|  |  |
| --- | --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ | |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  «Пермский государственный национальный исследовательский университет» | |
| **ЧИСЛЕННЫЕ МЕТОДЫ**  **«Интерполирование. Среднеквадратичное приближение»** | |
| *Лабораторная работа №5* | |
| Вариант(ы) № 14, 20 | |
|  | Работу выполнили студенты группы ПМИ – 1,2  Заманов Мухтар  Толов Ярослав |
| |  |  | | --- | --- | | Конспект |  | | Сроки |  | | Попытки |  | | Замечания к отчету |  | | ИТОГО: |  | | Проверил:  профессор, доктор физико-математических наук  С. В. Русаков  “\_\_\_\_” 20\_\_ г. |
|  |  |
| Пермь 2017 | |

Оглавление

[*Задание* 3](#_Toc500017125)

[*Исходные данные* 4](#_Toc500017126)

[*Результаты тестирования* 5](#_Toc500017127)

[*Вывод* 25](#_Toc500017128)

[*Код на C++* 26](#_Toc500017129)

# *Задание*

*Задача* - приблизить заданную функцию  на отрезке .

1. Построить таблицу



По полученной таблице произвести интерполяцию с помощью

* формулы Ньютона;
* кубических сплайнов дефекта 1.

Провести оценку погрешности в узлах 

Сравнить оценку погрешности с реальной погрешностью.

2) Выполнить среднеквадратичное приближение заданной функции на заданном отрезке c помощью полинома второго порядка

* дискретный вариант (по таблице из п.1);
* непрерывный (интегральный) вариант.

Провести оценку погрешности.

Построить график приближаемой и приближающих функций.

3) Методом обратного интерполирования, используя интерполяционную формулу Ньютона, найти корень уравнения



Константа *с* выбирается таким образом, чтобы существовал корень на отрезке .

*Указание.*

1) При построении параметров кубического сплайна воспользоваться алгоритмом прогонки. Выполнить оценку погрешности аппроксимации значений первой производной в узлах интерполяции, при этом использовать вариант для равномерной сетки.

# *Исходные данные*

Варианты исходных данных: 14 и 20.



Вариант 20:



# *Результаты тестирования*

Входные данные: Вариант 14

f(x)=2ex-3x+1

f’(x0)=f’(a)=2.43656

f’(x0)=f’(b) = 11.7781

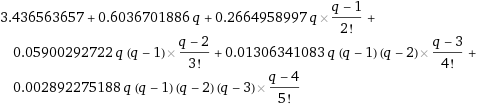
Интерполяционная формула Ньютона для равноотстоящих узлов

Таблица значений функции

|  |  |
| --- | --- |
|  |  |
| 1,0 | 3.436563657 |
| 1,2 | 4.040233845 |
| 1,4 | 4.910399934 |
| 1,6 | 6.106064849 |
| 1,8 | 7.699294929 |
| 2,0 | 9.778112198 |

Таблица конечных разностей

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 1,0 | 3.436563657 | 0.6036701886 | 0.2664958997 | 0.05900292722 | 0.01306341083 | 0.002892275188 |
| 1,2 | 4.040233845 | 0.870166088 | 0.3254988269 | 0.07206633805 | 0.01595568601 | 0,00000000 |
| 1,4 | 4.910399934 | 1.195664915 | 0.3975651649 | 0.08802202406 | 0,00000000 | 0,00000000 |
| 1,6 | 6.106064849 | 1.59323008 | 0.485587189 | 0,00000000 | 0,00000000 | 0,00000000 |
| 1,8 | 7.699294929 | 2.078817269 | 0,00000000 | 0,00000000 | 0,00000000 | 0,00000000 |
| 2,0 | 9.778112198 | 0,00000000 | 0,00000000 | 0,00000000 | 0,00000000 | 0,00000000 |

P(x) = 

,



В нашем случае:

f(6)(x) = 2\*e^x

M6 = max|f(6)(x)|= f(6)(2)=| 2\*e^2 |≈14.778112197861+

Оценка погрешности

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | + | + |
| 1,1 | 3.708332048 | 3.708343243 | 1.119496078e-05 | 1.939627226e-05 |
| 1,3 | 4.438593335 | 4.438589499 | 3.836224989e-06 | 6.465424087e-06 |
| 1,5 | 5.463378141 | 5.463380959 | 2.818616059e-06 | 4.618160062e-06 |
| 1,7 | 6.847894783 | 6.847890722 | 4.061526418e-06 | 6.465424087e-06 |
| 1,9 | 8.671788885 | 8.671801434 | 1.25490214e-05 | 1.939627226e-05 |

Интерполяционный кубический сплайн дефекта 1:

СЛАУ для определения величин ; параметры были вычислены с помощью алгоритма прогонки:

\* =

M = 11.05377208 15.49373252 20.91671246 27.54035512

m0= f’(a)=2.43656

m1= 3.64016284

m2= 5.110329133

m3= 6.905985676

m4= 9.099153091

m5= f’(b) = 11.7781

+



В нашем случае:

f(4)(x) = 2\*e^x

f(5)(x) = 2\*e^x

M4 = max|f(4)(x)|= f(4)(2)=| 2\*e^2 |≈14.778112197861+

M5 = max|f(5)(x)|= f(5)(2)=| 2\*e^2 |≈14.778112197861+

||f(x)-S3,1(f;x)||<= =0.0000812796+

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i | Xi | f(xi) | mi | f’(xi) | Реальная погрешность |
| 0 | 1 | 3,43656 | 2.43656 | 2,43656 | 0 |
| 1 | 1.2 | 4,04023 | 3.64016284 | 3,64023 | 6,716e-05 |
| 2 | 1.4 | 4,91039 | 5.110329133 | 5,11039 | 6,0867e-05 |
| 3 | 1.6 | 6,10606 | 6.905985676 | 6,90606 | 7,4324e-05 |
| 4 | 1.8 | 7,69929 | 9.099153091 | 9,09929 | 0,000136909 |
| 5 | 2 | 9,77811 | 11.7781 | 11,77811 | 1e-05 |

Оценка погрешности

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | + | + |
| 1,10000000 | 3.708332048 | 3.708308772 | 2.327628199e-05 | 8.127962095e-05 |
| 1,30000000 | 4.438593335 | 4.438562732 | 3.060298521e-05 | 8.127962095e-05 |
| 1,50000000 | 5.463378141 | 5.463340978 | 3.716299455e-05 | 8.127962095e-05 |
| 1,70000000 | 6.847894783 | 6.847850703 | 4.408001785e-05 | 8.127962095e-05 |
| 1,90000000 | 8.671788885 | 8.671729586 | 5.929889615e-05 | 8.127962095e-05 |



M5 = ≈14.778112197861

0.0001418382129<=0.0003940829919+

Среднеквадратичное приближение (дискретный вариант):

Система функции .

