МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по практической работе №4
по дисциплине «Вычислительная математика»
Тема: Решение нелинейных уравнений методами аппроксимации

Студент гр. 0304 Крицын Д. Р.
Преподаватель Попова Е. В.

Санкт-Петербург
2021

Цель работы.

Целью работы является нахождение корня уравнения $f(x)=0$ методами *Ньютона* и *простых итераций* с заданной точностью ε, исследование скорости сходимости и обусловленности .этих методов.

Основные теоретические положения.

Метод Ньютона. В случае, когда известно хорошее начальное приближение решения уравнения $f(x)=0$, эффективным методом повышения точности является метод *Ньютона*. Он состоит в построении итерационной последовательности (1)

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}, \qquad (1)$$

где $f'(x_n)\neq 0$. Последовательность сходится к корню c уравнения $f(x)=0$. По теореме о сходимости метода *Ньютона* c должен быть простым корнем уравнения $f(x)=0$; в отсекающем промежутке этого корня $[a,b]$ $f(a)\cdot f(b)<0$; функция $f(x)$ – дважды непрерывно дифференцируема и $f''(x)\neq 0$.

Для оценки погрешности n-го приближения корня предлагается пользоваться неравенством (2)

$$|c-x_n|\leq\frac{M_2}{2m_1}|x_n-x_{n-1}|^2, \qquad (2)$$

где M_2 - наибольшее значение модуля второй производной ¿$f''(x)\vee$¿ на отрезке [a,b]; m_1 - наименьшее значение модуля первой производной ¿$f'(x)\vee$¿ на отрезке [a,b]. Таким образом, если $|x_n-x_{n-1}|<\varepsilon$, то $|c-x_n|\leq\frac{M_2}{2m_1}\varepsilon^2$. Это означает, что при хорошем начальном приближении корня после каждой итерации число верных десятичных знаков в очередном приближении удваивается, т.е. процесс сходится очень быстро и имеет место квадратическая сходимость.

Если необходимо найти корень с точностью ε, то итерационный процесс можно прекращать, когда выполняется неравенство (3)

$$|x_n-x_{n-1}|<\varepsilon=\sqrt{\frac{2m_1\varepsilon}{M_2}}. \qquad (3)$$

Если на *(n-1)*-м шаге очередное приближение x_{n-1} не удовлетворяет условию окончания процесса, то вычисляются величины $f(x_{n-1}), f'(x_{n-1})$ и следующие приближение корня $x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$. При выполнении условия (3) величина x_n принимается за приближенное значение корня *c*, вычисленное с точностью ε.

Постановка задачи. В практической работе предполагается использование функций *newton* и *_round*. Этапы выполнения практической работы:

1 Графически или аналитически отделить корень уравнения (найти отрезки [left, right], на которых функция $f(x)$. удовлетворяет условиям теоремы о сходимости метода Ньютона).

2 Проверить функцию $f(x)$ на выпуклость вверх или вниз, выбрать начальное приближение корня $x_0 \epsilon [a, b]$,так чтобы $f(x_0) \cdot f''(x_0) > 0$.

3 Получить аналитическое выражение функций $f'(x)$ и $f''(x)$. Оценить снизу величину $m_1 = \min\limits_{x \epsilon [a,b]} |f'(x)|$, оценить сверху величину $M_2 = \max\limits_{x \epsilon [a,b]} |f''(x)|$.

4 По заданному ε сосчитать условие окончания итерационного процесса $\varepsilon_2 = \sqrt{\frac{2 m_1 Eps}{M_2}}$.

5 Составить подпрограммы-функции вычисления $f(x)$, $f'(x)$, предусмотрев округление их значений с заданной точностью Δ.

6 Составить головную программу, вычисляющую корень уравнения $f(x) = 0$ и содержащую обращение к подпрограммам $f(x)$, $f'(x)$, *_round, newton* и индикацию результатов.

7 Провести вычисления по программе. Исследовать скорость сходимости метода и чувствительность метода к ошибкам в исходных данных.

Метод простых итераций. Метод *простых итераций* решения уравнения $f(x)=0$ заключается в замене исходного уравнения эквивалентным ему уравнением $x=\varphi(x)$ и построении последовательности $x_{n+1}=\varphi(x_n)$, сходящейся при $n\to\infty$ к точному решению.

Корень уравнения $x=\varphi(x)$ является точкой пересечения двух графиков $y=x$ и $y=\varphi(x)$. Сходимость метода зависит от вида функции $\varphi(x)$. В зависимости от величины модуля первой производной ¿$\varphi'(x)\vee$¿ метод может сходиться и расходиться.

Достаточные условия сходимости метода *простых итераций* формулируются следующей теоремой:

Если функция $\varphi(x)$ определена, дифференцируема и принадлежит отрезку $[a,b]$, то существует числоq, такое что ¿$\varphi'(x)\vee\le q<1$ на $[a,b]$, и последовательность$x_{n+1}=\varphi(x_n)$,сходится к единственному решению на $[a,b]$ уравнения $x=\varphi(x)$ при $\forall\, x_0\in[a,b]$

$$\lim_{n\to\infty} x_n=\lim_{n\to\infty}\varphi(x_n)=c, f(c)=0, c\epsilon[a,b]. \qquad (1)$$

Если $\varphi'(x)>0$, то $|c-x_n|<\frac{q}{1-q}|x_n-x_{n-1}||$, если $\varphi'(x)<0$, то $|c-x_n|<|x_n-x_{n-1}||$.

