Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики
Факультет программной инженерии и компьютерной техники

Вычислительная математика
Лабораторная №3

Выполнил:
Беляков Дмитрий
Группа:
P3210
Преподаватель:
Перл О.В.

Санкт-Петербург
2020

Описание метода половинного деления

Начальный интервал [a; b] делим пополам, получая изначальное приближение:

$$x_0 = \frac{(a_0 + b_0)}{2}$$

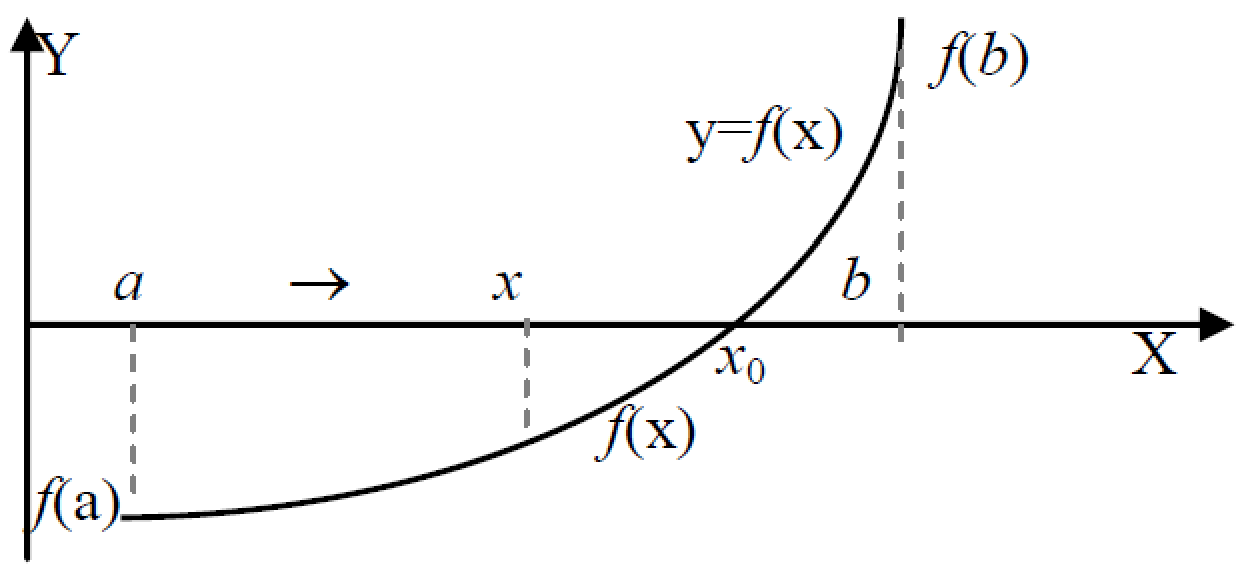
Вычисляем $f_0(x_0)$. Далее из получившихся отрезков $[a_0; x_0] u [x_0; b_0]$ выбираем тот, где функция имеет различные знаки на концах, оставшийся отрезок отбрасываем. Новый отрезок снова делим пополам, получая очередное приближение к корню и т. д:

$$x_i = \frac{(a_i + b_i)}{2}$$

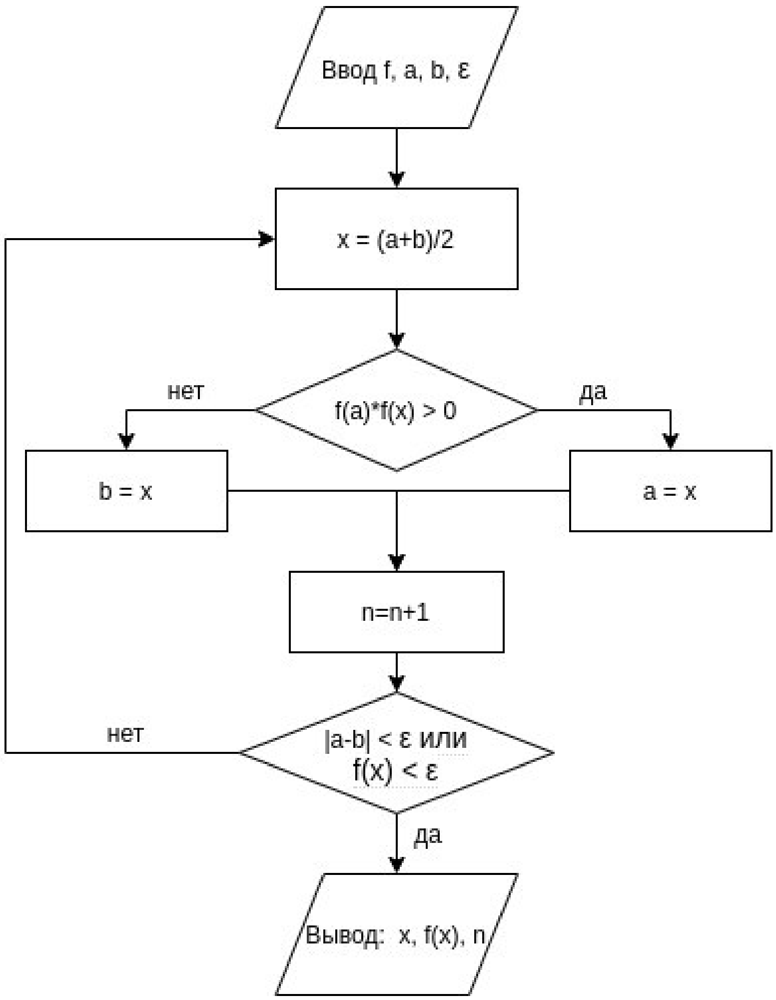
При этом критерием окончания может быть следущие условия:

1. $|(b_n - a_n)| \leq \varepsilon$ в данном случае размер отрезка будет настолько мал, что мы можем взять любое значение x из данного отрезка.
2. $|f(x)| \leq \varepsilon$

Графическое представление метода половинного деления



Блок-схема



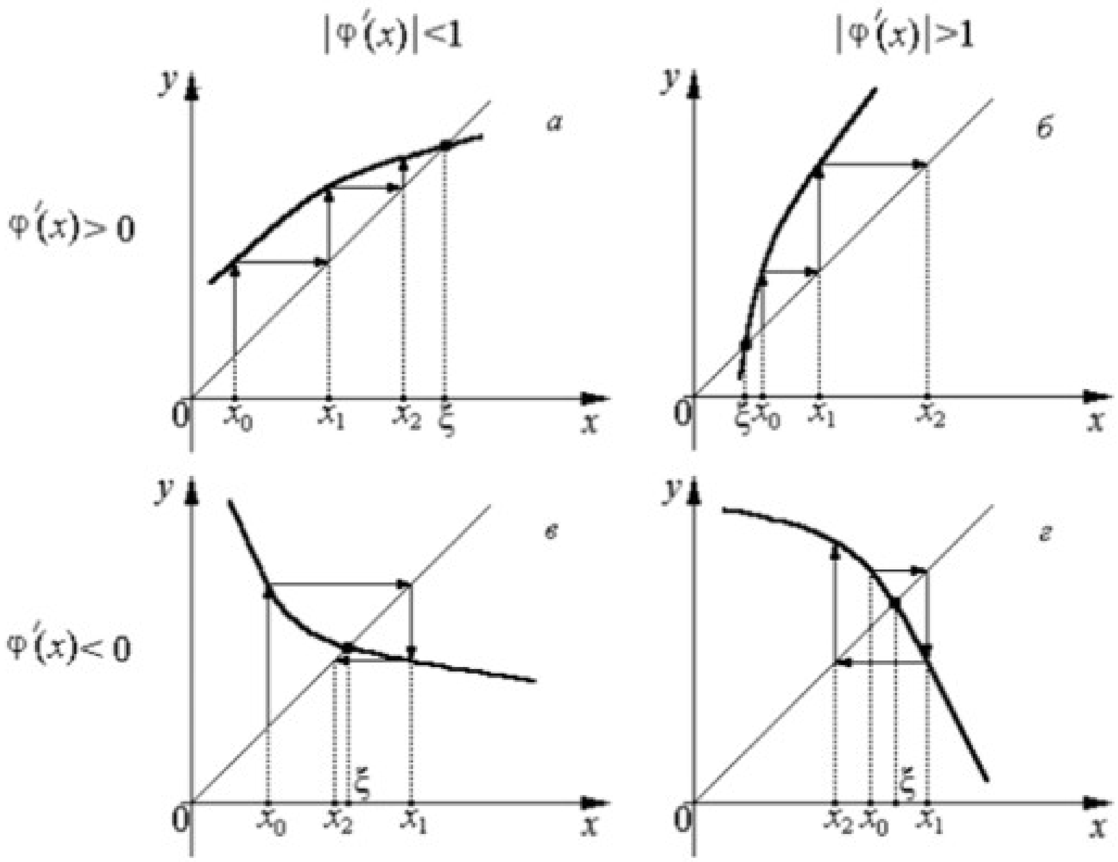
Листинг численного метода:

```
def comp_bisection_method(a, b, function, eps):

    log = []

    i = 1

    while(True):

        x = (a+b)/2
        log.append([a,b,x])
        if abs(a-b) <= eps or abs(function(x)) <= eps: break

        if function(a)*function(x) > 0:
            a = x
        else:
            b = x
        i+=1
```

```
        assert i < limit, "Не удалось найти решение за {0} итераций".format(limit)

    return i, x, log
```