Тогда искомый полином запишется: .

Таблица скалярных произведений

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 6 | 9 | 14,2 |
|  | 9 | 14,2 | 23,4 |
|  | 14,2 | 23,4 | 39,9664 |
|  | 35.97066941 | 58.3440632 | 98.56857464 |

Найденные коэффициенты : +

с1= 6.371167

с2= -7.483118

с3= 4.583925

g = 6.371167-7.483118\*x+4.583925\*x^2

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | 3.436563657 | 3.471973612 |
| 1,2 | 4.040233845 | 3.992276825 |
| 1,4 | 4.910399934 | 4.879294015 |
| 1,6 | 6.106064849 | 6.133025182 |
| 1,8 | 7.699294929 | 7.753470328 |
| 2 | 9.778112198 | 9.740629450 |

||f-g||=0.097918889+

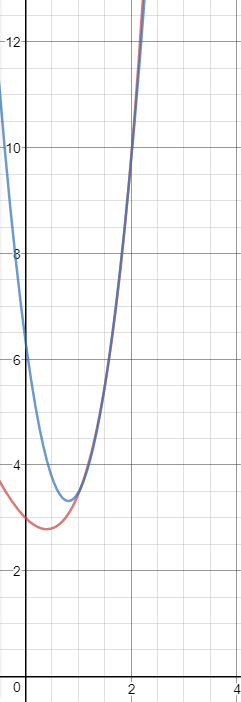
 где – g, а – f

График f(x) и g(x) на x=[0,3]

 где – g, а – f

График f(x) и g(x) на x=[1,2]

Среднеквадратичное приближение (интегральный вариант):

Система функции и вид искомой функции аналогичен дискретному варианту.

Таблица скалярных произведений

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 1.0 | 1.5 | 2.3333333 |
|  | 1.5 | 2.3333333 | 3.75 |
|  | 2.3333333 | 3.75 | 6.2 |
|  | 5.841548541 | 9.278112198 | 15.202994072 |

с1 = 6.442278

с2= -7.497366

с3= 4.562279

g = 6.442278 -7.497366\*x + 4.562279\*x2+

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | 3.436563657 | 3.507192159 |
| 1,2 | 4.040233845 | 4.015121965 |
| 1,4 | 4.910399934 | 4.888034126 |
| 1,6 | 6.106064849 | 6.125928642 |
| 1,8 | 7.699294929 | 7.728805513 |
| 2 | 9.778112198 | 9.696664739 |

||f-g||= 0.028491457+

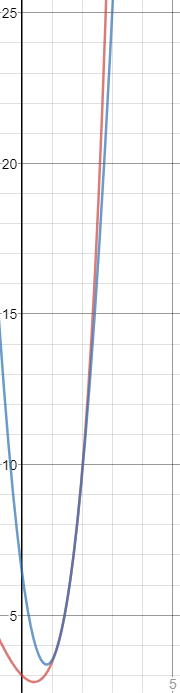
где – g, а – f

График f(x) и g(x) на x=[0,3]

 где – g, а – f

График f(x) и g(x) на x=[1,2]

Решение уравнения с помощью обратного интерполирования:

Решаем уравнение 

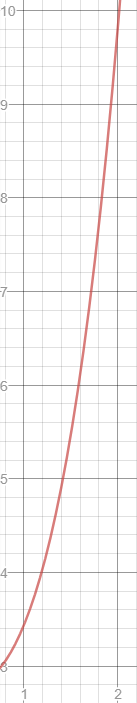


График функции f(x) на [1,2]

Таким образом, c∈[3,4;9,8]

Таблица значений функции

|  |  |
| --- | --- |
|  |  |
| 3.436563657 | 1,0 |
| 4.040233845 | 1,2 |
| 4.910399934 | 1,4 |
| 6.106064849 | 1,6 |
| 7.699294929 | 1,8 |
| 9.778112198 | 2,0 |

Построим интерполяционный многочлен , для которого .

Таблица разделенных разностей+

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 3.436563657 | 1,00000000 | 0.3313067363 | -0.0688445240 | 0.0144432814 | -0.0024059550 | 0.0003037317 |
| 4.040233845 | 1,20000000 | 0.2298411794 | -0.0302881671 | 0.0041873416 | -0.0004798258 | 0.0000000000 |
| 4.910399934 | 1,40000000 | 0.1672709448 | -0.0149664285 | 0.0014341594 | 0.0000000000 | 0.0000000000 |
| 6.106064849 | 1,60000000 | 0.1255311474 | -0.0079853532 | 0.0000000000 | 0.0000000000 | 0.0000000000 |
| 7.699294929 | 1,80000000 | 0.0962085523 | 0.0000000000 | 0.0000000000 | 0.0000000000 | 0.0000000000 |
| 9.778112198 | 2,00000000 | 0.0000000000 | 0.0000000000 | 0.0000000000 | 0.0000000000 | 0.0000000000 |

Pn(y)= 1,00000000 + 0.3313067363\*(x-3.436563657) - 0.0688445240\*(x-3.436563657)\*(x-4.040233845) + 0.0144432814\*(x-3.436563657)\*(x-4.040233845)\*(x-4.910399934) - 0.0024059550\*(x-3.436563657)\*(x-4.040233845)\*(x-4.910399934)\*(x-6.106064849) + 0.0003037317\*(x-3.436563657)\*(x-4.040233845)\*(x-4.910399934)\*(x-6.106064849)\*(x-7.699294929)

и будет решением исходного уравнения.