Рассмотрим один шаг итерационного процесса. Исходя из найденного на предыдущем шаге значения x_{n-1}, вычисляется $y=\varphi(x_{n-1})$. Если $|y-x_{n-1}|>\varepsilon$, то полагается $x_n=y$ и выполняется очередная итерация. Если же $|y-x_{n-1}|<\varepsilon$, то вычисления заканчиваются и за приближенное значение корня принимается величина $x_n=y$. Погрешность результата вычислений зависит от знака производной $\varphi'(x)$: при $\varphi'(x)>0$: погрешность определения корня составляет $\frac{q\varepsilon}{1-q}$, а при $\varphi'(x)<0$, погрешность не превышает ε. Здесь q - число, такое, что $|\varphi'(x)|\le q<1$ на отрезке [a,b]. Существование числа q является условием сходимости метода в соответствии с отмеченной выше теоремой.

Для применения метода *простых итераций* определяющее значение имеет выбор функции $\varphi(x)$, в уравнении $x=\varphi(x)$, эквивалентном исходному. Функцию $\varphi(x)$ необходимо подбирать так, чтобы $|\varphi'(x)|\le q<1$. Это

обусловливается тем, что если $\varphi'(x)<0$, на отрезке [a,b], то последовательные приближения $x_n=\varphi(x_{n-1})$. будут колебаться около корня *c*, если же $\varphi'(x)>0$, то последовательные приближения будут сходиться к корню *c* монотонно.

Число обусловленности метода *простых итераций* (2)

$$\nu_\Delta=\frac{1}{|1-\varphi'(x)|}.$$

Постановка задачи. В практической работе предлагается использовать программы функции *iter* и *phi.*

Необходимо осуществить следующие шаги:

1) Графически или аналитически отделить корень уравнения $f(x)=0$ и получить отрезок [left, right].

2) Сосчитать $f'(x)$, найти на отрезке [left, right] m – минимальное значение $f'(x)$, M – максимальное значение $f'(x)$.

3) Преобразовать уравнение $f(x)=0$ к виду, удобному для итераций $\varphi(x)=x-\frac{2}{m+M}f(x)$.

3) Выбрать начальное приближение x_0, лежащее на [left, right].

4) Составить подпрограммы для вычисления значений $\varphi(x)$,$\varphi'(x)$, предусмотрев округление вычисленных значений с точностью Delta.

5) Составить головную программу, вычисляющую корень уравнения и содержащую обращение к программам $\varphi(x)$,$\varphi'(x)$, *phi*, *iter* и индикацию результатов.

6) Провести вычисления по программе. Исследовать скорость сходимости и обусловленность метода.

Выполнение работы.

$$f(x)=\frac{1+\cos x}{3-\sin x}-x.$$

1. Локализуем корень функции графическим методом. График f(x) приведён на рис. 1.

