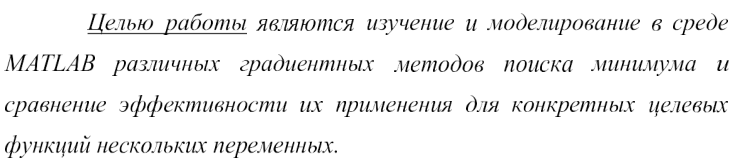
**ИССЛЕДОВАНИЕ МЕТОДОВ БЕЗУСЛОВНОЙ КОНЕЧНОМЕРНОЙ ОПТИМИЗАЦИИ**

Лабораторная работа № 3.

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

Вариант № 4.

Отчет выполнили: Гвазджяускас В.В.​, Седов А.О., Фадеев О.И.



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:





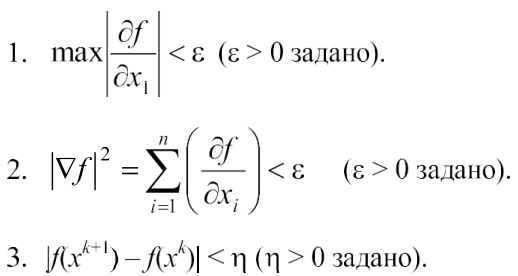
Наше начальное уравнение эквивалентно:

x(1)^2 + x(2)^2 + x(3)^3 – 2 \* x(3) \* x(1) – 4 \* x(2) + 4 – данный вид удобный для расчёта алгоритмов.

Каждый метод рассматривается при четырёх разных точностях.

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

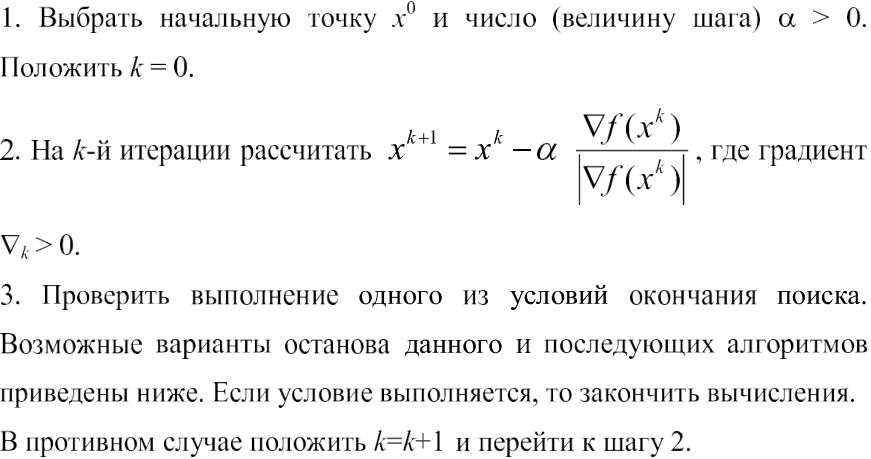
Для начала укажем критерии для определения окончания процесса вычислений:



Теперь перейдём к разбору методов.

Метод градиента с постоянным (заданным) шагом.

Алгоритм:



Программный код:

Файл GradConst.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-7)

E4 = 10^(-10)

% Значения коэффициентов

c11 = 1;

c22 = 1;

c33 = 1;

c13 = -2;

c2 = -4;

c = 4;

g = 0.1; % постоянная шага

%Начальная точка

x1 = 0;

x2 = 0;

x3 = 0;

k = 1; % Счетчик шагов

kmax = 1000; % Предельное число шагов задается для предотвращения зацикливания

%Массивы для хранения промежуточных координат

xm1 = [x1];

xm2 = [x2];

xm3 = [x3];

i = 2;

while k < kmax

% Спуск по координатам одновременный

gr1 = 2\*c11\*x1 + c13\*x3; % y

gr2 = 2\*c22\*x2 + c2; % z

gr3 = 2\*c33\*x3 + c13\*x1; % x

x1 = x1 + g\*gr1;

x2 = x2 + g\*gr2;

x3 = x3 + g\*gr3;

%Сохранение координат

xm1(i) = x1;

xm2(i) = x2;

xm3(i) = x3;

i = i + 1;

