|  |  |
| --- | --- |
| Московский авиационный институт | |
| (Национальный исследовательский университет) | |
| Факультет прикладной математики и физики | |
| Кафедра вычислительной математики и программирования | |
|  | |
| **Курсовой проект** | |
| по курсу «Численные методы» | |
| Вычисление многократных интегралов с использованием квадратурных формул и метода Монте-Карло. | |
|  | |
|  | Выполнил: Мариечев К.Д. |
|  | Группа: М8О-409Б-20 |
|  | Проверил: Пивоваров Д.Е. |
|  | Дата: |
|  | Оценка: |
| Москва, 2023 | |

**Оглавление**

[**Метод Монте-Карло** 3](#_Toc154351711)

[**Общие сведения.** 3](#_Toc154351712)

[**Постановка задачи и алгоритм решения.** 4](#_Toc154351713)

[**Квадратурные формулы** 6](#_Toc154351714)

[**Общие сведения.** 6](#_Toc154351715)

[**Постановка задачи и алгоритм решения.** 6](#_Toc154351716)

[**Сведения о программе** 9](#_Toc154351717)

[**Код** 10](#_Toc154351718)

[**Интерфейс** 22](#_Toc154351719)

[**Тестирование** 23](#_Toc154351720)

[**Исследование сходимости метода Монте-Карло** 28](#_Toc154351721)

[**Список источников** 32](#_Toc154351722)

# **Метод Монте-Карло**

## **Общие сведения.**

Группа численных методов для изучения [случайных процессов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81). Суть метода заключается в следующем: процесс описывается математической моделью с использованием [генератора случайных величин](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB), модель многократно обсчитывается, на основе полученных данных вычисляются [вероятностные](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C) характеристики рассматриваемого процесса. Методы используются для решения задач в различных областях [физики](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B7%D0%B8%D0%BA%D0%B0), [химии](https://ru.wikipedia.org/wiki/%D0%A5%D0%B8%D0%BC%D0%B8%D1%8F), [математики](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), [экономики](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D0%BA%D0%B0), [оптимизации](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), [теории управления](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) и др.

Сначала [Энрико Ферми](https://ru.wikipedia.org/wiki/%D0%A4%D0%B5%D1%80%D0%BC%D0%B8,_%D0%AD%D0%BD%D1%80%D0%B8%D0%BA%D0%BE) в [1930-х](https://ru.wikipedia.org/wiki/1930-%D0%B5) годах в Италии, а затем [Джон фон Нейман](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D0%BD_%D0%9D%D0%B5%D0%B9%D0%BC%D0%B0%D0%BD,_%D0%94%D0%B6%D0%BE%D0%BD) и [Станислав Улам](https://ru.wikipedia.org/wiki/%D0%A3%D0%BB%D0%B0%D0%BC,_%D0%A1%D1%82%D0%B0%D0%BD%D0%B8%D1%81%D0%BB%D0%B0%D0%B2_%D0%9C%D0%B0%D1%80%D1%82%D0%B8%D0%BD) в [1940-х](https://ru.wikipedia.org/wiki/1940-%D0%B5) в [Лос-Аламосе](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D1%81-%D0%90%D0%BB%D0%B0%D0%BC%D0%BE%D1%81) предположили, что можно использовать связь между стохастическими процессами и [дифференциальными уравнениями](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5) «в обратную сторону». Они предложили использовать стохастический подход для [аппроксимации](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D1%80%D0%BE%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) многомерных интегралов в уравнениях переноса, возникших в связи с задачей о движении нейтрона в [изотропной](https://ru.wikipedia.org/wiki/%D0%98%D0%B7%D0%BE%D1%82%D1%80%D0%BE%D0%BF%D0%B8%D1%8F) среде. Теория получила развитие с появлением первого электронного [компьютера](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) [ENIAC](https://ru.wikipedia.org/wiki/ENIAC), который мог с большой скоростью генерировать [псевдослучайные числа](https://ru.wikipedia.org/wiki/%D0%9F%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) и использовать их в математических моделях, возобновился интерес к стохастическим методам. После начала использования компьютеров произошёл большой прорыв, и метод Монте-Карло применялся во многих задачах, для решения которых стохастический подход оказался более эффективным, чем другие математические методы. Тем не менее, использование такой методики имело и свою ограниченность из-за необходимости очень большого количества вычислений для получения результатов с высокой точностью.

Годом рождения термина «метод Монте-Карло» считается 1949 год, когда в свет вышла статья Метрополиса и Улама «Метод Монте-Карло». Название метода происходит от названия коммуны в княжестве [Монако](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BD%D0%B0%D0%BA%D0%BE), широко известного своими многочисленными [казино](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%B7%D0%B8%D0%BD%D0%BE), поскольку именно рулетка является одним из самых широко известных [генераторов случайных чисел](https://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB).

В [1970-х](https://ru.wikipedia.org/wiki/1970-%D0%B5) годах в новой области математики — [теории вычислительной сложности](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2) было показано, что существует класс задач, сложность которых растёт с размерностью задачи [экспоненциально](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D1%81%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%80%D0%BE%D1%81%D1%82). Иногда можно, пожертвовав точностью, найти алгоритм, сложность которого растёт медленнее, но есть большое количество задач, для которого этого нельзя сделать и метод Монте-Карло является единственной возможностью для получения достаточно точного ответа за приемлемое время. В настоящее время основные усилия исследователей направлены на создание эффективных Монте-Карло алгоритмов различных физических, химических и социальных процессов для [параллельных вычислительных систем](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B).

## **Постановка задачи и алгоритм решения.**

Предположим, требуется вычислить определённый интеграл

Для определения площади под графиком функции можно использовать следующий стохастический алгоритм:

* ограничим функцию прямоугольником (*n*-мерным параллелепипедом в случае многих измерений), площадь которого���� можно легко вычислить; *любая сторона прямоугольника содержит хотя бы 1 точку графика функции, но не пересекает его;*
* «набросаем» в этот прямоугольник (параллелепипед) некоторое количество точек ( штук), координаты которых будем выбирать случайным образом;
* определим число точек ( штук), которые попадут под график функции;
* площадь области, ограниченной функцией и осями координат, даётся выражением

Для малого числа измерений интегрируемой функции производительность Монте-Карло интегрирования гораздо ниже, чем производительность детерминированных методов. Тем не менее, в некоторых случаях, когда функция задана неявно, а необходимо определить область, заданную в виде сложных неравенств, стохастический метод может оказаться более предпочтительным.

# **Квадратурные формулы**

## **Общие сведения.**

В [численном анализе](https://en.wikipedia.org/wiki/Numerical_analysis) *n*-точечное **квадратурное правило**Гаусса[, названное в честь](https://en.wikipedia.org/wiki/Carl_Friedrich_Gauss)Карла Фридриха Гаусса[представляет собой](https://en.wikipedia.org/wiki/Quadrature_rule)квадратурное правило[, предназначенное для получения точного результата для](https://en.wikipedia.org/wiki/Polynomial)многочленов степени  или меньше путем подходящего выбора узлов  и весов  для

Современная формулировка, использующая [ортогональные многочлены](https://en.wikipedia.org/wiki/Orthogonal_polynomial), была разработана [Карлом Густавом Якоби](https://en.wikipedia.org/wiki/Carl_Gustav_Jacobi) в 1826 году. Наиболее распространенной областью интегрирования для такого правила считается , поэтому правило формулируется как

которая точна для многочленов степени 2*n* − 1 или меньше. Это точное правило известно как квадратурное [правило](https://en.wikipedia.org/wiki/Gauss-Legendre_quadrature)Гаусса. Квадратурное правило будет точным приближением к приведенному выше интегралу только в том случае, если *f* (*x*) хорошо аппроксимируется многочленом степени 2*n* - 1 или меньше на .

## **Постановка задачи и алгоритм решения.**

В общем виде формула численного интегрирования записывается следующим образом:

– интегральная функция;

– веса интегрирования;

– система координат мастер-элемента;

– матрица Якоби для перехода на мастер-элемента

*Одномерный интеграл*

Одномерное интегрирование – это всегда интегрирование по отрезку

* Область интегрирования: отрезок ;
* Мастер-элемент: отрезок ;
* Переход на мастер-элемент: ;
* Переход с мастер-элемента: ;
* Якобиан: .

*Двумерный интеграл*

* Область интегрирования: прямоугольник
* Мастер-элемент: квадрат
* Переход на мастер-элемент:
* Переход с мастер-элемента:
* Якобиан:

*Трехмерный интеграл*

* Область интегрирования: прямоугольник
* Мастер-элемент: квадрат
* Переход на мастер-элемент:
* Переход с мастер-элемента:
* Якобиан:

# **Сведения о программе**

Программа написана на языке Python 3.9 в среде программирования PyCharm и состоит из следующих блоков:

* *main.py*
* *KP.py*
* *KP\_function1.py*
* *KP\_function2.py*
* *KP\_function3.py*

*main.py* – отрисовка интерфейса, графиков и взаимодействие с пользователем.

*KP.py* – вычисление решения задачи по параметрам, заданным пользователем.

*KP\_function1.py* – содержит параметры для решения одномерного интеграла.

*KP\_function2.py* – содержит параметры для решения двумерного интеграла.

*KP\_function3.py* – содержит параметры для решения трехмерного интеграла.

Программа позволяет решать интегралы с любым заданным порядком интегрирования метом Монте-Карло и при помощи квадратурных формул. Для решения интеграла методом Монте-Карло пользователю нужно задать дискретизацию пространства, число точек и параметры интеграла в соответствующем блоке. Для решения интеграла при помощи квадратурных формул пользователю нужно задать параметры интеграла в соответствующем блоке. В программе представлены следующие квадратурные формулы:

* Одномерный интеграл: Гаусс-3, Гаусс-4
* Двумерный интеграл: Гаусс-4, Гаусс-12
* Трехмерный интеграл: Гаусс-8, Гаусс-14