Пусть с = 5.0000000000

Решение уравнения = 1.4170950660+

Оценка погрешности по невязке: 0.0010455634+

Входные данные: Вариант 20



f(x)=2\*arctg(x) – x + 3

f’(x0)=f’(a)= 0

f’(x0)=f’(b) = -0.6

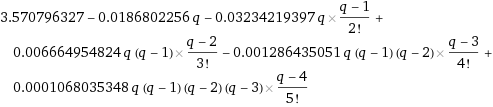
Интерполяционная формула Ньютона для равноотстоящих узлов

Таблица значений функции

|  |  |
| --- | --- |
|  |  |
| 1,0 | 3.570796327 |
| 1,2 | 3.552116101 |
| 1,4 | 3.501093682 |
| 1,6 | 3.424394023 |
| 1,8 | 3.327395645 |
| 2,0 | 3.214297436 |

Таблица конечных разностей

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 1,00000000 | 3.570796327 | -0.0186802256 | -0.03234219397 | 0.006664954824 | -0.001286435051 | 0.0001068035348 |
| 1,20000000 | 3.552116101 | -0.05102241957 | -0.02567723915 | 0.005378519773 | -0.001179631516 | 0,00000000 |
| 1,40000000 | 3.501093682 | -0.07669965872 | -0.02029871938 | 0.004198888257 | 0,00000000 | 0,00000000 |
| 1,60000000 | 3.424394023 | -0.0969983781 | -0.01609983112 | 0,00000000 | 0,00000000 | 0,00000000 |
| 1,80000000 | 3.327395645 | -0.1130982092 | 0,00000000 | 0,00000000 | 0,00000000 | 0,00000000 |
| 2,00000000 | 3.214297436 | 0,00000000 | 0,00000000 | 0,00000000 | 0,00000000 | 0,00000000 |

P(x) = 

,



В нашем случае:

f(6)(x) = -(480\*x\*(3 – 10\*x^2 + 3\*x^4))/(1 + x^2)^6

M6 = max|f(6)(x)|= f(6)(1)=| -(480\*1\*(3 – 10\*1^2 + 3\*1^4))/(1 + 1^2)^6 |≈30+

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | + |  |
| 1,10000000 | 3.565962533 | 3.56596872 | 6.186348371e-06 | 3.9375e-05 |
| 1,30000000 | 3.530201401 | 3.530199704 | 1.697551706e-06 | 1.3125e-05 |
| 1,50000000 | 3.465587446 | 3.46558845 | 1.003958992e-06 | 9.375e-06 |
| 1,70000000 | 3.378144519 | 3.378143347 | 1.171822066e-06 | 1.3125e-05 |
| 1,90000000 | 3.272636796 | 3.272639748 | 2.952001743e-06 | 3.9375e-05 |

Интерполяционный кубический сплайн дефекта 1:

СЛАУ для определения величин ; параметры были вычислены с помощью алгоритма прогонки:

\* =

M = -0.5227698388 -0.9579155872 -1.302735276 -1.575724405

m0= f’(a)=0

m1= -0.1803023336

m2= -0.3243303431

m3= -0.4382074685

m4= -0.5283103353

m5= f’(b) = -0.6

+



В нашем случае:

f(4)(x) = -(48\*x\*(-1 + x^2))/(1 + x^2)^4

f(5)(x) = (48\*(1 - 10\*x^2 + 5\*x^4))/(1 + x^2)^5

M4 = max|f(4)(x)|= f(4)(1.3764)=| -(48\*1\*(-1 + 1^2))/(1 + 1^2)^4 |≈ 0.8403760403+

M5 = max|f(5)(x)|= f(5)(1)=| (48\*(1 - 10\*2^2 + 5\*2^4))/(1 + 2^2)^5 |≈6

||f(x)-S3,1(f;x)||<= (0.8403760403/384 + 6×0.2/240)×0.2^4=0.0000115016+

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i | Xi | f(xi) | mi | f’(xi) | Реальная погрешность |
| 0 | 1 | 3,570796 | 0 | 0 | 0 |
| 1 | 1.2 | 3,552116 | 0.1803023336 | -0,1803278689 | 0,360630203 |
| 2 | 1.4 | 3,501094 | -0.3243303431 | -0,3243243243 | 6,0188e-06 |
| 3 | 1.6 | 3,424394 | -0.4382074685 | -0,4382022472 | 5,2213e-06 |
| 4 | 1.8 | 3,327396 | -0.5283103353 | -0,5283018868 | 8,4485e-06 |
| 5 | 2 | 3,214297 | -0.6 | -0,6 | 0 |

Оценка погрешности

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | + |  |
| 1,10000000 | 3.565962533 | 3.565963772 | 1.2389873e-06 | 8.00000038e-06 |
| 1,30000000 | 3.530201401 | 3.530205592 | 4.190539967e-06 | 8.00000038e-06 |
| 1,50000000 | 3.465587446 | 3.46559078 | 3.333903666e-06 | 8.00000038e-06 |
| 1,70000000 | 3.378144519 | 3.378147406 | 2.886452624e-06 | 8.00000038e-06 |
| 1,90000000 | 3.272636796 | 3.272638782 | 1.986298134e-06 | 8.00000038e-06 |



M5 = 6

2.553523157e-05<=0.00016+

Среднеквадратичное приближение (дискретный вариант)

Система функции .

Тогда искомый полином запишется: .

