Лабораторная работа №3

Вариант №27

Товмасян Арман М3132

Функция и отрезок:

$$f(x) = 2^x \in [-1, 2]$$

Аналитический Метод

1. Построим верхние и нижние суммы Дарбу для равномерного разбиения

Пусть:

$$\tau - \text{равномерное разбиение отрезка на } n \text{ частей}$$

Исходя из того что мы работаем с отрезком $[-1, 2]$ получим:

$$\Delta x_i = \frac{3}{n} \quad x_i = -1 + \frac{3i}{n} \quad i = 0, 1, \ldots, n$$

Посчитаем нижние и верхние суммы Дарбу:

Буду пользоваться формулой суммы геометрической прогрессии:

$$S_n = \frac{b_1 \cdot (q^n - 1)}{q - 1}$$

Нижняя сумма Дарбу

$$s_\tau = \sum_{i=0}^{n-1} \frac{2^{-1+\frac{3i}{n}} \cdot 3}{n} = \frac{3}{2n} \sum_{i=0}^{n-1} \left(8^{\frac{1}{n}}\right)^i = \frac{21}{2n(\sqrt[n]{8} - 1)}$$

Верхняя сумма Дарбу

$$S_\tau = \sum_{i=1}^{n} \frac{2^{-1+\frac{3i}{n}} \cdot 3}{n} = \frac{3}{2n} \sum_{i=1}^{n} \left(8^{\frac{1}{n}}\right)^i = \frac{21 \cdot \sqrt[n]{8}}{2n(\sqrt[n]{8}-1)}$$

---

2. Исследуем интегрируемость функции с помощью критерия Римана

Критерий Римана интегрируемости функции

$$\forall \varepsilon, \exists \tau : S_\tau - s_\tau < \varepsilon$$

Рассмотрим $S_\tau - s_\tau$:

$$\frac{21 \cdot \sqrt[n]{8}}{2n(\sqrt[n]{8}-1)} - \frac{21}{2n(\sqrt[n]{8}-1)} = \frac{21 \cdot \sqrt[n]{8} - 21}{2n(\sqrt[n]{8}-1)} = \frac{21\left(\sqrt[n]{8}-1\right)}{2n(\sqrt[n]{8}-1)} = \frac{21}{2n}$$

$$S_\tau - s_\tau = \frac{21}{2n} < \varepsilon$$

Найдем такое n_0 что $S_\tau - s_\tau < \varepsilon$:

$$\forall \varepsilon, \exists n_0 : \forall n \geq n_0 \Rightarrow \frac{21}{2n} < \varepsilon \Rightarrow$$

$$\Rightarrow n > \frac{21}{2\varepsilon} \Rightarrow n_0 = \left[\frac{21}{2\varepsilon}\right] + 1$$

Следовательно - функция интегрируема

Помимо критерия Римана, интегрируемость можно еще доказать исходя из понятия классов интегрируемых функций

Очевидно что функция $f(x) = 2^x$ монотонно возрастает на отрезке $[-1, 2]$

Это означает что заданная функция принадлежит классу монотонных функций, которые $\in R[a, b]$

3. Найдем пределы сумм Дарбу

Предел нижней суммы:

$$\lim_{n\to\infty} s_\tau = \lim_{n\to\infty} \frac{21}{2n(\sqrt[n]{8}-1)} = \frac{21}{2} \lim_{n\to\infty} \frac{1}{n \cdot \sqrt[n]{8} - n} =$$

Поделим числитель и знаменатель на n и воспользуемся правилом Лопиталя и получим:

$$\frac{21}{2}\lim_{n\to\infty}\frac{-\frac{1}{n^2}}{-\frac{\ln 8\cdot\sqrt[n]{8}}{n^2}}=\frac{21}{2\cdot\ln 8}\lim_{n\to\infty}\frac{1}{8^{\frac{1}{n}}}=\frac{7}{\ln 4}$$

Предел верхней суммы:

$$\lim_{n\to\infty}S_\tau=\lim_{n\to\infty}\frac{21\cdot\sqrt[n]{8}}{2n(\sqrt[n]{8}-1)}=\frac{21}{2}\lim_{n\to\infty}\frac{\frac{\sqrt[n]{8}}{n}}{\sqrt[n]{8}-1}=$$

Воспользуемся правилом Лопиталя и получим:

$$\frac{21}{2}\lim_{n\to\infty}\frac{\frac{-\ln 8\cdot 8^{\frac{1}{n}}+n\cdot 8^{\frac{1}{n}}}{n^3}}{-\frac{\ln 8\cdot 8^{\frac{1}{n}}}{n^2}}=\frac{21}{2}\lim_{n\to\infty}\frac{\ln 8+n}{n\ln 8}=\frac{21}{2\cdot\ln 8}\lim_{n\to\infty}\frac{\frac{\ln 8}{n}+1}{1}=\frac{7}{\ln 4}$$