## **Код**

*main.py*

|  |
| --- |
| import tkinter as tk  import matplotlib.pyplot as plt  import numpy as np  import KP\_function1 as kp1  import KP\_function2 as kp2  import KP\_function3 as kp3  import KP as kp  from tkinter import ttk  from matplotlib.backends.backend\_tkagg import (FigureCanvasTkAgg,  NavigationToolbar2Tk)  razb=0  N=1000  flag=0  legend=0  flagP=0  choice\_int=""  choice\_meth=''  sc=0  Pnts=[]  Methods = []  def Scale(ax, maxx, minx, maxy, miny, sc):  if sc==0:  dx=(maxx-minx)\*0.05  dy=(maxy-miny)\*0.05  ax.set\_xlim([minx-dx, maxx+dx])  ax.set\_ylim([miny-dy, maxy+dy])  else:  if maxx-minx>maxy-miny:  d=(maxx-minx)\*0.05  half=miny+(maxy-miny)/2  ax.set\_xlim([minx-d, maxx+d])  ax.set\_ylim([half-(maxx-minx)/2-d, half+(maxx-minx)/2+d])  else:  d = (maxy - miny) \* 0.05  half=minx+(maxx-minx)/2  print(half)  ax.set\_ylim([miny - d, maxy + d])  ax.set\_xlim([half - (maxy - miny) / 2 - d, half + (maxy - miny) / 2 + d])  def Scale3d(ax, maxx, minx, maxy, miny, maxz, minz, sc):  if sc==0:  dx=(maxx-minx)\*0.05  dy=(maxy-miny)\*0.05  dz=(maxz-minz)\*0.05  ax.set\_xlim([minx-dx, maxx+dx])  ax.set\_ylim([miny-dy, maxy+dy])  ax.set\_zlim([minz-dz, maxz+dz])  else:  if maxx-minx>maxy-miny and maxx-minx>maxz-minz:  d=(maxx-minx)\*0.05  half=miny+(maxy-miny)/2  half2=minz+(maxz-minz)/2  ax.set\_xlim([minx-d, maxx+d])  ax.set\_ylim([half-(maxx-minx)/2-d, half+(maxx-minx)/2+d])  ax.set\_zlim([half2-(maxx-minx)/2-d, half2+(maxx-minx)/2+d])  elif maxy-miny>maxx-minx and maxy-miny>maxz-minz:  d = (maxy - miny) \* 0.05  half=minx+(maxx-minx)/2  half2 = minz + (maxz - minz) / 2  ax.set\_ylim([miny - d, maxy + d])  ax.set\_xlim([half - (maxy - miny) / 2 - d, half + (maxy - miny) / 2 + d])  ax.set\_zlim([half2-(maxy-miny)/2-d, half2+(maxy-miny)/2+d])  else:  d = (maxz - minz) \* 0.05  half = minx + (maxx - minx) / 2  half2 = miny + (maxy - miny) / 2  ax.set\_zlim([minz - d, maxz + d])  ax.set\_xlim([half - (maxz - minz) / 2 - d, half + (maxz - minz) / 2 + d])  ax.set\_ylim([half2 - (maxz - minz) / 2 - d, half2 + (maxz - minz) / 2 + d])  def plot(razb, ax, Pnts, flag, flagP, choice\_int):  ax.clear()  fig.clear()  print(choice\_int)  if choice\_int=="Одномерный интеграл":  ax = fig.add\_subplot(111)  x = np.linspace(kp1.bx(), kp1.tx(), razb)  ax.axhline(y=0, color='k', linewidth=0.5)  ax.axvline(x=0, color='k', linewidth=0.5)  y = kp1.function(x)  ax.plot(x, y, label="Подинтегральная функция")  ax.plot([kp1.bx(), kp1.bx()], [0, y[0]], color="olive", label="Ограничения по x")  ax.plot([kp1.tx(), kp1.tx()], [0, y[-1]], color="olive")  Scale(ax, np.max(x), np.min(x), np.max(y), np.min(y), sc)  elif choice\_int=="Двумерный интеграл":  print("yes")  if kp2.function(2.4, 2.4)==1:  ax = fig.add\_subplot(111)  if kp2.ty(1)==kp2.ty(2):  ax.axhline(y=0, color='k', linewidth=0.5)  ax.axvline(x=0, color='k', linewidth=0.5)  ax.axhline(y=kp2.ty(0), color="orange", label="Ограничения по y")  ax.axhline(y=kp2.by(0), color="orange")  y = np.linspace(kp2.by(0), kp2.ty(0), razb)  if kp2.tx(2)==kp2.tx(3):  x = np.full(shape=razb, fill\_value=kp2.tx(2))  else:  x = kp2.tx(y)  maxx1=np.max(x); minx1=np.min(x); maxy1=np.max(y); miny1=np.min(y)  ax.plot(x, y, color="olive", label="Ограничения по x")  if kp2.bx(2)==kp2.bx(3):  x = np.full(shape=razb, fill\_value=kp2.bx(2))  else:  x = kp2.bx(y)  ax.plot(x, y, color="olive")  maxx2 = np.max(x); minx2 = np.min(x); maxy2 = np.max(y); miny2 = np.min(y)  Scale(ax, max(maxx1, maxx2), min(minx1, minx2), max(maxy1, maxy2), min(miny1, miny2), sc)  elif kp2.tx(1)==kp2.tx(2):  ax.axhline(y=0, color='k', linewidth=0.5)  ax.axvline(x=0, color='k', linewidth=0.5)  ax.axvline(x=kp2.tx(0), color="olive", label="Ограничения по x")  ax.axvline(x=kp2.bx(0), color="olive")  x = np.linspace(kp2.bx(0), kp2.tx(0), razb)  if kp2.ty(2)==kp2.ty(3):  y = np.full(shape=razb, fill\_value=kp2.ty(2))  else:  y = kp2.ty(x)  maxx1 = np.max(x); minx1 = np.min(x); maxy1 = np.max(y); miny1 = np.min(y)  ax.plot(x, y, color="orange", label="Ограничения по y")  if kp2.by(2)==kp2.by(3):  y=np.full(shape=razb, fill\_value=kp2.by(2))  else:  y = kp2.by(x)  ax.plot(x, y, color="orange")  maxx2 = np.max(x); minx2 = np.min(x); maxy2 = np.max(y); miny2 = np.min(y)  Scale(ax, max(maxx1, maxx2), min(minx1, minx2), max(maxy1, maxy2), min(miny1, miny2), sc)  else:  if kp2.ty(0)==kp2.ty(2):  y = np.linspace(kp2.by(0), kp2.ty(0), int(razb))  x = np.linspace(np.min(kp2.bx(y)), np.max(kp2.tx(y)), int(razb))  X, Y = np.meshgrid(x, y)  ax = fig.add\_subplot(111, projection='3d')  ax.plot\_wireframe(X, Y, np.zeros((len(x), len(y))), color="black")  ax.plot\_wireframe(X, Y, kp2.function(X, Y), color="blue", label="Подинтегральная функция")  z = np.linspace(np.min(kp2.function(X,Y)), np.max(kp2.function(X,Y)), int(razb))  x1 = kp2.tx(y)  Y, Z = np.meshgrid(y, z)  maxx1 = np.max(x1); minx1 = np.min(x1); maxy1 = np.max(y); miny1 = np.min(y)  ax.plot\_wireframe(kp2.tx(Y), Y, Z, color="olive", label="Ограничение по x")  x2 = kp2.bx(y)  ax.plot\_wireframe(kp2.bx(Y), Y, Z, color="olive")  maxx2 = np.max(x2); minx2 = np.min(x2); maxy2 = np.max(y); miny2 = np.min(y)  maxz=kp.Max\_Min2(x, y, kp2.function)[0]  minz=kp.Max\_Min2(x, y, kp2.function)[1]  Scale3d(ax, max(maxx1, maxx2), min(minx1, minx2), max(maxy1, maxy2), min(miny1, miny2), maxz, minz, sc)  elif kp2.tx(2)==kp2.tx(3):  x = np.linspace(kp2.bx(0), kp2.tx(0), int(razb))  y = np.linspace(np.min(kp2.by(x)), np.max(kp2.ty(x)), int(razb))  X, Y = np.meshgrid(x, y)  ax = fig.add\_subplot(111, projection='3d')  ax.plot\_wireframe(X, Y, np.zeros((len(x), len(y))), color="black")  ax.plot\_wireframe(X, Y, kp2.function(X, Y), color="blue", label="Подинтегральная функция")  z = np.linspace(np.min(kp2.function(X, Y)), np.max(kp2.function(X, Y)), int(razb))  y1 = kp2.ty(x)  X, Z = np.meshgrid(y, z)  maxx1 = np.max(x); minx1 = np.min(x); maxy1 = np.max(y1); miny1 = np.min(y1)  ax.plot\_wireframe(X, kp2.ty(X), Z, color="orange", label="Ограничение по y")  y2 = kp2.by(x)  ax.plot\_wireframe(X, kp2.by(X), Z, color="orange")  maxx2 = np.max(x); minx2 = np.min(x); maxy2 = np.max(y2); miny2 = np.min(y2)  maxz = kp.Max\_Min2(x, y, kp2.function)[0]  minz = kp.Max\_Min2(x, y, kp2.function)[1]  Scale3d(ax, max(maxx1, maxx2), min(minx1, minx2), max(maxy1, maxy2), min(miny1, miny2), maxz, minz, sc)  elif choice\_int=="Трехмерный интеграл":  #ax = fig.add\_subplot(111)  ax = fig.add\_subplot(111, projection='3d')  if kp3.order()[0]=='x':  if kp3.order()[1]=='y':  x = np.linspace(kp3.bx(0, 0), kp3.tx(0, 0), int(razb))  y = np.linspace(np.min(kp3.by(x, 0)), np.max(kp3.ty(x, 0)), int(razb))  X, Y = np.meshgrid(x, y)  ax.plot\_wireframe(X, Y, np.zeros((len(x), len(y))), color="black")  z = np.linspace(np.min(kp3.bz(X, Y)), np.max(kp3.tz(X, Y)), int(razb))  y1 = kp3.ty(x, 0)  Y, Z = np.meshgrid(y, z)  ax.plot\_wireframe(kp3.tx(Y, 0), Y, Z, color="olive", label="Ограничение по x")  ax.plot\_wireframe(kp3.bx(Y, 0), Y , Z, color="olive")  X, Z = np.meshgrid(x, z)  maxx1 = np.max(x)  minx1 = np.min(x)  maxy1 = np.max(y1)  miny1 = np.min(y1)  ax.plot\_wireframe(X, kp3.ty(X, 0), Z, color="orange", label="Ограничение по y")  y2 = kp3.by(x, 0)  ax.plot\_wireframe(X, kp3.by(X, 0), Z, color="orange")  X, Y = np.meshgrid(x, y)  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.tz), color="green", label="Ограничение по z")  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.bz), color="green")  maxx2 = np.max(x)  minx2 = np.min(x)  maxy2 = np.max(y2)  miny2 = np.min(y2)  maxz = np.max(kp3.tz(X, Y))  minz = np.min(kp3.bz(X, Y))  Scale3d(ax, max(maxx1, maxx2), min(minx1, minx2), max(maxy1, maxy2), min(miny1, miny2), maxz, minz, sc)  else:  x = np.linspace(kp3.bx(0, 0), kp3.tx(0, 0), int(razb))  z = np.linspace(np.min(kp3.bz(x, 