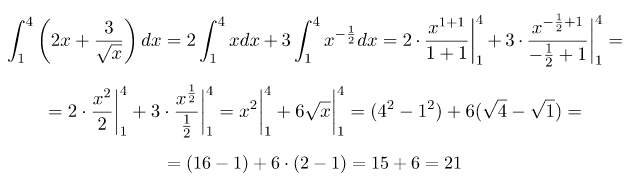
**Численное интегрирование**

**Функция**



от 1 до 4

**Численное значение функции по методу Ньютона-Лейбница:**



**Подпрограммы**

Для создания гибких, легко расширяемых консольных приложений, разработан модуль [FiniteConsole](https://github.com/cyrillelamal/FiniteConsole).

*# FiniteConsole.py*

"""

Finite state machine framework simplifies creation of console applications.

Describe your program with oriented graphs

"""

# Exceptions

class ProgramExistsException(Exception):

"""Rewriting singleton exception"""

pass

class MenuExistsException(Exception):

pass

class UndeterminedOption(Exception):

pass

# Realisations

class Program:

"""Singleton"""

\_PROGRAM = None

def \_\_init\_\_(self, init\_menu=None):

# Check the singleton

if Program.\_PROGRAM is not None:

raise ProgramExistsException

Program.\_PROGRAM = self

self.\_is\_running = False

# Initial state

self.\_init\_menu = init\_menu

# Current state

self.\_current\_menu = init\_menu

# {ID: 'Menu'}

self.menus = {}

if isinstance(init\_menu, Menu):

id\_ = init\_menu.id

self.menus[id\_] = init\_menu

# Actions' parameters and callback

self.result = None

self.args = []

self.kwargs = {}

@property

def init\_menu(self) -> 'Menu':

return self.\_init\_menu

@init\_menu.setter

def init\_menu(self, menu):

"""Set the initial menu. If the program is stopped, set the menu as the current menu"""

self.\_init\_menu = menu

if not self.\_is\_running:

self.\_current\_menu = menu

# New menus will be appended automatically if the program exists

def append\_menus(self, \*menus):

for menu in menus: # 'Menu'

new\_id = menu.id

# Check id

if new\_id in self.menus:

raise MenuExistsException(f'The menu with id "{new\_id}" already exists')

self.menus[new\_id] = menu

return self

def remove\_menus(self, \*menus):

for menu in menus:

# Remove by id

if isinstance(menu, str):

self.menus.pop(menu, None)

# Remove by 'Menu' object (value)

elif isinstance(menu, Menu):

self.menus = {k: v for k, v in self.menus.items() if v != menu}

return self

def resolve\_dependencies(self) -> str:

"""Check if all options and menus are correct"""

report = ''

if self.init\_menu is None:

report += 'The initial menu is undefined.\n'

for menu in self.menus.values(): # No need of colored graphs

# The menu has no options and is not a finite state

if not menu.options and not menu.is\_finite:

report += f'The menu "{menu}" has no options.\n'

continue

# Check if all options lead to menus

for opt in menu.options.values():

out = opt.out

# Lazy binding

if isinstance(out, str):

opt.out = self.menus.get(out, None)

if out is None:

report += f'The option "{opt}" in the menu "{menu}" is broken.\n'

print(report)

return report

def start\_loop(self):

"""Run the program"""

# There are problems

if self.resolve\_dependencies():

return

self.\_is\_running = True

while self.\_is\_running:

# Render

self.\_current\_menu.render()

# Get input

inp = self.\_current\_menu.read\_input()

# # Change the state

self.\_do\_mapping(inp)

def stop\_loop(self):

"""Stop the program"""

if self.\_is\_running:

self.\_is\_running = False

def \_do\_mapping(self, inp):

"""Change menu"""

new\_state = self.\_current\_menu.options.get(str(inp)).out # 'Menu'

if new\_state.is\_finite:

self.result = new\_state.action(\*self.args, \*\*self.kwargs)

self.args.clear()

self.kwargs.clear()

else:

self.\_current\_menu = new\_state

@staticmethod

def get\_program() -> 'Program':

return Program.\_PROGRAM

@staticmethod

def drop():

"""Replace the singleton"""

Program.\_PROGRAM = None

class Menu:

"""State"""

TEST = False

def \_\_init\_\_(self, id\_, action=None):

self.id = str(id\_) # Must be unique

# Check if the menu is a finite state

if callable(action):

is\_finite = True

else:

is\_finite = False

self.\_action = action

self.is\_finite = is\_finite

# {INP: 'Option'}

self.options = {}

# Register menu if the program exists

p = Program.get\_program()

if p is not None:

p.append\_menus(self)