В пункте 2, мною была доказана интегрируемость нашей функции. Исходя из этого можно утверждать что:

Так как функция монотонно возрастает на отрезке $[-1,2]$, то:

$$I_*=\sup s_\tau=\frac{7}{\ln 4}\quad\left(s_\tau<\frac{7}{\ln 4}\quad\&\&\quad\lim_{n\to\infty}s_\tau=\frac{7}{\ln 4}\right)$$

$$I^*=\inf S_\tau=\frac{7}{\ln 4}\quad\left(S_\tau>\frac{7}{\ln 4}\quad\&\&\quad\lim_{n\to\infty}S_\tau=\frac{7}{\ln 4}\right)$$

$$\Rightarrow I_*=I^*=I=\frac{7}{\ln 4}$$

4. Проверим результат с помощью формулы Ньютона-Лейбница

$$\int_{-1}^{2}2^x\,dx=\left.\frac{2^x}{\ln 2}\right|_{-1}^{2}=\frac{2^2}{\ln 2}-\frac{2^{-1}}{\ln 2}=\frac{7}{\ln 4}$$

Посчитав значение интеграла с помощью формулы Ньютона-Лейбница, удостоверились в правильности вычислений в пункте 3

Численный Метод

1. Напишем программу на языке Python 3 для вычисления интегральных сумм с заданными параметрами

In [1]:
```
import numpy as np  # Библиотека для работы с массивами чисел
import matplotlib.pyplot as plt # Библиотека для построения графиков
import pandas as pd # Библиотека для работы с датафреймами (таблицы)
import random
%matplotlib inline
```

In [2]:
```
plt.style.use(["science", "notebook", "grid", "high-vis"])  # Стиль координатной плоскос
```

In [3]:
```
def f(x):  # Желаемую функцию вводить сюда!
    return x**3 + x**2 + 9*x + 3
```

In [4]:
```
def compute_integral_sums(function, a, b, N, points_attachment): # Подсчет значения инте
    difference = (b - a) / N # Разбиение отрезка на N частей
    x = np.linspace(a, b, N+1) # Вектор который хранит в себе точки разбиения

    match points_attachment: # Смотрим какое оснащение выбрал пользователь
        case "left":
            left = x[:-1] # Берем левые точки (исключаем самую правую)
            return np.sum(function(left) * difference)
        case "right":
            right = x[1:] # Берем правые точки (исключаем самую левую)
            return np.sum(function(right) * difference)
        case "mid":
            mid = (x[:-1] + x[1:]) / 2 # Берем концы подотрезков, складываем, делим на 2
            return np.sum(function(mid) * difference)
        case "random":
            rand = np.random.uniform(x[:-1], x[1:]) # Функция uniform() генерирует случа
            # тем самым получая рандомную точку на каждом из подотрезков
            return np.sum(function(rand) * difference)
        case _: # Case default
            print("Please, enter 'points_attachment' option and determine the sum kind")
            print("Choose one of the options: [left, right, mid, random]")
```

In [5]:
```
compute_integral_sums(f, -3, 5, 100, "mid") # Проверить можно здесь)
```