0)), np.max(kp3.tz(x, 0)), int(razb))  X, Z = np.meshgrid(x, z)  #ax.plot\_wireframe(X, Y, np.zeros((len(x), len(y))), color="black")  y = np.linspace(np.min(kp3.by(X, Z)), np.max(kp3.ty(X, Z)), int(razb))  z1 = kp3.tz(x, 0)  Y, Z = np.meshgrid(y, z)  ax.plot\_wireframe(kp3.tx(Y, 0), Y, Z, color="olive", label="Ограничение по x")  ax.plot\_wireframe(kp3.bx(Y, 0), Y, Z, color="olive")  X, Z = np.meshgrid(x, z)  maxx1 = np.max(x)  minx1 = np.min(x)  maxz1 = np.max(z1)  minz1 = np.min(z1)  ax.plot\_wireframe(X, kp3.ty(X, Z), Z, color="orange", label="Ограничение по y")  ax.plot\_wireframe(X, kp3.by(X, Z), Z, color="orange")  z2 = kp3.bz(x, 0)  X, Y = np.meshgrid(x, y)  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.tz), color="green", label="Ограничение по z")  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.bz), color="green")  maxx2 = np.max(x)  minx2 = np.min(x)  maxz2 = np.max(z2)  minz2 = np.min(z2)  maxy = np.max(kp3.ty(X, Z))  miny = np.min(kp3.by(X, Z))  Scale3d(ax, max(maxx1, maxx2), min(minx1, minx2), maxy, miny, max(maxz1, maxz2), min(minz1, minz2), sc)  if kp3.order()[0]=='y':  if kp3.order()[1]=='x':  y = np.linspace(kp3.by(0, 0), kp3.ty(0, 0), int(razb))  x = np.linspace(np.min(kp3.bx(y, 0)), np.max(kp3.tx(y, 0)), int(razb))  X, Y = np.meshgrid(x, y)  ax.plot\_wireframe(X, Y, np.zeros((len(x), len(y))), color="black")  z = np.linspace(np.min(kp3.bz(X, Y)), np.max(kp3.tz(X, Y)), int(razb))  x1 = kp3.tx(y, 0)  Y, Z = np.meshgrid(y, z)  ax.plot\_wireframe(X, kp3.ty(X, 0), Z, color="orange", label="Ограничение по y")  ax.plot\_wireframe(X, kp3.by(X, 0), Z, color="orange")  X, Z = np.meshgrid(y, z)  maxy1 = np.max(y)  miny1 = np.min(y)  maxx1 = np.max(x1)  minx1 = np.min(x1)  ax.plot\_wireframe(kp3.tx(Y, 0), Y, Z, color="olive", label="Ограничение по x")  ax.plot\_wireframe(kp3.bx(Y, 0), Y, Z, color="olive")  x2 = kp3.bx(y, 0)  X, Y = np.meshgrid(x, y)  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.tz), color="green", label="Ограничение по z")  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.bz), color="green")  maxy2 = np.max(y)  miny2 = np.min(y)  maxx2 = np.max(x2)  minx2 = np.min(x2)  maxz = np.max(kp3.tz(X, Y))  minz = np.min(kp3.bz(X, Y))  Scale3d(ax, max(maxx1, maxx2), min(minx1, minx2), max(maxy1, maxy2), min(miny1, miny2), maxz, minz, sc)  else:  y = np.linspace(kp3.by(0, 0), kp3.ty(0, 0), int(razb))  z = np.linspace(np.min(kp3.bz(0, y)), np.max(kp3.tz(0, y)), int(razb))  Y, Z = np.meshgrid(y, z)  #ax.plot\_wireframe(X, Y, np.zeros((len(x), len(y))), color="black")  x = np.linspace(np.min(kp3.bx(Y, Z)), np.max(kp3.tx(Y, Z)), int(razb))  z1 = kp3.tz(0, y)  X, Z = np.meshgrid(x, z)  ax.plot\_wireframe(X, kp3.ty(X, 0), Z, color="orange", label="Ограничение по y")  ax.plot\_wireframe(X, kp3.by(X, 0), Z, color="orange")  Y, Z = np.meshgrid(y, z)  maxy1 = np.max(y)  miny1 = np.min(y)  maxz1 = np.max(z1)  minz1 = np.min(z1)  ax.plot\_wireframe(kp3.tx(Y, Z), Y, Z, color="olive", label="Ограничение по x")  ax.plot\_wireframe(kp3.bx(Y, Z), Y, Z, color="olive")  z2 = kp3.bx(0, y)  X, Y = np.meshgrid(x, y)  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.tz), color="green", label="Ограничение по z")  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.bz), color="green")  maxy2 = np.max(y)  miny2 = np.min(y)  maxz2 = np.max(z2)  minz2 = np.min(z2)  maxx = np.max(kp3.tx(Y, Z))  minx = np.min(kp3.bx(Y, Z))  Scale3d(ax, maxx, minx, max(maxy1, maxy2), min(miny1, miny2), max(maxz1, maxz2), min(minz1, minz2), sc)  if kp3.order()[0]=='z':  if kp3.order()[1]=='y':  z = np.linspace(kp3.bz(0, 0), kp3.tz(0, 0), int(razb))  y = np.linspace(np.min(kp3.by(0, z)), np.max(kp3.ty(0, z)), int(razb))  Y, Z = np.meshgrid(y, z)  x = np.linspace(np.min(kp3.bx(Y, Z)), np.max(kp3.tx(Y, Z)), int(razb))  y1 = kp3.ty(0, z)  Y, Z = np.meshgrid(y, z)  ax.plot\_wireframe(kp3.tx(Y, Z), Y, Z, color="olive", label="Ограничение по x")  ax.plot\_wireframe(kp3.bx(Y, Z), Y, Z, color="olive")  X, Z = np.meshgrid(x, z)  maxz1 = np.max(z)  minz1 = np.min(z)  maxy1 = np.max(y1)  miny1 = np.min(y1)  ax.plot\_wireframe(X, kp3.ty(0, Z), Z, color="orange", label="Ограничение по y")  ax.plot\_wireframe(X, kp3.by(0, Z), Z, color="orange")  y2 = kp3.by(x, 0)  X, Y = np.meshgrid(x, y)  ax.plot\_wireframe(X, Y, np.zeros((len(x), len(y))), color="black")  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.tz), color="green", label="Ограничение по z")  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.bz), color="green")  maxz2 = np.max(z)  minz2 = np.min(z)  maxy2 = np.max(y2)  miny2 = np.min(y2)  maxx = np.max(kp3.tx(X, Y))  minx = np.min(kp3.bx(X, Y))  Scale3d(ax, maxx, minx, max(maxy1, maxy2), min(miny1, miny2), max(maxz1, maxz2), min(minz1, minz2), sc)  else:  z = np.linspace(kp3.bz(0, 0), kp3.tz(0, 0), int(razb))  x = np.linspace(np.min(kp3.bx(0, z)), np.max(kp3.tx(0, z)), int(razb))  print(x[0], x[-1])  X, Z = np.meshgrid(x, z)  y = np.linspace(np.min(kp3.by(X, Z)), np.max(kp3.ty(X, Z)), int(razb))  x1 = kp3.tx(0, z)  ax.plot\_wireframe(X, kp3.ty(X, Z), Z, color="orange", label="Ограничение по y")  ax.plot\_wireframe(X, kp3.by(X, Z), Z, color="orange")  X, Y = np.meshgrid(x, y)  ax.plot\_wireframe(X, Y, np.zeros((len(x), len(y))), color="black")  Y, Z = np.meshgrid(y, z)  ax.plot\_wireframe(kp3.tx(0, Z), Y, Z, color="olive", label="Ограничение по x")  ax.plot\_wireframe(kp3.bx(0, Z), Y, Z, color="olive")  X, Z = np.meshgrid(y, z)  maxz1 = np.max(z)  minz1 = np.min(z)  maxx1 = np.max(x1)  minx1 = np.min(x1)  x2 = kp3.bx(0, z)  X, Y = np.meshgrid(x, y)  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.tz), color="green", label="Ограничение по z")  ax.plot\_wireframe(X, Y, kp.Border(X, Y, kp3.bz), color="green")  maxz2 = np.max(z)  minz2 = np.min(z)  maxx2 = np.max(x2)  minx2 = np.min(x2)  maxy = np.max(kp3.ty(X, Y))  miny = np.min(kp3.by(X, Y))  Scale3d(ax, max(maxx1, maxx2), min(minx1, minx2), maxy, miny, max(maxz1, maxz2), min(minz1, minz2), sc)  if flag==1:  if len(Pnts)==3:  plt.ion()  X=[]; Y=[]  for i in range(N):  if Pnts[2][i]==1:  X.append(Pnts[0][i])  Y.append(Pnts[1][i])  ax.scatter(X, Y, s=1, color="gray")  plt.ioff()  else:  plt.ion()  X = []; Y = []; Z=[]  for i in range(N):  if Pnts[3][i] == 1:  X.append(Pnts[0][i])  Y.append(Pnts[1][i])  Z.append(Pnts[2][i])  ax.scatter(X, Y, Z, s=2, color="gray")  plt.ioff()  if flagP==1:  if len(Pnts) == 3:  plt.ion()  X = []; Y = []  for i in range(N):  if Pnts[2][i] == 0:  X.append(Pnts[0][i])  Y.append(Pnts[1][i])  ax.scatter(X, Y, s=1, color="red")  plt.ioff()  else:  plt.ion()  X = []; Y = []; Z = []  for i in range(N):  if Pnts[3][i] == 0:  X.append(Pnts[0][i])  Y.append(Pnts[1][i])  Z.append(Pnts[2][i])  ax.scatter(X, Y, Z, s=2, color="red")  plt.ioff()  if legend==1:  ax.legend()  ax.set\_title("График подинтегрального выражения")  ax.grid()  canvas.draw()  def checkbutton\_changed():  global flag  if enabled.get() == 1:  flag=1  plot(razb, ax, Pnts, flag, flagP, choice\_int)  else:  flag=0  plot(razb, ax, Pnts, flag, flagP, choice\_int)  def checkbutton\_changed2():  global flagP  if enabled2.get() == 1:  flagP=1  plot(razb, ax, Pnts, flag, flagP, choice\_int)  else:  flagP=0  plot(razb, ax, Pnts, flag, flagP, choice\_int)  def display\_selected(choice):  global choice\_int  global choice\_1  global Methods  choice\_int=choice  if choice\_int=="Одномерный интеграл":  #Methods = ["Метод Монте-Карло", "Метод Гаусса-2"]  menu=combobox2['menu']  menu.delete(0, 'end')  combobox2.set\_menu("None", "Метод Монте-Карло", "Метод Гаусса-3", "Метод Гаусса-4")  elif choice\_int=="Двумерный интеграл":  menu = combobox2['menu']  menu.delete(0, 'end')  combobox2.set\_menu("None", "Метод Монте-Карло", "Метод Гаусса-4", "Метод Гаусса-12")  elif choice\_int=="Трехмерный