%Проверка условия останова

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

break; % Выход из цикла в случае выполнения условия

end

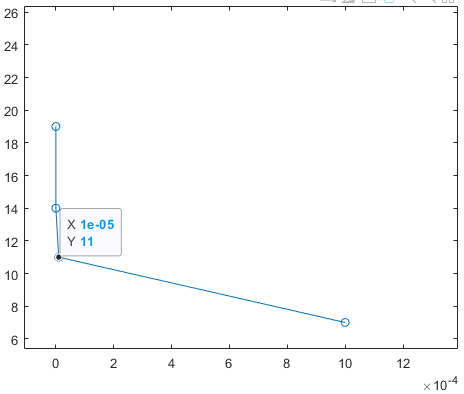
k = k + 1; % Счётчик итераций

end

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-7): | E1 = 10^(-10): |
| x(1) | 4.7e-03 | 6.1e-04 | 8.2e-06 | 7.6e-05 |
| x(2) | 2.0094 | 2.0000484 | 2.000661 | 2.0000001 |
| x(3) | 6.9e-03 | 1.477e-04 | 7.41e-06 | 6.82e-05 |
| F | 8.3e-04 | 2.3e-06 | 9.71e-08 | 1.42e-11 |
| k | 7 | 11 | 14 | 19 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

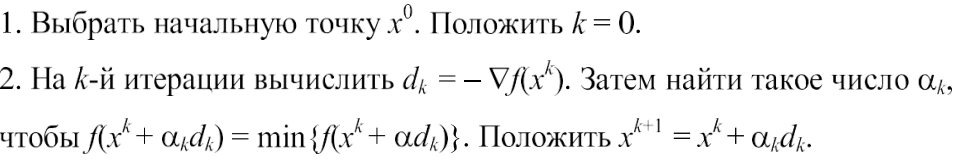
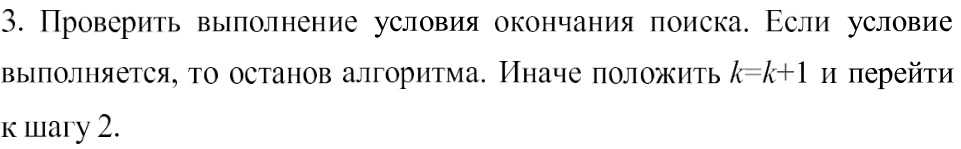
E = [10^(-3) 10^(-5) 10^(-7) 10^(-10)]

K = [7 11 14 19]

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

Метод наискорейшего спуска.

Алгоритм:

Программный код:

Файл FastSol.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-7)

E4 = 10^(-10)

clc

clear all

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-7)

E4 = 10^(-10)

% Функция

y=@(x)(x(3) - x(1))^2 + (2 - x(2));

% Производные

dyx1 = @(x)2 \* x(1) - 2 \* x(3);

dyx2 = @(x)2 \* x(2) - 4;

dyx3 = @(x)2 \* x(3) - 2 \* x(1);

% Метод наискорейшего спуска

a = 1;

x = [0, 0, 0];

k = 0;

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

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

local\_min = dihotomy(f, 0, 3, E1); % Поиск локального минимума методом дихотомии

a = local\_min(1);

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

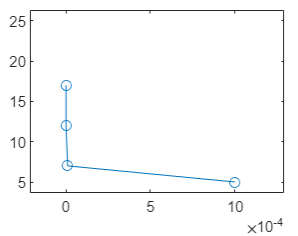
    ret = [x\_min, n];

end

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-7): | E1 = 10^(-10): |
| x(1) | 1.2e-03 | 4e-04 | 2.28e-05 | 5.121e-5 |
| x(2) | 1.982 | 1.9993 | 2.000147 | 2.000011 |
| x(3) | 9.67e-03 | 6.5e-04 | 1.134e-04 | 4.715e-5 |
| F | -3.2065e-04 | -4.28e-06 | -1.3401e-08 | 6.481e-11 |
| k | 5 | 7 | 12 | 17 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

