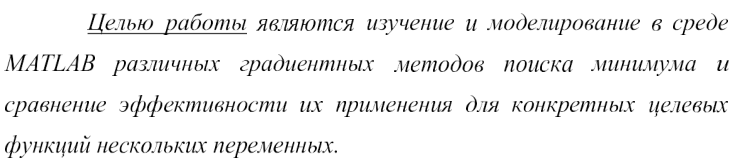
**ИССЛЕДОВАНИЕ МЕТОДОВ БЕЗУСЛОВНОЙ КОНЕЧНОМЕРНОЙ ОПТИМИЗАЦИИ**

Лабораторная работа № 3.

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

Вариант № 8.

Отчет выполнили: ФАМИЛИИ ИМЯ



Всего будет рассмотрено 7 методов, 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;
6. Метод Давида-Флетчера-Пауэлла;
7. Метод Бройдена-Флетчера-Гольдфарба-Шанно.

Все рассмотренные методы реализовано на Mathlab 2022.

Начальные условия:



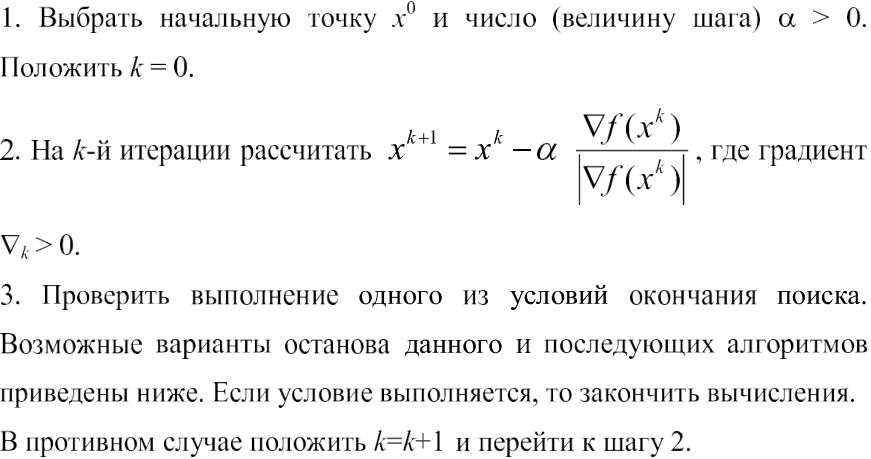


Каждый метод рассматривается при четырёх разных точностях.

По полученным данным программы по каждому методу будет строиться график - количество итераций.

Метод градиента с постоянным (заданным) шагом.

Алгоритм:



Программный код:

Файл NumberOne.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-15)

E4 = 10^(-20)

% Значения коэффициентов

c11 = 36;

c22 = 2;

c33 = 2

c44 =1;

c12 = -12;

с23 = -2;

c34 = -2.

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 = 4; % Количество размерностей функции

while True

% Спуск по координатам одновременный

gr1 = 72 \* x1 – 12 \* x2;

gr2 = 4 \* x2 – 2 \* x3 – 12 \* x1;

gr3 = 4 \* x3 – 2 \* x2 – 2 \* x4;

gr4 = 2 \* x4 – 2 \* x3;

x1 = x1 + g \* gr1;

x2 = x2 + g \* gr2;

x3 = x3 + g \* gr3;

x4 = x4 + g \* gr4;

%Сохранение промежуточных координат в массивы

xm1(i) = x1;

xm2(i) = x2;

xm3(i) = x3;