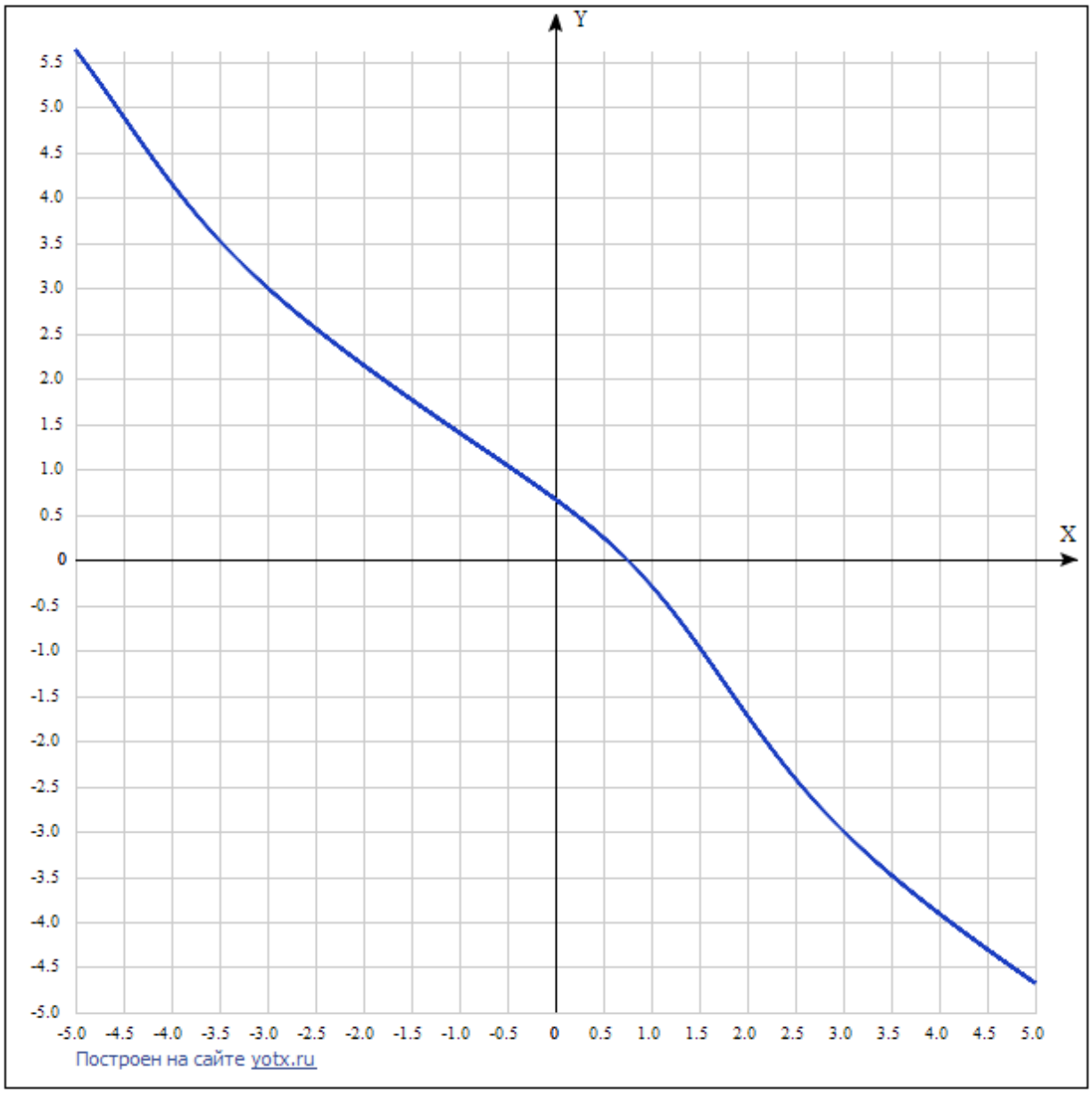

Рис. 1 — График функции f(x)

При помощи графика можно взять начальный промежуток аппроксимации [0; 1,5].

2. Вычислим первую и вторую производную функции f(x).

$$f'(x)=\left(\frac{1+\cos x}{3-\sin x}-x\right)'=\frac{(-\sin x)(3-\sin x)-(-\cos x)(1+\cos x)}{(3-\sin x)^2}-1=$$

$$=\frac{\sin^2 x-3\sin x+\cos x+\cos^2 x}{(3-\sin x)^2}-1=\frac{1-3\sin x+\cos x}{(3-\sin x)^2}-1.$$

$$f''(x)=\left(\frac{1-3\sin x+\cos x}{(3-\sin x)^2}-1\right)'=$$

$$=\frac{(-3\cos x-\sin x)(3-\sin x)^2+2(\cos x)(3-\sin x)}{(3-\sin x)^4}=$$

$$=\frac{-27\cos x-9\sin x+18\sin x\cos x+6\sin^2 x-3\cos x\sin^2 x-\sin^3 x}{(3-\sin x)^4}+$$

$$+\frac{6\cos x-2\cos x\sin x}{(3-\sin x)^4}=$$

$$=\frac{-21\cos x-9\sin x+10\sin 2x+6\sin^2 x-3\cos x\sin^2 x-\sin^3 x}{(3-\sin x)^4}.$$

Графики первой и второй производной находятся на рис. 2 и рис. 3 соответственно.

