "The approximated root equals: " << x[i-1] << endl;
  cout << "The number of iterations equals: " << i << endl;
}
void Newton(float a, float b, float epsilon)
{
  int i =1;
  cout << "Subsequent approximations: " << endl;
  if(f(a)*second_der(a) > 0)
  {
            x[0]=a;
            do
            {
                  //x[i]=x[i-1] - f(x[i-1])/first_der(x[i-1]);
                  forward(table, 5, x[i-1], 0.1);
                  x[i]=x[i-1] - f(x[i-1])/first_delta(table, 0.1, 5);
                  cout << x[i] << endl;
                  i++;
            }while(fabs(f(x[i-1])) > epsilon);
            //}while(fabs(x[i-1]-x[i-2]) > epsilon);
  }
  else if(f(b)*second_der(b) > 0)
  {
            x[0]=b;
            do
            {
                  //x[i]=x[i-1] - f(x[i-1])/first_der(x[i-1]);
                  forward(table, 5, x[i-1], 0.1);
                  x[i]=x[i-1] - f(x[i-1])/first_delta(table, 0.1, 5);
                  cout << x[i] << endl;
                  i++;
            }while(fabs(f(x[i-1])) > epsilon);
            //}while(fabs(x[i-1]-x[i-2]) > epsilon);
  }
  else
    cout << "No starting point could be found!" << endl;

     cout << "The approximated root equals: " << x[i-1] << endl;
     cout << "The number of iterations equals: " << i-1 << endl;
}
```

- ***Bernoulli method***

```
const int N = 100;
float ai[N];
int main(int argc, _TCHAR* argv[])
{
	float epsilon=0;
	int n=0, choice=0;
	ai[0]=1;
	/*ai[1]=-11;
	ai[2]=8;
	ai[3]=20;*/
	/*ai[1]=-36;
	ai[2]=111;
	ai[3]=2224;
	ai[4]=-4620;*/
	/*ai[1]=5.5;
	ai[2]=-138;
	ai[3]=-639.5;
	ai[4]=1771;
	ai[1]=5.1;
	ai[2]=-135.8;
	ai[3]=-611.5;
	ai[4]=1709.4;
	ai[1]=-17.01;
	ai[2]=-2.94;
	ai[3]=913.69;
	ai[4]=-1695.54;
	ai[1]=-64;
	ai[2]=1965;
	ai[3]=-22748;
	ai[4]=52860;
	ai[1]=-47;
	ai[2]=507;
	ai[3]=1003;
	ai[4]=-28814;
	ai[5]=50820;
	ai[1]=-58;
	ai[2]=1024;
	ai[3]=-4574;
	ai[4]=-40117;
	ai[5]=370744;
	ai[6]=-559020;
	ai[1]=-66;
	ai[2]=1191;
	ai[3]=-1106;
	ai[4]=-71340;
	ai[5]=138600;
	ai[1]=-96;
	ai[2]=3171;
	ai[3]=-36836;
	ai[4]=-38160;
	ai[5]=2278800;
	ai[6]=-4158000;*/
	cout << "Type in the order of polynomial n: ";
	cin >> n;
	cout << "Type in the accuracy epsilon: ";
	cin >> epsilon;
	cout << "Please choose: " << endl;
	cout << "1 - Highest modulus root searching" << endl;
	cout << "2 - Lowest modulus root searching" << endl;
```