Out[5]:
```
282.6496
```

2. Визуализируем результат работы программы

In [6]:
```
def riemann_sums_visualization(function, a, b, N):  # Визуализация интегральных сумм (с 
    # Подготовка нужных векторов (область определения, значений, разбиения)
    n = N
    x = np.linspace(a, b, N+1)
    y = function(x)

    X = np.linspace(a, b, n*N+1)
    Y = function(X)

    # Разобьем на 4 подграфика
    fig, axes = plt.subplots(4, 1, figsize=(15, 20))

    # График с оснащением в точках слева
    ax_left = axes[0]
    ax_left.axvline(x=0, lw=1, color="black")  # Построим x=0
    ax_left.axhline(y=0, lw=1, color="black")  # Построим y=0
```

```python
    ax_left.plot(X, Y, 'b') # График функции
    x_left = x[:-1] # Выбираем соответствующие точки (зависит от вида оснащения отрезков
    y_left = y[:-1]
    ax_left.plot(x_left, y_left, "b.", markersize=10) # Точки на самом графике функции
    ax_left.bar(x_left, y_left, width=(b - a)/N, alpha=0.2, align="edge", edgecolor="b")
    ax_left.set_title(f"Левая интегральная сумма, N = {N}", fontsize=25) # Название к гр
    # Аналогично для остальных графиков

    # График с оснащением в точках посередине
    ax_mid = axes[1]
    ax_mid.axvline(x=0, lw=1, color="black")
    ax_mid.axhline(y=0, lw=1, color="black")
    ax_mid.plot(X,Y,'b')
    x_mid = (x[:-1] + x[1:])/2
    y_mid = function(x_mid)
    ax_mid.plot(x_mid, y_mid,'b.',markersize=10)
    ax_mid.bar(x_mid, y_mid, width=(b-a)/N, alpha=0.2, edgecolor='b')
    ax_mid.set_title(f'Интегральная сумма с точками посередине, N = {N}', fontsize=25)

    # График с оснащением в точках справа
    ax_right = axes[2]
    ax_right.axvline(x=0, lw=1, color="black")
    ax_right.axhline(y=0, lw=1, color="black")
    ax_right.plot(X,Y,'b')
    x_right = x[1:]
    y_right = y[1:]
    ax_right.plot(x_right, y_right,'b.',markersize=10)
    ax_right.bar(x_right, y_right, width=-(b-a)/N, alpha=0.2, align='edge', edgecolor='b
    ax_right.set_title(f'Правая интегральная сумма, N = {N}', fontsize=25)

    # График с оснащением в точках рандомно расположенных
    ax_rand = axes[3]
    ax_rand.axvline(x=0, lw=1, color="black")
    ax_rand.axhline(y=0, lw=1, color="black")
    ax_rand.plot(X,Y,'b')
    x_rand = np.random.uniform(x[:-1], x[1:])
    y_rand = function(x_rand)
    ax_rand.plot(x_rand, y_rand,'b.',markersize=10)
    ax_rand.bar(x_rand, y_rand, width=(b-a)/N, alpha=0.2, align='edge', edgecolor='b')
    ax_rand.set_title(f'Интегральная сумма с рандомными точками, N = {N}', fontsize=25)

    fig.tight_layout(h_pad=2.0)
    plt.show()
```

In [7]:
```python
riemann_sums_visualization(f, -3, 5, 10)
```

3. Соберем в таблицу результаты обработки функции и отрезка из аналитической части и сравним их

In [8]:
```
def compute_error(integral_sum): # Подсчет ошибки относительно получившегося результата
    return np.abs(integral_sum - (7 / np.log(4)))
```

In [9]:
```
def get_data(): # Собираем значения интегральных сумм для разных разбиений и оснащений
    n_s = np.arange(1000, 105000, 5000)
    points_attachment = ["left", "right", "mid", "random"]
    for i in range(len(n_s)):
        rand_points_attachment = random.choice(points_attachment) # Выбираем рандомное о
        integral_sum = compute_integral_sums(lambda x: 2**x, -1, 2, n_s[i], rand_points_
        yield ["2^x", n_s[i], rand_points_attachment, integral_sum, compute_error(integr
```

In [10]:
```
information_list = []
for row in get_data():
    information_list.append(row)

# Собираем полученные данные в датафрейм
df = pd.DataFrame(information_list, columns=["Function", "N", "Points Attachment", "Valu
df
```

Out[10]:

	Function	N	Points Attachment	Value Of Riemann Integral Sum	Calculation Error
0	2^x	1000	left	5.044184	5.248180e-03
1	2^x	6000	random	5.049414	1.817605e-05
2	2^x	11000	mid	5.049433	7.518642e-09
3	2^x	16000	left	5.049105	3.281179e-04
4	2^x	21000	right	5.049683	2.500041e-04
5	2^x	26000	right	5.049635	2.019258e-04
6	2^x	31000	left	5.049263	1.693529e-04
7	2^x	36000	right	5.049578	1.458347e-04
8	2^x	41000	left	5.049305	1.280477e-04
9	2^x	46000	mid	5.049433	4.299432e-10
10	2^x	51000	random	5.049433	1.283114e-08
11	2^x	56000	random	5.049432	2.478716e-07
12	2^x	61000	mid	5.049433	2.444933e-10
13	2^x	66000	random	5.049433	9.622998e-08
14	2^x	71000	mid	5.049433	1.804716e-10
15	2^x	76000	left	5.049364	6.907863e-05
16	2^x	81000	left	5.049368	6.481454e-05
17	2^x	86000	random	5.049432	2.195806e-07
18	2^x	91000	left	5.049375	5.769209e-05
19	2^x	96000	random	5.049433	2.751031e-08
20	2^x	101000	left	5.049381	5.198002e-05

In [11]:
```
df.describe()["Calculation Error"] # Немного информации по ошибке подсчета
```

Out[11]:
```
count    2.100000e+01
mean     3.206580e-04
std      1.133066e-03
min      1.804716e-10
25%      2.751031e-08
50%      5.198002e-05
75%      1.458347e-04
```

