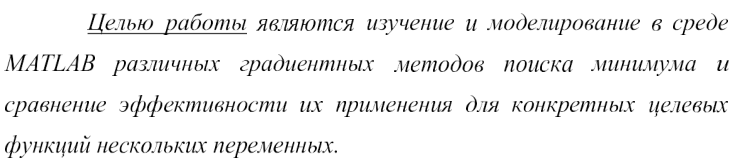
**ИССЛЕДОВАНИЕ МЕТОДОВ БЕЗУСЛОВНОЙ КОНЕЧНОМЕРНОЙ ОПТИМИЗАЦИИ**

Лабораторная работа № 3.

**ЧИСЛЕННЫЕ МЕТОДЫ ПОИСКА ЭКСТРЕМУМА ДЛЯ ДИФФЕРЕНЦИРУЕМЫХ ФУНКЦИЙ МНОГИХ ПЕРЕМЕННЫХ**

Вариант № 9.

Отчет выполнили: Фролов Никита, Учаев Сергей, Булыгин Василий



Всего будет рассмотрено 7 метода, один из которых имеет 3 подвида:

1. Метод градиента с постоянным (заданным) шагом;
2. Метод наискорейшего спуска;
3. Метод сопряжённого градиента для квадратичных функций;
4. Метод Фленча-Ривза;
5. Метод Ньютона (модифицированный метод Ньютона, метод Марквардта);
6. Метод Давида-Флетчера-Пауэлла;
7. Метод Бройдена-Флетчера-Гольдфарба-Шанно.

Все рассмотренные методы реализовано на Mathlab 2022.

Начальные условия:



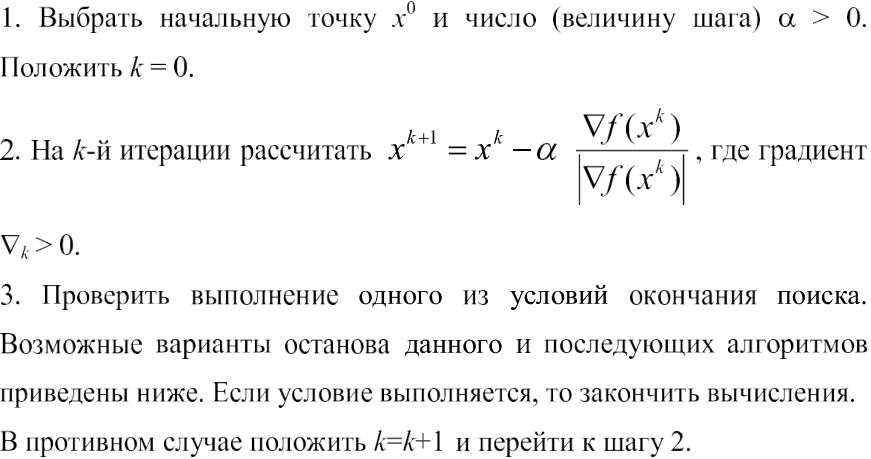


Каждый метод рассматривается при четырёх разных точностях.

По полученным данным из программы каждому методу будет строиться график количества итераций.

Метод градиента с постоянным (заданным) шагом.

Алгоритм:



Программный код:

Файл NumberOne.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-5)

E2 = 10^(-7)

E3 = 10^(-9)

E4 = 10^(-12)

% Значения коэффициентов

c11 = 3;

c22 = 2;

c33 = 6

c44 = 15;

c12 = -2

c34 = -8;

c14 = -8;

c24 = 12;

c13 = -4;

g = 0.1; % постоянная шага

%Начальная точка

x1 = 0;

x2 = 0;

x3 = 0

x4 = 0;

k = 1; % Счетчик шагов

%Массивы для хранения промежуточных координат

xm1 = [x1];

xm2 = [x2];

xm3 = [x3];

xm4 = [x4];

i = 2;

while True

% Спуск по координатам одновременный

gr1 = 6 \* x1 – 2 \* x2 – 4 \* x3;

gr2 = 4 \* x2 – 2 \* x1 + 12 \* x4;

gr3 = 4 \* x2 – 2 \* x1 + 6 \* x4;

gr4 = 26 \* x4 – 8 \* x1 + 6 \* x2;

x1 = x1 + g \* gr1;

x2 = x2 + g \* gr2;

x3 = x3 + g \* gr3;

x4 = x4 + g \* gr4;

%Сохранение координат

xm1(i) = x1;

xm2(i) = x2;

xm2(i) = x2;

xm4(i) = x4;

i = i + 1;