Таблица скалярных произведений

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 6 | 9 | 14,2 |
|  | 9 | 14,2 | 23,4 |
|  | 14,2 | 23,4 | 39,9664 |
|  | 20.59009321 | 30.63180427 | 47.95238746 |

Найденные коэффициенты :+

с1= 3.349788

с2= 0.516655

с3= -0.292854

g = 3.349788+0.516655\*x-0.292854\*x2

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | 3.570796327 | 3.573588918 |
| 1,2 | 3.552116101 | 3.548064029 |
| 1,4 | 3.501093682 | 3.499110793 |
| 1,6 | 3.424394023 | 3.426729207 |
| 1,8 | 3.327395645 | 3.330919274 |
| 2 | 3.214297436 | 3.211680992 |

||f-g||=0.007270799+

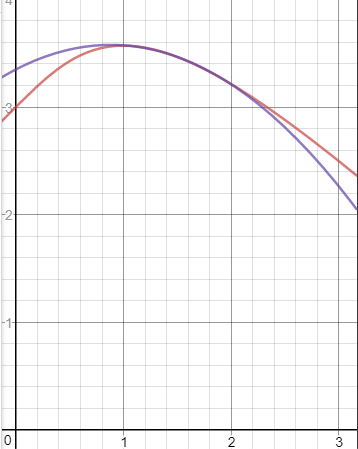
где – g, а – f

График f(x) и g(x) на x=[0,3]

 где – g, а – f

График f(x) и g(x) на x=[1,2]

Среднеквадратичное приближение (интегральный вариант)

Система функции и вид искомой функции аналогичен дискретному варианту.

Таблица скалярных произведений+

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 1.0 | 1.5 | 2.3333333 |
|  | 1.5 | 2.3333333 | 3.75 |
|  | 2.3333333 | 3.75 | 6.2 |
|  | 7.461435360 | 11.515709035 | 18.398483258 |

с1= 3.362579

с2= 0.505350

с3= -0.291041

g = 3.362579+0.505350\*x-0.291041\*x2 +

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | 3.436563657 | 3.576887677 |
| 1,2 | 4.040233845 | 3.549899580 |
| 1,4 | 4.910399934 | 3.499628184 |
| 1,6 | 6.106064849 | 3.426073489 |
| 1,8 | 7.699294929 | 3.329235495 |
| 2 | 9.778112198 | 3.209114202 |

||f-g||= sqrt(11.8557 - 11.8557) = 0



где – g, а – f

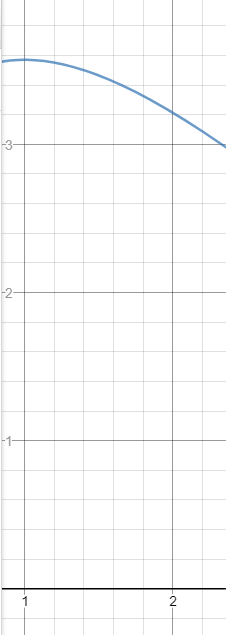
График f(x) и g(x) на x=[0,3]

 где – g, а – f

График f(x) и g(x) на x=[1,2]

Решение уравнения с помощью обратного интерполирования

Решаем уравнение 

 График функции f(x) на [1,2]

Таким образом, c∈[3,57;3,21]

Таблица значений функции

|  |  |
| --- | --- |
|  |  |
| 3.570796327 | 1,0 |
| 3.552116101 | 1,2 |
| 3.501093682 | 1,4 |
| 3.424394023 | 1,6 |
| 3.327395645 | 1,8 |
| 3.214297436 | 2,0 |

Построим интерполяционный многочлен , для которого .

Таблица разделенных разностей+

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 3.570796327 | 1,00000000 | -10.7065088130 | -97.3659368435 | -594.8779598981 | -2313.6208035863 | -6276.7877450673 |
| 3.552116101 | 1,20000000 | -3.9198454655 | -10.2744329797 | -31.7410784394 | -75.9529321299 | 0,00000000 |
| 3.501093682 | 1,40000000 | -2.6075735321 | -3.1415633465 | -6.0827602583 | 0,00000000 | 0,00000000 |
| 3.424394023 | 1,60000000 | -2.0618901462 | -1.3970505389 | 0,00000000 | 0,00000000 | 0,00000000 |
| 3.327395645 | 1,80000000 | -1.7683745957 | 0.0000000000 | 0,00000000 | 0,00000000 | 0,00000000 |
| 3.214297436 | 2,00000000 | 0.0000000000 | 0.0000000000 | 0,00000000 | 0,00000000 | 0,00000000 |

Pn(y)= 1,00000000 - 10.7065088130\*(x-3.570796327) - 97.3659368435\*(x-3.570796327)\*(x-3.552116101) - 594.8779598981\*(x-3.570796327)\*(x-3.552116101)\*(x-3.501093682) - 2313.6208035863\*(x-3.570796327)\*(x-3.552116101)\*(x-3.501093682)\*(x-3.424394023) -6276.7877450673\*(x-3.570796327)\*(x-3.552116101)\*(x-3.501093682)\*(x-3.424394023)\*(x-3.327395645)

и будет решением исходного уравнения.

Пусть с = 3.5000000000

Решение уравнения = 1.4021742581+

Оценка погрешности по невязке: 0.0003870074+

# *Вывод*

Были программно реализованы методы интерполяции функции с помощью формулы Ньютона и кубических сплайнов дефекта 1, также выполнено среднеквадратичное приближения заданной функции на заданном отрезке c помощью полинома второго порядка (дискретный и непрерывные варианты) и рассмотрена задача нахождения корней трансцендентных уравнений методом обратного интерполирования. Для каждого метода были получены теоретические оценки погрешности и проведено сравнение получившейся оценки с реальной погрешностью.

Интерполяционная формула Ньютона и кубический сплайн дефекта 1 дают довольно точное решение относительно обоих вариантов среднеквадратичного приближения.

Также замечено, что от выбора узлов интерполяции напрямую зависит качество интерполяции (сравнение погрешности обратной интерполяции и формулы Ньютона для равноотстоящих узлов).