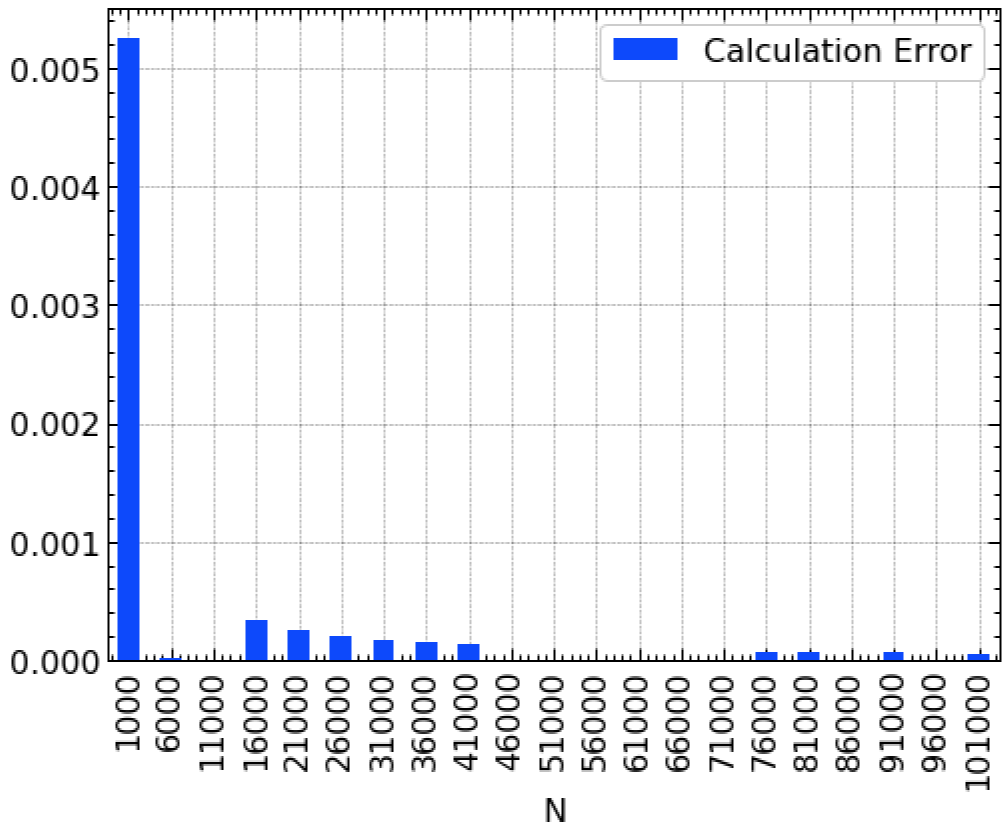
```
max      5.248180e-03
Name: Calculation Error, dtype: float64
```

Приведу график который отображает корреляцию ошибки подсчета относительно размера разбиения

Так как информация подсчитывается по разному, то после каждой генерации датафрейма, график будет меняться