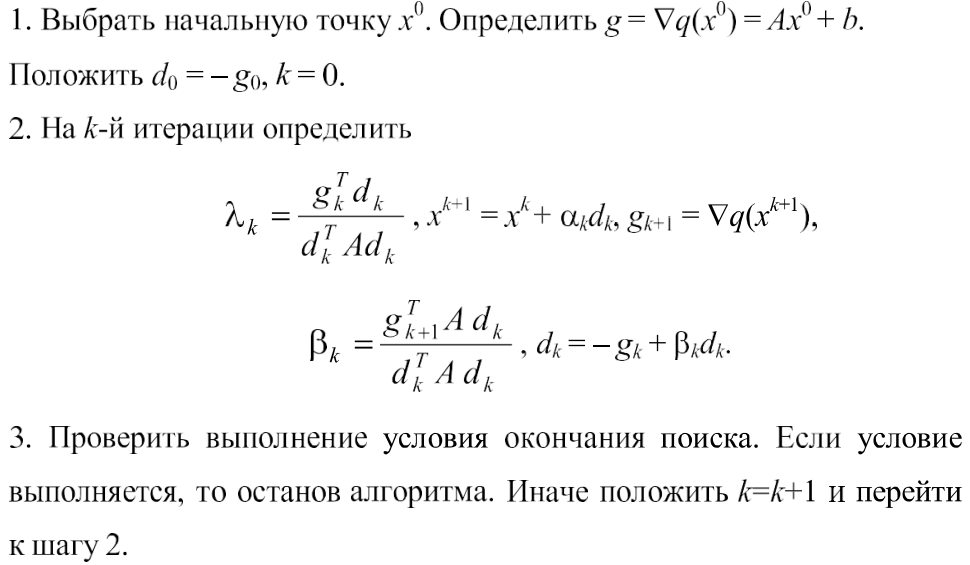
E = [10^(-3) 10^(-5) 10^(-7) 10^(-10)]

K = [5 7 12 17]

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

Метод сопряжённого градиента для квадратичных функций.

Алгоритм:



Программный код:

Файл GD.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-7)

E4 = 10^(-10)

x0 = (0, 0, 0); % Начальная точка

function f = GR(x0, E1)

x = x0;

syms xi yi a; % Определяют каждую символьную переменную

f = (x(3) - x(1))^2 + (2 - x(2))^2; % Наша функция

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

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

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

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

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

k = 0; % Счётчик итераций

while (sqrt (fx1 ^ 2 + fx2 ^ 2 + fx3 ^ 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}, 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);

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

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

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

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

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

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;

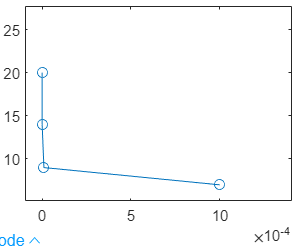
end

fmin = GR(x0, E1)

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-7): | E1 = 10^(-10): |
| x(1) | 1.7e-02 | 5.08e-04 | 3.32e-07 | 9.1e-08 |
| x(2) | 1.981 | 1.9941 | 1.9997 | 1.999998 |
| x(3) | 3.7e-03 | 8.6e-04 | 1.5e-08 | 8.7e-08 |
| F | 14.338e-03 | 8.47e-06 | 2.91e-08 | 4.84e-11 |
| k | 7 | 9 | 14 | 20 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

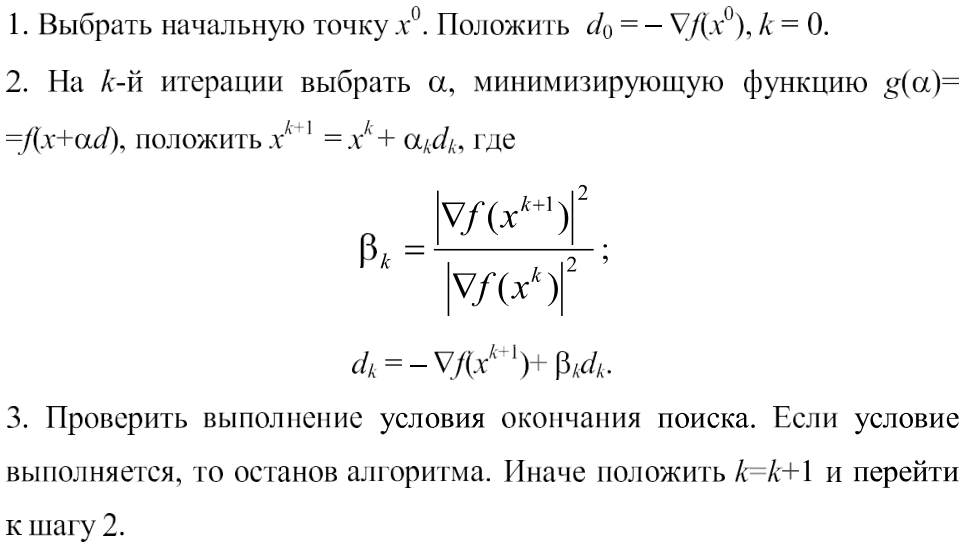
E = [10^(-3) 10^(-5) 10^(-7) 10^(-10)]

K = [7 11 14 19]

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

Метод Фленча-Ривза.