интеграл":  menu = combobox2['menu']  menu.delete(0, 'end')  combobox2.set\_menu("None", "Метод Монте-Карло", "Метод Гаусса-8", "Метод Гаусса-14")  print(choice\_int)  def display\_selected\_meth(choice):  global choice\_meth  global choice\_1  choice\_meth=choice  print(choice\_meth)  def click\_button():  global Pnts  global N  global razb  labelWarn.place\_forget()  N = int(PointsNumber.get())  razb = int(DiscretNumber.get())  checkBox\_Points.place(relx=0.05, rely=0.95)  checkBox\_Points2.place(relx=0.4, rely=0.95)  checkBox\_Scale.place(relx=0.7, rely=0.95)  checkBox\_Legend.place(relx=0.85, rely=0.95)  print(razb)  if choice\_int=="Одномерный интеграл" and choice\_meth=="Метод Монте-Карло":  checkBox\_Points.place(relx=0.05, rely=0.95)  checkBox\_Points2.place(relx=0.4, rely=0.95)  res, Pnts = kp.Monte\_Karlo\_1(N, razb)  labelRes["text"] = str(res)  plot(razb, ax, Pnts, flag, flagP, choice\_int)  elif choice\_int=="Двумерный интеграл" and choice\_meth=="Метод Монте-Карло":  checkBox\_Points.place(relx=0.05, rely=0.95)  checkBox\_Points2.place(relx=0.4, rely=0.95)  if kp2.function(2, 2)==kp2.function(3, 3):  res, Pnts = kp.Monte\_Karlo2(N, razb)  labelRes["text"] = str(res)  plot(razb, ax, Pnts, flag, flagP, choice\_int)  else:  res, Pnts = kp.Monte\_Karlo2\_3D(N, razb)  labelRes["text"]=str(res)  plot(razb, ax, Pnts, flag, flagP, choice\_int)  elif choice\_int=="Трехмерный интеграл" and choice\_meth=="Метод Монте-Карло":  checkBox\_Points.place(relx=0.05, rely=0.95)  checkBox\_Points2.place(relx=0.4, rely=0.95)  if kp3.function(2, 1 ,3)!=kp3.function(4, 3, 2) and kp3.function(2, 1,3)!=1:  labelWarn['text'] = "Этот интеграл нельзя вычислить данным методом"  labelWarn.place(relx=0.55, rely=0.2)  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  checkBox\_Legend.place\_forget()  checkBox\_Scale.place\_forget()  return  res, Pnts = kp.Monte\_Karlo2\_3D\_3(N, razb)  labelRes["text"]=str(res)  plot(razb, ax, Pnts, flag, flagP, choice\_int)  if choice\_int=="Одномерный интеграл" and choice\_meth=="Метод Гаусса-3":  res = kp.Gausse\_1(3)  labelRes["text"]=str(res)  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  plot(razb, ax, Pnts, flag, flagP, choice\_int)  if choice\_int == "Одномерный интеграл" and choice\_meth == "Метод Гаусса-4":  res = kp.Gausse\_1(4)  labelRes["text"] = str(res)  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  plot(razb, ax, Pnts, flag, flagP, choice\_int)  if choice\_int=="Двумерный интеграл" and choice\_meth=="Метод Гаусса-4":  if kp2.tx(1) != kp2.tx(2) or kp2.bx(1)!=kp2.bx(2) or kp2.by(1)!=kp2.by(2) or kp2.ty(1)!=kp2.ty(2):  labelWarn['text']="Этот интеграл нельзя вычислить данным методом"  labelWarn.place(relx=0.55, rely=0.2)  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  checkBox\_Legend.place\_forget()  checkBox\_Scale.place\_forget()  return  res = kp.Gausse\_2(4)  labelRes["text"] = str(res)  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  plot(razb, ax, Pnts, flag, flagP, choice\_int)  if choice\_int=="Двумерный интеграл" and choice\_meth=="Метод Гаусса-12":  if kp2.tx(1) != kp2.tx(2) or kp2.bx(1)!=kp2.bx(2) or kp2.by(1)!=kp2.by(2) or kp2.ty(1)!=kp2.ty(2):  labelWarn['text']="Этот интеграл нельзя вычислить данным методо"  labelWarn.place(relx=0.55, rely=0.2)  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  checkBox\_Legend.place\_forget()  checkBox\_Scale.place\_forget()  return  res = kp.Gausse\_2(12)  labelRes["text"] = str(res)  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  plot(razb, ax, Pnts, flag, flagP, choice\_int)  if choice\_int=="Трехмерный интеграл" and choice\_meth=="Метод Гаусса-8":  if kp3.tx(1, 1) != kp3.tx(2, 2) or kp3.bx(1, 1)!=kp3.bx(2, 2) or kp3.by(1, 1)!=kp3.by(2, 2) or kp3.ty(1, 1)!=kp3.ty(2, 2) or kp3.tz(1, 1)!=kp3.tz(2, 2) or kp3.bz(1, 1)!=kp3.bz(2, 2):  labelWarn['text']="Этот интеграл нельзя вычислить данным методо"  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  checkBox\_Legend.place\_forget()  checkBox\_Scale.place\_forget()  return  res = kp.Gausse\_3(8)  labelRes["text"] = str(res)  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  if kp3.function(2, 1 ,3)!=kp3.function(4, 3, 2) and kp3.function(2, 1,3)!=1:  labelWarn.place(relx=0.55, rely=0.2)  labelWarn["text"]="У данного интеграла нет геометрической интерпретации"  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  checkBox\_Legend.place\_forget()  checkBox\_Scale.place\_forget()  return  plot(razb, ax, Pnts, flag, flagP, choice\_int)  if choice\_int=="Трехмерный интеграл" and choice\_meth=="Метод Гаусса-14":  res = kp.Gausse\_3(14)  labelRes["text"] = str(res)  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  if kp3.function(2, 1 ,3)!=kp3.function(4, 3, 2) and kp3.function(2, 1,3)!=1:  labelWarn.place(relx=0.55, rely=0.2)  labelWarn["text"] = "У данного интеграла нет геометрической интерпретации"  checkBox\_Points.place\_forget()  checkBox\_Points2.place\_forget()  checkBox\_Legend.place\_forget()  checkBox\_Scale.place\_forget()  return  plot(razb, ax, Pnts, flag, flagP, choice\_int)  def checkbutton\_Scale\_changed():  global sc  if enabled1.get() == 1:  sc=1  plot(razb, ax, Pnts, flag, flagP, choice\_int)  else:  sc=0  plot(razb, ax, Pnts, flag, flagP, choice\_int)  def checkbutton\_Legend\_changed():  global legend  if enabled\_L.get()==1:  legend=1  plot(razb, ax, Pnts, flag, flagP, choice\_int)  else:  legend=0  plot(razb, ax, Pnts, flag, flagP, choice\_int)  root = tk.Tk()  root.geometry("1000x600")  root.title('Курсовой проект')  root.resizable(0, 0)  frame = tk.Frame(root, borderwidth=1, relief="solid")  frame.pack(anchor="nw", padx=20, pady=10)  fig, ax = plt.subplots(figsize=(5, 5), dpi=100)  canvas = FigureCanvasTkAgg(fig, master=frame)  canvas.get\_tk\_widget().pack()  toolbar = NavigationToolbar2Tk(canvas, root)  toolbar.update()  canvas.get\_tk\_widget().pack()  enabled = tk.IntVar()  checkBox\_Points=ttk.Checkbutton(text="Точки вне области", variable=enabled, command=checkbutton\_changed, master=frame)  #checkBox\_Points.place(relx=0.05, rely=0.95)  enabled2 = tk.IntVar()  checkBox\_Points2=ttk.Checkbutton(text="Точки в области", variable=enabled2, command=checkbutton\_changed2, master=frame)  #checkBox\_Points2.place(relx=0.4, rely=0.95)  enabled1 = tk.IntVar()  checkBox\_Scale=ttk.Checkbutton(text="1:1", variable=enabled1, command=checkbutton\_Scale\_changed, master=frame)  #checkBox\_Scale.place(relx=0.7, rely=0.95)  enabled\_L = tk.IntVar()  checkBox\_Legend=ttk.Checkbutton(text="Легенда", variable=enabled\_L, command=checkbutton\_Legend\_changed, master=frame)  #checkBox\_Legend.place(relx=0.85, rely=0.95)  choice\_=tk.StringVar()  choice\_.set("None")  Types=["None", "Одномерный интеграл", "Двумерный интеграл", "Трехмерный интеграл"]  combobox = ttk.OptionMenu(root, choice\_, \*Types, command=display\_selected)  combobox.place(relx=0.7, rely=0.02)  choice\_1=tk.StringVar()  choice\_1.set("None")  combobox2 = ttk.OptionMenu(root, choice\_1, \*Methods, command=display\_selected\_meth)  combobox2.place(relx=0.7, rely=0.07)  btn = ttk.Button(text="Вычислить интеграл", command=click\_button, master=root)  btn.place(relx=0.7, rely=0.8)  PointsNumber = ttk.Entry()  PointsNumber.place(relx=0.7, rely=0.5)  PointsNumber.insert(0, "1000")  DiscretNumber = ttk.Entry()  DiscretNumber.place(relx=0.7, rely=0.4)  DiscretNumber.insert(0, "10")  label1 = ttk.Label(text="Дискретизация области")  label1.place(relx=0.55, rely=0.4)  label2 = ttk.Label(text="Кол-во точек")  label2.place(relx=0.55, rely=0.5)  labelRes = ttk.Label()  labelRes.place(relx=0.7, rely=0.9)  labelWarn = ttk.Label(text="У данного интеграла нет геометрической интерпретации", foreground='red')  #plot(1000, ax, Px, Py, flag, choice\_int)  root.mainloop() |