Зачастую видно что по возрастанию N, ошибка уменьшается

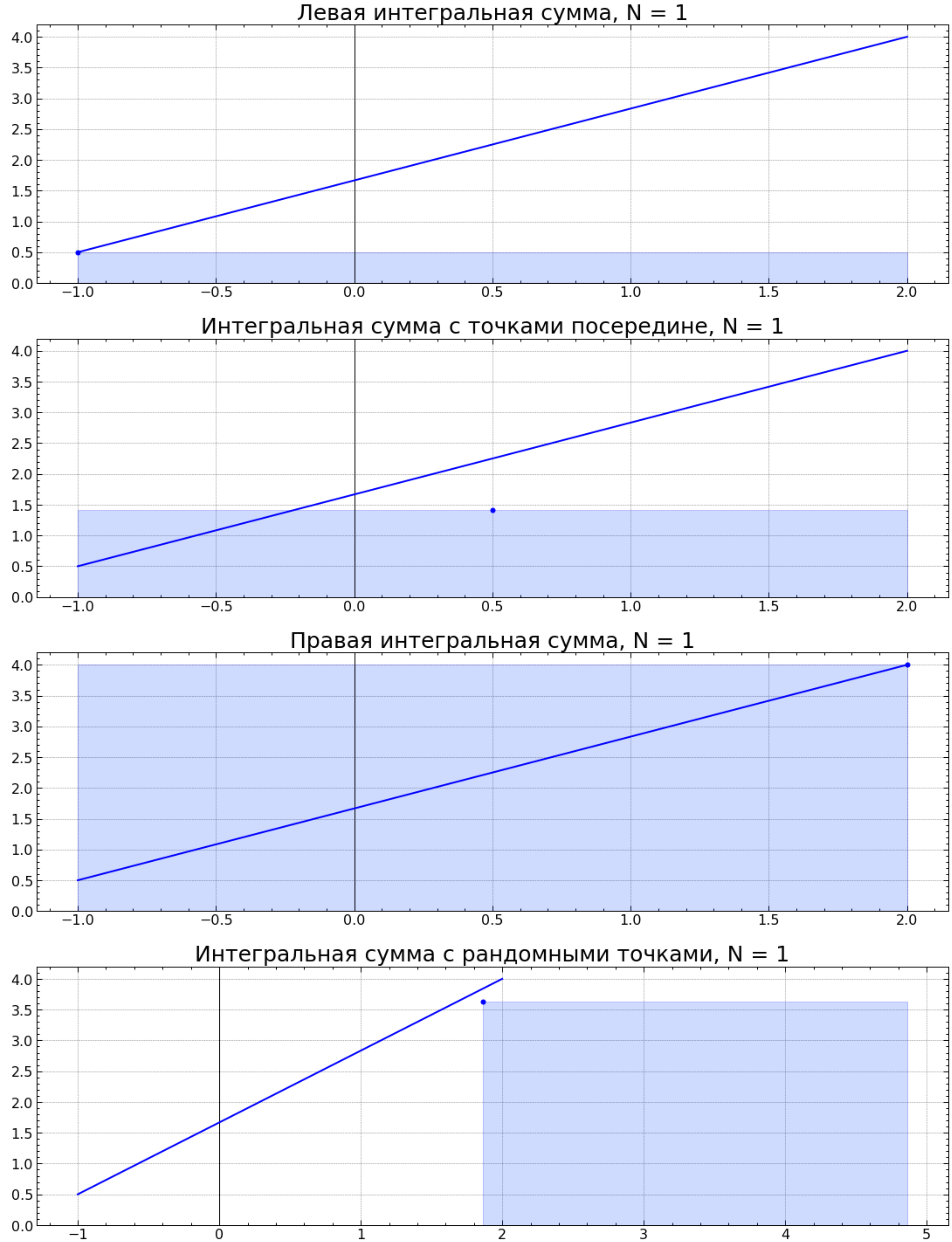
In [12]:
```
calc_error_corr = df.plot.bar(x="N", y="Calculation Error")
```



Визуализируем некоторые графики

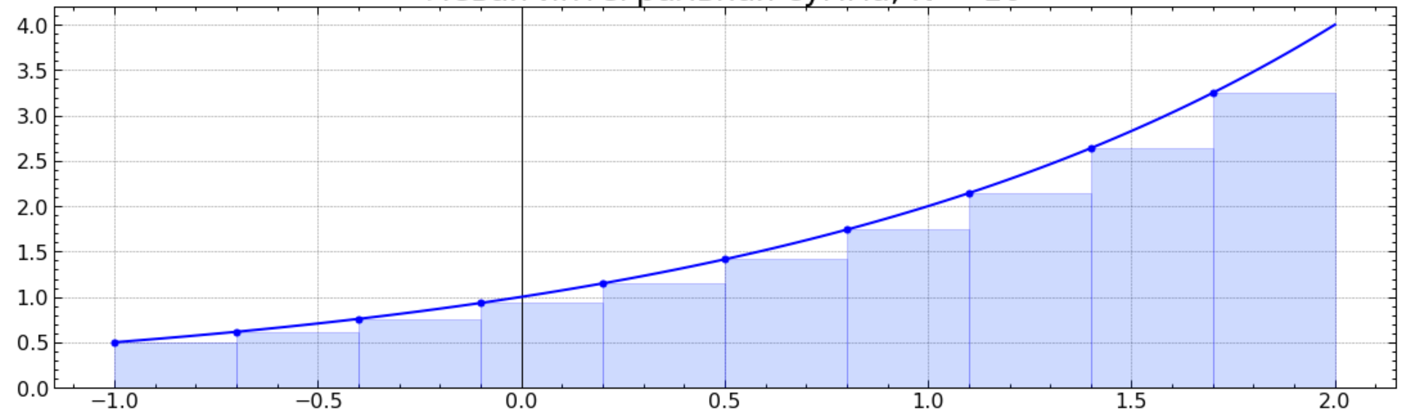
In [13]:
```
def func_to_vis(x):
    return 2**x
```