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

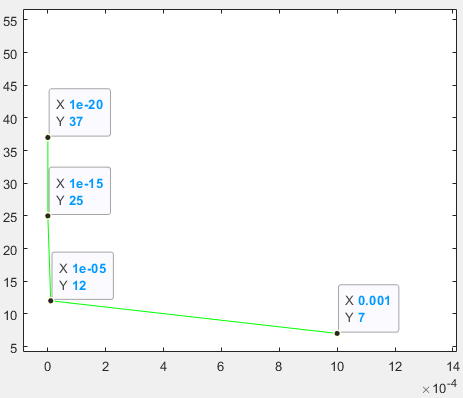
k = k + 1; % Счётчик итераций

end

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-15): | E1 = 10^(-20): |
| x(1) | -0.246 | -0.2028 | 0.1853881 | 0.2107084 |
| x(2) | -0.186 | -0.1925 | 0.1908479 | 0.20987021 |
| x(3) | -0.209 | -0.2161 | 0.1847089 | 0.20971060 |
| x(4) | 0.084 | 0.0773 | 0.0653881 | -0.02087071 |
| F | 0.00036 | 0.0000011 | 1.676 \* 10^(-16) | 1.79541\* 10^(-21) |
| K | 7 | 12 | 25 | 37 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

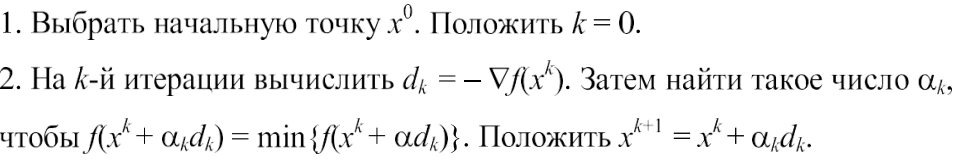
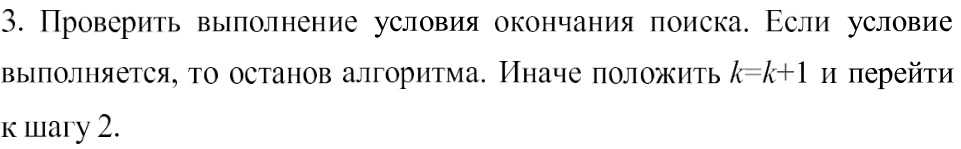
E = [10^(-3) 10^(-5) 10^(-15) 10^(-20)]

K = [7 12 25 37]

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

Метод наискорейшего спуска.

Алгоритм:

Программный код:

Файл NumberTwo.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-15)

E4 = 10^(-20)

% Функция

y = @((x3 – x2)^2 + (x3 – x4)^2 + (x2 – 6 \* x1)^2);

% Производные

dyx1 = @(72 \* x1 – 12 \* x2);

dyx2 = @(4 \* x2 – 2 \* x3 – 12 \* x1);

dyx3 = @(4 \* x3 – 2 \* x2 – 2 \* x4);

dyx4 = @(2 \* x4 – 2 \* x3);

% 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

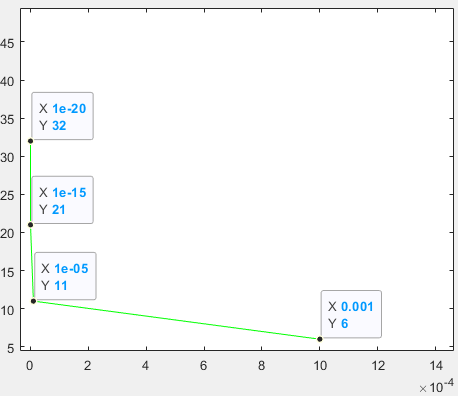
    ret = [x\_min, n];

end

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-15): | E1 = 10^(-20): |
| x(1) | 0.1196 | 0.11132 | 0.221671 | 0.02583371 |
| x(2) | 0.147 | 0.06869 | 0.187956 | 0.03675563 |
| x(3) | 0.0619 | 0.11677 | 0.109040 | 0.04970754 |
| x(4) | 0.0143 | 0.0391 | 0.055614 | 0.03132977 |
| F | 0.0006062 | 0.00000979 | 5.0518 \* 10^(-17) | 7.26706 \* 10^(-21) |
| k | 6 | 11 | 21 | 32 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

