**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 3**

по курсу «Численные методы»

Студент: Гаврилов М.С.

Группа: 80-306б

Преподаватель: Ревизников Д.Л.

Оценка:

Москва, 2022

1. **Постановка задачи**

1. Используя таблицу значений  функции , вычисленных в точках  построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки . Вычислить значение погрешности интерполяции в точке .

, a)  ; б) ; .

2. Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при  и . Вычислить значение функции в точке  = 3.0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
|  | 0.0 | 1.7 | 3.4 | 5.1 | 6.8 |
|  | 0.0 | 1.3038 | 1.8439 | 2.2583 | 2.6077 |

3. Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены a) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
|  | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
|  | 1.0 | 1.0032 | 1.0512 | 1.2592 | 1.8192 | 3.0 |

4. Вычислить первую и вторую производную от таблично заданной функции в точке .  0.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
|  | -0.2 | 0.0 | 0.2 | 0.4 | 0.6 |
|  | 1.7722 | 1.5708 | 1.3694 | 1.1593 | 0.9273 |

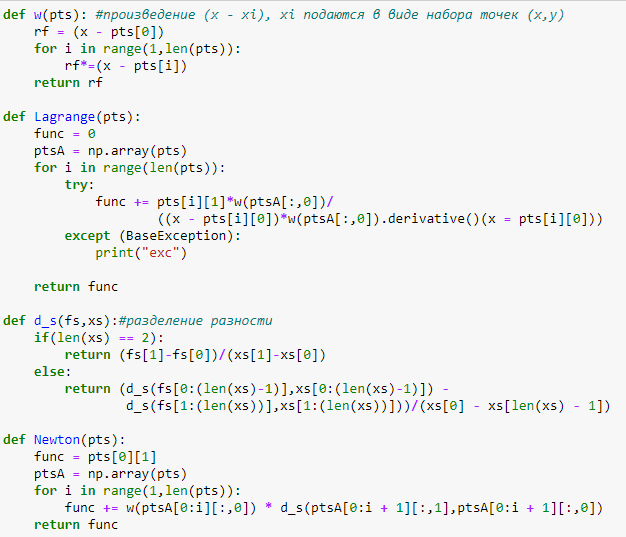
5. Вычислить определенный интеграл , методами прямоугольников, трапеций, Симпсона с шагами . Оценить погрешность вычислений, используя Ме­тод Рунге-Ромберга:

, ;

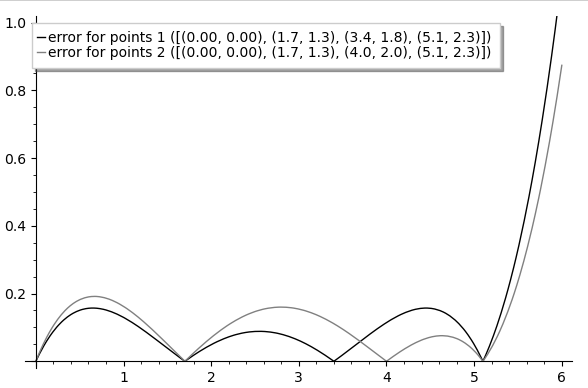
1. **Выполнение работы**

Задание 1.

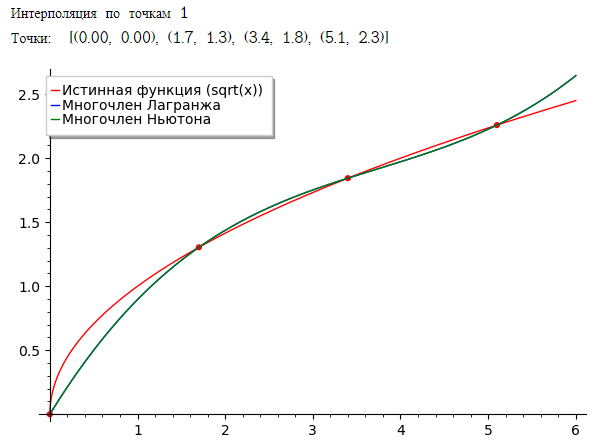
Для расчета многочленов Лагранжа и Ньютона использовались следующие функции:

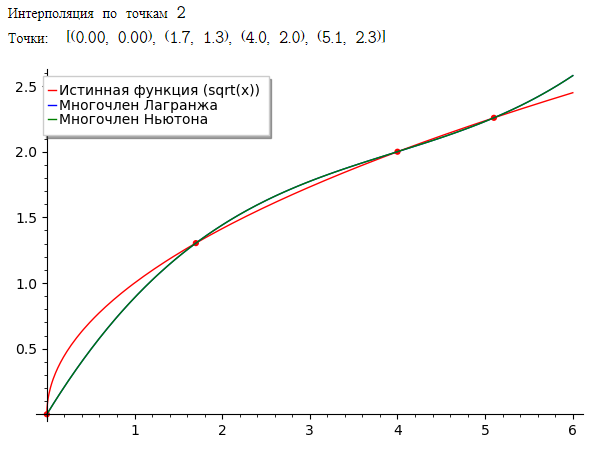


Функция ошибки:



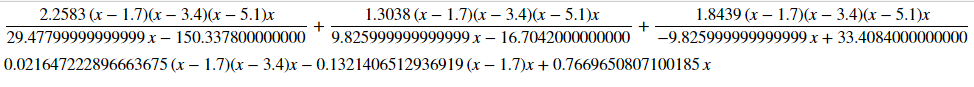
Построение интерполяционных многочленов по каждому из наборов точек:





Видно, что многочлены Ньютона и Лагранжа совпадают.

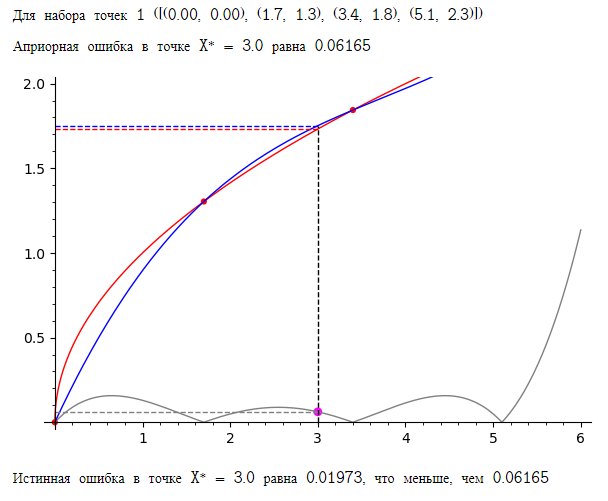
Многочлены Лагранжа (сверху) и Ньютона (снизу), построенные по набору точек 1:

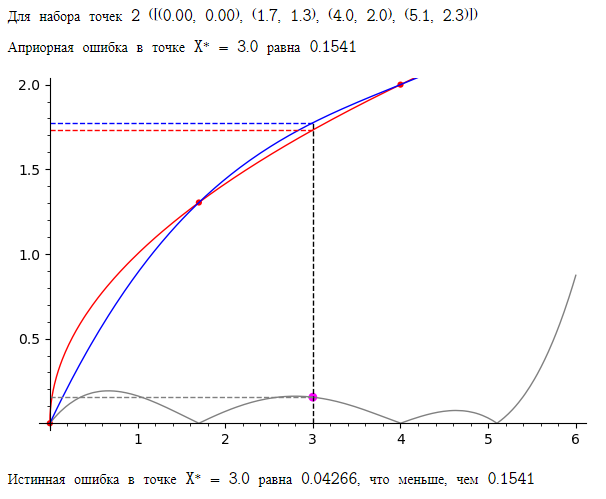


Расчет погрешности интерполяции в точке X\* = 3.0

Под истинной ошибкой понимается отличие значения интерполяционных многочленов в точке X\* от значения интерполируемой функции. Под априорной ошибкой понимается значение функции ошибки в точке X\*. Видно, что априорная ошибка выше истинной, а значит, функция ошибок действительно ограничивает погрешность сверху (по крайней мере в точке X\*).

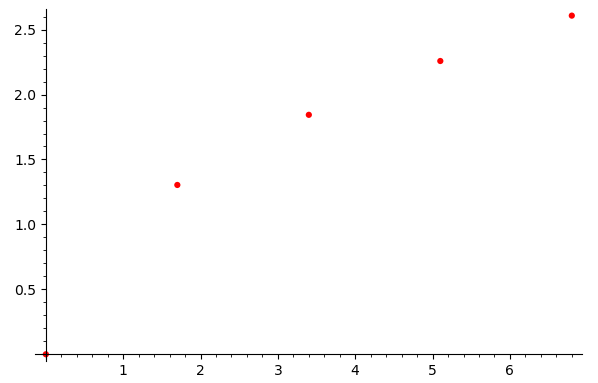
Графики функций и значения расчетов представлены на странице ниже.





Задание 2.