In [14]:
```
riemann_sums_visualization(func_to_vis, -1, 2, 1)
```

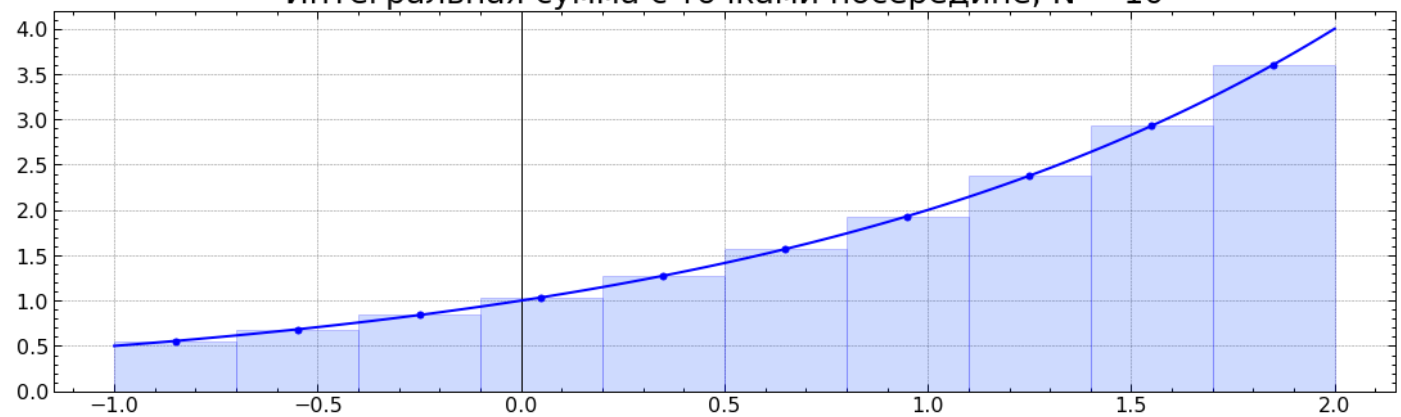


In [15]:
```
riemann_sums_visualization(func_to_vis, -1, 2, 10)
```

Левая интегральная сумма, N = 10

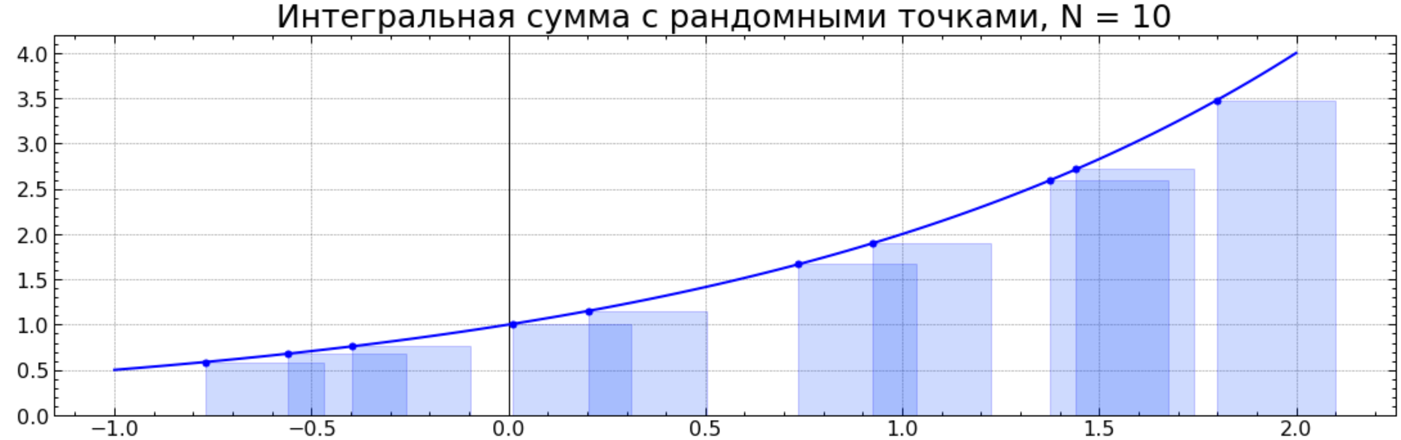


Интегральная сумма с точками посередине, N = 10