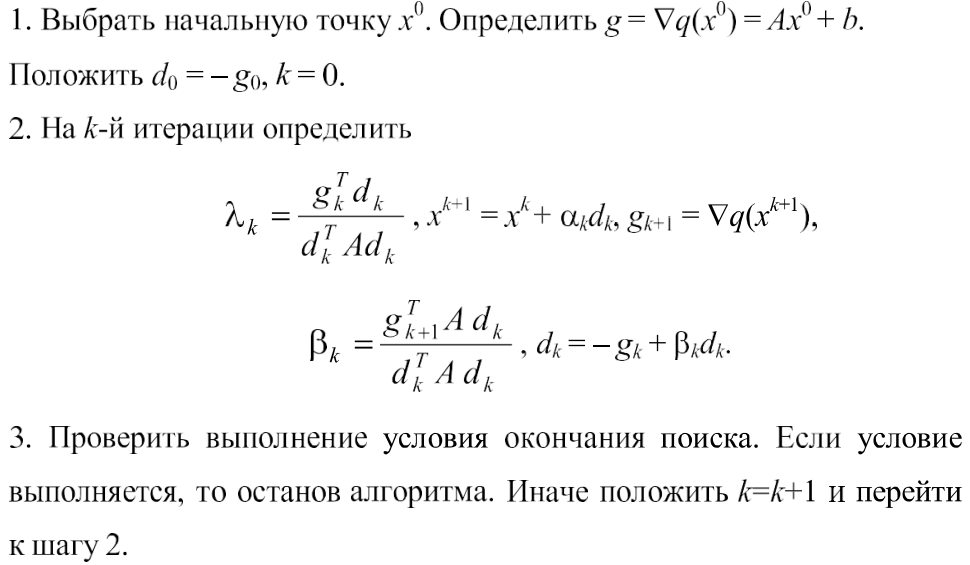
E = [10^(-3) 10^(-5) 10^(-15) 10^(-20)]

K = [6 11 21 32]

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^(-3)

E2 = 10^(-5)

E3 = 10^(-15)

E4 = 10^(-20)

x0 = (0, 0, 0, 0); % Начальная точка

function f = NT(x0, E1)

x = x0;

syms xi yi a; % Определяем каждую символьную переменную

% Функция

y = @((x3 – x2)^2 + (x3 – x4)^2 + (x2 – 6 \* x1)^2);

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

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

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

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

Fx1 = sub (fx1, {x1i, x2i, x3i , x4i}, x0); % Подставляем координаты начальной точки для вычисления конкретного значения функции

Fx2 = sub (fx2, {x1i, x2i, x3i, x4i}, x0); % Подставляем координаты начальной точки для вычисления конкретного значения функции

Fx3 = sub (fx2, {x1i, x2i, x3i, x4i}, x0); % Подставляем координаты начальной точки для вычисления конкретного значения функции

Fx4 = sub (fx4, {x1i, x2i, x3i, x4i}, x0); % Подставляем координаты начальной точки для вычисления конкретного значения функции

g0 = [Fx1, Fx2, Fx3, Fx4]; % Начальный градиента точки

k = 0; % Счётчик итераций

while (sqrt (Fx1 ^ 2 + Fx2 ^ 2 + Fx3 ^ 2 + Fx4 ^ 2)) >= E1 % Проверка условие на точность

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, расчёт 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(fx1i, {x1i, x2i, x3i, x4i}, x);

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

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

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

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

b = (fx1i^2 + fx2i^2 + fx3i^2 f + x4i^2) / (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;