%Проверка условия останова

if sqrt(gr1^2 + gr2^2 + gr3^2 + gr4^2) <= E1; % Тут меняем точность на необходимую

break; % Выход из цикла в случае выполнения условия

end

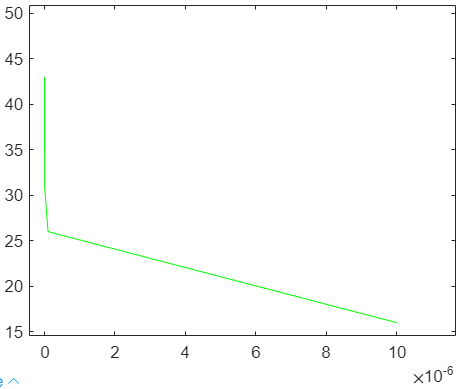
k = k + 1; % Счётчик итераций

end

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-5): | E2 = 10^(-7): | E1 = 10^(-9): | E1 = 10^(-12): |
| x(1) | -0.536 | -0.8438 | -0.43161 | -0.05246944 |
| x(2) | -0.148 | 1.7141 | -1.66163 | -0.41818462 |
| x(3) | -0.49 | -0.8692 | 1.4846 | 0.18289460 |
| x(4) | -1.962 | 1.2313 | 1.47567 | 0.27206111 |
| F | 1.63247 | 1.6324813 | 1.632481237 | 1.632481236677 |
| k | 16 | 26 | 31 | 43 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

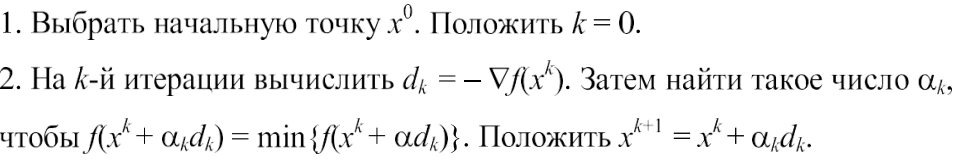
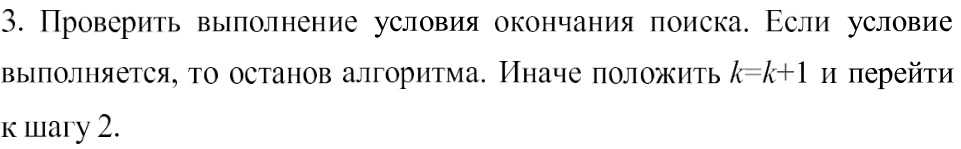
E = [10^(-5) 10^(-7) 10^(-9) 10^(-12)]

K = [16 26 31 43]

plot(E, K, 'g-');

Метод наискорейшего спуска.

Алгоритм:

Программный код:

Файл NumberTwo.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-5)

E2 = 10^(-7)

E3 = 10^(-9)

E4 = 10^(-12)

% Функция

Y = @(x) (x(1) – x(2))^2 + 4\* (x(3) – x(4))^2 + (x(2) + 6 \* x(4))^4 + 2\* (x(1) – x(3))^2

% Производные

dyx1 = @(x)6 \* x1 – 2 \* x2 – 4 \* x3;

dyx2 = @(x)-2 \* x1 + 4 \* x2 + 12 \* x4;

dyx3 = @(x)12 \* x3 – 8 \* x4 - 4 \* x1;

dyx4 = @(x)10 \* x4 – 8 \* x3 + 12 \* x2;

% Метод наискорейшего спуска

a = 1;

x = [0, 0, 0, 0];

k = 0;

while (sqrt(dyx1(x)^2 + dyx2(x)^2 + dyx3(x)^2 + dyx4(x)^2) > E1)

f=@(a)y(x – a \* [dyx1(x), dyx2(x), dyx3(x), dyx4(x)]);

local\_min = dihotomy(f, 0, 4, E1); % Поиск локального минимума методом дихотомии

a = local\_min(1);

x = x - a\*[dyx1(x), dyx2(x), dyx3(x), dyx4(x)];

k = k + 1;

end

Файл dihot.m:

function [ret] = dihotomy(cb, a, b, eps)

    d2 = eps / 10;

    n = 0;

    x\_min = a;

    while(b - a) > eps

        x\_min = (b + a) / 2;

        fp = cb(x\_min + d2);

        fm = cb(x\_min - d2);

        if fp < fm

            a = x\_min - d2;

        else

            b = x\_min + d2;

        end

        n = n + 1;

    end

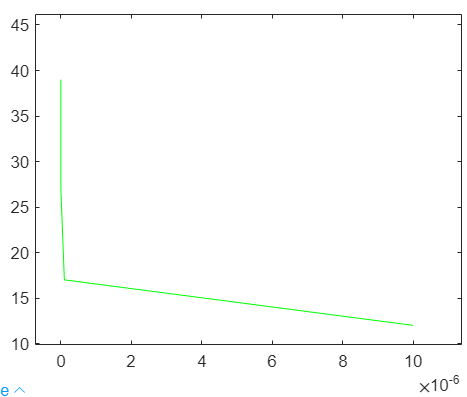
    ret = [x\_min, n];

end

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-5): | E2 = 10^(-7): | E1 = 10^(-9): | E1 = 10^(-12): |
| x(1) | 0.337 | 0.5304 | 1.492286 | -1.11288605 |
| x(2) | 0.261 | -1.8793 | -1.081179 | 1.42300345 |
| x(3) | -0.662 | 0.8845 | -0.804220 | -0.75354417 |
| x(4) | -0.136 | -1.3944 | 1.150003 | -0.07365147 |
| F | 1.63249 | 1.6324811 | 1.632481235 | 1.632481236675 |
| k | 12 | 17 | 27 | 39 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