Алгоритм:



Программный код:

Файл Rivz.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-7)

E4 = 10^(-10)

% Значения коэффициентов

c11 = 1;

c22 = 1;

c33 = 1;

c13 = -2

c2 = -4

c = 4

% Начальная точка

x1 = 0;

x2 = 0;

x3 = 0

k = 1; % Счетчик шагов

%Массивы для хранения промежуточных координат

x1trace = [x1];

x2trace = [x2];

x3trace = [x3];

i = 2;

while True;

% Вычисление коэффициента шага

gr1 = 2\*x1 + c13\*x3;

gr2 = 2\*x2 - 4;

gr3 = 2\*x3 + c13\*x1;

g = -(gr1^2 + gr2^2 + gr3^2) / (2\*c1\*gr1^2 + 2\*c2\*gr2^2 + 2\*c12\*gr1\*gr2);

Bk = abs((gr1 + gr2 + gr)^2 / abs(x1 + x2 + x3))

x1 = Bk - g\*gr1;

x2 = Bk - g\*gr2;

x3 = Bk - g\*gr3

%Сохранение координат

x1trace(i) = x1;

x2trace(i) = x2;

x3trace(i) = x3;

i = i + 1;

%Проверка условия останова

if sqrt(gr1^2 + gr2^2 + gr3^3) <= E1;

break;

%Выход из цикла в случае выполнения условия

end

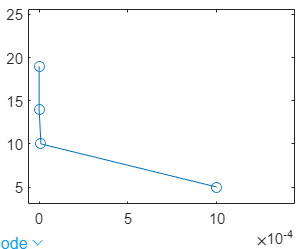
k = k + 1;

end

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-7): | E1 = 10^(-10): |
| x(1) | 7.5e-02 | 4.15e-04 | 7.9e-10 | 8.45e-08 |
| x(2) | 2.004 | 2 | 1.999997 | 2.0000144 |
| x(3) | 5.3e-03 | 6.27e-04 | 7.5e-09 | 1.7e-09 |
| F | 1.1e-04 | 9.17e-06 | 3.99e-08 | 4.54e-11 |
| k | 5 | 10 | 14 | 19 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

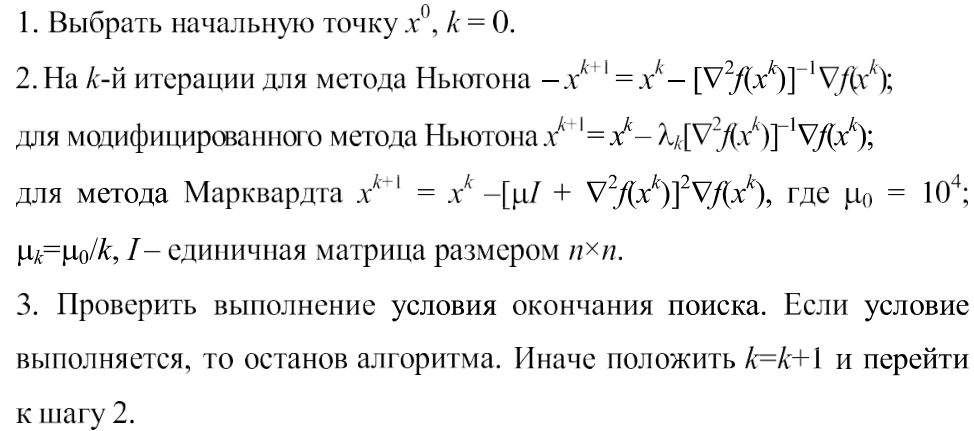
E = [10^(-3) 10^(-5) 10^(-7) 10^(-10)]

K = [5 10 14 19]

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

Метод Ньютона (обычный).