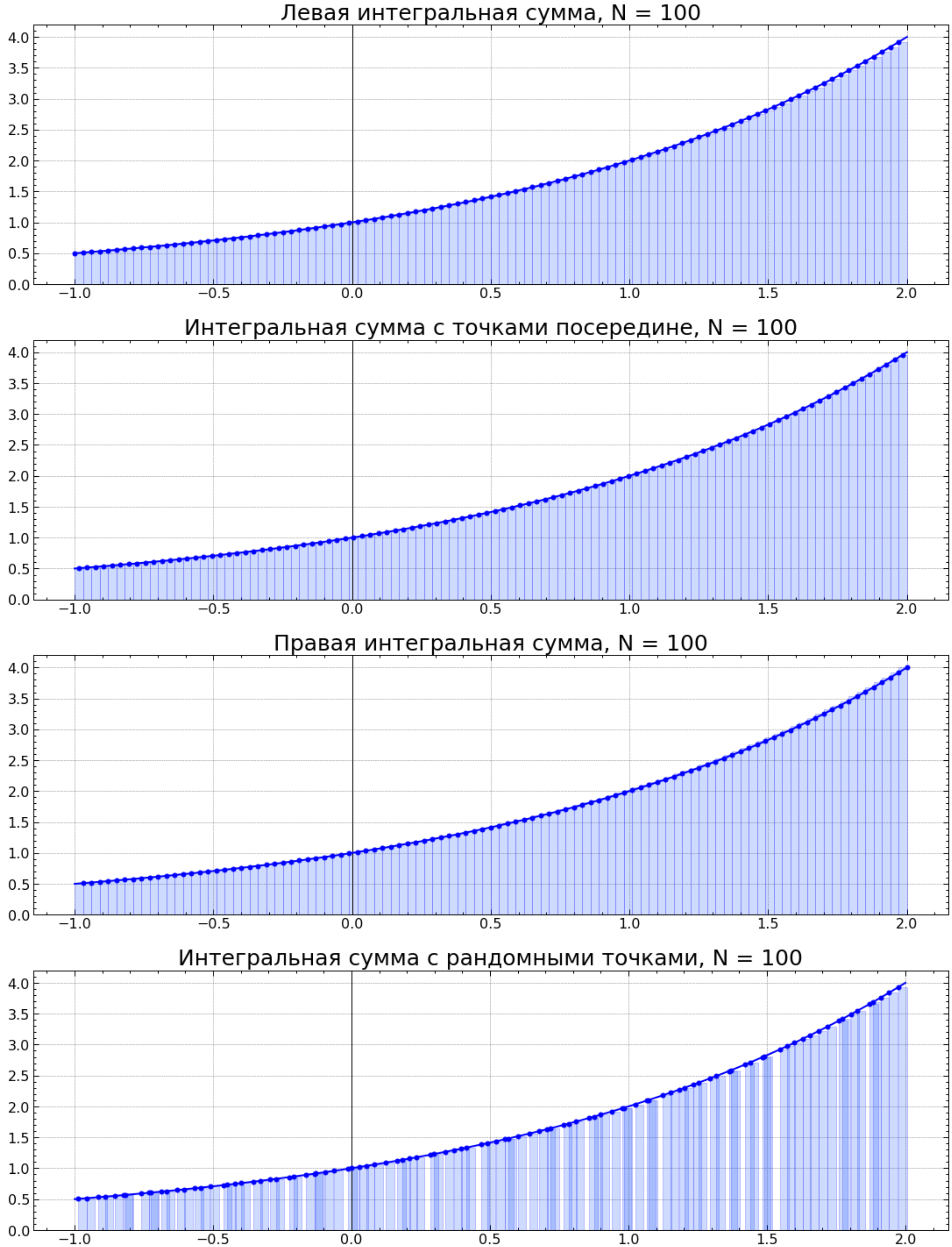


Правая интегральная сумма, N = 10

Интегральная сумма с рандомными точками, N = 10

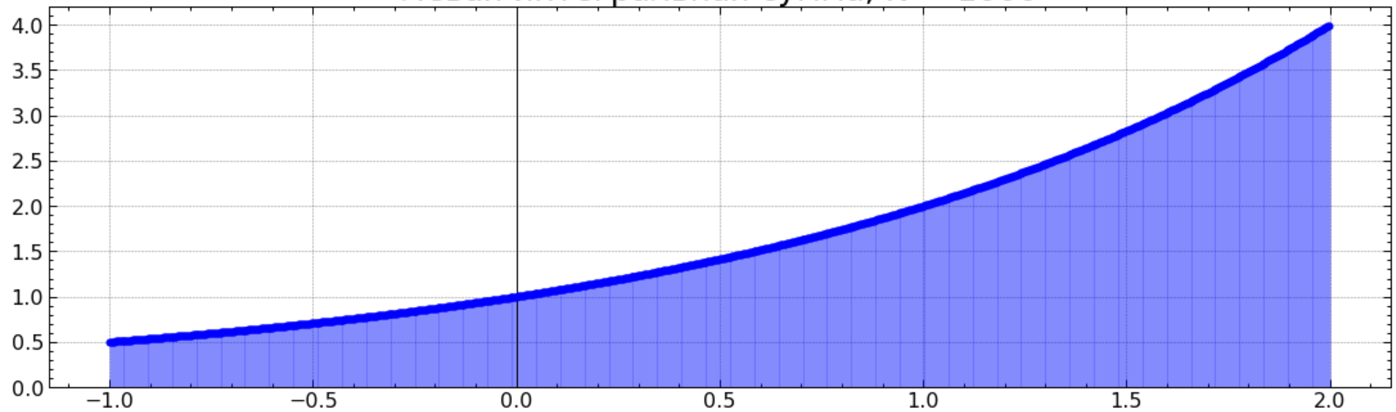


In [16]:
```
riemann_sums_visualization(func_to_vis, -1, 2, 100)
```

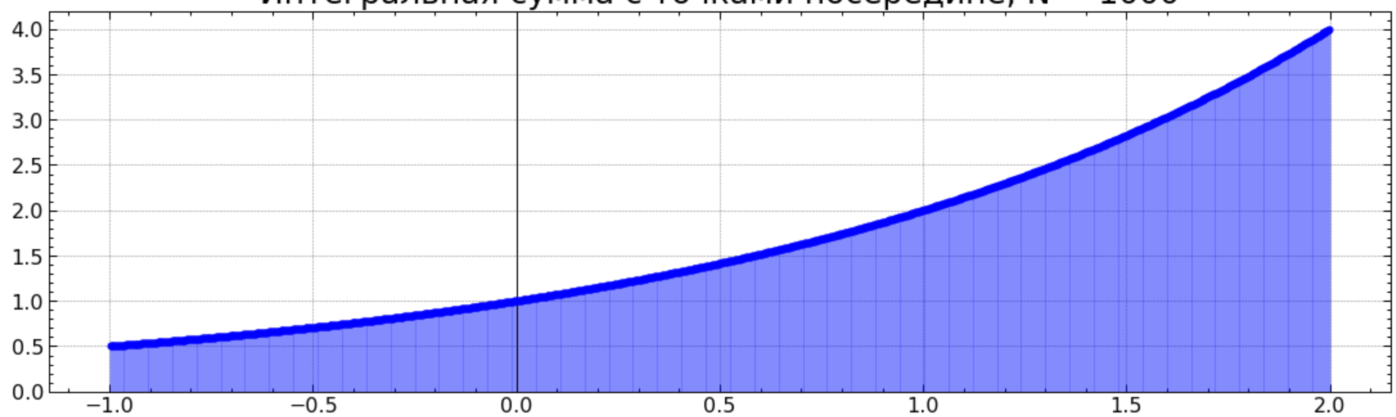