```
	cout << "3 - All roots searching" << endl;
	choice = getch();
	while((choice != '1') && (choice != '2') && (choice != '3'))
		choice = getch();
	putch(choice);
	cout << endl;
	switch(choice)
	{
	case '1': Bernoulli(n, epsilon, ai, false);
			break;
	case '2': Bernoulli(n, epsilon, ai, true);
			break;
	case '3': all_roots(n, epsilon, ai);
	}
	system("PAUSE");
	return 0;
}
float W(float x)
{
	return powf(x, 3) - 11*powf(x,2) + 8*x + 20;
	//return (x-30)*(x-11)*(x+7)*(x-2);
	//return (x+11.5)*(x-11)*(x+7)*(x-2);
	//return (x+11.1)*(x-11)*(x+7)*(x-2);
	//return (x-11.01)*(x-11)*(x+7)*(x-2);
	//return (x-30)*(powf(x, 2) - 32*x + 881)*(x-2);
	//return (x-30)*powf((x-11), 2)*(x+7)*(x-2);
	//return (x-30)*powf((x-11), 3)*(x+7)*(x-2);
	//return powf((x-30), 2)*(x-11)*(x+7)*(x-2);
	//return powf((x-30), 3)*(x-11)*(x+7)*(x-2);
}
void Bernoulli(int n, float epsilon, float *ai, bool M_L)
{
	float *lambda, *bi, *ci;
	lambda = new float[N];
	bi = new float[N];
	ci = new float[n+1];
	float xM=0;
	float swap=0;
	for(int i=0; i<n-2; i++)
		lambda[i]=0;
	for(int i=0; i<N; i++)
	{
		bi[i]=0;
	}
	lambda[n-2]=1;
	//lambda[n-2]=-0.5;
	//lambda[n-1]=-0.25;
	if(M_L)
	{
		for(int i=0; i<(n+1)/2; i++)
		{
			swap=ai[i];
			ai[i]=ai[n-i];
			ai[n-i]=swap;
		}
	}
	for(int i=0; i<n; i++)
		bi[i+1]=ai[i+1]/ai[0];
	int j=0, k=0;
	ci[n]=bi[n];
	do
```

```
	{
		for(int i=0; i<n; i++)
		{
			ci[i]=0;
		}
		j = 1;
		for(int i=n-1; i>0; i--)
		{
			if(i==1)
				ci[i]=-bi[i]-ci[i+1]*lambda[j-1];
			else
				ci[i]=bi[i]+ci[i+1]*lambda[j-1];
			j++;
		}
		lambda[n-1] = ci[1];
		for(int i=0; i<n-1; i++)
		{
			if(i==(n-2))
				lambda[i]=1.0/lambda[i+1];
			else
				lambda[i]=lambda[i+1];
		}
		k++;
		}while(fabs(W(/*1/lambda[n-1]*/lambda[n-1])) > epsilon);
	if(M_L)
	{
		cout << "Root of the lowest modulus equals: " << 1/lambda[n-1]
<< endl;
		cout << "Number of iterations equals: " << k << endl;
	}
	else
	{
		cout << "Root of the highest modulus equals: " << lambda[n-1]
<< endl;
		cout << "Number of iterations equals: " << k << endl;
	}
}
void all_roots(int n, float epsilon, float *ai)
{
	float *lambda, *bi, *ci;
	int tm=n;
	lambda = new float[N];
	bi = new float[N];
	ci = new float[n+1];
	for(int i=0; i<n-2; i++)
		lambda[i]=0;
	for(int i=0; i<N; i++)
		bi[i]=0;
	lambda[n-2]=1;
	for(int i=0; i<n; i++)
		bi[i+1]=ai[i+1]/ai[0];
	int j=0, k=0;
	while(n>0)
	{
		ci[n]=bi[n];
		do
		{
			for(int i=0; i<n; i++)
				ci[i]=0;
			j = 1;
			for(int i=n-1; i>0; i--)
```

```
                  {
                        if(i==1)
                              ci[i]=-bi[i]-ci[i+1]*lambda[j-1];
                        else
                              ci[i]=bi[i]+ci[i+1]*lambda[j-1];
                        j++;
                  }
                  lambda[n-1] = ci[1];
                  for(int i=0; i<n-1; i++)
                  {
                        if(i==(n-2))
                              lambda[i]=1.0/lambda[i+1];
                        else
                              lambda[i]=lambda[i+1];
                  }
                  k++;
            }while(fabs(W(lambda[n-1])) > epsilon);
            if(tm==n)
            {
                  cout << "Root of the highest modulus equals: " <<
lambda[n-1] << endl;
                  cout << "Number of iterations equals: " << k << endl;
            }
            else if(tm>0)
            {
                  cout << "Root of the next highest modulus equals: " <<
lambda[n-1] << endl;
                  cout << "Number of iterations equals: " << k << endl;
            }
            k=0;
            n--;
            for(int i=0; i<n; i++)
            {
                  ai[i+1]+=ai[i]*lambda[n];
                  bi[i+1]=ai[i+1]/ai[0];
            }
            if(n==1)
            {
                  cout << "Root of the lowest modulus equals: " << -bi[i]
<< endl;
                  break;
            }
      }
}
```