1 Kod Źródłowy

Implementacja frameworku symulacji:

```
import matplotlib.pyplot as plt
import os

class Simulation:
    """ Class generating and managing a simulation """

    def __init__(self, n_values, repeats, fun):
        """
        Constructor
        :param n_values: numbers of points for which simulation is carried out
        :type n_values: list of int
        :param repeats: simulation repeats number
        :type repeats: int
        :param fun: function calculating simulation value for given number of points
        :type fun: (int) -> float
        """
        self.reps = repeats
        self.x_values = n_values.copy()
        self.y_series = [[fun(n) for _ in range(repeats)] for n in n_values]

    def add_series_to_chart(self, label='', color='b', marker='s', marker_size=10) -> None:
        """
        Adding calculated series to chart
        :param label: series label
        :type label: str
        :param color: marker color
        :type color: str
        :param marker: marker shape type
        :type marker: str
        :param marker_size: marker size
        :type marker_size: int
        :return: None
        """
        for rep in range(self.reps):
            plt.scatter(self.x_values, [s[rep] for s in self.y_series],
                        label=label, color=color, marker=marker, s=marker_size)

    def add_averages_to_chart(self, label='', color='r', marker='s', marker_size=10) -> None:
        """
        Adding series representing calculated averages to chart
        :param label: series label
        :type label: str
        :param color: marker color
        :type color: str
        :param marker: marker shape type
        :type marker: str
        :param marker_size: marker size
        :type marker_size: int
        :return: None
        """
        plt.scatter(self.x_values, [sum(s) / len(s) for s in self.y_series],
                    label=label, color=color, marker=marker, s=marker_size)
```

```
    def add_line(self, fun, label='', color='y', marker='s', marker_size=10) -> None:
        """
        Adding additional series representing given function to chart
        :param fun: function representing series
        :type fun: (int) -> float
        :param label: series label
        :type label: str
        :param color: marker color
        :type color: str
        :param marker: marker shape type
        :type marker: str
        :param marker_size: marker size
        :type marker_size: int
        :return: None
        """
        plt.scatter(self.x_values, [fun(x) for x in self.x_values],
                    color=color, marker=marker, s=marker_size, label=label)

    def add_names(self, x_axis_name, y_axis_name, title_name) -> None:
        """
        Adding chart description
        :param x_axis_name: X axis description
        :type x_axis_name: str
        :param y_axis_name: Y axis description
        :type y_axis_name: str
        :param title_name: chart title
        :type title_name: str
        :return: None
        """
        plt.xlabel(x_axis_name)
        plt.ylabel(y_axis_name)
        plt.title(title_name)

    def add_legend(self, location='upper right') -> None:
        """
        Showing chart legend
        :param location: legend location on chart
        :type location: str
        :return: None
        """
        plt.legend(loc=location)

    def generate_chart(self, result_file_name) -> None:
        """
        Saving generated chart to plots directory as PNG and showing it
        :param result_file_name: name of saved file
        :type result_file_name: str
        :return: None
        """
        plt.savefig(os.path.dirname(os.path.abspath(__file__)) + '/plots/' +
                    result_file_name + '.png')
        plt.show()
```

Implementacja całkowania:

```
from random import uniform as rand
from simulation_framework import Simulation


def approximate_function_integral(fun, fun_name, var_name, lower_bound, upper_bound, supremo,
                                  calc_value) -> None:
    """
    Simulation of approximating integral of given function

    :param fun: function to approximate
    :type fun: (float) -> float
    :param fun_name: string of function name in latex format
    :type fun_name: str
    :param var_name: variable name
    :type var_name: str
    :param lower_bound: lower bound of integration
    :type lower_bound: float
    :param upper_bound: upper bound of integration
    :type upper_bound: float
    :param supremo: supremo of the function on [lower_bound, upper_bound]
    :type supremo: float
    :param calc_value: exact value of function integral
    :type calc_value: float

    :return: None
    """

    sim = Simulation(list(range(50, 5000, 50)), 50,
                     lambda n:
                     sum([rand(0, supremo) <= fun(rand(lower_bound, upper_bound))
                         for _ in range(n)]) / n * (upper_bound - lower_bound) * supremo)

    sim.add_series_to_chart()
    sim.add_averages_to_chart(label='average value', marker_size=20)
    sim.add_line(lambda x: calc_value, label='exact value', marker_size=3)
    sim.add_names('Points number used in approximating',
                  'Approximated integral value',
                  'Approximating $\int_{}^{} {} \,d{}$'.format(lower_bound, upper_bound,
                                                               fun_name, var_name))
    sim.add_legend()
    sim.generate_chart('integrate ' + fun_name)
```