@property

def action(self):

return self.\_action

@action.setter

def action(self, action\_):

self.\_action = action\_

self.is\_finite = True

if not action\_ or action\_ is None:

self.is\_finite = False

def render(self):

"""Display options in the CLI"""

str\_ = '\n'.join([str(opt) for opt in self.options.values()])

print(str\_)

return str\_

def read\_input(self) -> str:

"""Get input for mapping (by user)"""

# FOR UNIT TESTS

if Menu.TEST:

return '1'

inp = None

while inp not in self.options:

inp = input('Choose an option: ')

return inp

def append\_options(self, \*options):

"""Append 'Option'(s) to the menu"""

for opt in options: # 'Option'

new\_inp = opt.inp

# The option is not registered yet

if new\_inp not in self.options:

self.options[new\_inp] = opt

else:

raise UndeterminedOption(f'The same input "{new\_inp}" in the menu "{self}"')

return self

def remove\_options(self, \*options):

"""Remove options from the menu by list of inputs or by list of 'Option' objects"""

# Make list of inputs

options = [opt.inp if isinstance(opt, Option) else opt for opt in options]

# Remove by inputs

self.options = {k: v for k, v in self.options.items() if k not in options}

return self

def remove(self):

"""Remove the menu from the program"""

p = Program.get\_program()

p.remove\_menus(self)

return self

def \_\_str\_\_(self):

return f'{self.id}'

class Option:

"""Mapping"""

def \_\_init\_\_(self, inp, out, description=''):

# Ready for user's CLI input

if not isinstance(inp, str):

inp = str(inp)

self.inp = inp

# Bind 'Menu'

if isinstance(out, (str, Menu,)):

# Bind 'Menu' or string to lazy binding

if isinstance(out, str):

p = Program.get\_program()

if p is not None:

out = p.menus.get(out, out)

else:

raise AttributeError('Inappropriate type of input')

self.out = out

self.description = description

def \_\_str\_\_(self):

return f'{self.inp}. {self.description}'

С использованием разработанного модуля, создана программа для численного интегрирования, представленная в двух файлах: основном файле-скелете меню *lab1.py* и файле *funcs.py*, содержащем логику состояний меню, то есть, производящем основные вычисления. Таким образом неявно использован паттерн проектирования «представление-контроллер»: логика перемещения через состояния программы отделена от реализаций состояний.

*# lab1.py*

import sys

from FiniteConsole.FiniteConsole import Menu, Option, Program

from lab1 import funcs as f

p = Program()

p.init\_menu = Menu('main')

p.init\_menu.append\_options(

Option(1, 'const\_step', 'Методы с постоянным шагом.'),

Option(2, 'var\_step', 'Методы с переменным шагом.'),

Option(3, 'exit', 'Выйти из программы.')

)

# Const steps

Menu('const\_step').append\_options(

Option('1', 'left\_t', 'Метод левых прямоугольников'),

Option('2', 'right\_t', 'Метод правых прямоугольников'),

Option('3', 'trap', 'Метод трапеций'),

Option('4', 'parabola', 'Метод парабол'),

Option('5', 'main', 'В главное меню')

)

# Var steps

Menu('var\_step').append\_options(

Option('1', 'alg\_1', 'Алгоритм 1'),

Option('2', 'alg\_2', 'Алгоритм 2'),

Option('3', 'main', 'В главное меню')

)

# Finite states

# Exit

Menu('exit', lambda: sys.exit())

# Const steps

Menu('left\_t', f.left\_rectangles)

Menu('right\_t', f.right\_rectangles)

Menu('trap', f.trapezium)

Menu('parabola', f.parable)

# Var steps

Menu('alg\_1', f.alg1)

Menu('alg\_2', f.alg2)

def start():

p.start\_loop()

if \_\_name\_\_ == '\_\_main\_\_':

start()

*# funcs.py*

import math

# USER'S DEFINITIONS

# Edges

A = 1

B = 4

# Integrated function

def func(x):

return 2\*x + 3/math.sqrt(x) # 21

# Function maximum

MAX = math.inf

# Constant steps

NUMBER\_OF\_STEPS = [10\*\*2, 10\*\*3, 10\*\*4]

E = 10\*\*(-5)

