**Покоординатный спуск:**

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <math.h>*

*#define sqr(x) ((x)\*(x)) // макрос для квадрата числа*

*// Объявление функции, которую мы хотим минимизировать*

*double f(double\*);*

*int main(){*

*// Определение переменных, включая массивы для хранения*

*// точек x и их производных df, шага h и точности eps*

*double \*x, \*df, h, eps = 0.0001, f0, df1, s, grad, dx = 0.0001;*

*int i, m, t, n = 2;*

*// Выделение памяти под массивы x и df*

*x = (double\*) malloc(sizeof(double)\*n);*

*df = (double\*) malloc(sizeof(double)\*n);*

*printf("Величина точности eps установлена равной = %lf\n",eps);*

*// Запрашиваем у пользователя шаг h*

*printf("Введите величину шага h = ");*

*scanf("%lf",&h);*

*// Запрашиваем у пользователя максимальное число итераций m*

*printf("Введите предельное число итераций m = ");*

*scanf("%d",&m);*

*// Запрашиваем у пользователя начальные значения для каждого x[i]*

*puts("Введите начальное значение переменных");*

*for(i=0;i<n;i++){*

*printf("x[%d] = ",i+1);*

*scanf("%lf",&x[i]);*

*}*

*t = 0;*

*// Начало цикла градиентного спуска*

*do{*

*f0 = f(x); // Значение функции в текущей точке*

*s = 0;*

*t++;*

*// Расчет градиента функции в точке x*

*for(i=0;i<n;i++){*

*x[i] += dx;*

*df[i] = (f(x)-f0)/dx;*

*x[i] -= dx;*

*}*

*// Движение в направлении антиградиента*

*for(i=0;i<n;i++){*

*x[i] -= h\*df[i];*

*}*

*df1 = f(x)-f0;*

*// Если функция увеличивается, уменьшаем шаг*

*if(df1 > 0) h /= 2;*

*// Расчет нормы градиента*

*for(i=0;i<n;i++){*

*s += sqr(df[i]);*

*}*

*grad = sqrt(s);*

*} while(grad > eps && t <= m); // Повторяем, пока градиент больше eps и число итераций меньше m*

*// Проверяем, достигнуто ли решение*

*if(grad < eps && t <= m){*

*puts("Результаты оптимизации");*

*// Выводим результаты оптимизации*

*for(i=0;i<n;i++){*

*printf("\nx[%d] = %5.10lf",i+1,x[i]);*

*}*

*printf("\nзначение функции цели = %5.10lf",f0);*

*}*

*else{*

*// Выводим сообщение об ошибке, если решение не было найдено*

*puts("Решение не найдено");*

*puts("Уменьшите/увеличите величину шага или увеличьте предельное число итераций");*

*}*

*fflush(stdin);*

*getchar();*

*return 0;*

*}*

*// Определение функции, которую мы минимизируем*

*double f(double \* x){*

*return 4 \* pow(x[0], 2) + 2 \* x[0] \* x[1] + 5 \* pow(x[1], 2) + 9 \* x[0];*

*}*

**Ньютон-рафсон:**

***#include <stdio.h>***

***#include <stdlib.h>***

***#include <math.h>***

***#define sqr(x) ((x)\*(x))***

***const double dx = 0.0001;***

***double \*df,\*\*d2f;***

***double f(double \* x){***

***return 4 \* pow(x[0], 2) + 2 \* x[0] \* x[1] + 5 \* pow(x[1], 2) + 9 \* x[0];***

***}***

***void invert(int n, int &q);***

***void for\_mat\_d2f(int n, double &grad, double &h, double \*x);***

***int main(){***

***int i,j,n = 2, q;***

***double eps = 0.0001, grad, s, h, \*x;***

***// Выделение памяти для переменных и матриц***

***d2f = (double\*\*)malloc(sizeof(double\*)\*n);***

***for(i=0;i<n;i++)***

***d2f[i] = (double\*)malloc(sizeof(double)\*n);***

***x = (double\*) malloc(n\*sizeof(double));***

***df = (double\*) malloc(n\*sizeof(double));***

***// Инициализируем начальное значение переменных***

***x[0] = 1.0;***

***x[1] = 1.0;***

***// Цикл Ньютона для оптимизации функции***

***do{***

***for\_mat\_d2f(n, grad, h,x); // Вычисление производных и шага***

***invert(n,q); // Определение обратной матрицы вторых производных***

***if(q)***

***{***

***puts("Определитель равен нулю"); // Если матрица не обратима, завершаем программу***

***fflush(stdin);***

***getchar();***

***return 0;***

***}***

***for(i=0;i<n;i++) // Обновление переменных для следующего шага***

***{***

***s=0;***

***for(j=0;j<n;j++)***

***s+= d2f[i][j]\*df[j];***

***x[i] -= h\*s;***

***}***

***} while (grad >= eps); // Продолжаем до тех пор, пока норма градиента не станет меньше eps***

***// Вывод результатов оптимизации***

***puts("\nРезультаты оптимизации:\n");***

***for(i=0;i<n;i++)***

***printf("x[%d]=%5.10lf\n",i+1,x[i]);***

***printf("Значение функции цели = %5.10lf",f(x));***

***// Освобождение выделенной памяти***

***for(i=0;i<n;i++)***

***free(d2f[i]);***

***free(d2f);***

***free(x);***

***free(df);***

***fflush(stdin);***

***getchar();***

***return 0;***

***}***

***// Эта функция пытается найти обратную матрицу для матрицы вторых производных. Если определитель равен нулю (матрица необратима), q меняется на 1***