In [17]:
```
riemann_sums_visualization(func_to_vis, -1, 2, 1000)
```

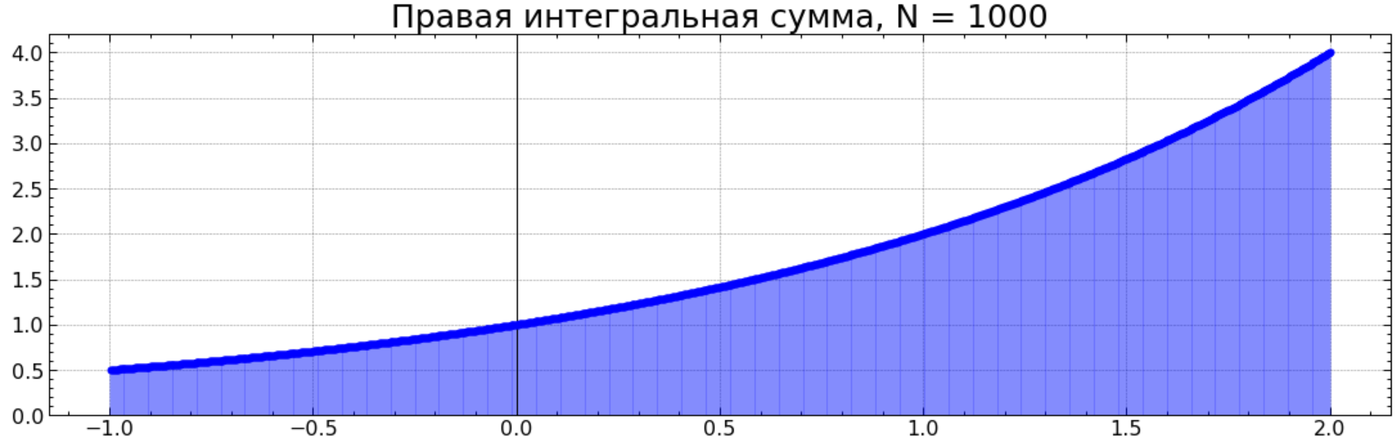
Левая интегральная сумма, N = 1000



Интегральная сумма с точками посередине, N = 1000



Правая интегральная сумма, N = 1000



Интегральная сумма с рандомными точками, N = 1000