PRECISION = 5

# END OF USER"S DEFINITIONS

# Deep dark backends

# Generators for sums

def left\_rectangles\_generator(h, start\_from=None):

x = A if start\_from is None else start\_from

# x = A # From 0

while x <= B - h: # To n-1

yield func(x)

x += h

def right\_rectangles\_generator(h):

x = A + h # From 1

while x <= B: # To n

yield func(x)

x += h

def trapezium\_generator(h):

x = A + h # From x0

while x <= B - h: # To n-1

yield func(x)

x += h

# 'View' functions

def left\_rectangles():

for n in NUMBER\_OF\_STEPS:

h = (B - A) / n

i = round(h \* sum(left\_rectangles\_generator(h)), PRECISION)

print(f'Для {n} шагов: {i}')

def right\_rectangles():

for n in NUMBER\_OF\_STEPS:

h = (B - A) / n

i = round(h \* sum(right\_rectangles\_generator(h)), PRECISION)

print(f'Для {n} шагов: {i}')

def trapezium():

for n in NUMBER\_OF\_STEPS:

h = (B - A) / n

i = round(

h \* ((func(A) + func(B)) / 2 + sum(trapezium\_generator(h))),

PRECISION

)

print(f'Для {n} шагов: {i}')

def parable(): # Simpson

for n in NUMBER\_OF\_STEPS:

if n % 2 == 1:

raise ValueError('Number of steps must be even')

h = (B - A) / n / 2

odd\_sum = 0

for i in range(1, n):

odd\_sum += func(2 \* h \* i + A)

even\_sum = 0

for i in range(1, n + 1):

even\_sum += func(h \* (-1 + 2 \* i) + A)

i = round(

h/3 \* (func(A) + func(B) + 2\*odd\_sum + 4\*even\_sum),

PRECISION

)

print(f'Для {n} шагов: {i}')

def alg1():

"""Recount every integral"""

h = math.sqrt(E)

n\_init = int((B - A) / h)

r = abs((B - A)\*\*3 / (12 \* n\_init\*\*2) \* MAX)

print('Остаточный член: {:.5f}'.format(r))

current\_integral = h \* sum(trapezium\_generator(h))

previous\_integral = current\_integral

h /= 2

n = n\_init \* 2

current\_integral = h \* sum(trapezium\_generator(h))

while abs(current\_integral - previous\_integral) >= E:

# Integrate with a new step

previous\_integral = current\_integral

h /= 2

n \*= 2

current\_integral = h \* sum(trapezium\_generator(h))

print(f'Начальное количество шагов: {n\_init}. '

f'Финальное количество шагов: {n}')

print(f'Результат: {round(current\_integral, PRECISION)}')

def alg2():

n\_init = NUMBER\_OF\_STEPS[0]

hv = (B - A) / n\_init # Base step

previous\_integral = hv \* sum(left\_rectangles\_generator(hv))

hs = hv / 2 # Bias step

hd = hv # New step for sum: previous hv

start\_from = A + hs # Bias from x0

current\_integral = hv \* sum(left\_rectangles\_generator(hd, start\_from))

hv = hs # new hv

n = n\_init \* 2

while abs(current\_integral - previous\_integral) >= E:

n \*= 2

previous\_integral = current\_integral

hs = hv / 2 # Bias step

hd = hv # New step for sum: previous hv

start\_from = A + hs

current\_integral = hv \* sum(left\_rectangles\_generator(hd, start\_from))

hv = hs # new hv

print(f'Начальное количество шагов: {n\_init}. '

f'Финальное количество шагов: {n}')

print(f'Результат: {round(current\_integral, PRECISION)}')

Для агрегирования и запуска программ будет разработан еще один модуль, на данный момент — это 'w1.py'.

**Результаты**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Кол-во шагов | Результат | Абсолютная погрешность | Относительная погрешность |
| Левых прямоугольников | 100 | 20.93260 | 0,0674 | 0,32095 |
| 1000 | 20.96477 | 0,03523 | 0,16776 |
| 10000 | 20.99648 | 0,00352 | 0,01676 |
| Правых прямоугольников | 100 | 21.06760 | 0,0676 | 0,32190 |
| 1000 | 20.97825 | 0,02175 | 0,10357 |
| 10000 | 20.99783 | 0,00217 | 0,01033 |
| Трапеций | 100 | 21.00010 | 1E-04 | 0,00047 |
| 1000 | 20.97152 | 0,02848 | 0,13561 |
| 10000 | 20.99715 | 0,00285 | 0,01357 |
| Парабол | 100 | 21.00000 | 0 | 0 |
| 1000 | 21.00000 | 0 | 0 |
| 10000 | 21.00000 | 0 | 0 |
| Алгоритм 1 | 948=>3883008 | 20.99999 | 1E-05 | 4,76E-05 |
| Алгоритм 2 | 100=>6553600 | 20.99999 | 1E-05 | 4,76E-05 |
| Метод Ньютона-Лейбница | - | 21,00000 | - | - |



1. Вторая производная:

2. Максимум второй производной: inf

3. Для метода трапеций остаточный член: inf