Описание метода простых итераций

В данном методе $f(x)=0$ приводят к виду $x=\varphi(x)$.
Далее выбирают изначальное приближение $x_0 \in [a;b]$ и вычисляют последующие приближения:
$x_i=\varphi(x_{i-1})$
Метод сходится если в некоторой окрестности корня уравнения $f(x)=0$ функция $x=\varphi(x)$ дифференцируема и удовлетворяет неравнеству $|\varphi'(x)|<q$, где $0 \le q<1$ - постоянная, то независимо от начального приближения итерационная последовательность x_n не выходит из этой окрестности.
Хорошо это отображает следующее изображение:



Здесь мы видим, что $\varphi(x_i)$ с каждым приближением только отдаляется от искомого корня.
Способы получения $x=\varphi(x)$ **:**
Основной проблемой данного методая является получение функции $x=\varphi(x)$.
Для этого можно просто выразить x из f(x) путем алгебраических преобразований, но обычно поступают по-другому.
$f(x)=0$
$f(x)*\lambda=0$

$$f(x)*\lambda+x=x$$

Тогда:

$$\varphi(x)=x+\lambda f(x)$$

Тогда из условия теоремы сходимости:

$$\varphi'(x)=1+\lambda f'(x)$$

$$|1+\lambda f'(x)|<1$$

$$-2<\lambda f(x)<0$$

$$|\lambda|\leq\left|\frac{2}{max(f'(x))}\right|$$

Иногда берут не константу, а функцию: $\lambda=s(x)$, или $\lambda=\frac{1}{f'(x_0)}$

Условием окончания будет: $|x_n-x_{n-1}|\leq\varepsilon$

Графическое представление метода простых итераций

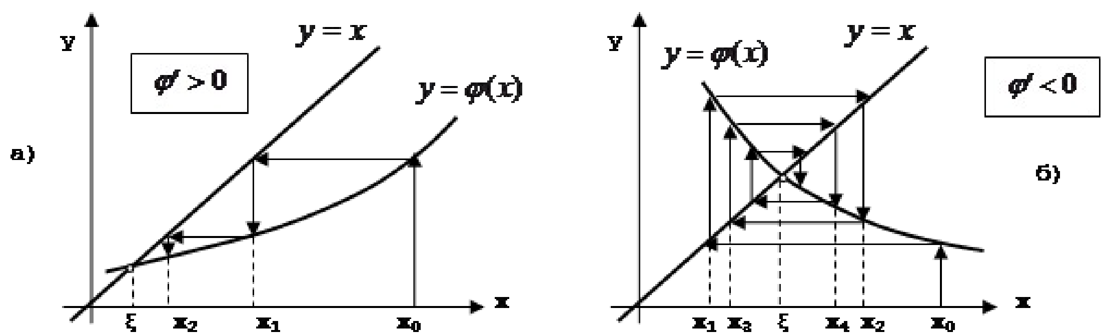
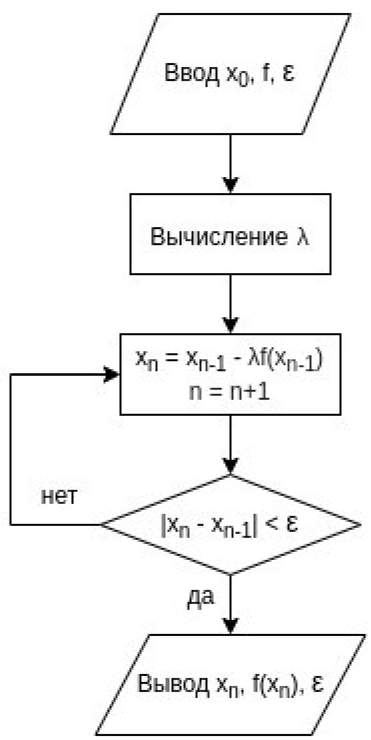