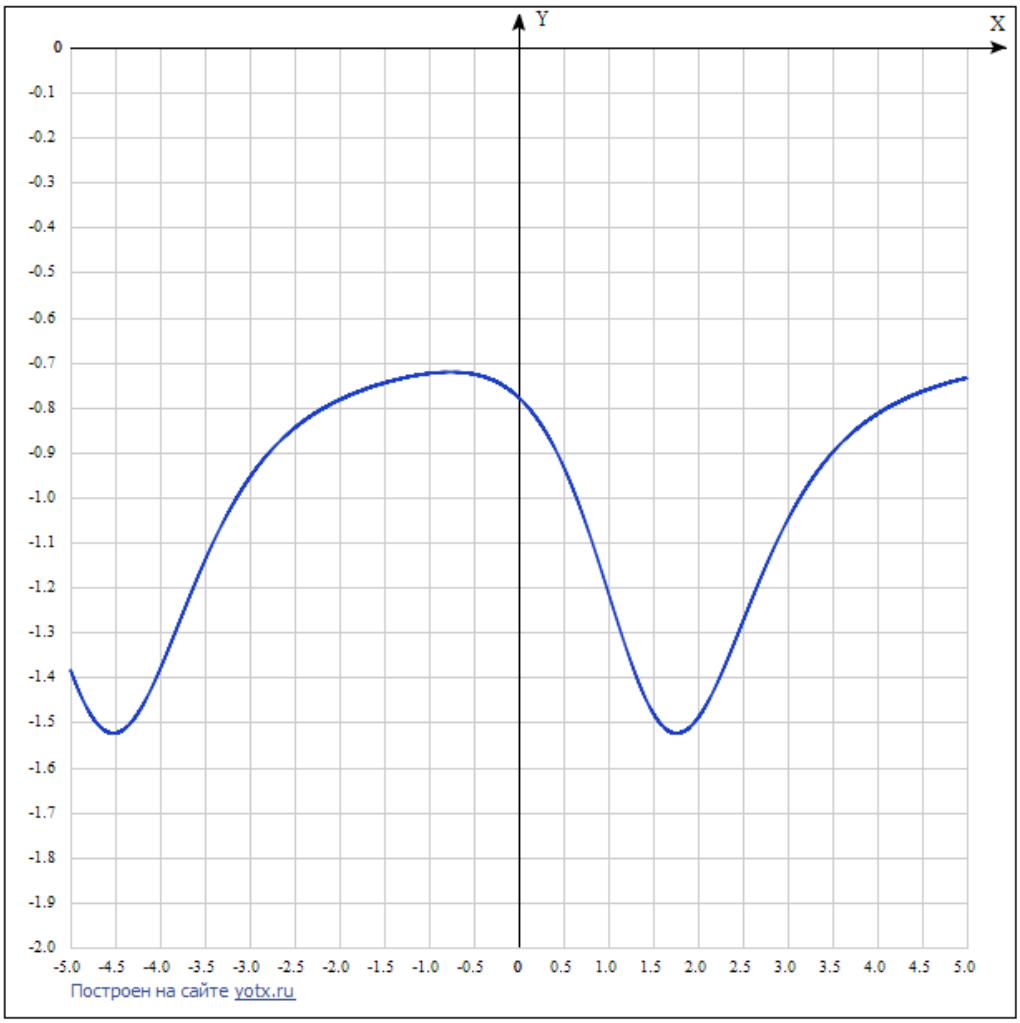

Рис. 2 — График производной функции f(x)

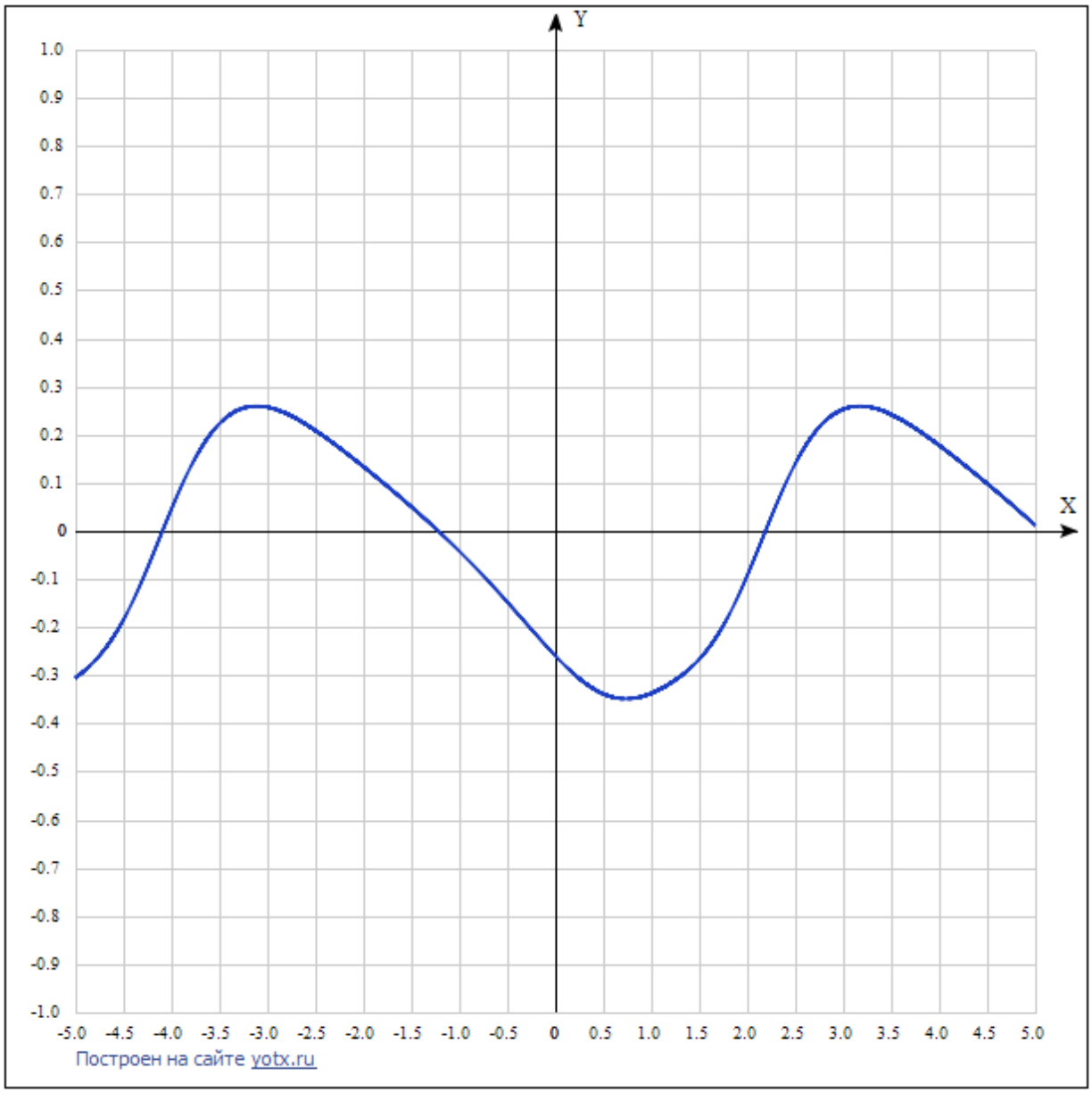

Рис. 3 — График второй производной функции f(x)

Как видно по графикам 2 и 3, обе производные функции знакопостоянны, а значит условия теоремы о сходимости метода Ньютона выполнены.

Метод Ньютона.

1. Значение второй производной на выбранном начальном отрезке аппроксимации [0; 1,5] всегда отрицательно. Значит, среди двух точек — границ этого отрезка — в качестве точки входа в методе Ньютона подходит только $x_0 = 1{,}5$, т. к. $f(x_0)f''(x_0) < 0$.