# *Код на C++*

#include <iostream>

#include <iomanip>

#include <string>

#include <algorithm>

using namespace std;

#define VARIANT 14

void Print(double\* a, int n, string info = "");

void SrednekvDiskret(double \*x, double \*F);

void SrednekvNepr(double \*x, double \*F);

void reverse(double(\*f)(double), double \*x);

double\* GetFuncTable(double(\*f)(double), const double a, const double b, const int n);

double\*\* DividedDeltaTable(double\* functTable, const double a, const double b, const int n);

double\*\* GetFiniteDeltaTable(double\* functTable, const double a, const double b, const int n);

double\* GetFiniteDeltaTablePolinom(double\*\* finiteDeltaTable, const double a, const double b, const int n);

double\*\* GetErrorTable(double(\*f)(double), double(\*df6)(double), double\*\* finiteDeltaTable, const double a, const double b, const int n);

double\*\* GetSoLE();

double\* GetRightPart(double(\*f)(double), double(\*df)(double), double\* functTable, const double a, const double b, const int n);

double\*\* init(int n, int m, bool fillByZero = false);

double\* Solve(double\*\* A, double\* b, int n);

double\*\* GetMateForSpline(double(\*f)(double), double(\*df4)(double), double(\*df5)(double), double\* M, const double a, const double b, const int n);

int factorial(int i);

void PrintInterpStr(double\*\* finiteDeltaTable);

const int a = 1;

const int b = 2;

const int n = 5;

double f\_14(double x)

{

return 2 \* exp(x) - 3 \* x + 1;

}

double f\_20(double x)

{

return 2 \* atan(x) - x + 3;

}

double f\_t(double x)

{

//return (pow(3.0, x - 1) + 4 - x);

return pow(3, x) - 2 \* x + 5;

}

double df\_t(double x)

{

//return (pow(3.0, x - 1)\*log(3) - 1);

return pow(3, x) \* log(3) - 2;

}

double df\_14(double x)

{

return -3 + 2 \* exp(x);

}

double df\_20(double x)

{

return -1 + 2 / (1 + x\*x);

}

double df4\_t(double x)

{

//return (pow(3.0, 2 - 1)\*(pow(log(3), 4)));

return pow(3, x) \* pow(log(3), 4);

}

double df4\_14(double x)

{

return 2 \* exp(x);

}

double df4\_20(double x)

{

return -(48 \* x \*(-1 + x\*x)) / pow((1 + x\*x), 4);

}

double df5\_t(double x)

{

//return (pow(3.0, 2 - 1)\*(pow(log(3), 5)));

return pow(3, x) \* pow(log(3), 5);

}

double df5\_14(double x)

{

return 2 \* exp(x);

}

double df5\_20(double x)

{

return (48 \* (1 - 10 \* x\*x + 5 \* x\*x\*x\*x)) / pow((1 + x\*x), 5);

}

double df6\_t(double x)

{

//return (pow(3.0, 2 - 1)\*(pow(log(3), 6)));

return pow(3, x) \* pow(log(3), 6);

}

double df6\_14(double x)

{

return 2 \* exp(x);

}

double df6\_20(double x)

{

return -(480 \* x\*(3 - 10 \* x\*x + 3 \* x\*x\*x\*x)) / pow((1 + x\*x), 6);

}

void main()

{

setlocale(LC\_ALL, "Russian");

const int a = 1;

const int b = 2;

const int n = 5;

double h = ((double)(b - a)) / n;

double(\*f)(double) = NULL;

double(\*df)(double) = NULL;

double(\*df4)(double) = NULL;

double(\*df5)(double) = NULL;

double(\*df6)(double) = NULL;

switch (VARIANT)

{

case 14:

f = f\_14;

df = df\_14;

df4 = df4\_14;

df5 = df5\_14;

df6 = df6\_14;

break;

case 20:

f = f\_20;

df = df\_20;

df4 = df4\_20;

df5 = df5\_20;

df6 = df6\_20;

break;

default:

f = f\_t;

df = df\_t;

df4 = df4\_t;

df5 = df5\_t;

df6 = df6\_t;

break;

}

double \*w = new double[6];

double \*F = new double[6];

double \*x = new double[6];

for (int i = 0; i < 6; i++)

{

w[i] = 0;

x[i] = 1 + i\*0.2;

F[i] = f(x[i]);

}

std::cout << "df(a) = " << df(a) << endl;

std::cout << "df(b) = " << df(b) << endl;

std::cout << "Интерполяционная формула Ньютона для равноотстоящих узлов\n";

double\* functTable = GetFuncTable(f, a, b, n);

Print(functTable, n + 1, "\nТаблица значений функции");

double\*\* finiteDeltaTable = GetFiniteDeltaTable(functTable, a, b, n);

std::cout << "\nТаблица конечных разностей:\n";

for (int i = 0; i < n + 1; i++, std::cout <<endl)

Print(finiteDeltaTable[i], n + 2);

std::cout << "\nИнтерполяционная формула Ньютона для равноотстоящих узлов:\n";

PrintInterpStr(finiteDeltaTable);

double\*\* errorTable = GetErrorTable(f,df6,finiteDeltaTable, a, b, n);

for (int i = 0; i < 5; i++, std::cout <<endl)

Print(errorTable[i], n);

std::cout << "\nИнтерполяционный кубический сплайн\n";

double\*\* SoLE = GetSoLE();

for (int i = 0; i < n - 1; i++, std::cout<<endl)

Print(SoLE[i], n - 1);

double\* rPart = GetRightPart(f, df, functTable, a, b, n);

Print(rPart, n - 1);

double\* M = new double[n + 1];

M[0] = df(a);

M[n] = df(b);

rPart[0] -= 0.5 \* M[0];

rPart[3] -= 0.5 \* M[n];

double\* ans = Solve(SoLE, rPart,n-1);

for (int i = 1; i < n; i++)

M[i] = ans[i - 1];

Print(M, n+1,"\nОтвет M[], найденный с помощью метода прогонки");

double M5 = fabs(df5(a));

double M4 = fabs(df4(a));

for (double i = a + h; i <= b; i += h)

{

double temp = fabs(df5(i));

if (temp > M5)

M5 = temp;

temp = fabs(df4(i));

if (temp > M4)

M4 = temp;

}

std::cout << "\nM4 = " << M4 << "\nM5 = " << M5 << endl;

double\*\* MateForSpline = GetMateForSpline(f, df4, df5, M, a, b, n);

for (int i = 0; i < 5; i++, std::cout << endl)

{

Print(MateForSpline[i], n);

}

std::cout << endl << "Вторая оценка:" << endl;

double\* ocenka = new double[6];

for (int i = 0; i < 5; i++)

ocenka[i] = abs(df(finiteDeltaTable[i][0]) - M[i]);

std::cout << \*max\_element(ocenka, ocenka + 6) << "<=" << M5 / 60 \* pow(h, 4);

std::cout << endl << "Среднеквадратичное приближение (дискретный вариант)" << endl;