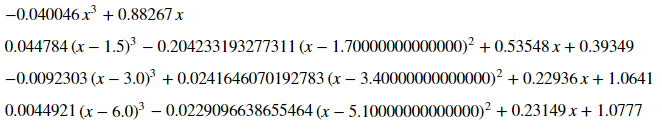
Точки, по которым надо построить сплайн на графике:



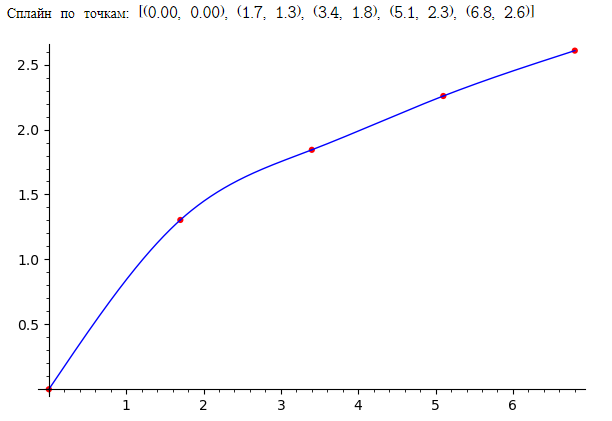
Сплайн строится следующей функцией:

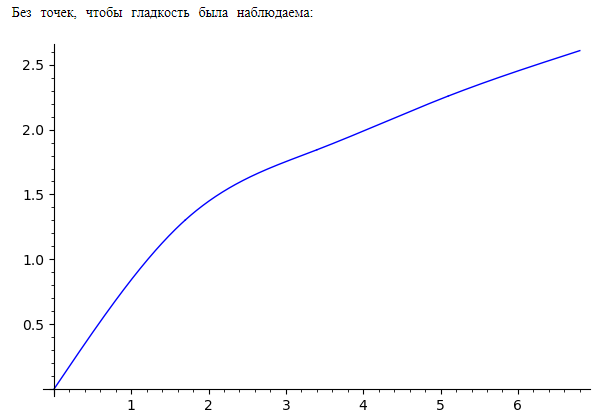
|  |
| --- |
| **def** SplineCube**(**pts**):**  h **=** **[]**  **for** i **in** **range(**1**,len(**pts**)):**  h**.**append**(**pts**[**i**][**0**]** **-** pts**[**i**-**1**][**0**])**  ptsA **=** np**.**array**(**pts**)**  f **=** ptsA**[:,**1**]**  n **=** **len(**pts**)** **-** 2  names **=** "" #инициализация переменных  **for** i **in** **range** **(len(**pts**)** **-** 1**):**  names **+=** "с" **+** **(str(**i**))** **+** " "  c **=** var**(**names**)**    #построение системы уравнений  eqs **=** **[**c**[**0**]** **==** 0**,** 2**\*(**h**[**0**]** **+** h**[**1**])\***c**[**1**]** **+** h**[**1**]\***c**[**2**]** **==**  3**\*((**f**[**2**]-**f**[**1**])/**h**[**1**]** **-** **(**f**[**1**]-**f**[**0**])/**h**[**0**])]**  **for** i **in** **range(**2**,** n**):**  eqs**.**append**(**h**[**i**-**1**]\***c**[**i**-**1**]** **+** 2**\*(**h**[**i**-**1**]+**h**[**i**])\***c**[**i**]** **+** h**[**i**]\***c**[**i**+**1**]** **==**  3**\*((**f**[**i**+**1**]-**f**[**i**])/**h**[**i**]** **-** **(**f**[**i**]-**f**[**i**-**1**])/**h**[**i**-**1**]))**  eqs**.**append**(**h**[**n**-**1**]\***c**[**n**-**1**]** **+** 2**\*(**h**[**n**-**1**]+**h**[**n**])\***c**[**n**]** **==**  3**\*((**f**[**n**+**1**]-**f**[**n**])/**h**[**n**]** **-** **(**f**[**n**]-**f**[**n**-**1**])/**h**[**n**-**1**]))**    sols **=** solve**(**eqs**,**c**)** # решение системы уравнений    C **=** **[]**  **for** a **in** sols**[**0**]:**  C**.**append**(**a**.**rhs**())** #сборка решений в массив C    #отыскание остальных коэфициентов  a **=** **[]**  b **=** **[]**  d **=** **[]**  **for** i **in** **range(**1**,len(**pts**)):**  a**.**append**(**pts**[**i**-**1**][**1**])**  **if** **(**i **<** **len(**pts**)** **-** 1**):**    b**.**append**((**pts**[**i**][**1**]** **-** pts**[**i**-**1**][**1**])/**h**[**i**-**1**]** **-**  **(**1**/**3**)\*(**h**[**i**-**1**]\*(**C**[**i**]** **+** 2**\***C**[**i**-**1**])))**  d**.**append**((**C**[**i**]** **-** C**[**i**-**1**])/(**3**\***h**[**i**-**1**]))**  **else:**  b**.**append**((**pts**[**i**][**1**]** **-** pts**[**i**-**1**][**1**])/**h**[**i**-**1**]** **-**  **(**2**/**3**)\*(**h**[**i**-**1**]\***C**[**i**-**1**]))**  d**.**append**((-** C**[**i**-**1**])/(**3**\***h**[**i**-**1**]))**    #построение систмы функций  func**=** **[]**  **for** i **in** **range(len(**pts**)** **-** 1**):**  func**.**append**(**a**[**i**]** **+** b**[**i**]\*(**x **-** pts**[**i**][**0**])\*\***1**+** C**[**i**].**n**()\*(**x **-**  pts**[**i**][**0**])\*\***2 **+** d**[**i**]\*(**x **-** pts**[**i**][**0**])\*\***3**)**    **return** func |