2. С помощью функций *min_f()* и *max_f()* были посчитаны минимальное значение первой (m_1) и маскимальное значение второй (M_2) производных на данном отрезке.

$$m_1 = \min_{x\in[a,b]} |f'(x)| = -1{,}479269, M_2 = \max_{x\in[a,b]} |f''(x)| = -2{,}131890.$$

3. Зададим $\Delta = 10^{-6}$. Пусть значение ε варьируется от 10^{-6} до 10^{-1}. Результаты работы представлены на рис. 4.

Метод Ньютона, Δ = 1e-6, 1e-6 <= ε <= 1e-1

left	right	ε	x0	ε2	итераций
0	1.5	1e-06	0.747111	0.00117803	4
0	1.5	1e-05	0.747113	0.00372526	3
0	1.5	0.0001	0.747113	0.0117803	3
0	1.5	0.001	0.747113	0.0372526	3
0	1.5	0.01	0.749803	0.117803	2
0	1.5	0.1	0.749803	0.372526	2

Рис. 4 — Результат работы метода Ньютона при постоянном Δ

Видно, что с увеличением значения ε увеличивается значение минимального шага метода Ньютона — условия окончания этого метода. Благодаря этому метод заканчивает работу за меньшее количество итераций.

4. Пусть $\varepsilon = 10^{-3}$. Значение Δ варьируется от 10^{-6} до 10^{-2}. Проверим обусловленность метода Ньютона. Коэффициент обусловленности для этого метода считается по формуле

$$\nu = \frac{1}{|f'(x_0)|} = \frac{1}{\left|\frac{1-3\sin x+\cos x}{(3-\sin x)^2}-1\right|}.$$

Результаты работы метода приведены на рис. 5.

Метод Ньютона, ε = 1e-3, 1e-6 <= Δ <= 1e-1

left	right	Δ	x0	N	N max	итераций
0	1.5	1e-06	0.747113	0.946407	1000	3
0	1.5	1e-05	0.74711	0.946409	100	3
0	1.5	0.0001	0.7471	0.946414	10	3
0	1.5	0.001	0.747	0.946465	1	3
0	1.5	0.01	0.75	0.944932	0.1	3

Рис. 5 — Результат работы метода Ньютона при постоянном ε

Видно, что с убыванием значения Δ максимальное число обусловленности возрастает, а число обусловленности почти не меняется. При $\Delta \le 10^{-3}$ метод считается хорошо обусловленным.

Метод простых измерений.

1. Найдём минимальное и максимальное значения первой производной f(x), воспользовавшись функциями *min_f(), max_f()*.

$$m_1 = \min_{x\in[a,b]} |f'(x)| = -1{,}479269, M_1 = \max_{x\in[a,b]} |f'(x)| = -0.777778.$$

Данные значения будут использованы для задания функции приближения $\varphi(x)$.

2. Функция, используемая в этом методе, имеет вид $\varphi(x) = x - \alpha f(x)$, где $\alpha = \frac{2}{m_1 + M_1}$. Производная данной функции имеет вид $\varphi'(x) = x' - \alpha f'(x) = 1 - \alpha f'(x) = 1 - \frac{2f'(x)}{m_1 + M_1}$, и значит, максимальное значение производной по модулю составляет $|\varphi(x_{max})| = |1 - 2\frac{M_1}{m_1 + M_1}| = |\frac{M-m}{M+m}| = q$, минимальное - $|\varphi(x_{min})| = |1 - \frac{2m}{m+M}| = |\frac{m-M}{M+m}| = q = |\varphi(x_{max})|$. Для проведения вычислений данным методом в качестве точки входа будем использовать правую границу отрезка локализации.

3. Пусть $\Delta = 10^{-6}$, а значение ε варьируется от 10^{-6} до 10^{-1}. Результаты работы представлены на рис. 6.

Метод простых итераций, Δ = 1e-6, 1e-6 <= ε <= 1e-1

left	right	ε	x0	phi'	итераций
0	1.5	1e-06	0.747111	0.0637091	6
0	1.5	1e-05	0.747109	0.0637101	5
0	1.5	0.0001	0.747073	0.0637283	4
0	1.5	0.001	0.74651	0.064013	3
0	1.5	0.01	0.738	0.0683009	2
0	1.5	0.1	0.738	0.0683009	2

Рис. 6 — Результат работы метода простых итераций при постоянном Δ

Видно, что с возрастанием значения ε уменьшается количество итераций метода простых итераций. Например, для $\varepsilon = 10^{-6}$ требуется 6 итераций, а для $\varepsilon = 10^{-2}$ — 2 итерации.

4. Пусть $\varepsilon = 10^{-3}$, а значение Δ варьируется от 10^{-6} до 10^{-1}. Проверим обусловленность метода простых итераций. Число обусловленности этого метода считается по формуле $\nu = \frac{1}{|1-\varphi'(x)|}$, где $\varphi(x) = x - \frac{2f(x)}{m_1+M_1}$. Тогда $\nu = \frac{1}{|1-1+\alpha f'(x)|} = |\frac{1}{\alpha f'(x)}| = |\frac{m_1+M_1}{2f'(x)}|$. Результаты работы представлены на рис. 7.