SrednekvDiskret(x, F);

std::cout << endl << "Среднеквадратичное приближение (интегральный вариант)" << endl;

SrednekvNepr(x, F);

cout << endl << "Обратное интерполирование:" << endl;

reverse(f,x);

cin.get();

}

void Print(double\* a, int n, string info)

{

std::cout << info;

if (info != "")

std::cout << ":\n ";

for (int i = 0; i < n; i++)

{

//cout << fixed;

std::cout.precision(10);

std::cout << a[i] << " ";

}

}

double\*\* init(int n, int m, bool fillByZero)

{

double\*\* A = new double\*[n];

for (int i = 0; i < n; i++)

A[i] = new double[m];

if (fillByZero)

for (int i = 0; i < n; i++)

for (int j = 0; j < m; j++)

A[i][j] = 0.0;

return A;

}

int factorial(int i)

{

if (i == 0) return 1;

else return i\*factorial(i - 1);

}

void del(double\*\* M, int n)

{

for (int i = 0; i < n; i++)

delete[] M[i];

delete[] M;

}

double\* GetFuncTable(double(\*f)(double), const double a, const double b, const int n)

{

double h = (b - a) / n;

double t = a;

double\* funcTable = new double[n + 1];

for (int i = 0; t <= b; i++, t += h)

{

funcTable[i] = f(t);

//cout << "f(" << t << ") = " << f(t) << endl;

}

return funcTable;

}

double\*\* DividedDeltaTable(double\* functTable, const double a, const double b, const int n)

{

double\*\* newtonTable = init(n + 1, n + 2,true);

double h = (b - a) / n;

for (int i = 0; i < n + 1; i++)

{

newtonTable[i][0] = a + h\*i;

newtonTable[i][1] = functTable[i];

}

for (int j = 2; j < n + 2; j++)

for (int i = 0; i < (n + 1) - j + 1; i++)

newtonTable[i][j] = (newtonTable[i + 1][j - 1] - newtonTable[i][j - 1]) / (newtonTable[j + i - 1][0] - newtonTable[i][0]);

return newtonTable;

}

double\*\* GetFiniteDeltaTable(double\* functTable, const double a, const double b, const int n)

{

double\*\* finiteDeltaTable = init(n + 1, n + 2, true);

double h = (b - a) / n;

for (int i = 0; i < n + 1; i++)

{

finiteDeltaTable[i][0] = a + h\*i;

finiteDeltaTable[i][1] = functTable[i];

}

for (int j = 2; j < n + 2; j++)

for (int i = 0; i < (n + 1) - j + 1; i++)

finiteDeltaTable[i][j] = finiteDeltaTable[i + 1][j - 1] - finiteDeltaTable[i][j - 1];

return finiteDeltaTable;

}

void PrintInterpStr(double\*\* finiteDeltaTable)

{

std::cout << finiteDeltaTable[0][1] << " + " << finiteDeltaTable[0][2] << "\*q";

for (int i = 3; i < n + 2; i++)

{

std::cout << " + ";

std::cout << finiteDeltaTable[0][i] << "\*q";

for (int j = 1; j + 1 < i; j++)

std::cout << "(q - " << j << ")";

std::cout << "/" << i - 1 << "!";

}

std::cout << endl;

}

double\* GetFiniteDeltaTablePolinom(double\*\* finiteDeltaTable, const double a, const double b, const int n)

{

double\* dividedDeltaTable = new double[n];

double h = (b - a) / n;

int i = 0;

for (double x = a+0.5\*h; x <= b ; x+=h,i++)

{

dividedDeltaTable[i] = finiteDeltaTable[0][1];

double qpoly = 1.0;

double q = (x - a) / h;

for (int j = 0; j < n; j++)

{

qpoly \*= (q - j);

dividedDeltaTable[i] += qpoly \* finiteDeltaTable[0][j + 2] / factorial(j + 1);

}

}

return dividedDeltaTable;

}

double\*\* GetErrorTable(double(\*f)(double), double(\*df6)(double), double\*\* finiteDeltaTable, const double a, const double b, const int n)

{

double\*\* errorTable = init(n, 5);

double h = (b - a) / n;

int i = 0;

for (double x = a + 0.5\*h; x <= b && i<n; x += h, i++)

{

errorTable[0][i] = x;

errorTable[1][i] = f(x);

}

errorTable[2] = GetFiniteDeltaTablePolinom(finiteDeltaTable, a, b, n);

double M6 = df6\_t(a);

for (double i = a + h; i <= b; i += h)

{

double temp = df6(i);

if (temp > M6)

M6 = temp;

}

for (int i = 0; i < n; i++)

{

errorTable[3][i] = fabs(errorTable[1][i] - errorTable[2][i]);

double omega = 1;

for (int j = 0; j < n+1; j++)

omega \*= errorTable[0][i] - finiteDeltaTable[j][0];

errorTable[4][i] = (fabs(M6) \* fabs(omega))/factorial(n+1);

}

return errorTable;

}

double\*\* GetSoLE()

{

double\*\* SoLE = init(n - 1, n - 1, true);

double mu = 0.5;

double lambda = mu;

for (int i = 0; i < n - 1; i++)

{

if (i - 1 >= 0) SoLE[i][i - 1] = lambda;

SoLE[i][i] = 2;

if (i + 1 < n - 1) SoLE[i][i + 1] = mu;

}

return SoLE;

}

double\* GetRightPart(double(\*f)(double), double(\*df)(double), double\* functTable, const double a, const double b, const int n)

{

double\* rPart = new double[n - 1];

double mu = 0.5;

double lambda = mu;

double h = (b - a) / n;

for (int i = 1; i < n; i++)

rPart[i - 1] = 3 \* lambda \* (functTable[i] - functTable[i - 1]) / h + 3 \* mu \* (functTable[i + 1] - functTable[i]) / h;

return rPart;

}

double\* Solve(double\*\* matrix, double\* rPart, int n)

{

n = n + 1;

double\* alpha = new double[n-1];

double\* betta = new double[n-1];

int N1 = n - 2;

double y = matrix[0][0];

alpha[0] = (-matrix[0][1])/ y;

betta[0] = rPart[0] / y;

for (int i = 1; i < n-1; i++)