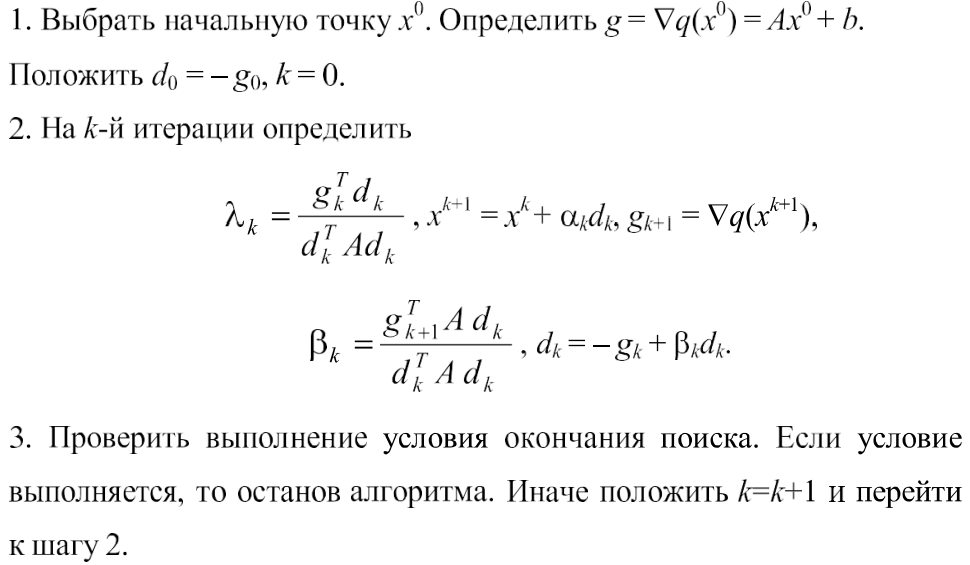
E = [10^(-5) 10^(-7) 10^(-9) 10^(-12)]

K = [12 17 27 39]

plot(E, K, 'g-');

Метод сопряжённого градиента для квадратичных функций.

Алгоритм:



Программный код:

Файл NumberThree.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-5)

E2 = 10^(-7)

E3 = 10^(-9)

E4 = 10^(-12)

x0 = (0, 0, 0, 0); % Начальная точка

function f = NT(x0, E1)

x = x0;

syms xi yi a; % Определяют каждую символьную переменную

% Функция

Y = @(x) (x(1) – x(2))^2 + 4\* (x(3) – x(4))^2 + (x(2) + 6 \* x(4))^4 + 2\* (x(1) – x(3))^2

fx1 = diff (f, x1i); % Найти f частную производную первого порядка по x1

fx2 = diff (f, x2i); % найти частную производную f первого порядка от x2

fx3 = diff (f, x3i); % найти частную производную f первого порядка от x3

fx4 = diff (f, x4i); % найти частную производную f первого порядка от x4

fx1 = sub (fx1, {x1i, x2i, x3i , x4i}, x0); % Подставляют координаты начальной точки для вычисления конкретного значения функции

fx2 = sub (fx2, {x1i, x2i, x3i, x4i}, x0); % Подставляют координаты начальной точки для вычисления конкретного значения функции

fx3 = sub (fx2, {x1i, x2i, x3i, x4i}, x0); % Подставляют координаты начальной точки для вычисления конкретного значения функции

fx4 = sub (fx4, {x1i, x2i, x3i, x4i}, x0); % Подставляют координаты начальной точки для вычисления конкретного значения функции

g0 = [fx1, fx2, fx3, fx4]; % Начального градиента точки

k = 0; % Счётчик итераций

while (sqrt (fx1 ^ 2 + fx2 ^ 2 + fx3 ^ 2 + fx4 ^ 2)) >= E1 % Проверка конца программы условие

if k <= 0;

d = -g0; % Первое направление поиска является направлением отрицательного градиента начальной точки

else

d = di;

end

x = x + a \* d; % Итеративная формула расчета, рассчёт координаты следующий точки

f = sub (f, {x1i, x2i, x3i, x4i}, x); % Построение унарной функцию m (a) of a, чтобы найти лучший шаг a

f1 = diff (f); % Дифференцирует m (a)

f1 = solve(f1, а); % Получение лучшего шага а

if f1 ~= 0;

ai = f1;

else

break; % Если a = 0, вырвитесь из цикла, эта точка является минимальной точкой

end

x = subs (x, a, ai); % Рассчитать значение координаты следующей точки

fx1i = diff(f, x1i);

fx2i = diff(f, x2i);

fx3i = diff(f, x3i);

fx4i = diff(f, x4i);

fx1i = subs(fx2i, {x1i, x2i, x3i, x4i}, x);

fx2i = subs(fx2i, {x1i, x2i, x3i, x4i}, x);

fx3i = subs(fx2i, {x1i, x2i, x3i, x4i}, x);

fx4i = subs(fx2i, {x1i, x2i, x3i, x4i}, x);

gi = [fx1i, fx2i, fx3i, fx4i]; % Следующего направления градиента

b = (fx1i^2 + fx2i^2 + fx3i^2 f + x4i^3) / (fx1^2 + fx2^2 + fx3^2 + fx4^2);

di = -gi + b \* d;% Рекурсивная формула для направления поиска сопряженного, следующего направления поиска

k = k + 1; % Номер итерации плюс 1

fx1 = fx1i;

fx2 = fx2i; % Обновление параметра градиента условия итерации

fx3 = fx3i;

fx4 = fx4i;