Рис.9 – Схемы сходимости метода простых итераций

Блок-схема



Листинг численного метода

```
def comp_fixed_point_method(a, b, function, eps):

    x = (a+b)/2
    log_values = [x]
    lambda_ = -1e+20
    step = a
    for i in range(int((b-a)/eps)):
        y = function.comp_derivative(step)
        lambda_ = max(lambda_, y)
        step += eps
    if lambda_ == 0:
        lambda_ = 1
    lambda_ = 1/lambda_

    i = 1

    while(True):
        x_0 = x
        try:
            x = x_0 - lambda_*function(x)
        except OverflowError:
            raise AssertionError("Произошло переполнение, не удалось найти решение")
        except ValueError:
            raise AssertionError("X вышел за границы float, не удалось найти решение")
        assert abs(1-lambda_*function.comp_derivative(x)) < 1, "Не удалось достичь сходимости"
        log_values.append(x)
        assert i < limit, "Не удалось найти решение за {0} итераций".format(limit)
        if abs(x_0 - x) < eps: break
        i+=1

    return i, x, {
        "lambda": lambda_,
        "logs": log_values
    }
```

Описание метода Ньютона для решения систем нелинейных уравнений

Имеем систему:

$f_0(x_0, x_1, \ldots, x_n) = 0$

$f_0(x_0, x_1, \ldots, x_n) = 0$

…

$f_0(x_0, x_1, \ldots, x_n) = 0$

Тогда система может быть записана в виде:

$f = (f_0, f_1, \ldots, f_n)$

$f(x) = 0$, где $x = (x_0, x_1, \ldots, x_n)$

Идея метода:

Вблизи точки х каждая функция f_i может быть разложена в ряд Тейлора:

$$f_i(x+\Delta x)=f_i(x)+\sum_{j=1}^{n}\frac{\delta f_i}{\delta x_j}\Delta x_j+\ldots$$

или

$$f(x+\Delta x)=f(x)+J\,\Delta x+\ldots$$, где J - матрица Якоби

Далее, ограничиваясь только первыми 2 членами правой части выражения, мы можем апроксимировать нашу функцию.

Учитывая, что $f_i(x+\Delta x)=0$ получим:

$$f(x)+J\,\Delta x=0$$

$$\Delta x=-J^{-1}f(x)$$

Так как:

$$x_{k+1}=x_k+\Delta x$$

Получим:

$$x_{k+1}=x_k-J^{-1}f(x)$$

Условием окончания будет:

$$max(|x_n-x_{n-1}|)\leq\varepsilon$$

Зачастую обратную матрицу вычислить сложно, поэтому обычно поступают так:

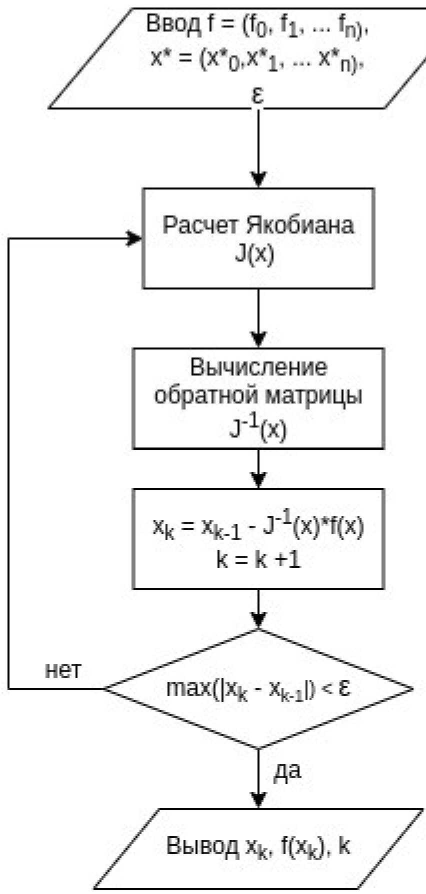
$$\Delta x=-J^{-1}f(x)$$

Домножим слева на J

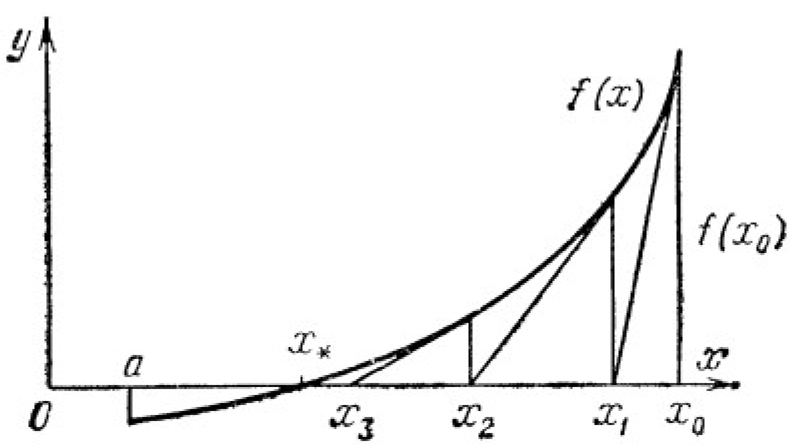
$$J\,\Delta x=f(x)$$

Остаётся лишь решить данное СЛАУ относительно Δx .

Блок-схема



Так как метод Ньютона для решения систем и уравнений аналогичен, то приведу **графическое изображения метода** для 1 уравнения:



Листинг численного метода

```
def comp_newtons_method(vector_x, functions, eps):

    log = []
    vector_x = np.array(vector_x)
```

```
    log.append(vector_x)
    i = 1
    while(True):
        jacoby_m = []
        vector_f = []
        for function in functions:
            try:
                jacoby_m.append(function.comp_derivative(*vector_x))
                vector_f.append(function(*vector_x))
            except ValueError:
                raise AssertionError("X вышел за границы float, не удалось найти решение")
        vector_x_0 = vector_x.copy()
        try:
            jacoby_m = np.linalg.inv(jacoby_m)
        except Exception:
            raise AssertionError("Не удалось найти обратную матрицу")
        delta_x = np.dot(jacoby_m, vector_f)
        vector_x = vector_x - delta_x
        log.append(vector_x)
        if max(abs(vector_x-vector_x_0))<eps:
            break
        assert i < limit, "Не удалось найти решение за {0} итераций".format(limit)
        i+=1
    return i, vector_x, np.array(log)
```

Вывод:

Для решения нелинейных уравнений численными методами прежде всего нужно выполнить очень важный шаг: изоляцию корня — определения интервала [a;b], на котором содержится только один корень уравнения. Без этого шага при решении могут возникнуть ошибки. В большинстве случаев методы позволяют быстро и легко найти корни уровнения, однако некоторые методы весьма чувствительны к начальным приближениям или заданным коэффициентам и функциях (пример — метод простых итераций)

Метод половинного деления:

- Плюсы:
  - Простота реализации
  - Абсолютная сходимость
- Минусы:
  - Ищет корни только на заданном промежутке
  - Медленный метод: линейная скорость
  - Неоднозначность результатов — если на промежутке имеется несколько корней, непонятно к какому относится найденный корень

Метод простых итераций:

- Плюсы:
  - Простота реализации
  - При хорошо подробанном приближении и $\varphi(x)$ метод быстро сходится
- Минусы

    - Чем дальше приближенное значение x от корня, тем больше итераций потребуется, возможен так же вариант сходимости к другому корню
    - Сложность подбора $\varphi(x)$ или λ

Метод Ньютона

- Плюсы:
    - Высокая скорость сходимости — квадратичная
- Минусы:
    - Необходимость вычисления производной на каждой итерации, а для систем уравнений — всех частных производных
    - Как и метод простых итераций — чувствителен к начальным приближениям
    - Для системы — сложность вычисления обратной матрицы или СЛАУ