{

y = matrix[i][i] + matrix[i][i - 1] \* alpha[i - 1];

alpha[i] = (-matrix[i][i + 1]) / y;

betta[i] = (rPart[i] - matrix[i][i - 1] \* betta[i - 1]) / y;

}

betta[N1] = (rPart[N1] - matrix[N1][N1 - 1] \* betta[N1 - 1]) / (matrix[N1][N1] + matrix[N1][N1 - 1] \* alpha[N1 - 1]);

double\* x = new double[n];

x[N1] = (rPart[N1] - matrix[N1][N1 - 1] \* betta[N1 - 1]) / (matrix[N1][N1] + matrix[N1][N1 - 1] \* alpha[N1 - 1]);

for (int i = N1-1; i >= 0; i--)

x[i] = alpha[i] \* x[i + 1] + betta[i];

delete[] alpha;

delete[] betta;

return x;

}

double z0(double x)

{

return (1 + 2 \* x) \* (1 - x) \* (1 - x);

}

double z1(double x)

{

return x \* (1 - x) \* (1 - x);

}

double fi0(double tau)

{

return (1 + 2 \* tau)\*(1 - tau)\*(1 - tau);

}

double fi1(double tau)

{

return tau\*(1 - tau)\*(1 - tau);

}

void splainRes(double \*f, double \*m, double \*s)

{

double tau = 0.5, h = 0.2;

for (int i = 0; i<5; i++)

s[i] = fi0(tau)\*f[i] + fi0(1 - tau)\*f[i + 1] + h\*(fi1(tau)\*m[i] - fi1(1 - tau)\*m[i + 1]);

}

double\*\* GetMateForSpline(double(\*f)(double), double(\*df4)(double), double(\*df5)(double), double\* M, const double a, const double b, const int n)

{

double\*\* mateForSpline = init(n, 5);

double h = (b - a) / n;

double t = a;

int i = 0;

for (double x = a + 0.5 \* h; x <= b; x += h, i++)

{

mateForSpline[0][i] = x;

mateForSpline[1][i] = f(x);

}

int k = 0;

double \*F = new double[5];

int j = 0;

for (double i = 1; i < 2; i += 0.2, j++)

F[j] = f(i);

splainRes(F, M, mateForSpline[2]);

for (double x = a + 0.5 \* h; x <= b; x += h, k++)

{

double t = (x - mateForSpline[0][k]) / h;

//double b = 2;// mateForSpline[0][4];

mateForSpline[3][k] = fabs(mateForSpline[1][k] - mateForSpline[2][k]);

mateForSpline[4][k] = (fabs(df4(b)) / 384 + fabs(df5(b)) \* h / 240) \* powf(h, 4);

}

return mateForSpline;

}

void Initialize(double\*\* matrix, int n)

{

for (int i = 0; i < n; i++)

matrix[i] = new double[n];

}

void SwapRows(double\*\* M, int a, int b)

{

double\* t = M[a];

M[a] = M[b];

M[b] = t;

}

void SwapL(double\*\* L, int a, int b)

{

double t;

for (int i = 0; i < b; i++)

{

t = L[b][i];

L[b][i] = L[a][i];

L[a][i] = t;

}

}

void LUP(double\*\* A, double\*\* U, double\*\*L, int\* P, int n, int count)

{

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

U[i][j] = A[i][j];

for (int i = 0; i < n; i++)

{

double maxVal = 0;

int swap = -1;

for (int row = i; row < n; row++)

{

if (fabs(U[row][i]) > maxVal)

{

maxVal = fabs(U[row][i]);

swap = row;

}

}

if (maxVal != 0)

{

int t = P[swap]; P[swap] = P[i]; P[i] = t;

SwapRows(U, swap, i);

SwapL(L, swap, i);

count++;

L[i][i] = U[i][i];

for (int j = i + 1; j < n; j++)

L[j][i] = U[j][i];

for (int k = i; k < n; k++)

{

U[i][k] /= L[i][i];

for (int j = i + 1; j < n; j++)

{

U[j][k] = U[j][k] - U[i][k] \* L[j][i];

}

}

}

}

}

void Pb(double \*b, int\* P, int n)

{

double\* fakeb = new double[n];

for (int i = 0; i < n; i++)

fakeb[i] = b[i];

for (int i = 0; i < n; i++)

{

b[i] = fakeb[P[i]];

}

}

void Findx(double b[], double x[], double \*\*L, double \*\*U, int n)

{

double \*y = new double[n];

double sum;

y[0] = b[0] / L[0][0];

for (int i = 1; i < n; i++)

{

sum = 0.0;

for (int j = 0; j < i; j++)

sum += L[i][j] \* y[j];

y[i] = ((-1)\*sum + b[i]) / L[i][i];

}

x[n - 1] = y[n - 1];

for (int i = n - 2; i >= 0; i--)

{

sum = 0;

for (int j = n - 1; j >i; j--)

sum += U[i][j] \* x[j];

x[i] = ((-1)\*sum + y[i]);

}

delete[] y;

}

void findc(double \*\*gmatrix, double \*c)

{

double \*\*A, \*b, \*b2, \*x, \*x2;

A = new double\*[3];

for (int i = 0; i < 3; i++)

{

A[i] = new double[3];

}

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

A[i][j] = gmatrix[i][j];

b = new double[3]; b2 = new double[3];

x = new double[3];

x2 = new double[3];

for (int i = 0; i < 3; i++)

b[i] = gmatrix[3][i];

double \*\*L = new double\*[3];

Initialize(L, 3);

double \*\*U = new double\*[3];

Initialize(U, 3);

int \*P = new int[3];

for (int i = 0; i < 3; i++)

P[i] = i;

int count = 0;

for (int i = 0; i < 3; i++)

b2[i] = b[i];

LUP(A, U, L, P, 3, count);

int\* fakeP = new int[3];

for (int i = 0; i<3; i++)

fakeP[i] = P[i];

Pb(b, fakeP, 3);

Findx(b, x2, L, U, 3);

std::cout << endl << " Решение с помощью LU разложения: " << endl;

std::cout << " c1" << " " << " c2" << " " << " c3" << endl;

for (int i = 0; i < 3; i++)