Построенный по данному набору точек сплайн, представленный как набор функций:

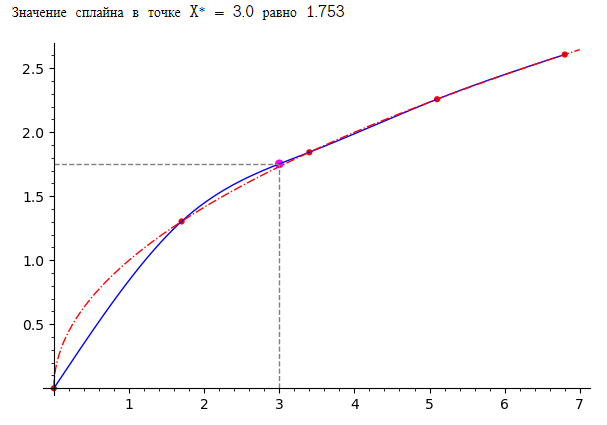


Его график:



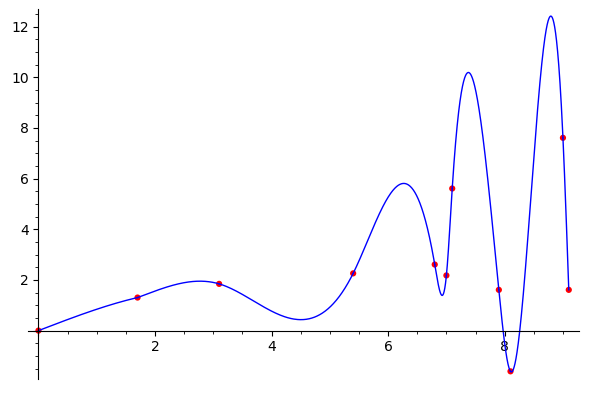


Поиск значения функции в точке X\* = 3.0



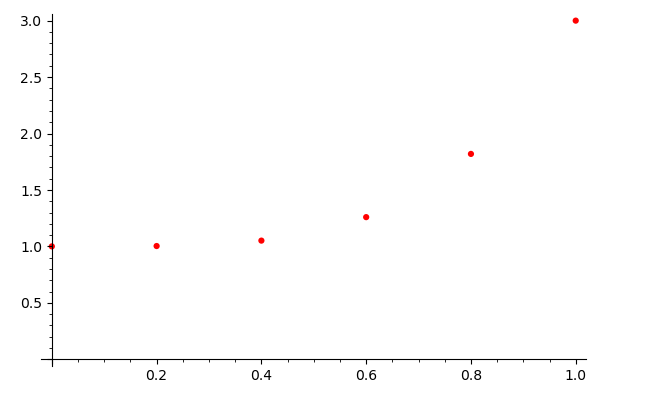
Пунктирной линией обозначен график функции , к которой, очевидно, и относятся те точки, по которым строился сплайн.

Для того, чтобы убедиться, что реализованная функция построения сплайна работает правильно, я применил ее к более сложному набору точек. Как видно, получившийся объект определяет гладкую функцию



Задание 3.

Точки, к которым задана функция.



Решение системы МНК осуществляется этой функцией:

|  |
| --- |
| **def** Least\_squares**(**pts**,**n**):** #порядок приближаемой функции    N **=** **len(**pts**)** #число точек  ptsA **=** np**.**array**(**pts**)**  X **=** ptsA**[:,**0**]**  Y **=** ptsA**[:,**1**]**  n**+=**1  names **=** "" #инициализация переменных  **for** i **in** **range** **(**n**):**  names **+=** "a" **+** **(str(**i**))** **+** " "  a **=** var**(**names**)**    eqsys **=** **[]** #построение системы  **for** k **in** **range** **(**n**):**  eqls **=** 0  eqrs **=** 0  **for** i **in** **range(**n**):**  eqa **=** 0  **for** j **in** **range(**N**):**  eqa **+=** X**[**j**]\*\*(**i**+**k**)**  eqls **+=** a**[**i**]\***eqa  **for** j **in** **range(**N**):**  eqrs **+=** Y**[**j**]\*(**X**[**j**]\*\***k**)**  eqsys**.**append**(**eqls **==** eqrs**)**    sols **=** solve**(**eqsys**,**a**)** #решение системы    func **=** 0  **for** i **in** **range(len(**sols**[**0**])):**  func **+=** sols**[**0**][**i**].**rhs**()\***x**\*\***i    **return** func |