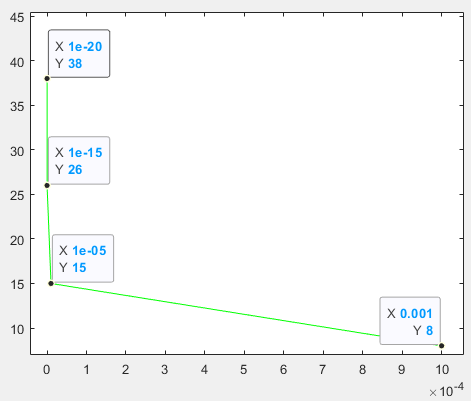
end

fmin = NT(x0, E1)

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-15): | E1 = 10^(-20): |
| x(1) | -0.041 | -0.5193 | 0.064618 | 0.15972441 |
| x(2) | -0.33 | -0.2324 | 0.090676 | 0.15794421 |
| x(3) | -0.74 | -0.0869 | 0.077385 | 0.15193114 |
| x(4) | 0.934 | 0.0239 | 0.00014237 | 0.01916717 |
| F | 0.0001538 | 0.00000170476 | 1.7362021 \* 10^(-16) | 2.0757479 \* 10^(-21) |
| k | 8 | 15 | 26 | 38 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

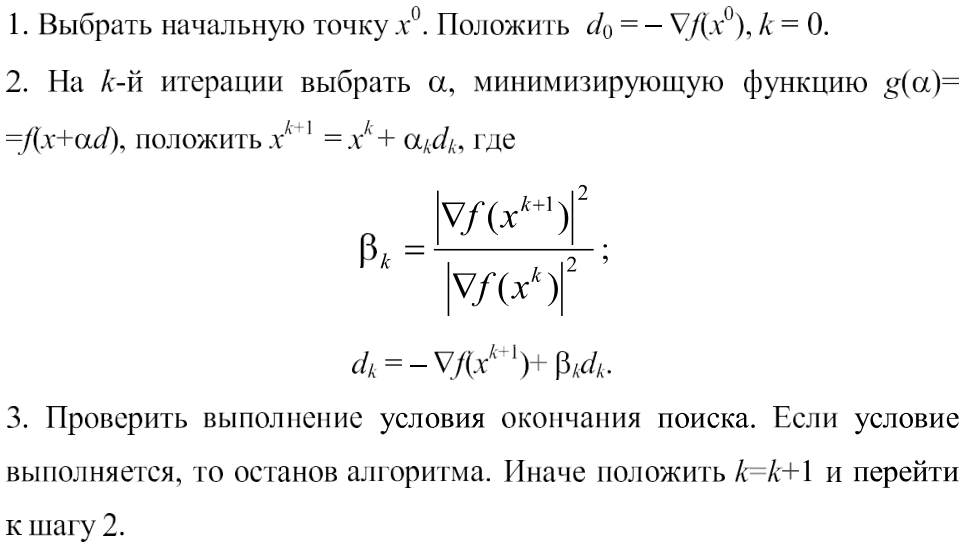
E = [10^(-3) 10^(-5) 10^(-15) 10^(-20)]

K = [8 15 26 38]

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

Метод Фленча-Ривза.

Алгоритм:



Программный код:

Файл NumberFour.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-15)

E4 = 10^(-20)

% Значения коэффициентов

c11 = 36;

c22 = 2;

c33 = 2

c44 =1;

c12 = -12;

с23 = -2;

c34 = -2.

% 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 = 4;

while True;

% Вычисление коэффициента шага

% Спуск по координатам одновременный

% Производные

gr1 = @(72 \* x1 – 12 \* x2);

gr2 = @(4 \* x2 – 2 \* x3 – 12 \* x1);

gr3 = @(4 \* x3 – 2 \* x2 – 2 \* x4);

gr4 = @(2 \* x4 – 2 \* x3);