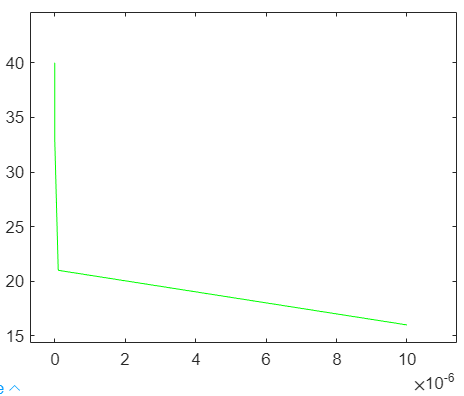
end

fmin = NT(x0, E1)

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-5): | E2 = 10^(-7): | E1 = 10^(-9): | E1 = 10^(-12): |
| x(1) | 1.392 | -1.8881 | -1.97458 | -1.42069754 |
| x(2) | -1.226 | 1.1386 | 0.231691 | 0.39024672 |
| x(3) | 1.067 | -1.476 | 0.228805 | -0.25944612 |
| x(4) | 1.756 | -0.3485 | 0.332846 | -0.47222471 |
| F | 1.63249 | 1.6324811 | 1.632481235 | 1.632481236675 |
| k | 16 | 21 | 33 | 40 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

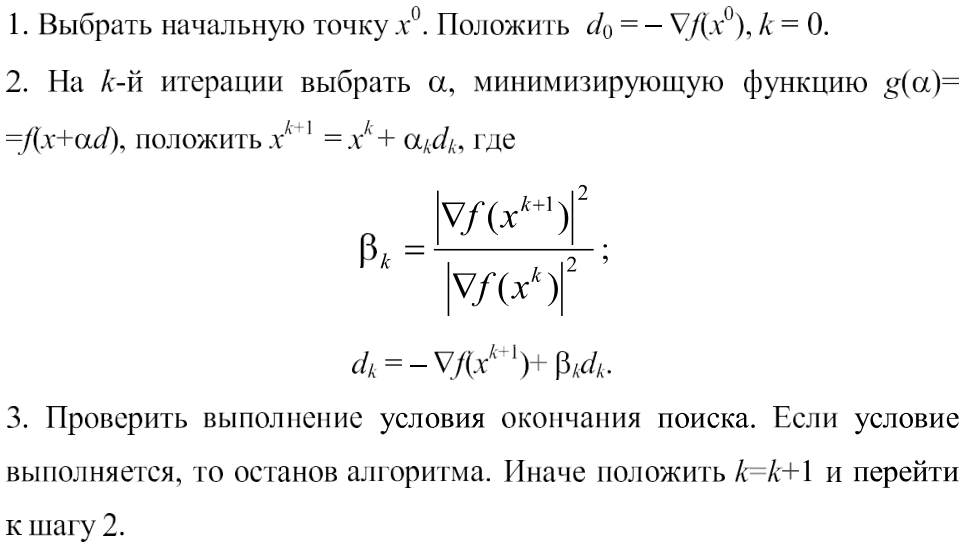
E = [10^(-5) 10^(-7) 10^(-9) 10^(-12)]

K = [16 21 33 40]

plot(E, K, 'g-');

Метод Фленча-Ривза.

Алгоритм:



Программный код:

Файл NumberFour.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-5)

E2 = 10^(-7)

E3 = 10^(-9)

E4 = 10^(-12)

% Значения коэффициентов

c11 = 3;

c22 = 2;

c33 = 6

c44 = 15;

c12 = -2

c34 = -8;

c14 = -8;

c24 = 12;

c13 = -4;

% Начальная точка

x1 = 0;

x2 = 0;

x3 = 0;

x4 = 0

k = 1; % Счетчик шагов

%Массивы для хранения промежуточных координат

x1trace = [x1];

x2trace = [x2];

x3trace = [x3];

x4trace = [x4];

i = 2;

while True;

% Вычисление коэффициента шага

% Спуск по координатам одновременный

% Производные

gr1 = @(x)6 \* x1 – 2 \* x2 – 4 \* x3;

gr2 = @(x)-2 \* x1 + 4 \* x2 + 12 \* x4;

gr3 = @(x)12 \* x3 – 8 \* x4 - 4 \* x1;

gr4 = @(x)10 \* x4 – 8 \* x3 + 12 \* x2;

g = -(gr1^2 + gr2^2 + gr3^2 + gr4^2) / (gr1^2 \* gr2^2 \* gr3^2 \* gr4^2);