Метод простых итераций, ε = 1e-3, 1e-6 <= Δ <= 1e-1

left	right	Δ	x0	N	N max	итераций
0	1.5	1e-06	0.74651	0.946716	1000	3
0	1.5	1e-05	0.74651	0.946716	100	3
0	1.5	0.0001	0.7465	0.946721	10	3
0	1.5	0.001	0.747	0.946465	1	3
0	1.5	0.01	0.75	0.944932	0.1	4

Рис. 7 — Результат работы метода простых итераций при постоянном ε

Видно, что меньшему значению Δ соответствует большее максимальное число обусловленности, а число обусловленности почти не меняется. Метод простых итераций считается обусловленным при $\Delta \leq 10^{-3}$.

5. Пусть $\varepsilon = \Delta = 10^{-6}$. Будем сдвигать отрезок локализации, и вместе с ним точку входа (правую границу отрезка локализации) — обе границы на -0,5 с

шагом 0,05. При изменении отрезка локализации будут изменяться минимальное и максимальное значения производной f'(x), а соответственно и значение коэффициента α. С изменением α будет изменяться и функция-приближение φ(x). Результат работы метода представлен на рис. 8.

Метод простых итераций, Δ = 1e-6, ε = 1e-6, -0,5 <= left <= 0, 1 <= right <= 1,5

left	right	x0	min f'(x)	max f'(x)	q
0	1.5	0.747111	-1.47924	-0.777778	-0.31079
-0.05	1.45	0.747111	-1.46104	-0.769017	-0.310317
-0.1	1.4	0.747111	-1.44017	-0.761212	-0.308423
-0.15	1.35	0.747111	-1.41686	-0.7543	-0.305164
-0.2	1.3	0.747111	-1.3914	-0.748221	-0.300605
-0.25	1.25	0.747111	-1.3641	-0.742915	-0.294817
-0.3	1.2	0.747111	-1.33527	-0.738326	-0.287878
-0.35	1.15	0.747111	-1.30524	-0.7344	-0.279874
-0.4	1.1	0.747111	-1.27435	-0.731087	-0.270895
-0.45	1.05	0.747111	-1.2429	-0.728337	-0.261037
-0.5	1	0.747111	-1.21122	-0.726107	-0.250402

Рис. 8 — Результат работы метода простых итераций при изменении отрезка локализации корня

Видно, что при изменении точки входа значение найденного корня меняется не очень сильно, однако значение *q*, являющееся границей значения по модулю φ'(x) уменьшается в связи с изменением минимального и максимального значений f'(x).

Сравнение методов решения нелинейных уравнений.

1. На примере данной функции сравним работу методов численного решения нелийных уравнений: методы бисекции, хорд, Ньютона, простых итераций. Для этого используем заданный ранее отрезок локализации [0; 1,5] и $\Delta = 10^{-6}$, $10^{-6} \leq \varepsilon \leq 10^{-1}$. Результаты работы программы представлены на рис. 10.

```
Сравнение методов:
Метод бисекции, x0 = [0.000000; 1.500000]
|       ε|        x0|итераций|
|  1e-06| 0.747112|       20|
|  1e-05| 0.747105|       17|
| 0.0001| 0.747253|       13|
|  0.001| 0.748535|       10|
|   0.01| 0.738281|        7|
|    0.1|   0.5625|        3|

Метод хорд, x0 = [0.000000; 1.500000]
|       ε|        x0|итераций|
|  1e-06|  0.74711|        8|
|  1e-05| 0.747107|        7|
| 0.0001| 0.747088|        6|
|  0.001| 0.746363|        4|
|   0.01| 0.742858|        3|
|    0.1| 0.723028|        2|

Метод Ньютона, x0 = 1.500000
|       ε|        x0|итераций|
|  1e-06| 0.747111|        4|
|  1e-05| 0.747113|        3|
| 0.0001| 0.747113|        3|
|  0.001| 0.747113|        3|
|   0.01| 0.749803|        2|
|    0.1| 0.749803|        2|

Метод простых итераций, x0 = 1.500000
|       ε|        x0|итераций|
|  1e-06| 0.747111|        6|
|  1e-05| 0.747109|        5|
| 0.0001| 0.747073|        4|
|  0.001|  0.74651|        3|
|   0.01|    0.738|        2|
|    0.1|    0.738|        2|
```

Рис. 9 — Сравнение результатов работы различных методов

Видно, что метод бисекции является самым медленным: для $\varepsilon = 10^{-6}$ он прекращает работу за 20 итераций. При таком же значении ε метод хорд

работает в течении 8 итераций, метод Ньютона — 4, метод простых итераций — 6. Конечно, не стоит забывать, что метод Ньютона корректен только для гладких на отрезке локализации функций.

Вывод.