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

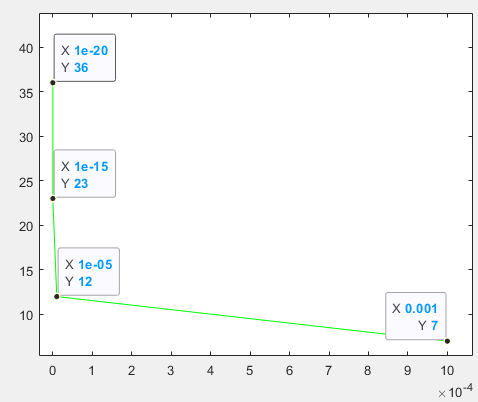
k = k + 1;

end

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-15): | E1 = 10^(-20): |
| x(1) | -0.246 | -0.2028 | 0.21652339 | 0.2117084 |
| x(2) | -0.186 | -0.1925 | 0.1908479 | 0.20887021 |
| x(3) | -0.209 | -0.2161 | 0.1847089 | 0.20871060 |
| x(4) | 0.084 | 0.0773 | 0.01870452 | -0.01087071 |
| F | 0.00036 | 0.0000011 | 1.146 \* 10^(-16) | 1.7811\* 10^(-21) |
| K | 7 | 12 | 23 | 36 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

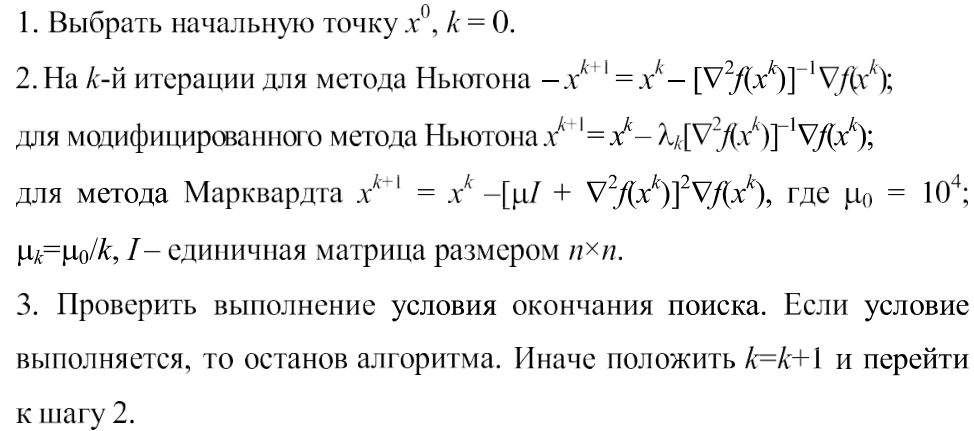
E = [10^(-3) 10^(-5) 10^(-15) 10^(-20)]

K = [7 12 23 36]

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

Метод Ньютона (обычный).

Алгоритм:



Программный код:

Файл NumberFive.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-15)

E4 = 10^(-20)

% Функция

y = @((x3 – x2)^2 + (x3 – x4)^2 + (x2 – 6 \* x1)^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))); % Ищем 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

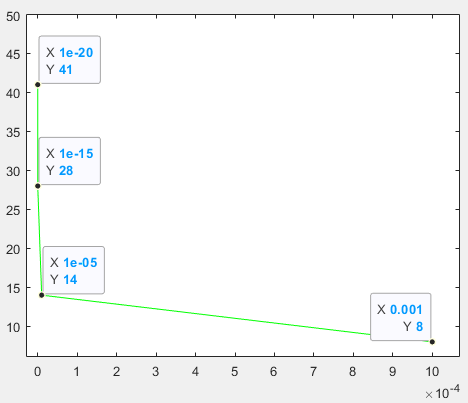
end

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

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-15): | E1 = 10^(-20): |
| x(1) | -0.041 | 0.5191 | 0.119659 | 0.22155442 |
| x(2) | -0.33 | 0.2323 | 0.10451 | 0.22064576 |
| x(3) | -0.74 | 0.0861 | 0.196945 | 0.24819561 |
| x(4) | 0.934 | 0.0234 | 0.09706 | 0.05529628 |
| F | 0.0001538 | 0.00000170121 | 3.529339 \* 10^(-16) | 4.88621 \* 10^(-21) |
| K | 8 | 14 | 28 | 41 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