Bk = abs((gr1 + gr2 + gr3 + gr4)^2 / abs(x1 + x2 + x3 + x4))

x1 = Bk – g \* gr1;

x2 = Bk – g \* gr2;

x3 = Bk – g \* gr3;

x4 = Bk – g \* gr4

%Сохранение координат

x1trace(i) = x1;

x2trace(i) = x2;

x3trace(i) = x3;

x4trace(i) = x4;

i = i + 1;

%Проверка условия останова

if sqrt(gr1^2 + gr2^2 + gr3^2 + gr4^2) <= E1;

break;

%Выход из цикла в случае выполнения условия

end

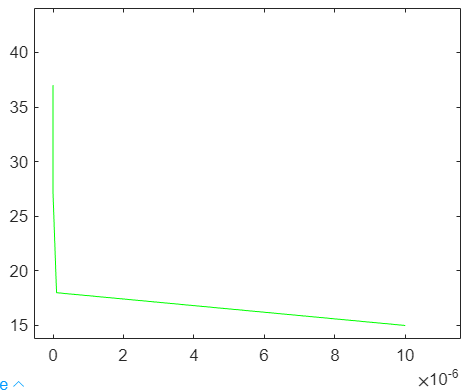
k = k + 1;

end

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-5): | E2 = 10^(-7): | E1 = 10^(-9): | E1 = 10^(-12): |
| x(1) | -0.671 | -1.37714 | -0.364553 | 0.48214132 |
| x(2) | 0.598 | 1.9203 | 0.845537 | 0.45365636 |
| x(3) | 0.282 | -0.24602 | -0.976412 | -0.83581279 |
| x(4) | 0.528 | 1.81001 | 1.086528 | 0.15853532 |
| F | 1.63247 | 1.6324813 | 1.632481235 | 1.632481236675 |
| K | 15 | 18 | 27 | 37 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

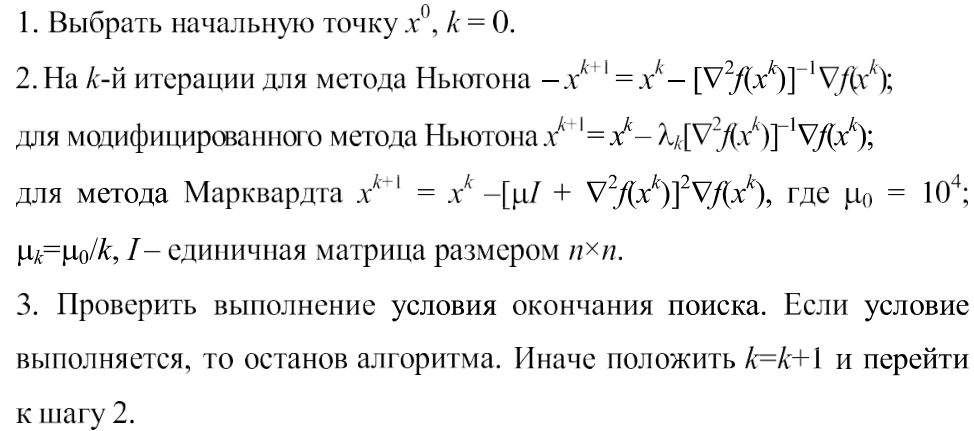
E = [10^(-5) 10^(-7) 10^(-9) 10^(-12)]

K = [15 18 27 37]

plot(E, K, 'g-');

Метод Ньютона (обычный).

Алгоритм:



Программный код:

Файл NumberFive.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-5)

E2 = 10^(-7)

E3 = 10^(-9)

E4 = 10^(-12)

% Функция

Y = @(x) (x(1) – x(2))^2 + 4\* (x(3) – x(4))^2 + (x(2) + 6 \* x(4))^4 + 2\* (x(1) – x(3))^2

function [fxin, xmin] = Newton(f, E1)

a = -1;

b = 2; % Границы

df = char(diff(sym(f))); % Символьно ищем первую

ddf = char(diff(sym(df))); % Вторую производные

F = inline(f); % Преобразуем в функции

F1 = inline(df);

F2 = inline(ddf);

eps = E1; % Задаем точность

if F(a) \* F2(a) > 0 % Проверка с какой границы начинать искать

xmin = b;

else

xmin = a;

end

while abs(F1(xmin)) > eps

X0 = xmin; % X0 - значение предыдущего шага

xmin = X0 - (F1(X0) / (F2(X0))); % Расчет нового значения

fxmin = F(xmin);

end

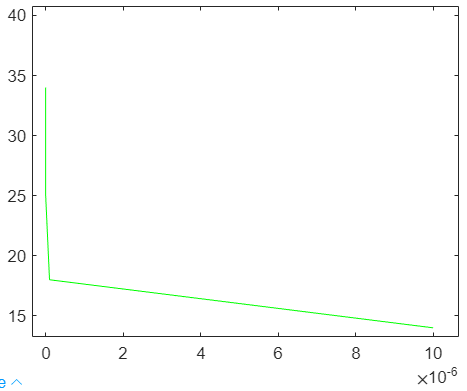
end

[fxin, xmin] = Newton(f, E1)