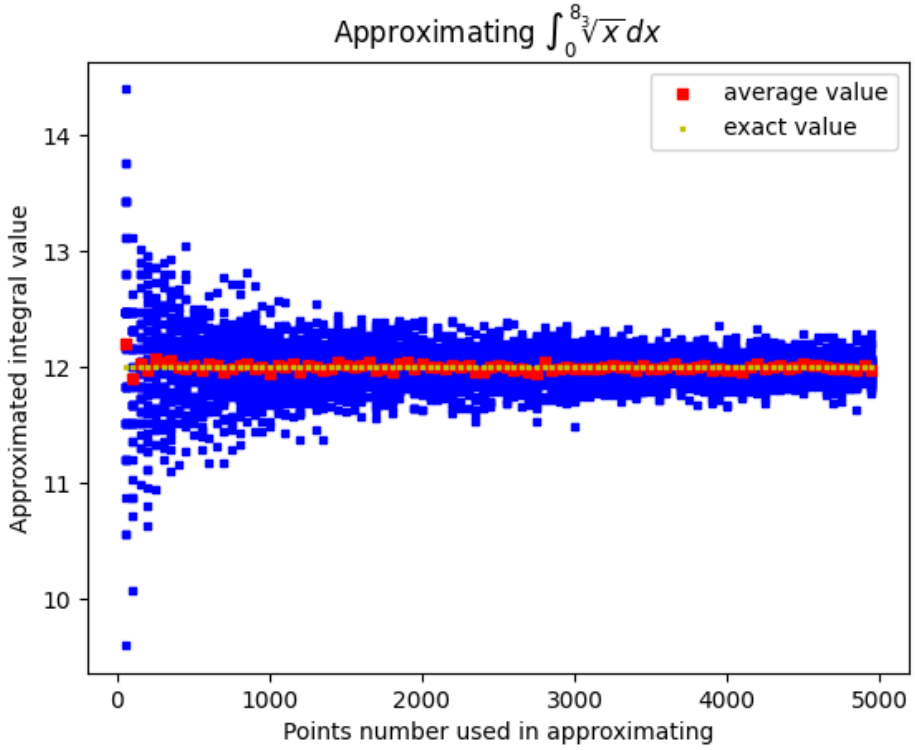
Implementacja wyznaczania liczby π:

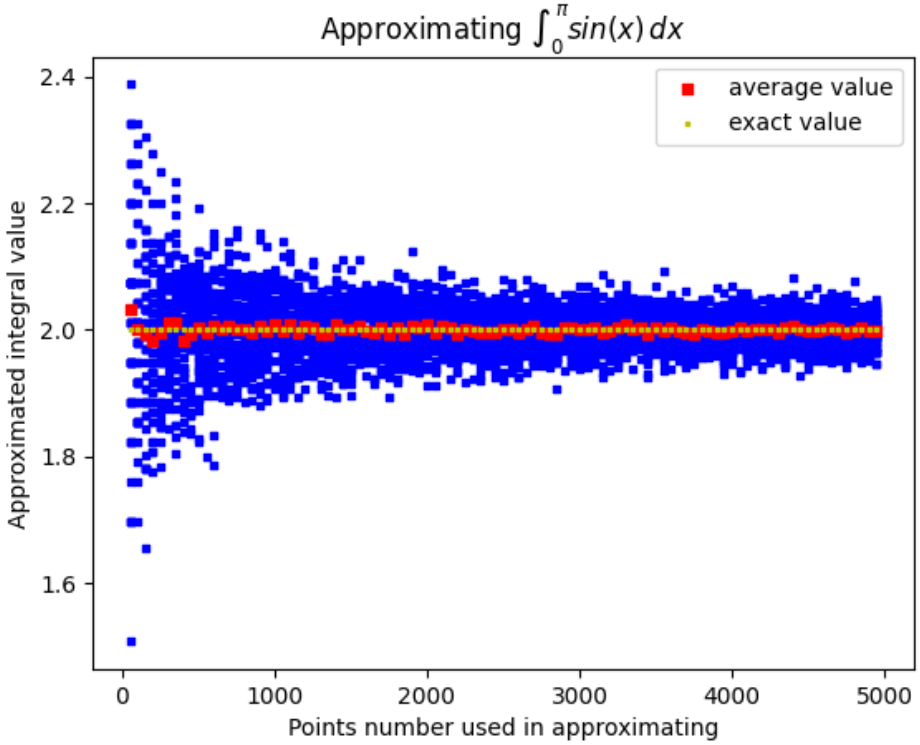
```
from random import uniform as rand
import math
from simulation_framework import Simulation


def approximate_pi() -> None:
    """
    Simulation of approximating PI

    :return: None
    """

    sim = Simulation(list(range(100, 10000, 100)), 50,
                     lambda n:
                     sum([math.sqrt(pow(rand(-1, 1), 2) + pow(rand(-1, 1), 2)) <= 1
                          for _ in range(n)]) / n * 4)

    sim.add_series_to_chart()
    sim.add_averages_to_chart(label='average value', marker_size=20)
    sim.add_line(lambda x: math.pi, label='exact value', marker_size=3)
    sim.add_names('Points number used in approximating',
                  'Approximated $\pi$ value',
                  'Approximating $\pi$')
    sim.add_legend()
    sim.generate_chart('pi generating')
```

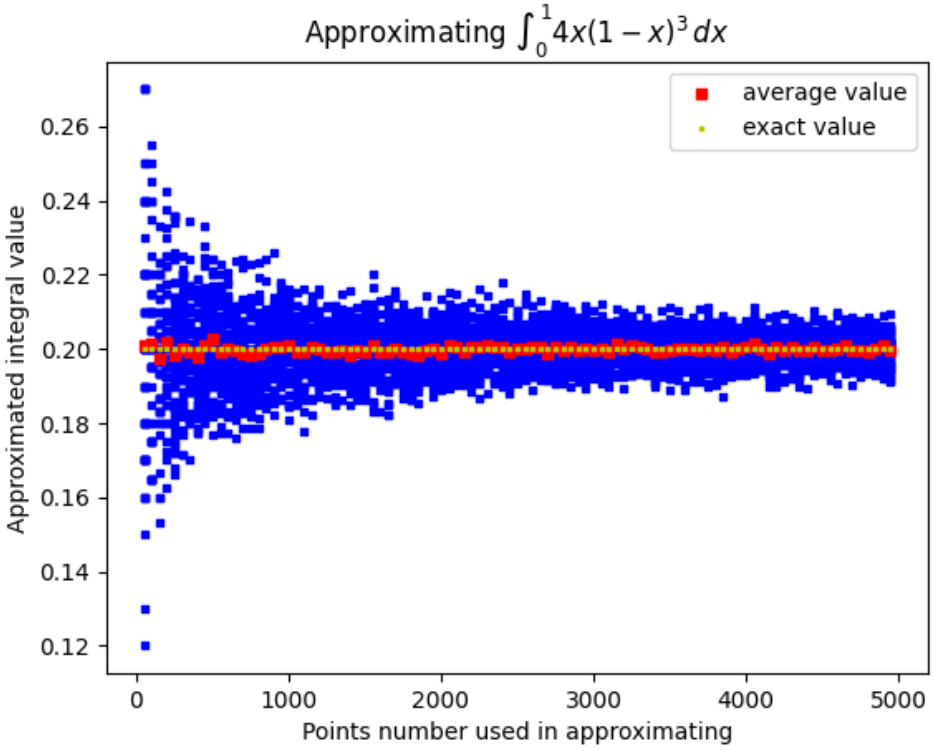
Generowanie wynikow:

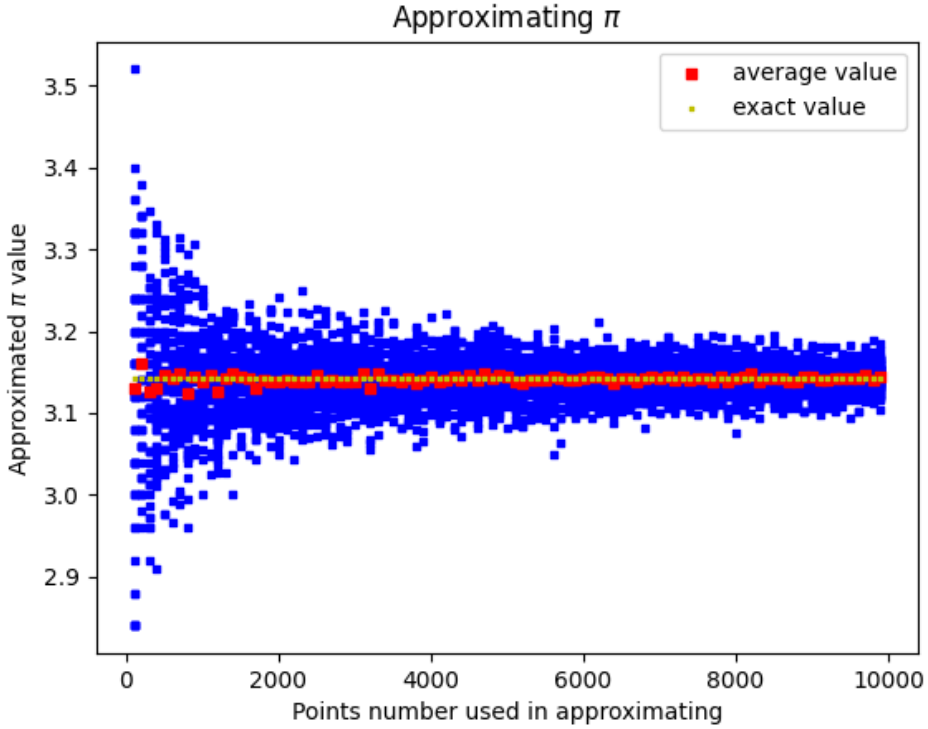
```
from integrals_approximation import approximate_function_integral
from pi_approximation import approximate_pi
from math import pow, sin, pi

approximate_function_integral(lambda x: pow(x, 1/3), '\sqrt[3]{x}', 'x',
                              0, 8, 2, 12)

approximate_function_integral(lambda x: sin(x), 'sin(x)', 'x',
                              0, pi, 1, 2)

approximate_function_integral(lambda x: 4 * x * pow(1 - x, 3), '4x(1-x)^3', 'x',
                              0, 1, 0.5, 0.2)

approximate_pi()
```

2 Wyniki


Approximating $\int_0^8 \sqrt[3]{x}\,dx$
average value
exact value
Approximated integral value
Points number used in approximating
10
11
12
13
14
0
1000
2000
3000
4000
5000


Approximating $\int_0^{\pi} sin(x)\,dx$
average value
exact value
Approximated integral value
Points number used in approximating
1.6
1.8
2.0
2.2
2.4
0
1000
2000
3000
4000
5000


Approximating $\int_0^1 4x(1-x)^3 dx$
average value
exact value
Approximated integral value
0.26
0.24
0.22
0.20
0.18
0.16
0.14
0.12
0
1000
2000
3000
4000
5000
Points number used in approximating


Approximating π
average value
exact value
Approximated π value
3.5
3.4
3.3
3.2
3.1
3.0
2.9
0
2000
4000
6000
8000
10000
Points number used in approximating

3 Obserwacje i Wnioski

- Wraz z wzrostem liczby punktów użytych w symulacji otrzymywane wartości w kolejnych powtórzeniach eksperymentu znajdują się w mniejszym przedziale.
- Wraz z wzrostem liczby punktów użytych w symulacji średnie wartości wyników powtórzeń eksperymentu zbiegają do dokładnej wartości całki lub wartości liczby π.
- Użycie już kilku tysięcy punktów pozwala na całkiem dokładne oszacowanie wyznaczanej wartości.