*KP.py*

|  |
| --- |
| import KP\_function1 as kp1  import KP\_function2 as kp2  import numpy as np  import random  import KP\_function3 as kp3  from inspect import signature  def Max\_Min(razb, tx, bx, ty, by, tz, bz, function):  dlt = (tx(2, 0) - bx(2, 0)) / razb  x = np.arange(bx(2, 0), tx(2, 0) + dlt, dlt)  if kp3.order()[0]=='z' or (kp3.order()[0]=='y' and kp3.order()[1]=='z'):  y\_max = np.max(ty(0, x))  y\_min = np.min(by(0, x))  else:  y\_max = np.max(ty(x, 0))  y\_min = np.min(by(x, 0))  x\_max = tx(0, 0)  x\_min = bx(0, 0)  X = np.linspace(x\_min, x\_max, razb)  Y = np.linspace(y\_min, y\_max, razb)  if kp3.order()[0]=='y' and kp3.order()[1]=='x':  z\_max1, z\_min1 = Max\_Min3(Y, X, function)  z\_max = max(z\_max1, Max\_Min2(Y, X, tz)[0])  z\_min = min(z\_min1, Max\_Min2(Y, X, bz)[1])  else:  z\_max1, z\_min1 = Max\_Min3(X, Y, function)  z\_max=max(z\_max1, Max\_Min2(X, Y, tz)[0])  z\_min=min(z\_min1, Max\_Min2(X, Y, bz)[1])  return x\_max, x\_min, y\_max, y\_min, z\_max, z\_min  def Border(X, Y, func):  sig = signature(func)  param = sig.parameters  if len(param)==2:  Z = np.zeros((len(X), len(Y)))  for i in range(len(X)):  for j in range(len(Y)):  Z[i][j]=func(X[i][j], Y[i][j])  else:  Z = np.zeros((len(X), len(Y)))  for i in range(len(X)):  for j in range(len(Y)):  Z[i][j] = func(X[i][j], Y[i][j], 0)  return Z  def Max\_Min2(X, Y, function):  mmax = function(X[0], Y[0])  mmin = function(X[0], Y[0])  for i in range(len(X)):  for j in range(len(Y)):  if function(X[i], Y[j])>mmax:  mmax=function(X[i], Y[j])  if function(X[i], Y[j])<mmin:  mmin=function(X[i], Y[j])  return mmax, mmin  def Max\_Min3(X, Y, function):  mmax = function(X[0], Y[0], 0)  mmin = function(X[0], Y[0], 0)  for i in range(len(X)):  for j in range(len(Y)):  if function(X[i], Y[j], 0)>mmax:  mmax=function(X[i], Y[j], 0)  if function(X[i], Y[j], 0)<mmin:  mmin=function(X[i], Y[j], 0)  return mmax, mmin  def Monte\_Karlo\_1(N, razb):  a=kp1.bx()  b=kp1.tx()  dlt=(b-a)/razb  x=np.arange(a, b+dlt, dlt)  mx = np.max(kp1.function(x))  mn = np.min(kp1.function(x))  print(mx, mn)  x\_max = b  x\_min = a  y\_max = mx  y\_min = 0  if mn<0:  y\_min=mn  cntP=0  cntO=0  Pnts=np.ones((3,N))  for i in range(N):  xi=x\_min+(x\_max-x\_min)\*random.random()  yi=y\_min+(y\_max-y\_min)\*random.random()  Pnts[0][i]=xi  Pnts[1][i]=yi  #print(xi)  if kp1.function(xi)\*yi>0:  if kp1.function(xi)>0 and kp1.function(xi)>yi:  Pnts[2][i]=0  cntP+=1  elif kp1.function(xi)<0 and kp1.function(xi)<yi:  cntO+=1  Pnts[2][i]=0  dolP=cntP/N  dolO=cntO/N  S=(x\_max-x\_min)\*(y\_max-y\_min)  print(cntP, cntO)  res=S\*dolP-S\*dolO  return res, Pnts  def Monte\_Karlo2(N, razb):  if kp2.ty(2)==kp2.ty(1):  dlt = (kp2.ty(2) - kp2.by(2)) / razb  y = np.arange(kp2.by(2), kp2.ty(2) + dlt, dlt)  x\_max= np.max(kp2.tx(y))  x\_min= np.min(kp2.bx(y))  y\_max=kp2.ty(0)  y\_min=kp2.by(0)  else:  dlt = (kp2.tx(2) - kp2.bx(2)) / razb  x = np.arange(kp2.bx(2), kp2.tx(2) + dlt, dlt)  y\_max = np.max(kp2.ty(x))  y\_min = np.min(kp2.by(x))  x\_max = kp2.tx(0)  x\_min = kp2.bx(0)  cnt = 0  Pnts=np.ones((3,N))  print(x\_max, x\_min, y\_max, y\_min)  for i in range(N):  xi=x\_min+(x\_max-x\_min)\*random.random()  yi=y\_min+(y\_max-y\_min)\*random.random()  Pnts[0][i]=xi  Pnts[1][i]=yi  if kp2.ty(xi)>yi and kp2.by(xi)<yi:  if kp2.tx(yi)>xi and kp2.bx(yi)<xi:  cnt+=1  Pnts[2][i]=0  dol=cnt/N  S=(x\_max-x\_min)\*(y\_max-y\_min)  print(cnt)  res=S\*dol  return res, Pnts  def Monte\_Karlo2\_3D(N, razb):  if kp2.ty(2)==kp2.ty(1):  dlt = (kp2.ty(2) - kp2.by(2)) / razb  y = np.arange(kp2.by(2), kp2.ty(2) + dlt, dlt)  x\_max= np.max(kp2.tx(y))  x\_min= np.min(kp2.bx(y))  y\_max=kp2.ty(0)  y\_min=kp2.by(0)  else:  dlt = (kp2.tx(2) - kp2.bx(2)) / razb  x = np.arange(kp2.bx(2), kp2.tx(2) + dlt, dlt)  y\_max = np.max(kp2.ty(x))  y\_min = np.min(kp2.by(x))  x\_max = kp2.tx(0)  x\_min = kp2.bx(0)  X = np.linspace(x\_min, x\_max, razb)  Y = np.linspace(y\_min, y\_max, razb)  z\_max, z\_min = Max\_Min2(X, Y, kp2.function)  cntP = 0  cntO = 0  Pnts=np.ones((4,N))  print(x\_max, x\_min, y\_max, y\_min, z\_max, z\_min)  for i in range(N):  xi=x\_min+(x\_max-x\_min)\*random.random()  yi=y\_min+(y\_max-y\_min)\*random.random()  zi=z\_min+(z\_max-z\_min)\*random.random()  Pnts[0][i]=xi  Pnts[1][i]=yi  Pnts[2][i]=zi  if kp2.ty(xi)>yi and kp2.by(xi)<yi:  if kp2.tx(yi)>xi and kp2.bx(yi)<xi:  if kp2.function(xi, yi)>0 and kp2.function(xi, yi)>zi and zi>0:  cntP+=1  Pnts[3][i] = 0  elif kp2.function(xi, yi)<0 and kp2.function(xi, yi)<zi and zi<0:  cntO += 1  Pnts[3][i] = 0  dol1=cntP/N  dol2=cntO/N  S=(x\_max-x\_min)\*(y\_max-y\_min)\*(z\_max-z\_min)  print(S)  res=S\*dol1-S\*dol2  return res, Pnts  def Monte\_Karlo2\_3D\_3(N, razb):  if kp3.order()[0]=='x':  if kp3.order()[1]=='y':  x\_max, x\_min, y\_max, y\_min, z\_max, z\_min = Max\_Min(razb, kp3.tx, kp3.bx, kp3.ty, kp3.by, kp3.tz, kp3.bz, kp3.function)  else:  x\_max, x\_min, z\_max, z\_min, y\_max, y\_min = Max\_Min(razb, kp3.tx, kp3.bx, kp3.tz, kp3.bz, kp3.ty, kp3.by, kp3.function)  if kp3.order()[0]=='y':  if kp3.order()[1]=='x':  y\_max, y\_min, x\_max, x\_min, z\_max, z\_min = Max\_Min(razb, kp3.ty, kp3.by, kp3.tx, kp3.bx, kp3.tz, kp3.bz, kp3.function)  else:  y\_max, y\_min, z\_max, z\_min, x\_max, x\_min = Max\_Min(razb, kp3.ty, kp3.by, kp3.tz, kp3.bz, kp3.tx, kp3.bx, kp3.function)  if kp3.order()[0]=='z':  if kp3.order()[1]=='x':  z\_max, z\_min, x\_max, x\_min, y\_max, y\_min = Max\_Min(razb, kp3.tz, kp3.bz, kp3.tx, kp3.bx, kp3.ty, kp3.by, kp3.function)  else:  z\_max, z\_min, y\_max, y\_min, x\_max, x\_min = Max\_Min(razb, kp3.tz, kp3.bz, kp3.ty, kp3.by, kp3.tx, kp3.bx, kp3.function)  cntP = 0  cntO = 0  Pnts = np.ones((4, N))  print(x\_max, x\_min, y\_max, y\_min, z\_max, z\_min)  for i in range(N):  xi=x\_min+(x\_max-x\_min)\*random.random()  yi=y\_min+(y\_max-y\_min)\*random.random()  zi=z\_min+(z\_max-z\_min)\*random.random()  Pnts[0][i]=xi  Pnts[1][i]=yi  Pnts[2][i]=zi  if kp3.ty(xi, zi)>yi and kp3.by(xi, zi)<yi:  if kp3.tx(yi, zi)>xi and kp3.bx(yi, zi)<xi:  if kp3.function(xi, yi, 0)==kp3.function(xi+1, yi+1, 0) and kp3.function(xi, yi, zi)==1:  if zi>0 and kp3.tz(xi, yi)>zi:  cntP+=1  Pnts[3][i]=0  elif zi<0 and kp3.bz(xi, yi)<zi:  cntO += 1  Pnts[3][i] = 0  dol1=cntP/N  dol2=cntO/N  S=(x\_max-x\_min)\*(y\_max-y\_min)\*(z\_max-z\_min)  print(S)  res=S\*dol1-S\*dol2  return res, Pnts  def Master\_element\_1(x0, x1, psi):  return ((x1-x0)\*(psi+1))/2+x0  def Gausse\_1(n):  if n==3:  psi=[-np.sqrt(3/5), 0, np.sqrt(3/5)]  w = [5/9, 8/9, 5/9]  elif n==4:  psi = [-np.sqrt(3/7 - 2/7\*np.sqrt(6/5)), np.sqrt(3/7 - 2/7\*np.sqrt(6/5)), -np.sqrt(3/7 + 2/7\*np.sqrt(6/5)), np.sqrt(3/7 + 2/7\*np.sqrt(6/5))]  w = [(18+np.sqrt(30))/36, (18+np.sqrt(30))/36, (18-np.sqrt(30))/36, (18-np.sqrt(30))/36]  res=0  for i in range(len(psi)):  res+=w[i]\*(kp1.tx()-kp1.bx())/2\*kp1.function(Master\_element\_1(kp1.bx(), kp1.tx(), psi[i]))  return res  def Gausse\_2(n):  if n==4:  psi = [-np.sqrt(1/3), -np.sqrt(1/3), np.sqrt(1/3), np.sqrt(1/3)]  eta = [-np.sqrt(1/3), np.sqrt(1/3), -np.sqrt(1/3), np.sqrt(1/3)]  w = [1, 1, 1, 1]  elif n==12:  psi = [-np.sqrt(6/3), np.sqrt(6/3), 0, 0, -np.sqrt((114-3\*np.sqrt(583))/287), np.sqrt((114-3\*np.sqrt(583))/287), -np.sqrt((114-3\*np.sqrt(583))/287), np.sqrt((114-3\*np.sqrt(583))/287), -np.sqrt((114+3\*np.sqrt(583))/287), np.sqrt((114+3\*np.sqrt(583))/287), -np.sqrt((114+3\*np.sqrt(583))/287), np.sqrt((114+3\*np.sqrt(583))/287)]  eta = [0, 0, -np.sqrt(6/3), np.sqrt(6/3), -np.sqrt((114-3\*np.sqrt(583))/287), -np.sqrt((114-3\*np.sqrt(583))/287), np.sqrt((114-3\*np.sqrt(583))/287), np.sqrt((114-3\*np.sqrt(583))/287), -np.sqrt((114+3\*np.sqrt(583))/287), -np.sqrt((114+3\*np.sqrt(583))/287), np.sqrt((114+3\*np.sqrt(583))/287), np.sqrt((114+3\*np.sqrt(583))/287)]  w = [98/405, 98/405, 98/405, 98/405, 307/810+923/(270\*np.sqrt(583)), 307/810+923/(270\*np.sqrt(583)), 307/810+923/(270\*np.sqrt(583)), 307/810+923/(270\*np.sqrt(583)), 307/810-923/(270\*np.sqrt(583)), 307/810-923/(270\*np.sqrt(583)), 307/810-923/(270\*np.sqrt(583)), 307/810-923/(270\*np.sqrt(583))]  res=0  for i in range(len(psi)):  res += w[i] \* (kp2.tx(0)-kp2.bx(0))\*(kp2.ty(0)-kp2.by(0))/4 \* kp2.function(Master\_element\_1(kp2.bx(0), kp2.tx(0), psi[i]), Master\_element\_1(kp2.by(0), kp2.ty(0), eta[i]))  #print(Master\_element\_1(kp2.bx(0), kp2.tx(0), psi[i]))  #print(Master\_element\_1(kp2.by(0), kp2.ty(0), psi[i]))  return res  def Gausse\_3(n):  if n==8:  psi = [-np.sqrt(1 / 3), -np.sqrt(1 / 3), -np.sqrt(1 / 3), -np.sqrt(1 / 3), np.sqrt(1 / 3), np.sqrt(1 / 3), np.sqrt(1 / 3), np.sqrt(1 / 3)]  eta = [-np.sqrt(1 / 3), -np.sqrt(1 / 3), np.sqrt(1 / 3), np.sqrt(1 / 3), -np.sqrt(1 / 3), -np.sqrt(1 / 3), np.sqrt(1 / 3), np.sqrt(1 / 3)]  ksi = [-np.sqrt(1 / 3), np.sqrt(1 / 3), -np.sqrt(1 / 3), np.sqrt(1 / 3), -np.sqrt(1 / 3), np.sqrt(1 / 3), -np.sqrt(1 / 3), np.sqrt(1 / 3)]  w = [1, 1, 1, 1, 1, 1, 1, 1]  elif n==14:  psi = [-np.sqrt(19/30), np.sqrt(19/30), 0, 0, 0, 0, -np.sqrt(19/33), -np.sqrt(19/33), -np.sqrt(19/33), -np.sqrt(19/33), np.sqrt(19/33), np.sqrt(19/33), np.sqrt(19/33), np.sqrt(19/33)]  eta = [0, 0, -np.sqrt(19/30), np.sqrt(19/30), 0, 0, -np.sqrt(19/33), -np.sqrt(19/33), np.sqrt(19/33), np.sqrt(19/33), -np.sqrt(19/33), -np.sqrt(19/33), np.sqrt(19/33), np.sqrt(19/33)]  ksi = [0, 0, 0, 0, -np.sqrt(19/30), np.sqrt(19/30), -np.sqrt(19/33), np.sqrt(19/33), -np.sqrt(19/33), np.sqrt(19/33), -np.sqrt(19/33), np.sqrt(19/33), -np.sqrt(19/33), np.sqrt(19/33)]  w = [320/361, 320/361, 320/361, 320/361, 320/361, 320/361, 121/361, 121/361, 121/361, 121/361, 121/361, 121/361, 121/361, 121/361]  res = 0  for i in range(len(psi)):  res += w[i] \* (kp3.tx(0, 0) - kp3.bx(0, 0)) \* (kp3.ty(0, 0) - kp3.by(0, 0))\*(kp3.tz(0,0)-kp3.bz(0,0))/ 8 \* kp3.function(Master\_element\_1(kp3.bx(0, 0), kp3.tx(0, 0), psi[i]), Master\_element\_1(kp3.by(0, 0), kp3.ty(0, 0), eta[i]), Master\_element\_1(kp3.bz(0,0), kp3.tz(0,0), ksi[i]))  return res  #print(Gausse\_2(1000))  #print(Monte\_Karlo2(5000, 1000))  #print(Monte\_Karlo\_1(50000000, 100)) |

*KP\_function1.py*

|  |
| --- |
| import numpy as np  def function(x):  return #Подинтегральная функция одномерного интеграла  def tx():  return #Верхнее ограничение по x одномерного интеграла  def bx():  return #Нижнее ограничение по x одномерного интеграла |