Полученные решения

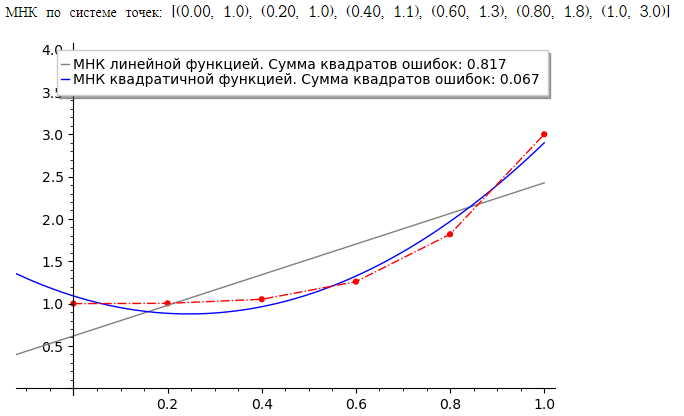
Для линейной функции:



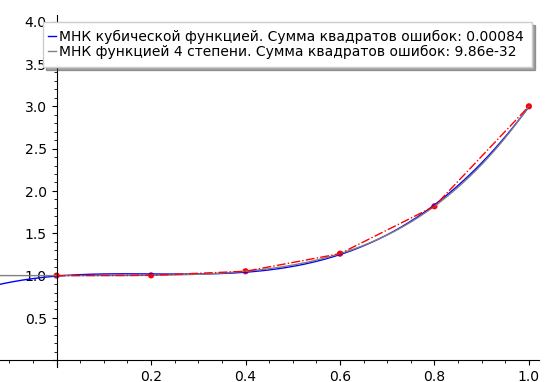
Для квадратичной функции:



Решения на графике:



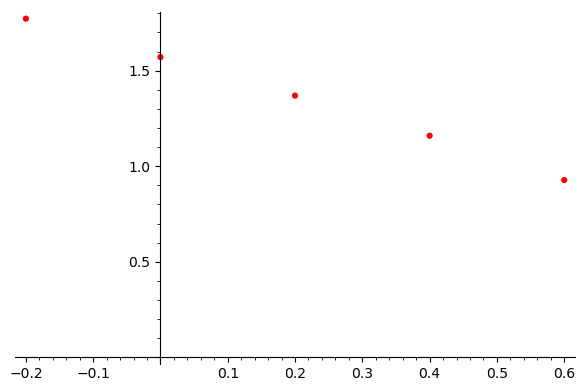
Заметив, как сильно возросла точность приближения многочленом второй степени в сравнении с первой, я рассчитал приближение по МНК многочленами 3 и 4 степени. Получилось очень точно:



Мне даже кажется, что точки и были сгенерированы многочленом 4 степени.

Задание 4.

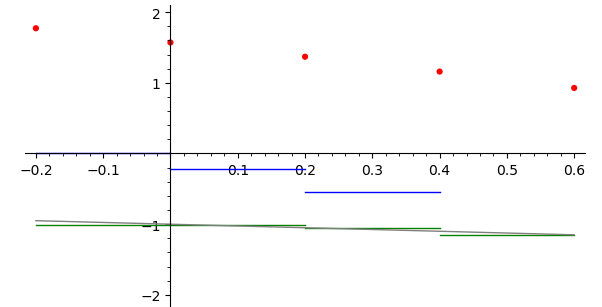
Точки, которыми задана функция на графике



Для вычисления производных используются следующие функции:

|  |
| --- |
| **def** s\_numeric\_derivative**(**pts**,**i**):**    ptsA **=** np**.**array**(**pts**)**  X **=** ptsA**[:,**0**]**  Y **=** ptsA**[:,**1**]**      res **=** **(**Y**[**i**+**1**]** **-** Y**[**i**])/(**X**[**i**+**1**]** **-** X**[**i**])**    **return** res  **def** numeric\_derivative**(**pts**,**i**):**    ptsA **=** np**.**array**(**pts**)**  X **=** ptsA**[:,**0**]**  Y **=** ptsA**[:,**1**]**      res **=** **(**Y**[**i**+**1**]** **-** Y**[**i**])/(**X**[**i**+**1**]** **-** X**[**i**])**  rf **=** **(((**Y**[**i**+**2**]** **-** Y**[**i**+**1**])/(**X**[**i**+**2**]** **-** X**[**i**+**1**]))** **-** res**)** **/** **(**X**[**i**+**2**]** **-** X**[**i**])**    rf**\*=(**2**\***x **-** X**[**i**]** **-** X**[**i **+** 1**])**    **return** res **+** rf    **def** numeric\_derivative\_2**(**pts**,**i**):**    ptsA **=** np**.**array**(**pts**)**  X **=** ptsA**[:,**0**]**  Y **=** ptsA**[:,**1**]**      res **=** **((**Y**[**i**+**2**]** **-** Y**[**i**+**1**])/(**X**[**i**+**2**]** **-** X**[**i**+**1**])** **-** **(**Y**[**i**+**1**]** **-** Y**[**i**])/(**X**[**i**+**1**]** **-** X**[**i**]))** **/** **(**X**[**i**+**2**]** **-** X**[**i**])**    **return** 2**\***res |

Значения производных по точкам на графике:



Серой линией обозначена производная построенного по точкам приближения с помощью МНК. По ней видно, что производные примерно совпадают с тем, что можно ожидать от функции, проходящей через данные точки.