В этой работе были исследованы методы аппроксимации корней линейных уравнений — метод Ньютона и метод простых итераций. Данные методы были проверены на обусловленность и скорость сходимости. Они работают быстрее методов бисекции и хорд, но сходятся далеко не для всех функций.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <math.h>
#include <errno.h>

const double delta_min = 1e-6;

#define ERRNO_APPROX_INTERVAL 1
#define ERRNO_APPROX_PRECISION 2
#define ERRNO_APPROX_INVALID_DERIVATIVE 3

#define ERRNO_ROUND_PRECISION 2

double _round(double x, double delta)
{
         if(delta <= 1e-9){
          errno = ERRNO_ROUND_PRECISION;
          return NAN;
         }
         return delta * floor((x / delta) + (x > 0 ? 0.5 : -0.5));
}

double min_f(double (*f)(double), double left, double right,
          int abs, double delta)
{
         double _min = INFINITY;
         if(abs)
          for(double x = left; x <= right+delta; x += delta)
            { if(fabs(f(x)) < _min) _min = fabs(f(x)); }
         else
          for(double x = left; x <= right+delta; x += delta)
            { if(f(x) < _min) _min = f(x); }
         return _min;
}
```

```
double max_f(double (*f)(double), double left, double right,
          int abs, double delta)
{
         double _max = abs ? 0 : -INFINITY;
         if(abs)
          for(double x = left; x <= right+delta; x += delta)
            { if(fabs(f(x)) > _max) _max = fabs(f(x)); }
         else
          for(double x = left; x <= right+delta; x += delta)
            { if(f(x) > _max) _max = f(x); }
         return _max;
}


double bisect(double (*f)(double),
          double left, double right,
          double eps, int* N)
{
         double e = fabs(eps) * 2;
         double fleft = f(left);
         double fright = f(right);
         double x = (left + right) / 2, y;

         if(fleft*fright > 0){
          errno = ERRNO_APPROX_INTERVAL;
          return NAN;
         }
         if(eps <= 0){
          errno = ERRNO_APPROX_PRECISION;
          return NAN;
         }

         *N = 0;

         if(fleft == 0) return left;
         if(fright == 0) return right;
```

```
        while((right - left) >= e)
        {
         x = (left + right) / 2;
         y = f(x);
         if(y == 0)
           return x;
         if(y*fleft < 0)
           right = x;
         else {
           left = x;
           fleft = y;
         }

         (*N)++;
        }

        return x;
}

double chord(double (*f)(double), double left, double right, double
eps, int* N)
{
        double fleft = f(left);
        double fright = f(right);
        double x, y;

        if(fleft*fright > 0){
         errno = ERRNO_APPROX_INTERVAL;
         return NAN;
        }
        if(eps <= 0){
         errno = ERRNO_APPROX_PRECISION;
         return NAN;
        }
```

```
          *N = 0;

          if(fleft == 0) return left;
          if(fright == 0) return right;

          do{
           x = left - (right - left) * fleft / (fright - fleft); //
предполагаемый корень хорды
           y = f(x);

           if(y == 0)
             return x;
           if(y * fleft < 0){
             right = x;
             fright = y;
           }
           else{
             left = x;
             fleft = y;
           }

           (*N)++;
          } while(fabs(y) >= eps);

          return x;
}

double newton(double (*f)(double), double (*fd1)(double), double m1,
double M2,
           double x, double eps, int* N, double* eps2_out)
{
          if(eps <= 0){
           errno = ERRNO_APPROX_PRECISION;
           return NAN;
          }
```

```
        double y, y1, dx, eps0;
        *N = 0;

        eps0 = sqrt(2 * m1 * eps / M2);

        do {
         y = f(x);
         if(y == 0)
           return x;
         y1 = fd1(x);
         if(y1 == 0){
           errno = ERRNO_APPROX_INVALID_DERIVATIVE;
           return NAN;
         }

         dx = y / y1;
         x -= dx;
         (*N)++;
        }
        while(fabs(dx) >= eps0);

        *eps2_out = eps0;
        return x;
}

double phi(double (*f)(double), double (*fd1)(double), double x, double
left, double right, double delta)
{
        if(x == 0)
         return NAN;

        double _min = min_f(fd1, left, right, 0, delta_min);
        double _max = max_f(fd1, left, right, 0, delta_min);
        double s = x - 2 / (_min + _max) * f(x);
        s = _round(s, delta);
        return s;
```

```
}

double iter(double (*f)(double), double (*fd1)(double), double x0,
double left, double right,
          double eps, double delta, int* N)
{
         if(eps <= 0){
          errno = ERRNO_APPROX_PRECISION;
          return NAN;
         }

         double x1 = phi(f, fd1, x0, left, right, delta);
         double x2 = phi(f, fd1, x1, left, right, delta);

         for(*N = 2; pow(x1 - x2, 2) > fabs((2*x1 - x0 - x2) * eps); ++
(*N))
         {
          x0 = x1; x1 = x2;
          x2 = phi(f, fd1, x1, left, right, delta);
         }

         return x2;
}



double f(double x) { return (1 + cos(x)) / (3 - sin(x)) - x; }
double fd1(double x) { return (1 - 3*sin(x) + cos(x)) / pow(3 - sin(x),
2) - 1; }
double fd2(double x) { return (-21*cos(x) - 9*sin(x) + 10*sin(2*x) +
6*pow(sin(x), 2) - 3*cos(x)*pow(sin(x), 2) - pow(sin(x), 3)) / (3 -
pow(sin(x), 4)); }
double f_cond(double x) { return 1 / fabs(fd1(x)); }



int main()
{
```

```
        double left = 0, right = 1.5;

        double m1 = min_f(&fd1, left, right, 0, delta_min);
        double M1 = max_f(&fd1, left, right, 0, delta_min);
        double m2 = min_f(&fd2, left, right, 0, delta_min);
        double M2 = max_f(&fd2, left, right, 0, delta_min);
        printf("m1 = %lf, M1 = %lf, m2 = %lf, M2 = %lf\n", m1, M1, m2,
M2);
        double alpha = 2 / (m1 + M1);