*KP\_function2.py*

|  |
| --- |
| import numpy as np  def function(x, y):  return #Подинтегральная функция двумерного интеграла  def tx(y):  return #Верхнее ограничение по x двумерного интеграла  def bx(y):  return #Нижнее ограничение по x двумерного интеграла  def ty(x):  return #Верхнее ограничение по y двумерного интеграла  def by(x):  return #Нижнее ограничение по y двумерного интеграла |

*KP\_function3.py*

|  |
| --- |
| *import numpy as np*  *def function(x, y, z):*  *return* #Подинтегральная функция трехмерного интеграла  *def tx(y, z):*  *return #Верхнее ограничение по x трехмерного интеграла*  *def bx(y, z):*  *return #Нижнее ограничение по x трехмерного интеграла*  *def ty(x, z):*  *return #Верхнее ограничение по y трехмерного интеграла*  *def by(x, z):*  *return #Нижнее ограничение по y трехмерного интеграла*  *def tz(x, y):*  *return #Верхнее ограничение по z трехмерного интеграла*  *def bz(x, y):*  *return #Нижнее ограничение по z трехмерного интеграла*  *def order():*  *return['x', 'y', 'z'] #Порядок интегрирования* |

## **Интерфейс**

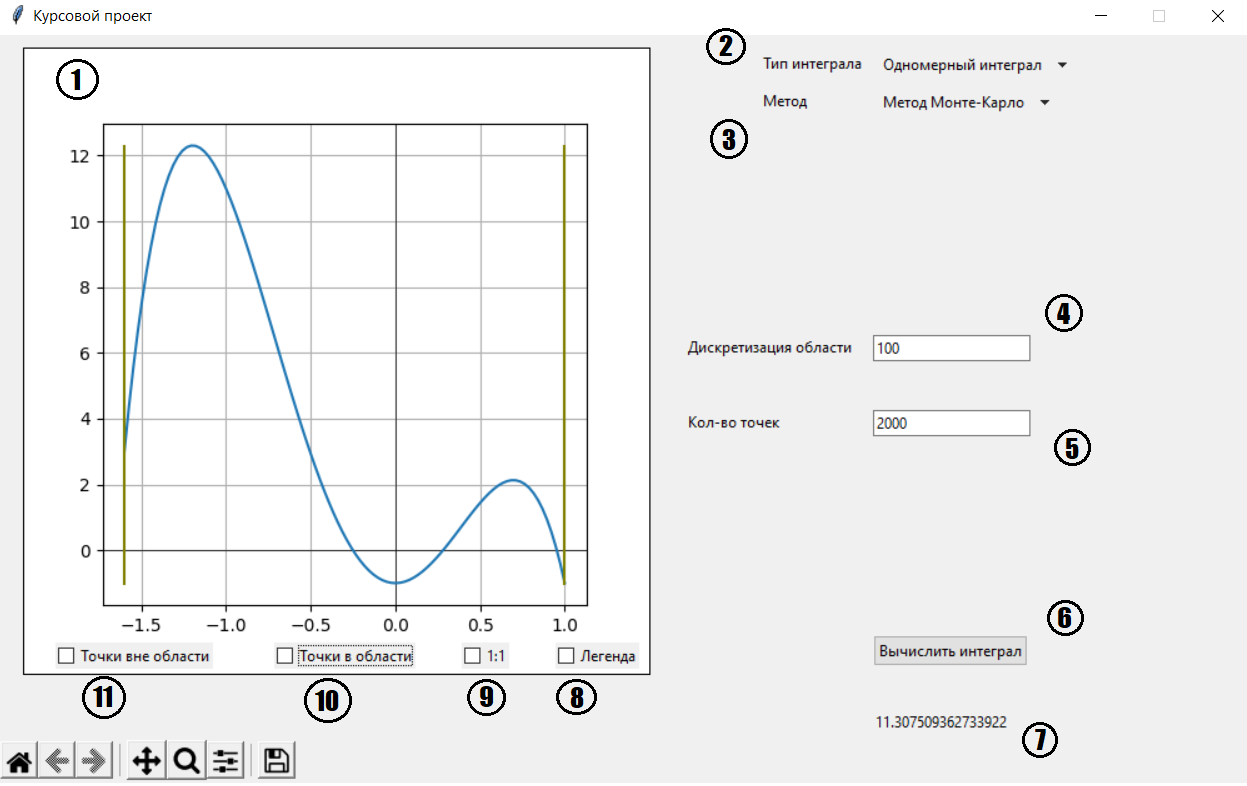


Рисунок 1. Окно программы

1) Область для построения графиков

2) Меню для выбора тип интеграла

3) Меню для выбора метода решения интеграла

4) Область для ввода дискретизации области

5) Область для ввода числа точек (для Метода Монте-Карло)

6) Кнопка для вычисления решения интеграла

7) Результаты вычисления интеграла

8) Легенда графика

9) Соотношение масштаба по осям 1:1

10) Отображение точек, попавших в область интегрирования (для метода Монте-Карло)

11) Отображение точек, не попавших в область интегрирования (для метода Монте-Карло)