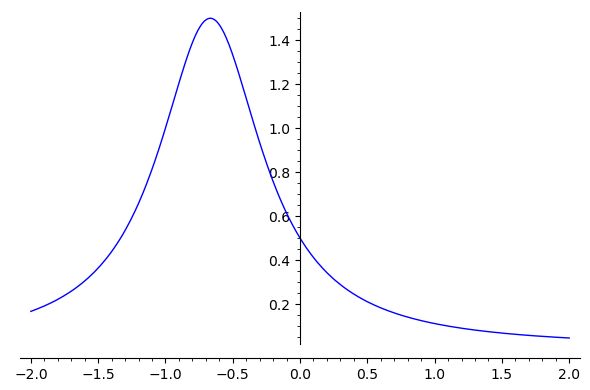
Значения первой и второй производной в точке X\* = 0.2





Задание 5.

Заданная функция:



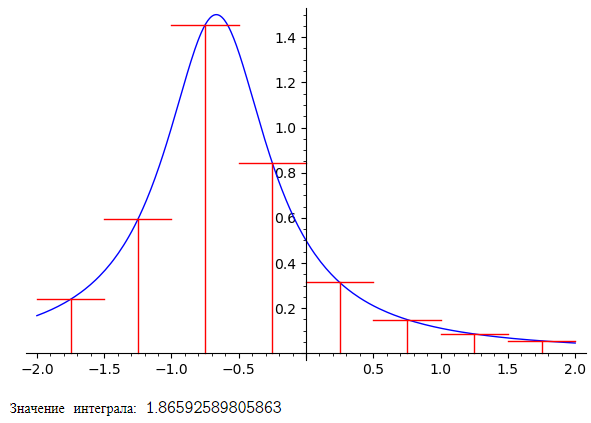
Расчет интегралов осуществляется следующими функциями:

|  |
| --- |
| **def** integrate**(**func**,**xmin**,**xmax**,**h**,**method**,**trace **=** false**,**eps **=** 0.1**):**  **if(**h **<=** 0**):** #чтоб не упал в бесконечный цикл  **return** false    step **=** h  xcur **=** xmin  res **=** 0    traceplt **=** 0    **if(**h **<** 0.01**):**#если шаг очень мал, то трэйсить не надо  trace **=** **False**    **if(**trace**):**  traceplt **=** plot**(**func**,**xmin **=** xmin**,** xmax **=** xmax**)**    **if(**method **==** "simpson"**):**  **while** **(**xcur **+** 2**\***step **<=** xmax **+** h**\***eps**):**  **if(**trace**):**  fi1 **=** func**(**xcur**)**  fi2 **=** func**(**xcur **+** step**)**  fi3 **=** func**(**xcur **+** 2**\***step**)**    traceplt **+=** line**([(**xcur**,**0**),(**xcur**,**fi1**)],**color **=** "red"**)**  traceplt **+=** line**([(**xcur **+** step**,**0**),(**xcur **+** step**,**fi2**)],**color **=** "red"**)**  traceplt **+=** line**([(**xcur **+** 2**\***step**,**0**),(**xcur **+** 2**\***step**,**fi3**)],**color **=** "red"**)**    traceplt **+=** plot\_sq\_int**([(**xcur**,**fi1**),(**xcur **+** step**,**fi2**),(**xcur **+** 2**\***step**,**fi3**)])**  res **+=** func**(**xcur**)** **+** 4**\***func**(**xcur **+** step**)** **+** func**(**xcur **+** 2**\***step**)**  xcur **+=** step**\***2  res **\*=** step**/**3  **else:**  **if(**method **==** "trap"**):**  **while** **(**xcur **+** step **<=** xmax **+** h**\***eps**):**  **if(**trace**):**  fi1 **=** func**(**xcur**)**  fi2 **=** func**(**xcur **+** step**)**  traceplt **+=** line**([(**xcur**,**0**),(**xcur**,**fi1**)],**color **=** "red"**)**  traceplt **+=** line**([(**xcur **+** step**,**0**),(**xcur **+** step**,**fi2**)],**color **=** "red"**)**  traceplt **+=** line**([(**xcur**,** fi1**),(**xcur **+** step**,** fi2**)],**color **=** "red"**)**  res **+=** func**(**xcur**)** **+** func**(**xcur **+** step**)**  xcur **+=** step  res **\*=** step**/**2  **else:**  **if(**method **==** "rect"**):**  **while** **(**xcur **+** step **<=** xmax **+** h**\***eps**):**  **if(**trace**):**  fi **=** func**(**xcur **+** step**/**2**)**  traceplt **+=** line**([(**xcur **+** step**/**2**,**0**),(**xcur **+** step**/**2**,**fi**)],**color **=** "red"**)**  traceplt **+=** line**([(**xcur**,** fi**),(**xcur **+** step**,** fi**)],**color **=** "red"**)**    res **+=** func**(**xcur **+** step**/**2**)**  xcur **+=** step  res**\*=**step  **else:**  **print(**"ERROR: wrong method"**)**  **return** false  **if(**trace**):**  show**(**traceplt**)**  **return** res**.**n**()** |