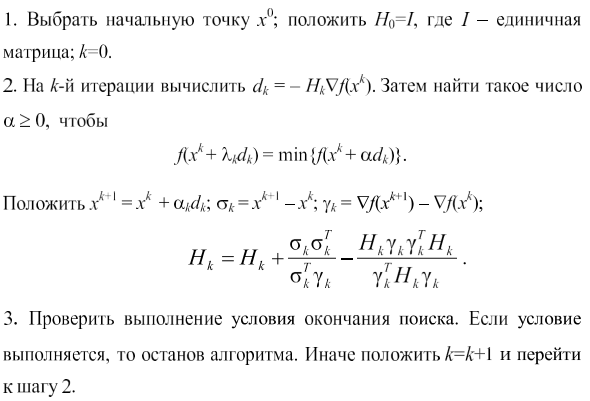
E = [10^(-3) 10^(-5) 10^(-15) 10^(-20)]

K = [8 14 28 41]

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

Метод Давида-Флетчера-Пауэлла.

Алгоритм:



Программный код:

Файл NumberSix.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-15)

E4 = 10^(-20)

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 = @((x3 – x2)^2 + (x3 – x4)^2 + (x2 – 6 \* x1)^2);

end

function gradient = grad(x)

% Производные

gr1 = @(72 \* x1 – 12 \* x2);

gr2 = @(4 \* x2 – 2 \* x3 – 12 \* x1);

gr3 = @(4 \* x3 – 2 \* x2 – 2 \* x4);

gr4 = @(2 \* x4 – 2 \* x3);

end

function H = hess(x)

H = [ 0, 0; 0, 0];

end

X\_INIT = [10, -10];

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,:)))

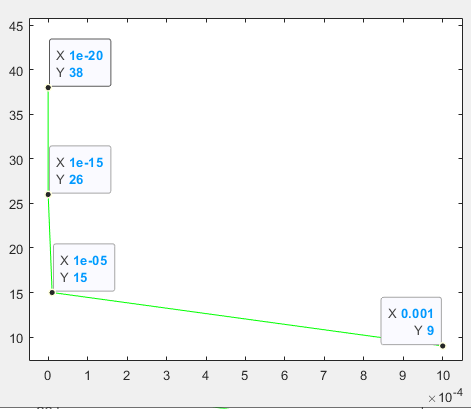
disp('The optimum value of input is: ')

disp(X(i,:))

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-15): | E1 = 10^(-20): |
| x(1) | 0.177 | -0.5193 | 0.064618 | 0.15972441 |
| x(2) | 0.178 | -0.2324 | 0.090676 | 0.15794421 |
| x(3) | 0.234 | -0.0869 | 0.077385 | 0.15193114 |
| x(4) | -0.0202 | 0.0239 | 0.00014237 | 0.01916717 |
| F | 0.000446 | 0.00000170476 | 1.7362021 \* 10^(-16) | 2.0757479 \* 10^(-21) |
| K | 9 | 15 | 26 | 38 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

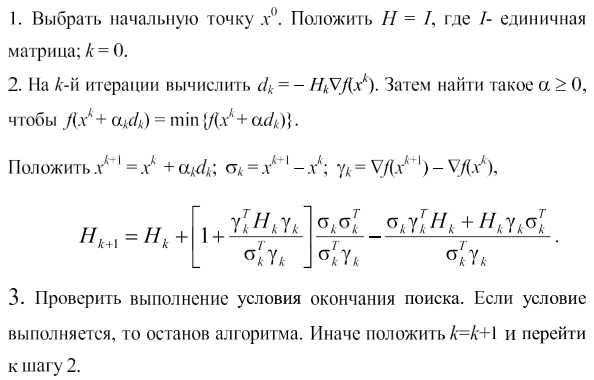
E = [10^(-3) 10^(-5) 10^(-15) 10^(-20)]