## **Тестирование**

*Одномерный интеграл*

Аналитическое решение: 11.979

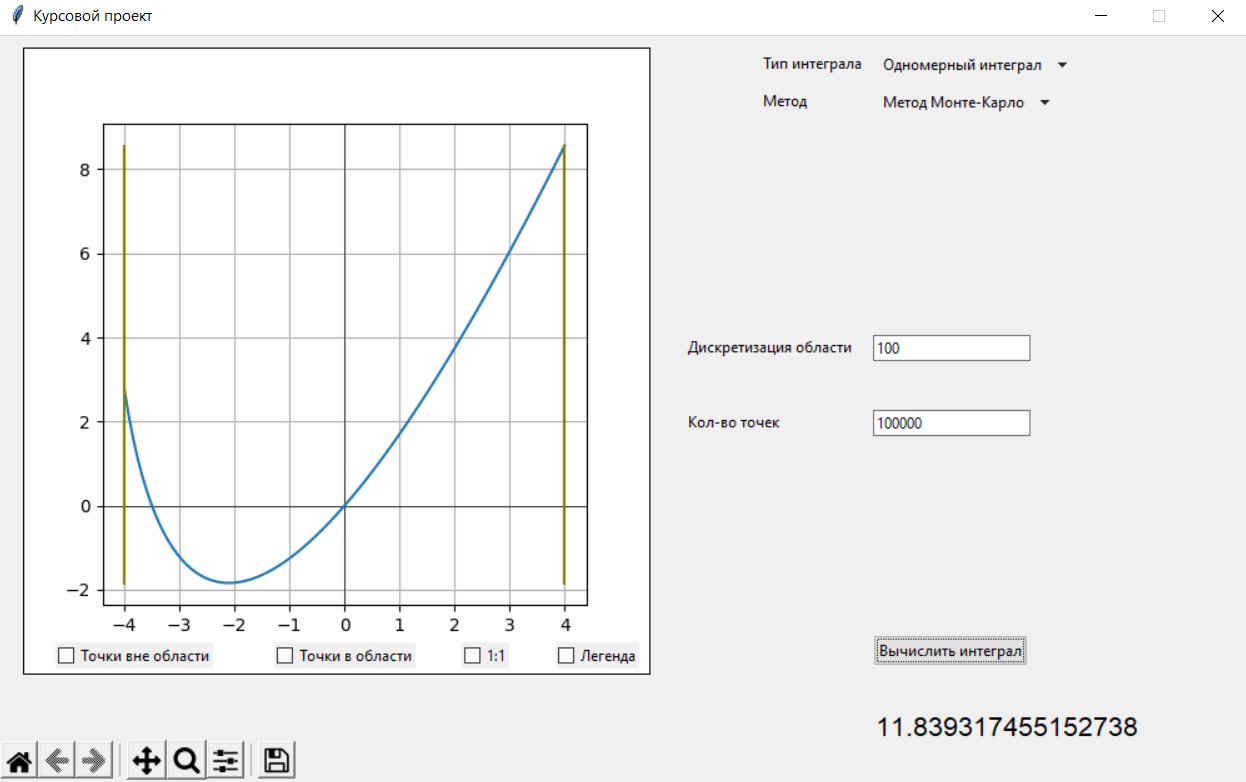


Рисунок 2. Решение одномерного интеграла методом Монте-Карло

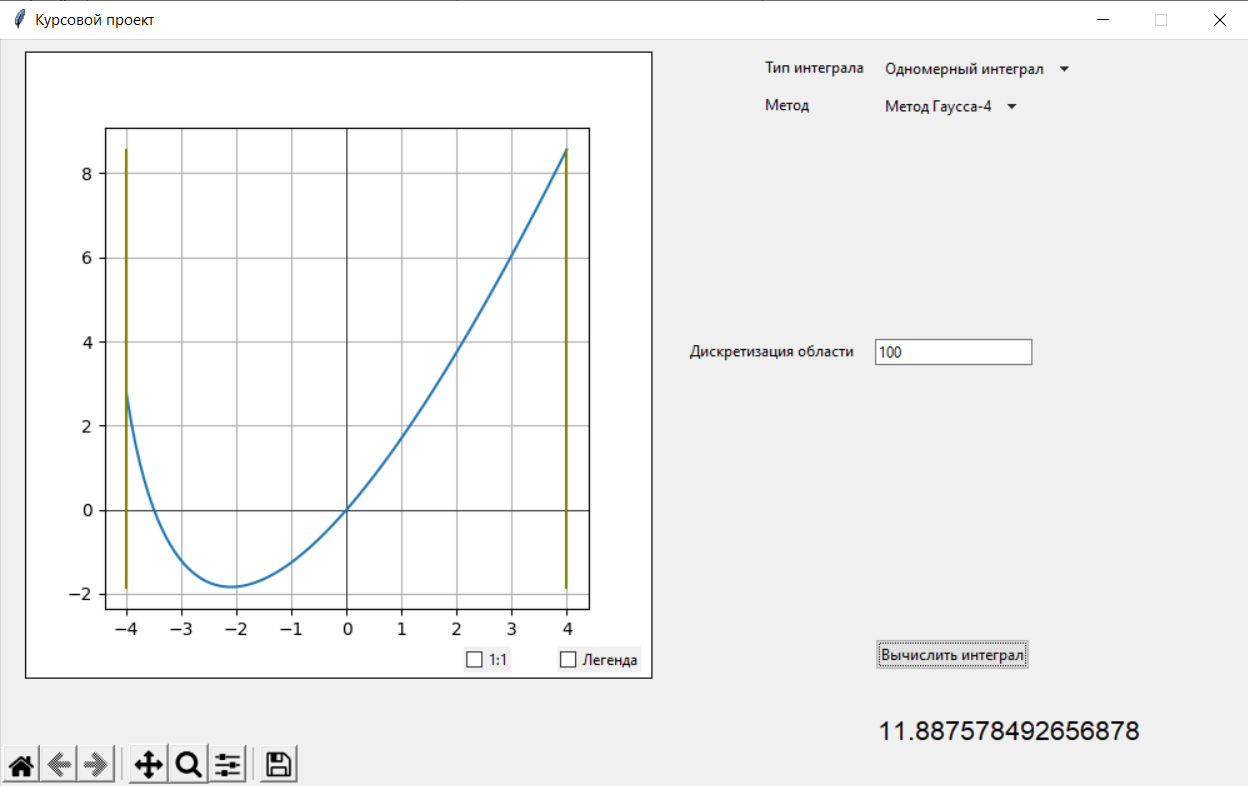


Рисунок 3. Решение одномерного интеграла при помощи квадратурной формулы

*Двумерный интеграл*

Аналитическое решение: -1485

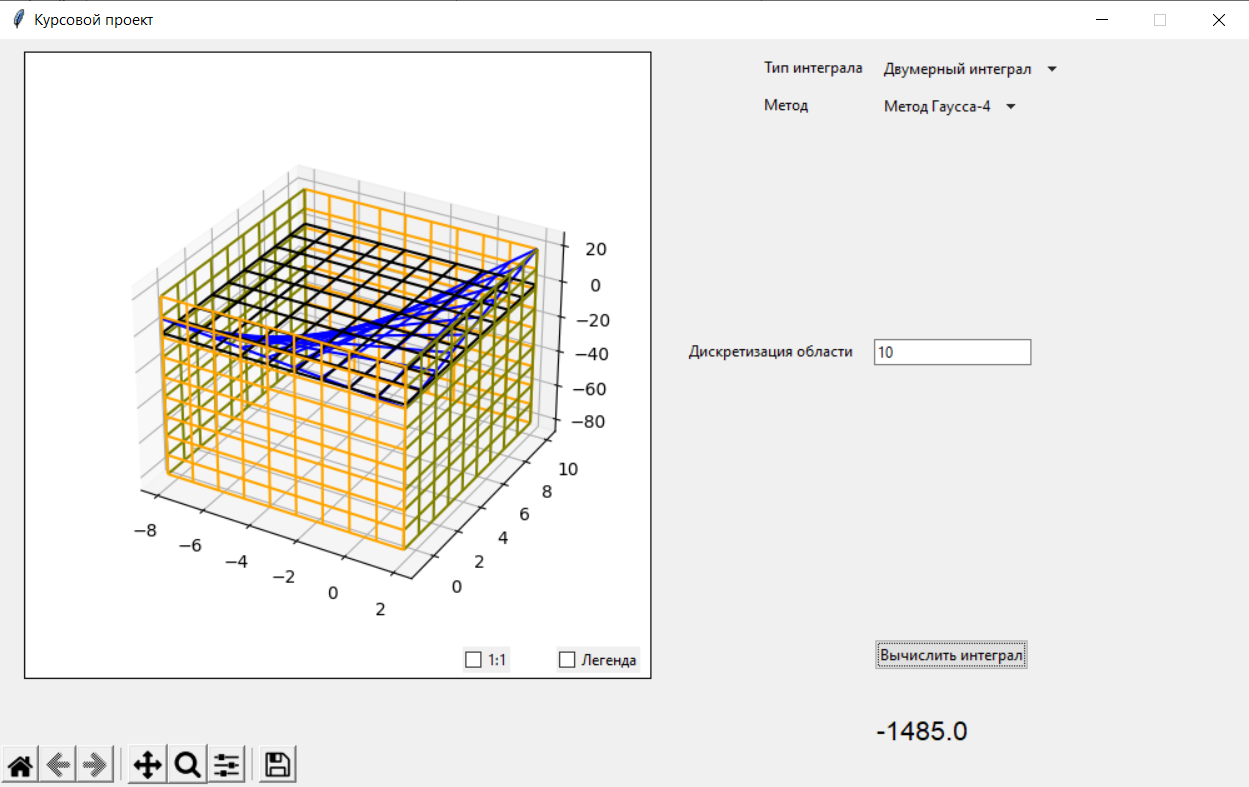


Рисунок 4. Решение двумерного интеграла при помощи квадратурных формул

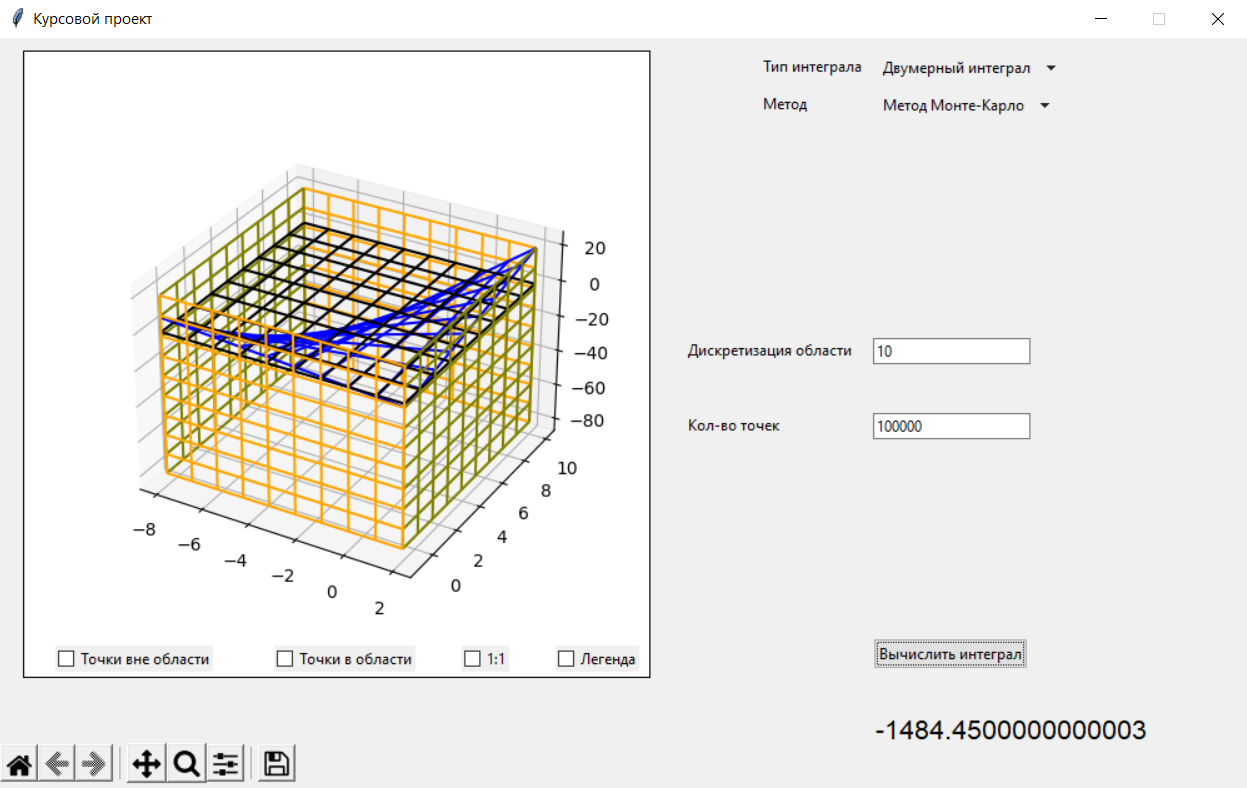


Рисунок 5. Решение двумерного интеграла методом Монте-Карло

*Трехмерный интеграл*

Аналитическое решение: 4.5

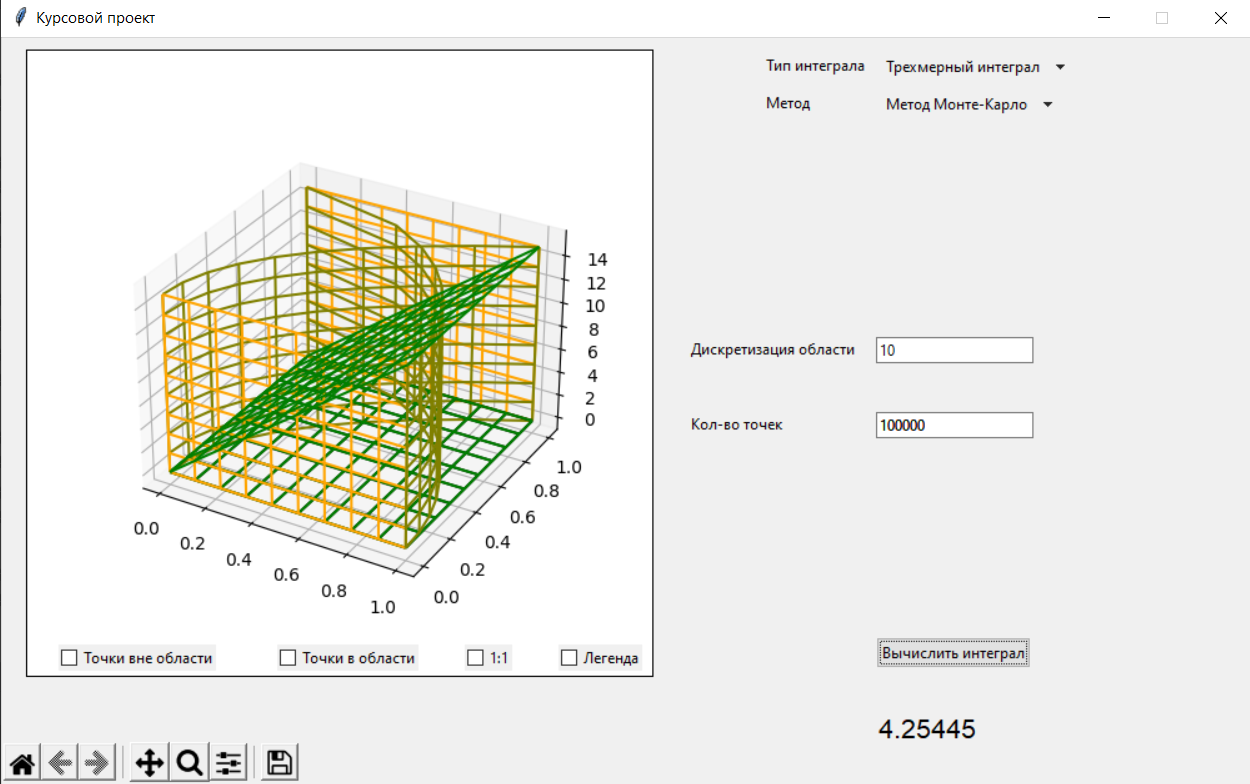


Рисунок 6. Решение трехмерного интеграла методом Монте-Карло

# **Исследование сходимости метода Монте-Карло**

Исследуем сходимость метода Монте-Карло на примере двухмерных интегралов. Рассмотрим следующий интеграл:

Аналитическое решение: 10

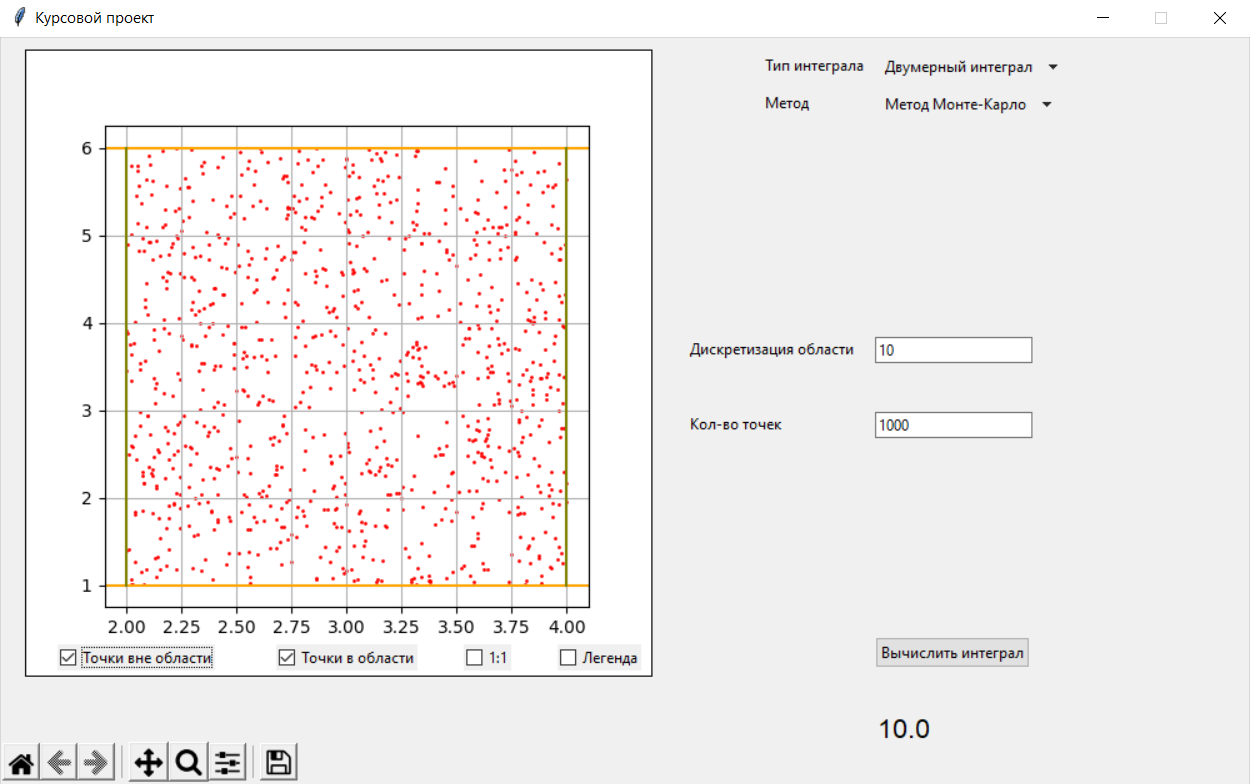


Рисунок 7. Решение интеграла методом Монте-Карло. Точки, попавшие в область интегрирования помечены красным, а не попавшие - серым

Решение данного интеграла методом Монте-Карло всегда будет совпадать с аналитическим решением при любом количестве точек, т.к. область интегрирования совпадает с прямоугольником, ограничивающим область интегрирования.