Алгоритм:



Программный код:

Файл Newton.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-7)

E4 = 10^(-10)

y=@(x)(x(3) - x(1))^2 + (2 - x(2)); % Наша функция

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

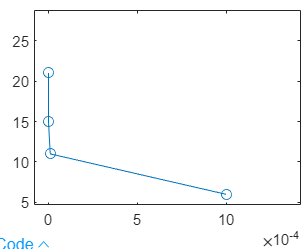
end

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

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-7): | E1 = 10^(-10): |
| x(1) | 0.9410 | 6.7e-02 | 4.9e-08 | 3.84-e11 |
| x(2) | 2 | 1.997 | 1.9999981 | 2.0000007 |
| x(3) | 0.9347 | 7.5e-03 | 8.94e-09 | 3.81e-11 |
| F | 4.8e-04 | 8.6e-06 | 5.73e-08 | 8.1e-11 |
| k | 6 | 11 | 15 | 21 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

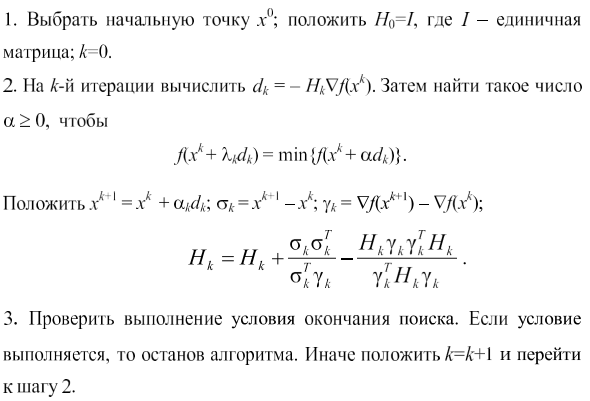
E = [10^(-3) 10^(-5) 10^(-7) 10^(-10)]

K = [7 11 14 19]

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

Метод Давида-Флетчера-Пауэлла.

Алгоритм:



Программный код:

Файл DavidF.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-7)

E4 = 10^(-10)

num\_var = 2;

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

Файд func.m (функци):

function objective\_fn = f(x)

objective\_fn = (x(3) – x(1))^2 + (2 – x(2));

end

Файл grad.m (вспомогательная функция):

function gradient = grad(x)

gradient = [6\*x(1); 8\*x(2)];

end

Файл hess.m (вспомогательный файл):

function H = hess(x)

H = [ 6, 0; 0, 8];

end

Файл main.m (запуск):

X\_INIT = [5, -2];

ALPHA = 0.01;

MAX\_ITER = 50;

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,:)))

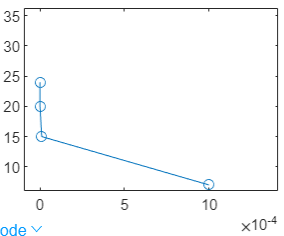
disp('The optimum value of input is: ')

disp(X(i,:))

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-7): | E1 = 10^(-10): |
| x(1) | 1.5 | 8.2e-03 | 5.77e-05 | 7.34e-04 |
| x(2) | 2 | 1.99914 | 1.999954 | 2.0000002 |
| x(3) | 1.52 | 9.96e-02 | 5.9e-05 | 6.88e-05 |
| F | 7.4e-04 | 9.57e-06 | 4.62e-08 | 2.39e-12 |
| k | 7 | 15 | 20 | 24 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

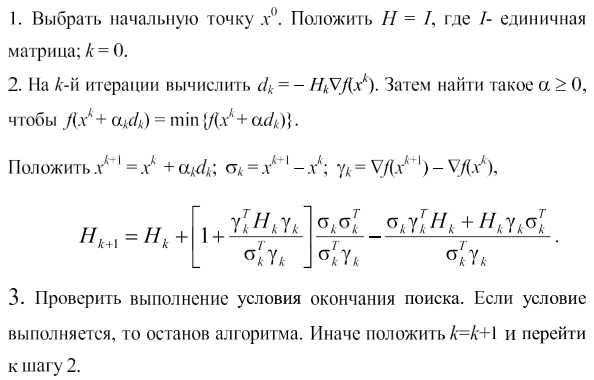
E = [10^(-3) 10^(-5) 10^(-7) 10^(-10)]

K = [7 15 20 24]

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

Метод Бройдена-Флетчера-Гольдфарба-Шанно.