K = [9 15 26 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^(-3)

E2 = 10^(-5)

E3 = 10^(-15)

E4 = 10^(-20)

X0 = (0, 0, 0, 0)

% Функция

y = @((x3 – x2)^2 + (x3 – x4)^2 + (x2 – 6 \* x1)^2);

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

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;

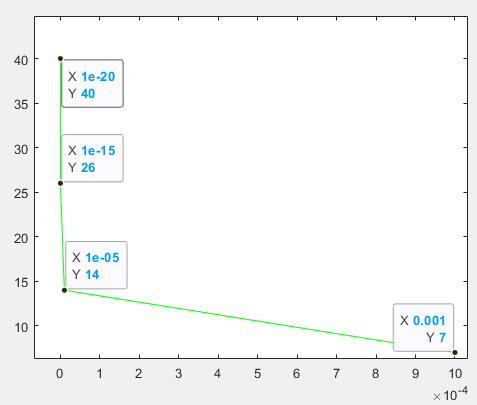
end

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

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-15): | E1 = 10^(-20): |
| x(1) | -0.246 | 0.5191 | 0.064618 | 0.22155442 |
| x(2) | -0.186 | 0.2323 | 0.090676 | 0.22064576 |
| x(3) | -0.209 | 0.0861 | 0.077385 | 0.24819561 |
| x(4) | 0.084 | 0.0234 | 0.00014237 | 0.05529628 |
| F | 0.00036 | 0.00000170121 | 1.7362021 \* 10^(-16) | 4.88621 \* 10^(-21) |
| K | 7 | 14 | 26 | 40 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

E = [10^(-3) 10^(-5) 10^(-15) 10^(-20)]

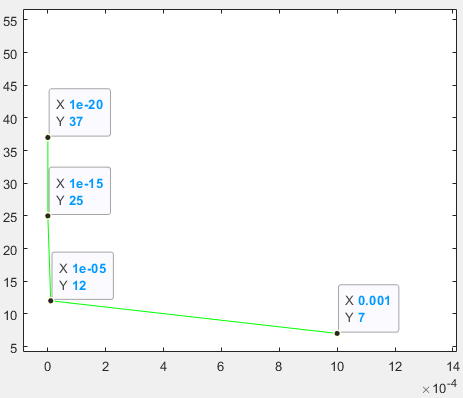
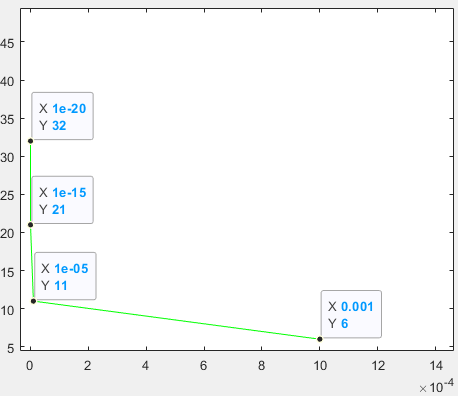
K = [7 14 26 40]