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-5): | E2 = 10^(-7): | E1 = 10^(-9): | E1 = 10^(-12): |
| x(1) | 0.204 | 1.9428 | -0.788733 | 0.98089062 |
| x(2) | 0.157 | 1.49915 | -0.287549 | 0.67987650 |
| x(3) | 0.223 | 1.4287 | - 0.275818 | 0.81494892 |
| x(4) | 1.919 | -1.7256 | -0.905808 | 0.53513584 |
| F | 1.63247 | 1.6324813 | 1.632481235 | 1.632481236675 |
| K | 14 | 18 | 25 | 34 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

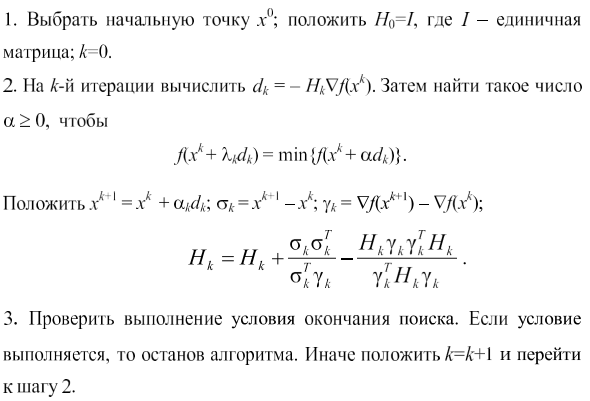
E = [10^(-5) 10^(-7) 10^(-9) 10^(-12)]

K = [14 18 25 34]

plot(E, K, 'g-');

Метод Давида-Флетчера-Пауэлла.

Алгоритм:



Программный код:

Файл NumberSix.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-5)

E2 = 10^(-7)

E3 = 10^(-9)

E4 = 10^(-12)

function [X, g, i] = DFP(func, grad, hess, x\_init, alpha, max\_iter, permissible\_error)

X = zeros(MAX\_ITER, num\_var);

g = zeros(MAX\_ITER, num\_var);

X(1,:) = x\_init;

B = inv(hess(X(1,:)));

I = 1;

for k = 1:max\_iter

g(k,:) = grad(X(k,:));

X(k+1,:) = X(k,:) - alpha\*B\*g(k,:);

g(k+1,:) = grad(X(k+1,:));

p\_k = X(k+1,:) - X(k,:);

q\_k = g(k+1,:)- g(k,:);

B = B - (B\*(p\_k \* p\_k') \* B) / (p\_k' \* B \* p\_k) + (q\_k \* q\_k') / (p\_k' \* q\_k);

i = i + 1;

disp('Input value:');

disp(X(k + 1,:));

disp('Function value:');

disp( func(X(k + 1,:)));

if func(X(k,:)) - func(X(k + 1,:)) < permissible\_error

i = k;

break;

end

end

end

function objective\_fn = f(x)

% Функция

Y = @(x) (x(1) – x(2))^2 + 4\* (x(3) – x(4))^2 + (x(2) + 6 \* x(4))^4 + 2\* (x(1) – x(3))^2

end

function gradient = grad(x)

% Производные

gr1 = @(x)6 \* x1 – 2 \* x2 – 4 \* x3;

gr2 = @(x)-2 \* x1 + 4 \* x2 + 12 \* x4;

gr3 = @(x)12 \* x3 – 8 \* x4 - 4 \* x1;

gr4 = @(x)10 \* x4 – 8 \* x3 + 12 \* x2;

end

function H = hess(x)

H = [ 3, 0; 1, 6];

end

X\_INIT = [5, -2];

ALPHA = 0.01;

max\_error = 0.001;

[X, g, i] = DFP(f, grad, hess, X\_INIT, ALPHA, MAX\_ITER, max\_error);

disp('The optimized value of the funciton is: ')

disp(f(X(i,:)))

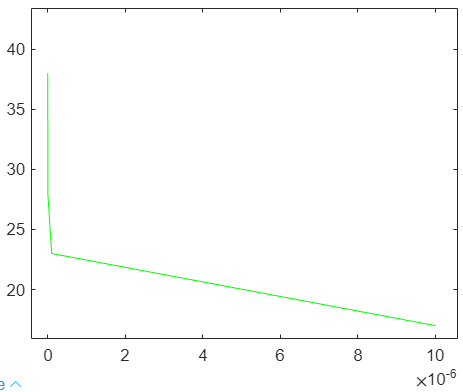
disp('The optimum value of input is: ')

disp(X(i,:))

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-5): | E2 = 10^(-7): | E1 = 10^(-9): | E1 = 10^(-12): |
| x(1) | -0.0212 | 0.47194 | -0.823203 | -1.22470750 |
| x(2) | 0.914 | 1.3096 | -0.691042 | -0.92592949 |
| x(3) | -1.215 | 0.5635 | 0.772563 | -1.82320793 |
| x(4) | -1.153 | 0.3201 | 0.742903 | 1.86797116 |
| F | 1.63249 | 1.6324812 | 1.632481236 | 1.632481236676 |
| K | 17 | 23 | 28 | 38 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