        puts("Метод Ньютона, Δ = 1e-6, 1e-6 <= ε <= 1e-1");
        puts("|left|right|      ε|      x0|        ε2|итераций|");
        for(double delta = 1e-6, eps = 1e-6; eps <= 1e-1 + 1e-3; eps
*= 10)
        {
         int N; double eps2;
         double x0 = newton(&f, &fd1, m1, M2,
                    right, eps, &N, &eps2);
         x0 = _round(x0, delta);

         printf("|%4g|%5g|%7g|%7g|%10g|%8d|\n", left, right, eps, x0,
eps2, N);
        }

        puts("Метод Ньютона, ε = 1e-3, 1e-6 <= Δ <= 1e-1");
        puts("|left|right|      Δ|      x0|        N|  N max|
итераций|");
        for(double delta = 1e-6, eps = 1e-3; delta <= 1e-2 + 1e-3;
delta *= 10)
        {
         int N; double eps2;
         double x0 = newton(&f, &fd1, m1, M2,
                    right, eps, &N, &eps2);
         x0 = _round(x0, delta);

         double cond = f_cond(x0);
```

```
          double cond_max = eps/delta;
          printf("|%4g|%5g|%7g|%8g|%9g|%7g|%8d|\n", left, right, delta,
x0, cond, cond_max, N);
          }

          puts("Метод простых итераций, Δ = 1e-6, 1e-6 <= ε <= 1e-1");
          puts("|left|right|      ε|      x0|       phi'|итераций|");
          for(double delta = 1e-6, eps = 1e-6; eps <= 1e-1 + 1e-3; eps
*= 10)
          {
          int N;
          double x0 = iter(&f, &fd1, right, left, right, eps, delta,
&N);
          x0 = _round(x0, delta);
          double dphi = 1 - alpha * fd1(x0);

          printf("|%4g|%5g|%7g|%8g|%11g|%8d|\n", left, right, eps, x0,
dphi, N);
          }

          puts("Метод простых итераций, ε = 1e-3, 1e-6 <= Δ <= 1e-1");
          puts("|left|right|      Δ|      x0|        N|  N max|
итераций|");
          for(double delta = 1e-6, eps = 1e-3; delta <= 1e-2 + 1e-3;
delta *= 10)
          {
          int N;
          double x0 = iter(&f, &fd1, right, left, right, eps, delta,
&N);
          x0 = _round(x0, delta);

          double cond = f_cond(x0);
          double cond_max = eps/delta;
          printf("|%4g|%5g|%7g|%8g|%9g|%7g|%8d|\n", left, right, delta,
x0, cond, cond_max, N);
          }
```

```
        puts("Метод простых итераций, Δ = 1e-6, ε = 1e-6, -0,5 <= left
<= 0, 1 <= right <= 1,5");
        puts("|left | right|       x0|min f'(x)|max f'(x)|
q|");
        for(double delta = 1e-6, eps = 1e-6; right >= 1.0 - 0.01;
right -= 0.05, left -= 0.05)
        {
         int N;
         double x0 = iter(&f, &fd1, right, left, right, eps, delta,
&N);
         x0 = _round(x0, delta);
         double dphi = 1 - alpha * fd1(x0);

         double m1 = min_f(&fd1, left, right, 0, delta_min);
         double M1 = max_f(&fd1, left, right, 0, delta_min);
         double q = (M1 - m1) / (M1 + m1);
         printf("|%5g|%6g|%9g|%9g|%9g|%9g|\n", left, right, x0, m1,
M1, q);
        }
        left = 0; right = 1.5;

        puts("Сравнение методов:");
        printf("Метод бисекции, x0 = [%lf; %lf]\n", left, right);
        puts("|      ε|       x0|итераций|");
        for(double delta = 1e-6, eps = 1e-6; eps <= 1e-1 + 1e-3; eps
*= 10)
        {
         int N;
         double x0 = bisect(&f, left, right, eps, &N);
         x0 = _round(x0, delta);

         printf("|%7g|%9g|%8d|\n", eps, x0, N);
        }
        printf("\nМетод хорд, x0 = [%lf; %lf]\n", left, right);
        puts("|      ε|       x0|итераций|");
```

```
        for(double delta = 1e-6, eps = 1e-6; eps <= 1e-1 + 1e-3; eps
*= 10)
        {
         int N;
         double x0 = chord(&f, left, right, eps, &N);
         x0 = _round(x0, delta);

         printf("|%7g|%9g|%8d|\n", eps, x0, N);
        }
        printf("\nМетод Ньютона, x0 = %lf\n", right);
        puts("|      ε|       x0|итераций|");
        for(double delta = 1e-6, eps = 1e-6; eps <= 1e-1 + 1e-3; eps
*= 10)
        {
         int N; double eps2;
         double x0 = newton(&f, &fd1, m1, M2,
                    right, eps, &N, &eps2);
         x0 = _round(x0, delta);

         printf("|%7g|%9g|%8d|\n", eps, x0, N);
        }
        printf("\nМетод простых итераций, x0 = %lf\n", right);
        puts("|      ε|       x0|итераций|");
        for(double delta = 1e-6, eps = 1e-6; eps <= 1e-1 + 1e-3; eps
*= 10)
        {
         int N;
         double x0 = iter(&f, &fd1, right, left, right, eps, delta,
&N);
         x0 = _round(x0, delta);

         printf("|%7g|%9g|%8d|\n", eps, x0, N);
        }
}
```