1. Описание метода. Расчётные формулы.

Методы численного интегрирования:

- Правило прямоугольников:

$$\int_a^b f(x)dx = (b-a)f\left(\frac{a+b}{2}\right)$$

- Правило трапеций:

$$\int_a^b f(x)dx = (b-a)f\left(\frac{f(a)+f(b)}{2}\right)$$

- Правило Симпсона:

$$\int_a^b f(x)dx = \frac{b-a}{6}\left(f(a)+f\left(\frac{a+b}{2}\right)+f(b)\right)$$

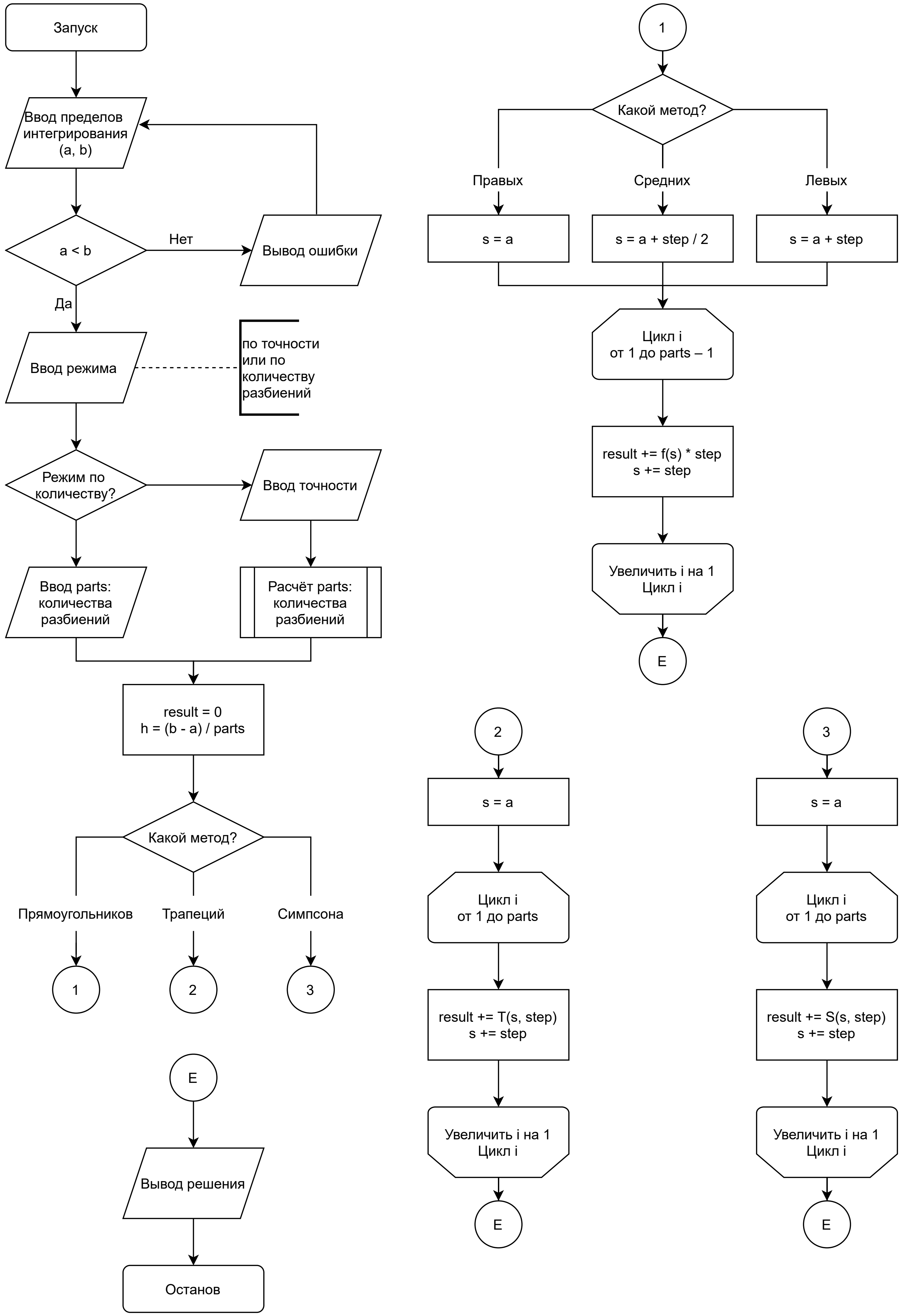
Данные методы используют в комбинации с разбиением исходного отрезка на n частей, чтобы повысить точность. Чем менее точен метод, тем на большее количество нужно разбить отрезок.