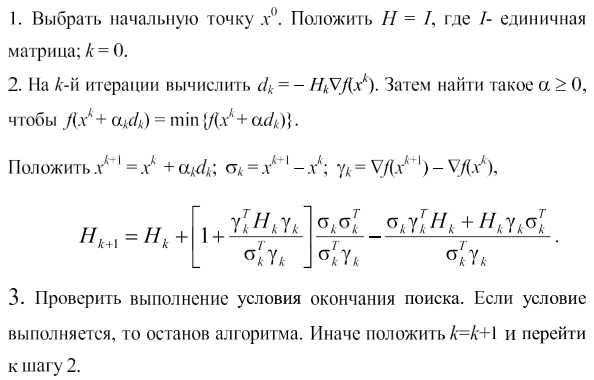
E = [10^(-5) 10^(-7) 10^(-9) 10^(-12)]

K = [17 23 28 38]

plot(E, K, 'g-');

Метод Бройдена-Флетчера-Гольдфарба-Шанно.

Алгоритм:



Программный код:

Файл NumberSeven.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-5)

E2 = 10^(-7)

E3 = 10^(-9)

E4 = 10^(-12)

X0 = (0, 0, 0)

% Функция

Y = @(x) (x(1) – x(2))^2 + 4\* (x(3) – x(4))^2 + (x(2) + 6 \* x(4))^4 + 2\* (x(1) – x(3))^2

function [x1, fn, k] = BFGS(fx, x0, E1)

gradToler = 1e-10;

XToler = E1;

k = 0;

n = length(x0);

Sm = zeros(n, m);

Ym = zeros(n, m);

[f0, g0] = feval(myFx, x0);

[alpha, f1, g1] = strongwolfe(myFx, -g0, x0, f0, g0);

x1 = x0 - alpha\*g0;

k =1;

while true

if k > maxIter

break;

end

fnorm = norm(g0);

if fnorm < gradToler

break;

end

s0 = x1 - x0;

y0 = g1 - g0;

hdiag = s0' \* y0 / (y0' \* y0);

p = zeros(length(g0), 1);

if (k <= m)

Sm(:,k) = s0;

Ym(:,k) = y0;

p = -getHg\_lbfgs(g1, Sm(:,1:k), Ym(:,1:k), hdiag);

elseif (k>m)

Sm(:,1:(m-1)) = Sm(:,2:m);

Ym(:,1:(m-1)) = Ym(:,2:m);

Sm(:,m) = s0;

Ym(:,m) = y0;

p = -getHg\_lbfgs(g1, Sm, Ym, hdiag);

end

[alpha, fs, gs] = strongwolfe(myFx, p, x1, f1, g1);

x0 = x1;

g0 = g1;

x1 = x1 + alpha \* p;

f1 = fs;

g1 = gs;

k = k + 1;

end

k = k - 1;

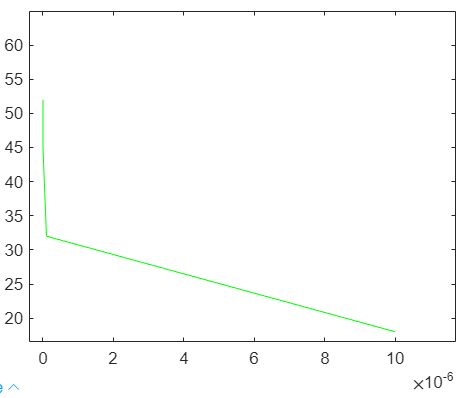
end

[xmin, fxmin] = BFGS(fx, x0, E1)

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-5): | E2 = 10^(-7): | E1 = 10^(-9): | E1 = 10^(-12): |
| x(1) | -1.693 | -0.5296 | 0.161619 | 0.68581332 |
| x(2) | 1.1154 | -0.69456 | -0.066172 | 1.22760092 |
| x(3) | -1.961 | -1.1614 | -0.500440 | 1.56436126 |
| x(4) | -1.569 | 0.86515 | 0.172948 | 1.18696399 |
| F | 1.63249 | 1.6324812 | 1.632481236 | 1.632481236676 |
| K | 18 | 32 | 45 | 52 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

E = [10^(-5) 10^(-7) 10^(-9) 10^(-12)]