Рассмотрим следующий интеграл:

Аналитическое решение: 2.82

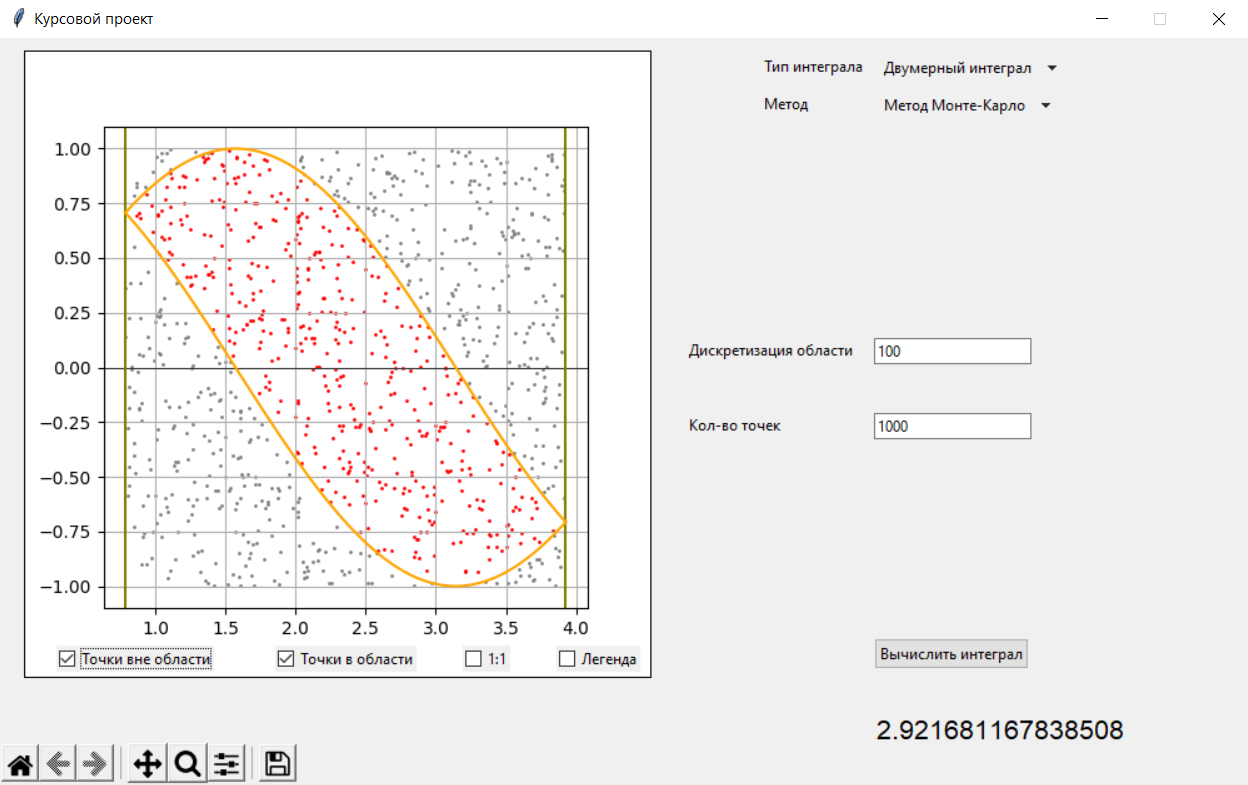


Рисунок 8.

Посчитаем 10 раз этот интеграл методом Монте-Карло для 1000, 10000, 100000 и 1000000 точек. В результате получим следующие значения.

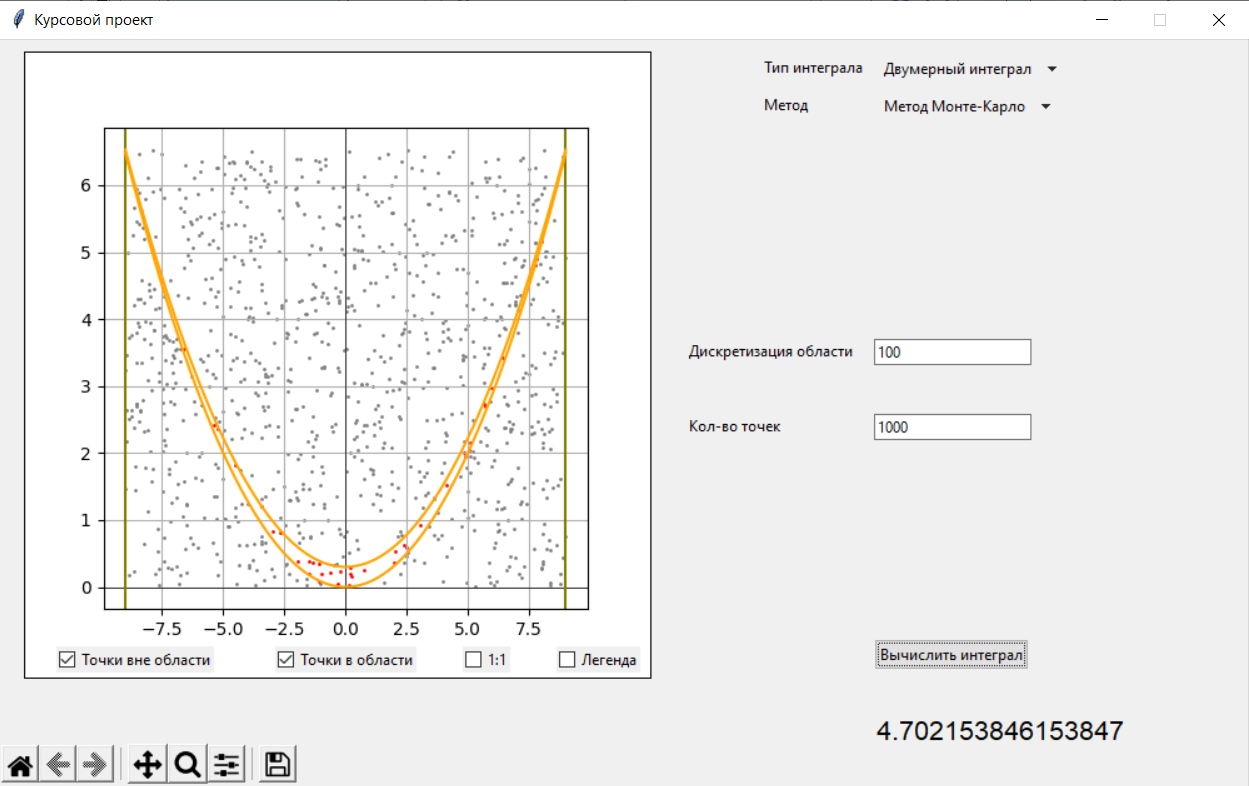
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № решения | 1000 точек | 10000 точек | 100000 точек | 1000000 точек |
| 1 | 2.82743 | 2.82869 | 2.83440 | 2.82664 |
| 2 | 2.75831 | 2.86387 | 2.81731 | 2.82316 |
| 3 | 2.80858 | 2.83560 | 2.82887 | 2.82965 |
| 4 | 2.83371 | 2.75517 | 2.81819 | 2.82813 |
| 5 | 2.62008 | 2.80544 | 2.82699 | 2.82737 |
| 6 | 2.89654 | 2.82869 | 2.83553 | 2.83117 |
| 7 | 2.67035 | 2.80104 | 2.82177 | 2.82585 |
| 8 | 2.93424 | 2.86199 | 2.82850 | 2.82774 |
| 9 | 2.72061 | 2.81685 | 2.82894 | 2.83015 |
| 10 | 2.90283 | 2.84816 | 2.82498 | 2.82216 |

, – вычисленное решение, - аналитическое решение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0.081682 | 0.02485 | 0.006452 | 0.001202 |

Рассмотрим следующий интеграл:

Аналитическое решение: 3.90461



Посчитаем 10 раз этот интеграл методом Монте-Карло для 1000, 10000, 100000 и 1000000 точек. В результате получим следующие значения.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № решения | 1000 | 10000 | 100000 | 1000000 |
| 1 | 4.46704 | 3.73821 | 3.89690 | 3.87798 |
| 2 | 4.23193 | 4.25544 | 3.88515 | 3.89220 |
| 3 | 3.17395 | 3.78523 | 3.99565 | 3.89796 |
| 4 | 3.87927 | 3.51486 | 3.92394 | 3.88550 |
| 5 | 4.81970 | 3.65592 | 3.94628 | 3.93123 |
| 6 | 3.64416 | 4.12613 | 3.86281 | 3.90255 |
| 7 | 5.17236 | 3.99683 | 3.88868 | 3.88574 |
| 8 | 3.29105 | 4.16140 | 3.92512 | 3.91089 |
| 9 | 4.58461 | 3.85576 | 4.00153 | 3.91736 |
| 10 | 5.05481 | 4.07911 | 3.87575 | 3.89643 |

, – вычисленное решение, - аналитическое решение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0.65328 | 0.206893 | 0.038323 | 0.013956 |

Вывод: Вычисленное решение методом Монте-Карло сходится к аналитическому решению. Скорость сходимости зависит от области интегрирования. Чем больше отношение площади интегрирования к площади прямоугольника, ограничивающего область интегрирования, тем быстрее сходимость метода. Аналогично для объема интегрирования и параллелепипеда, ограничивающего область интегрирования.

# **Список источников**

1. Соболь И.М. [Метод Монте-Карло](http://ilib.mccme.ru/plm/ann/a46.htm). — М.: Наука, 1968. — 64 с.
2. Мысовских И. П. Интерполяционные кубатурные формулы. — Москва: Наука, 1981. — С. 336.
3. Список квадратурных формул - Википедия [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/Список\_квадратурных\_формул (дата обращения: 18.12.2023)
4. Numerical methods [Электронный ресурс] URL: <http://hoster.bmstu.ru/~fn1/wp-content/uploads/2011/08/uchebno-metod/Blumin_Fed_Hrap_chisl_met.pdf> (дата обращения 18.12.2023)