Дополнительно метод прямоугольников делится также на метод левых, правых и средних прямоугольников, для которых вычисляется функция от a, b или $\frac{a+b}{2}$ соответственно.

2. Блок-схема

$$T(s, step) = (f(s)+f(s+step)) * \frac{step}{2}$$

$$S(s, step) = \left(f(s)+4f\left(s+\frac{step}{2}\right)+f(s+step)\right) * \frac{step}{6}$$


Запуск
Ввод пределов интегрирования (a, b)
a < b
Нет
Вывод ошибки
Да
Ввод режима
по точности или по количеству разбиений
Режим по количеству?
Ввод точности
Ввод parts: количества разбиений
Расчёт parts: количества разбиений
result = 0
h = (b - a) / parts
Какой метод?
Прямоугольников
Трапеций
Симпсона
1
2
3
E
Вывод решения
Останов
1
Какой метод?
Правых
Средних
Левых
s = a
s = a + step / 2
s = a + step
Цикл i
от 1 до parts – 1
result += f(s) * step
s += step
Увеличить i на 1
Цикл i
E
2
s = a
Цикл i
от 1 до parts
result += T(s, step)
s += step
Увеличить i на 1
Цикл i
E
3
s = a
Цикл i
от 1 до parts
result += S(s, step)
s += step
Увеличить i на 1
Цикл i
E

3. Листинг численного метода

```
@dataclass()
class IntegratorParamSpec(ParamSpec):
    right_limit: NUMBER
    left_limit: NUMBER

    def convert(self) -> tuple[Decimal, Decimal]:
        right, left = map(number_to_decimal, (self.right_limit, self.left_limit))
        return right, left


class Integrator(Solver):
    def _function_or_break(self, equation: AnyEquation, x: Decimal) -> Decimal:
        try:
            return equation.function(x)
        except DecimalException:
            return (equation.function(x - self.precision) + equation.function(x +
self.precision)) / 2

    def _solve(self, equation: AnyEquation, a: Decimal, b: Decimal, step_size:
Decimal) -> Decimal:
        raise NotImplementedError()

    def solve(self, equation: AnyEquation, params: IntegratorParamSpec) ->
Decimal:
        a, b = params.convert()
        step_size: Decimal = (b - a) / self.separations
        return self._solve(equation, a, b, step_size)


class RectangleIntegratorABS(Integrator):
    def _step_start(self, a: Decimal, step_size: Decimal):
        raise NotImplementedError()

    def _solve(self, equation: AnyEquation, a: Decimal, b: Decimal, step_size:
Decimal) -> Decimal:
        result: Decimal = Decimal()
        step_start: Decimal = self._step_start(a, step_size)
        for _ in range(self.separations - 1):
            result += self._function_or_break(equation, step_start) * step_size
            step_start += step_size
        return result


class RightRectangleIntegrator(RectangleIntegratorABS):
    def _step_start(self, a: Decimal, step_size: Decimal):
        return a


class MiddleRectangleIntegrator(RectangleIntegratorABS):
    def _step_start(self, a: Decimal, step_size: Decimal):
```

```
        return a + step_size / 2


class LeftRectangleIntegrator(RectangleIntegratorABS):
    def _step_start(self, a: Decimal, step_size: Decimal):
        return a + step_size


class ComplexIntegratorABS(Integrator):
    def _calc_step(self, f_start: Decimal, f_mid: Decimal, f_next: Decimal,
half_step_size: Decimal):
        raise NotImplementedError()

    def _solve(self, equation: AnyEquation, a: Decimal, b: Decimal, step_size:
Decimal) -> Decimal:
        result: Decimal = Decimal()
        step_size /= 2
        step_start: Decimal = a
        function_start: Decimal = self._function_or_break(equation, step_start)
        function_next: Decimal
        for _ in range(self.separations):
            step_start += step_size
            function_mid = self._function_or_break(equation, step_start)
            step_start += step_size
            function_next = self._function_or_break(equation, step_start)
            result += self._calc_step(function_start, function_mid, function_next,
step_size)
            function_start = function_next
        return result


class TrapezoidalIntegrator(ComplexIntegratorABS):
    def _calc_step(self, f_start: Decimal, f_mid: Decimal, f_next: Decimal,
half_step_size: Decimal):
        return (f_start + f_next) * half_step_size


class SimpsonsIntegrator(ComplexIntegratorABS):
    def _calc_step(self, f_start: Decimal, f_mid: Decimal, f_next: Decimal,
half_step_size: Decimal):
        return (f_start + 4 * f_mid + f_next) * half_step_size / 3
```