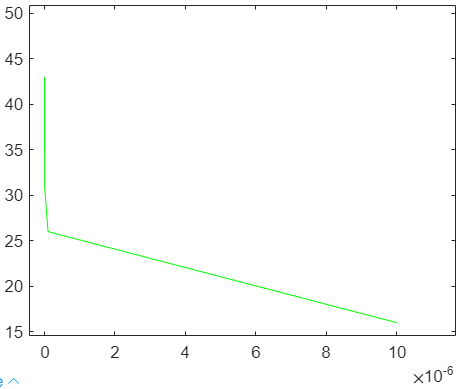
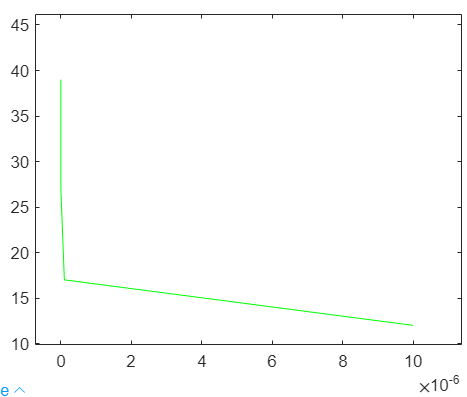
K = [18 32 45 52]

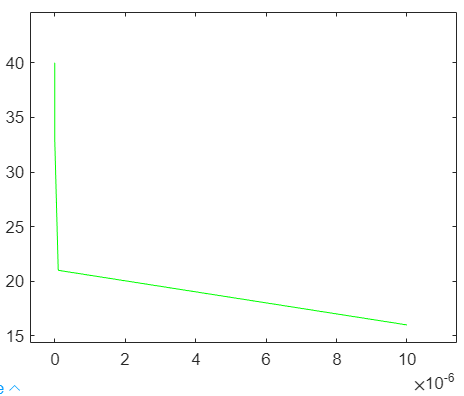
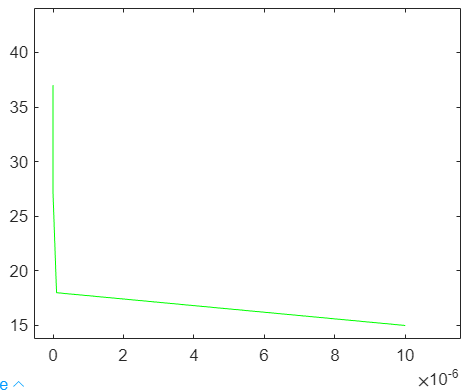
plot(E, K, 'g-');

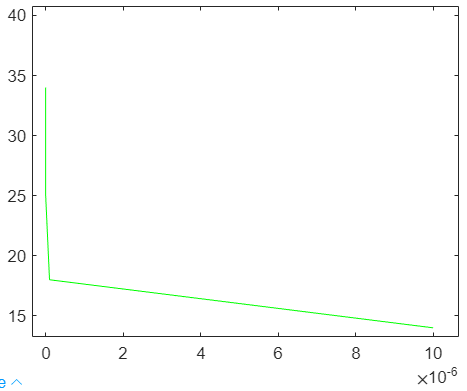
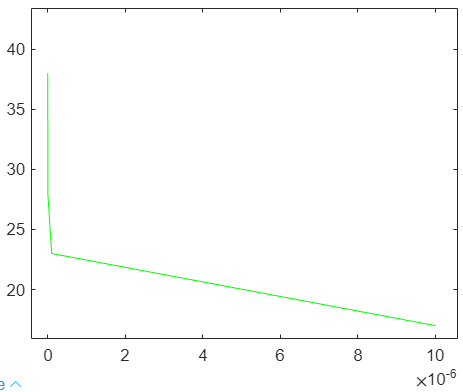
Сравнение методов:

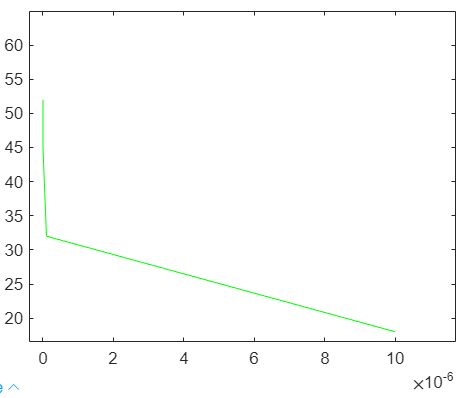
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер метода | E1 = 10^(-5)  K: | E2 = 10^(-7)  K: | E1 = 10^(-9)  K: | E1 = 10^(-12)  K: |
| Метод градиента с постоянным шагом | 16 | 26 | 31 | 43 |
| Метод наискорейшего спуска | 12 | 17 | 27 | 39 |
| Метод сопряжённого градиента для квадратичных функций | 16 | 21 | 33 | 40 |
| Метод Фленча-Ривза | 15 | 18 | 27 | 37 |
| Улучшенный метод Ньютона | 14 | 18 | 25 | 34 |
| Метод Давида-Флетчера-Пауэлла | 17 | 23 | 28 | 38 |
| Метод Бройдена-Флетчера-Гольдфарба-Шанно | 18 | 32 | 45 | 32 |

Графическое сравнение методов:

№ 1)  № 2) 

№ 3)  № 4) 

№ 5)  №6) 

№7) 

Выводы:

Самым эффективным методом поиска минимума при оказался Метод Бройдена-Флетчера-Гольдфарба-Шанно.

Почти все методы является сложнопрограммируемыми для функций нескольких переменных, в связи с чем, составить универсальную программу – затруднительно. Все формулы градиентов были высчитаны вручную.