Алгоритм:



Программный код:

Файл Bro.m:

clear all % Очищаем всю область и переменные

% Точности

E1 = 10^(-3)

E2 = 10^(-5)

E3 = 10^(-7)

E4 = 10^(-10)

X0 = (0, 0, 0)

f = (x(3) – x(1))^2 + (2 – x(2))

function [x1,f1,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;

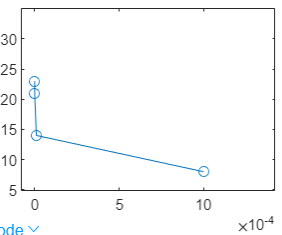
end

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

Значения функции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | E1 = 10^(-3): | E2 = 10^(-5): | E1 = 10^(-7): | E1 = 10^(-10): |
| x(1) | 9.17e-02 | 2.04e-03 | 6.27e-05 | 4.15e-06 |
| x(2) | 2.0001 | 2.000008 | 1.99993 | 2 |
| x(3) | 9.42e-02 | 4.14e-03 | 8.45e-05 | 7.05e-06 |
| F | 8.8e-04 | 1.5e-06 | 6e-08 | 4.5e-11 |
| k | 8 | 14 | 21 | 23 |

Графики зависимостей количества итераций от точности решений:



Код построения графиков:

clear all

E = [10^(-3) 10^(-5) 10^(-7) 10^(-10)]

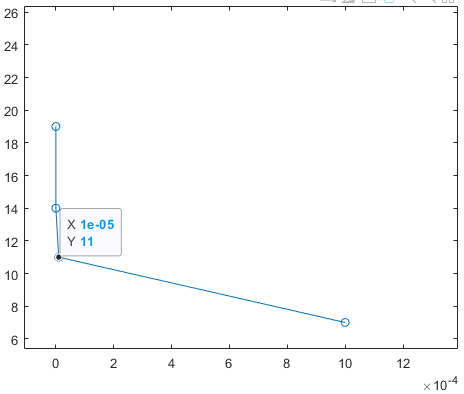
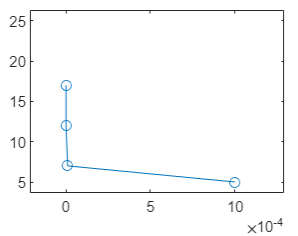
K = [8 14 21 23]

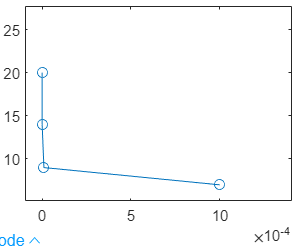
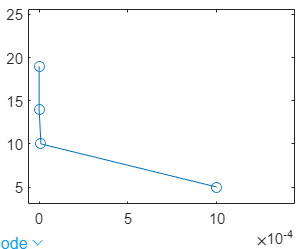
plot(E, K,'o-');

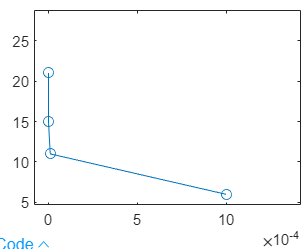
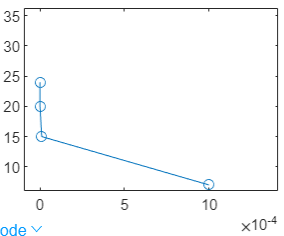
Сравнение методов:

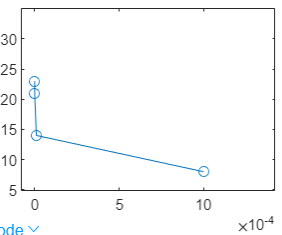
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер метода | E1 = 10^(-3)  K: | E2 = 10^(-5)  K: | E1 = 10^(-7)  K: | E1 = 10^(-10)  K: |
| 1 | 7 | 11 | 14 | 19 |
| 2 | 5 | 7 | 12 | 17 |
| 3 | 7 | 9 | 14 | 20 |
| 4 | 5 | 10 | 14 | 19 |
| 5 | 6 | 11 | 15 | 21 |
| 6 | 7 | 15 | 20 | 24 |
| 7 | 8 | 14 | 21 | 23 |

Графическое сравнение:

№ 1)  № 2) 

№ 3) № 4) 

№ 5)  №6) 

№7) 

Выводы:

Самым эффективным методом поиска минимума оказался метод наискорейшего спуска, что подтверждает таблица итераций методов (см. выше).

Также нужно отметить, что если функция заранее определена, то метод лёгкопрограммируем.