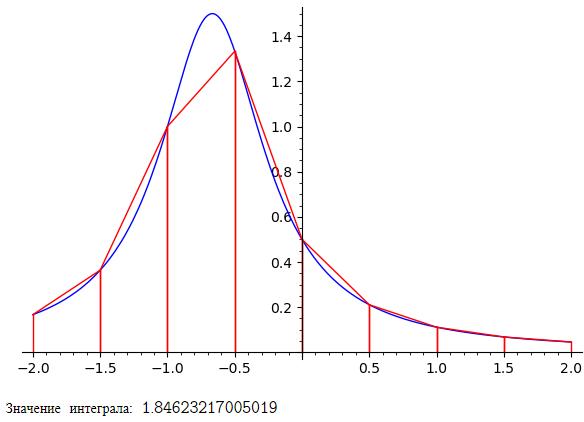
Расчет интегралов с трассировкой вычислений.

С шагом 0.5:

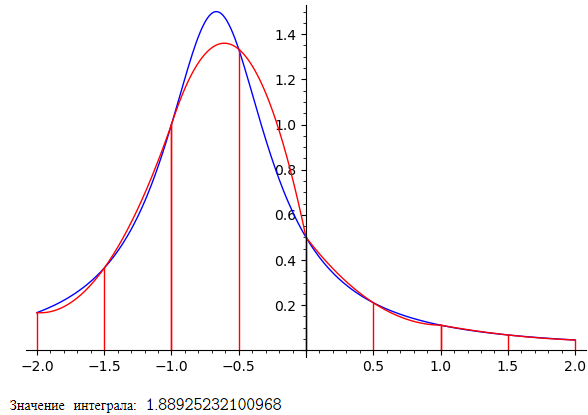
Метод прямоугольников:



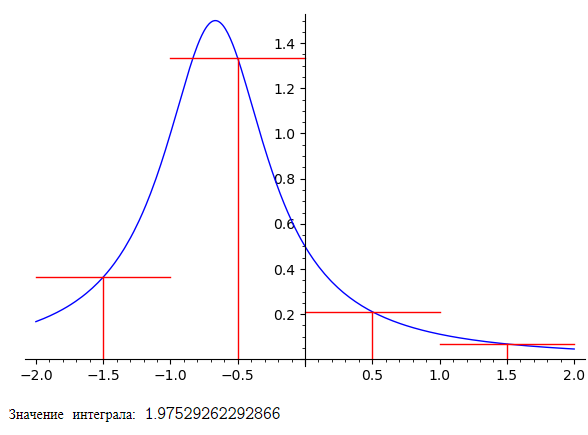
Метод трапеций:

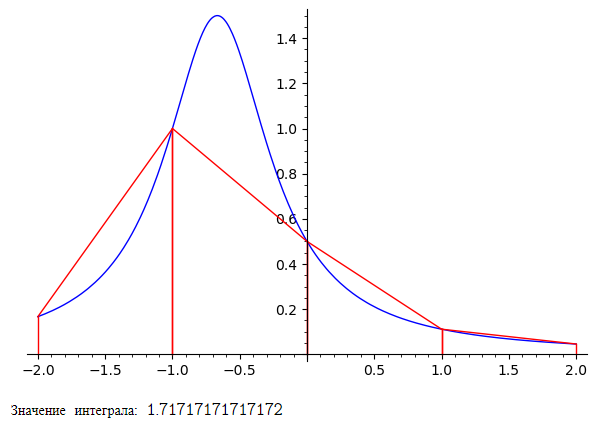


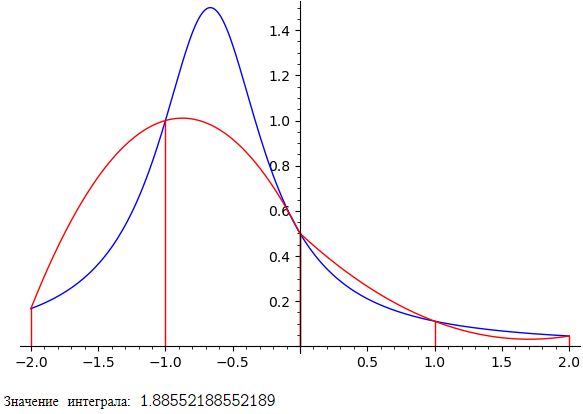
Метод Симпсона:



С шагом 1:



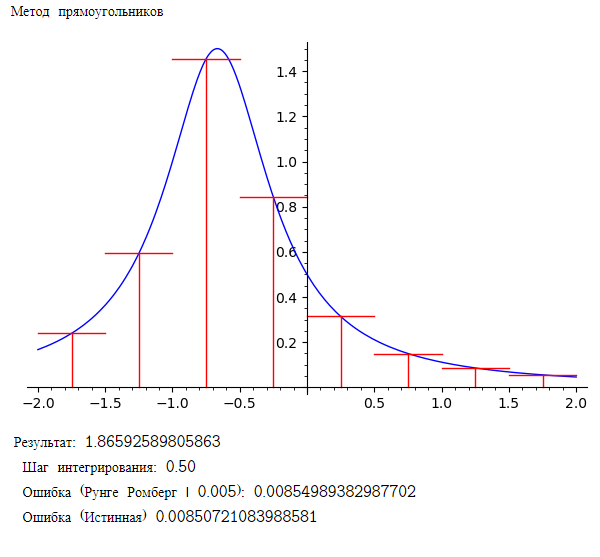


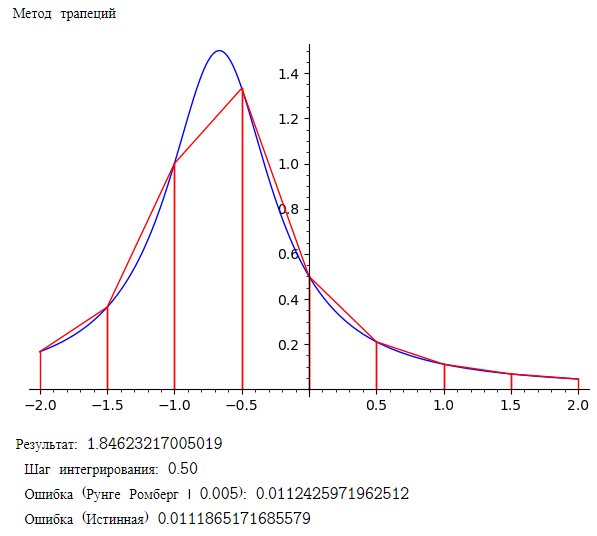


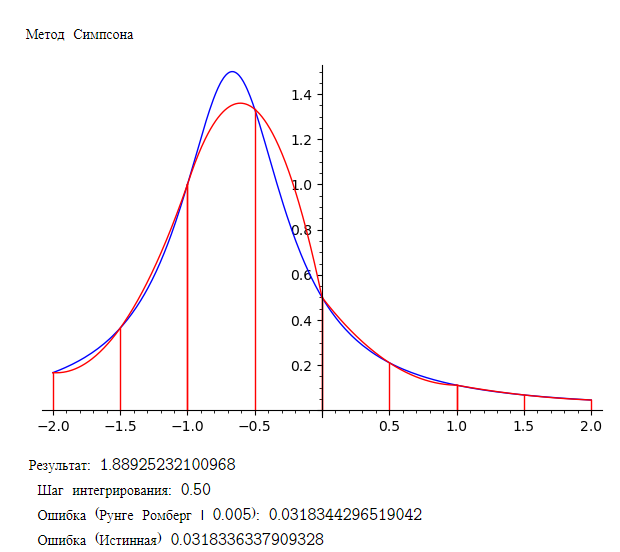
Вычисление ошибки методом Рунге-Ромберга:

|  |
| --- |
| **def** runge\_romberg\_error**(**f**,**xmin**,**xmax**,**h**,**k**,**method**):** #счет ошибки методом Рунге-Ромберга  r1 **=** integrate**(**f**,**X0**,**X1**,**h**,**method**)**  r2 **=** integrate**(**f**,**X0**,**X1**,**h**\***k**,**method**)**    a **=** 1  **if** method **==** "simpson"**:**  a **=** 2    err **=** **(**r1 **-** r2**)/(**k**\*\***a **-** 1**)**    **return** err |

Для проверки корректности рассчитанной данным образом ошибки, я использовал истинное значение интеграла, вычисленное аналитические.







Видно, что метод прямоугольников, на удивление необычайно точен. Также ошибка, вычисленная методом Рунге-Ромберга весьма точна.

1. **Вывод**

В ходе выполнения этой лабораторной работы я получил опыт в реализации всяческих методов аппроксимации, а также методов численного дифференцирования и интегрирования. На мой взгляд, это очень интересные темы, думаю, мне так кажется не в последнюю очередь потому, что они весьма понятны.