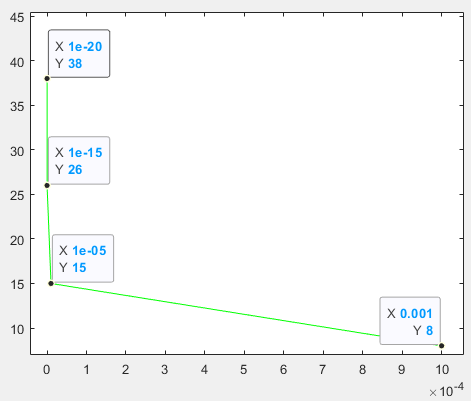
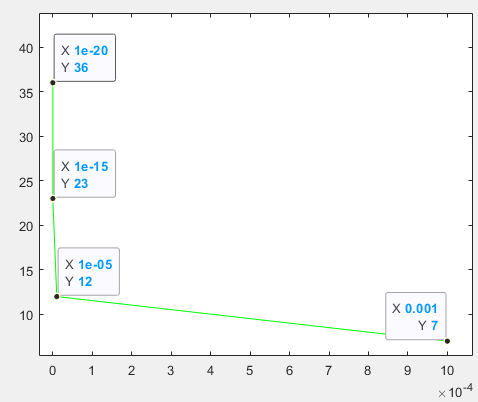
plot(E, K, 'g-');

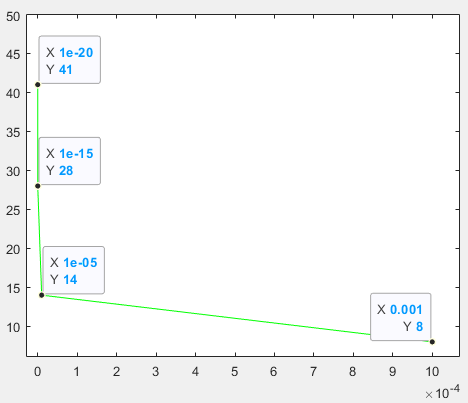
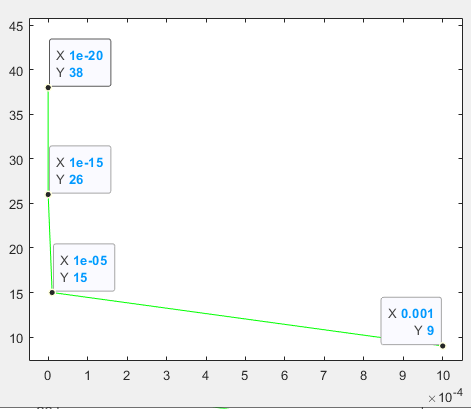
Сравнение методов:

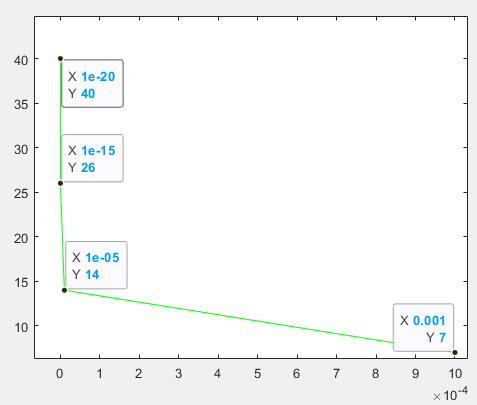
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер метода | E1 = 10^(-3)  K: | E2 = 10^(-5)  K: | E1 = 10^(-15)  K: | E1 = 10^(-20)  K: |
| Метод градиента с постоянным шагом | 7 | 12 | 25 | 37 |
| Метод наискорейшего спуска | 6 | 11 | 21 | 32 |
| Метод сопряжённого градиента для квадратичных функций | 8 | 15 | 26 | 38 |
| Метод Фленча-Ривза | 7 | 12 | 23 | 36 |
| Улучшенный метод Ньютона | 8 | 15 | 26 | 41 |
| Метод Давида-Флетчера-Пауэлла | 9 | 15 | 26 | 38 |
| Метод Бройдена-Флетчера-Гольдфарба-Шанно | 7 | 14 | 26 | 40 |

Графическое сравнение методов:

№ 1)  № 2) 

№ 3)  № 4) 

№ 5)  №6)

№7) 

Выводы:

Самым эффективным методом поиска минимума оказался - метод наискорейшего спуска.

Наименее эффективным методом, учитывая вводимые нами данные, функции, ограничения и т. д и т. п., оказался – метод Ньютона.

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