{

std::cout << fixed << setprecision(6) << x2[i] << " ";

c[i] = x2[i];

}

std::cout << endl;

}

void SrednekvDiskret(double \*x, double \*F)

{

double \*\*gmatrix;

gmatrix = new double\*[4];

for (int i = 0; i < 4; i++)

gmatrix[i] = new double[3];

for (int i = 0; i < 4; i++)

for (int j = 0; j < 3; j++)

gmatrix[i][j] = 0;

/\*for (int i = 0; i < 4; i++)

for (int j = 0; j < 3; j++)

for (double k = 1; k <= 2; k += 0.2)

gmatrix[i][j] += g(i, k)\*g(j, k);\*/

gmatrix[0][0] = 6;

gmatrix[0][1] = 9;

gmatrix[0][2] = 14.2;

gmatrix[1][0] = 9;

gmatrix[1][1] = 14.2;

gmatrix[1][2] = 23.4;

gmatrix[2][0] = 14.2;

gmatrix[2][1] = 23.4;

gmatrix[2][2] = 39.9664;

gmatrix[3][0] = 0;

gmatrix[3][1] = 0;

gmatrix[3][2] = 0;

for (int i = 0; i < 6; i++)

{

gmatrix[3][0] += F[i];

gmatrix[3][1] += x[i] \* F[i];

gmatrix[3][2] += x[i] \* x[i] \* F[i];

}

cout << endl << "Таблица среднеквадратичных приближений:" << endl;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 3; j++)

cout << gmatrix[i][j] << " ";

cout << endl;

}

double \*c = new double[3];

double \*gx = new double[6];

double normf = 0, normg = 0;

findc(gmatrix, c);

std::cout << endl << "Коэффициенты ci:" << endl;

for (int i = 0; i < 3; i++)

std::cout << c[i] << endl;

for (int i = 0; i < 6; i++)

gx[i] = c[0] + c[1] \* x[i] + c[2] \* x[i] \* x[i];

std::cout << endl << setw(3) << "x" << " " << setw(10) << "f(x)" << " " << setw(15) << "g(x)";

for (int i = 0; i < 6; i++)

{

cout << endl << fixed << setprecision(2) << x[i] << " " << fixed << setprecision(9) << setw(10) << F[i] << " " << setw(15) << gx[i];

normf += F[i] \* F[i];

normg += gx[i] \* gx[i];

}

cout << endl << "Оценка погрешности: " << sqrt(normf - normg) << endl;

}

void SrednekvNepr(double \*x, double \*F)

{

double \*\*gmatrix;

gmatrix = new double\*[4];

for (int i = 0; i < 4; i++)

gmatrix[i] = new double[3];

gmatrix[0][0] = 1;

gmatrix[0][1] = 1.5;

gmatrix[0][2] = 2.333333333;

gmatrix[1][0] = 1.5;

gmatrix[1][1] = 2.333333333;

gmatrix[1][2] = 3.75;

gmatrix[2][0] = 2.333333333;

gmatrix[2][1] = 3.75;

gmatrix[2][2] = 6.2;

gmatrix[3][0] = 5.841548540943209;//t7.461435359761024//14 5.841548540943209//20 3.441507812507310327653412937306385646731504662009915208426

gmatrix[3][1] = 9.278112197861300;//t11.51570903459456//14 9.278112197861300//20 5.131613928842222562520672275919600424918320193986276989561

gmatrix[3][2] = 15.202994072137843;//t18.39848325815298//14 15.202994072137843//20 7.936624629928235498075084294328284090157426282214543968043

cout << endl << "Таблица среднеквадратичных приближений:" << endl;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 3; j++)

cout << gmatrix[i][j] << " ";

cout << endl;

}

double \*c = new double[3];

double \*gx = new double[6];

double normf = 0, normg = 0;

findc(gmatrix, c);

cout << endl << "Коэффициенты ci:" << endl;

for (int i = 0; i < 3; i++)

cout << c[i] << endl;

for (int i = 0; i < 6; i++)

gx[i] = c[0] + c[1] \* x[i] + c[2] \* x[i] \* x[i];

cout << endl << setw(3) << "x" << " " << setw(10) << "f(x)" << " " << setw(15) << "g(x)";

for (int i = 0; i < 6; i++)

{

cout << endl << fixed << setprecision(2) << x[i] << " " << fixed << setprecision(9) << setw(10) << F[i] << " " << setw(15) << gx[i];

normf += F[i] \* F[i];

normg += gx[i] \* gx[i];

}

cout << endl << "Оценка погрешности: " << sqrt(37.43261176314579494247509791456102868274932999690982591091 - 37.4318) << endl;

cout << endl << "Оценка погрешности: " << sqrt(11.8557 - 11.8557) << endl;

}

void reverse(double (\*f)(double),double \*x)

{

double \*y = GetFuncTable(f, 1, 2, n);

double \*\* matr = init(6, 6, true);

for (int i = 0; i < 6; i++)

matr[i][0] = x[i];

int k = 6;

for (int j = 1; j < 6; j++)

{

k--;

for (int i = 0; i < k; i++)

matr[i][j] = (matr[i + 1][j - 1] - matr[i][j - 1]) / (y[(i + j)] - y[i]);

}

for (int i = 0; i < 6; i++, std::cout << endl)

{

cout << y[i] << " ";

Print(matr[i], n + 1);

}

double c, res = 0;

//c = 5;

c = 3.5;

res = matr[0][0] + matr[0][1] \* (c - y[0]) + matr[0][2] \* (c - y[0])\*(c - y[1]) + matr[0][3] \* (c - y[0])\*(c - y[1])\*(c - y[2]) + matr[0][4] \* (c - y[0])\*(c - y[1])\*(c - y[2])\*(c - y[3]) + matr[0][5] \* (c - y[0])\*(c - y[1])\*(c - y[2])\*(c - y[3])\*(c - y[4]);

cout << endl << "Пусть с = " << c << endl << "Решение уравнения = " << res;

cout << endl << "Погрешность:" << abs(f(res) - c) << endl;

}