4. Примеры работы программы

Пример 1

$$\int_{-3}^{10} 5x^2 + 3x + 2 = 1874.1(6)$$

```
LeftRectangleIntegrator:       1874.162961684975
RightRectangleIntegrator:      1874.15653968590448902
```

```
MiddleRectangleIntegrator:     1874.15975067445474725
TrapezoidalIntegrator:         1874.166684975
SimpsonsIntegrator:            1874.1666666666666667
NewtonLeibnizRule:             1874.1666666666666667
```

Пример 2

$$\int_{-7}^{4} sgn(x) = -3$$

```
LeftRectangleIntegrator:       -2.999997
RightRectangleIntegrator:      -3.000019
MiddleRectangleIntegrator:     -3.000019
TrapezoidalIntegrator:         -2.9997
SimpsonsIntegrator:            -3.043333333333333333
NewtonLeibnizRule:             -3
```

Пример 3

$$\int_{2}^{12} sinc(x) \approx -0.10044173527632148$$

```
LeftRectangleIntegrator:       -0.100443784943594
RightRectangleIntegrator:      -0.10043879130495682
MiddleRectangleIntegrator:     -0.10044128813064347
TrapezoidalIntegrator:         -0.10044169282255945
SimpsonsIntegrator:            -0.10044174622893035
NewtonLeibnizRule:             -0.10044173527632148
```

5. Вывод

Выполнив эту работу я изучил 3 разных метода численного интегрирования, их краткое сравнение:

- Метод прямоугольников: самый простой в реализации, но и самый неточный для сложных функций (а точнее затратный по необходимому количеству разбиений)
- Метод трапеций: не сильно более сложен в реализации, но теперь учитывает значения функции на концах отрезка, а не только в одной его точки, что сильно повышает точность.
- Метод Симпсона: приближает исходную функцию параболами, что в разы увеличивает его точность, но и значительно усложняет реализацию

Также при известной первообразной применима формула Ньютона-Лейбница, которая в точности ограничена только разрядной сеткой компьютера. Именно по этому она использовалась для проверки точностей других методов.

Нельзя забывать и про область значений функции, помнить про точки разрыва. Последние существуют в двух типах: первого (устранимые разрывы и скачки) и второго (полюс и колебания) родов. Разрывы первого рода просто устранить: пусть на t функция имеет такой разрыв, тогда её значение в этой точке можно заменить по формуле (e – некое небольшое число):

$$y(t) = \frac{f(x+e)+f(x-e)}{2}$$

Погрешности методов:

- Метод прямоугольников:

$$|R| \le \frac{(b-a)^3}{24n^2} \max_{x\in[a,b]} |f''(x)|$$

- Метод трапеций:

$$|R| \le \frac{(b-a)^3}{12n^2} \max_{x\in[a,b]} |f''(x)|$$

- Метод Симпсона:

$$|R| \le \frac{(b-a)^5}{180n^4} \max_{x\in[a,b]} |f''''(x)|$$