***void invert(int n, int &q)***

***{***

***double \*\*a,t;***

***int i,j,k,m;***

***m = 2\*n;***

***a = (double\*\*)malloc(sizeof(double\*)\*n);***

***for(i=0;i<n;i++)***

***a[i] = (double\*)malloc(sizeof(double)\*m);***

***q = 0;***

***for(i=0;i<n;i++)***

***for(j=0;j<m;j++)***

***if (j<n) a[i][j] = d2f[i][j];***

***else***

***if (j ==n+i) a[i][j]=1;***

***else a[i][j]=0;***

***for(i=0;i<n;i++){***

***k=i;***

***do{***

***if(!a[k][i])***

***{***

***q=1;***

***if(k < n-1) k++;***

***else return;***

***}***

***} while(q);***

***if(q)***

***for(j=0;j<m;j++)***

***{***

***t=a[k][j];***

***a[k][j]=a[i][j];***

***a[i][j]=t;***

***}***

***for(j=m-1;j>=i;j--) a[i][j] /= a[i][i];***

***for(k=0;k<n;k++)***

***if(k!=i)***

***for(j=m-1;j>=0;j--)***

***a[k][j] -= a[i][j]\*a[k][j];***

***}***

***q=0;***

***for(i=0;i<n;i++)***

***for(j=0;j<n;j++)***

***d2f[i][j]=a[i][j+n];***

***for(i=0;i<n;i++)***

***free(a[i]);***

***free(a);***

***}***

***// Эта функция вычисляет первые производные, вторые производные и смешанные производные для функции, а также норму градиента (grad) и шаг h.***

***void for\_mat\_d2f(int n,double &grad, double &h, double\*x)***

***{***

***double s,s1,f0;***

***int i,j;***

***//расчет первых производных***

***f0=f(x); //s=0;***

***for(i=0;i<n;i++)***

***{***

***x[i] += dx;***

***df[i]=(f(x)-f0)/dx;***

***s = sqr(df[i]);***

***x[i] -=dx;***

***}***

***grad = sqrt(s);***

***//расчет вторых производных***

***for(i=0; i<n;i++)***

***{***

***s=-2\*f(x);***

***x[i] += dx;***

***s += f(x);***

***x[i] -=2\*dx;***

***s += f(x);***

***x[i] += dx;***

***d2f[i][i] = s/sqr(dx);***

***}***

***//расчет смешанных производных***

***for(i=0;i<n-1;i++)***

***for(j=i+1;j < n; j++)***

***{***

***s=f(x);***

***x[i] -= dx; x[j] -=dx;***

***s += f(x);***

***x[j] += dx;***

***s -= f(x);***

***x[i] += dx; x[j]-= dx;***

***s-= f(x);***

***x[j] += dx;***

***d2f[i][j] = s/sqr(dx);***

***d2f[j][i] = d2f[i][j];***

***}***

***s=0; s1= 0;***

***for(i=0;i<n;i++)***

***s+=sqr(df[i]);***

***for(i=0;i<n;i++)***

***for(j=0;j<n;j++)***

***s1+= d2f[i][j]\*df[i]\*df[j];***

***h=s/s1; grad = sqrt(s);***

***}***

**Симплекс:**

*#include <iostream>*

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <string.h>*

*#include <math.h>*

*#define sqr(x) ((x)\*(x))*

*double f(double\*);*

*int Knum(const char\*);*

*void Fsi(double\*);*

*int n;*

*double \*\*s; // Глобальные переменные; s - двумерный массив для хранения вектор-столбцов симплекса и значений функции f в вершинах симплекса.*

*int main() {*

*double \*x, \*xc, t, d1, d2, eps, m;*

*int k, i, j;*

*n = 2;*

*printf("Размерность оптимизации = 2");*

*// printf("Введите размерность оптимизации n = ");*

*// scanf("%d", &n);*

*// создаем память для указателей x и xc*

*x = (double\*)malloc(sizeof(double) \* n);*

*xc = (double\*)malloc(sizeof(double) \* n);*

*// создаем память для массива симплекса*

*s = (double\*\*)malloc((n + 1) \* sizeof(double\*));*

*// создаем память для строк массива s*

*for (i = 0; i <= n; i++) {*

*s[i] = (double\*)calloc(n + 1, sizeof(double));*

*}*

*eps = 0.0001;*

*printf("\nТочность вычисления = 0.0001 ");*

*// printf("Введите точность вычисления eps = ");*

*// scanf("%lf", &eps);*

*printf("\nВведите начальную длину ребра симплекса m = ");*

*scanf("%lf", &m);*

*puts("\nВведите координаты начальной точки симплекса");*

*for (j = 0; j < n; j++) {*

*printf("x[%d]=", j + 1);*

*scanf("%lf", &s[1][j]);*

*}*

*//начальный симплекс*

*d1 = ((sqrt(n + 1) - 1) / (n \* sqrt(2))) \* m;*

*d2 = ((sqrt(n + 1) + n - 1) / (n \* sqrt(2))) \* m;*

*// инициализация начальной пространственной точки*

*for (i = 1; i <= n; i++)*

*for (j = 0; j < n; j++)*

*if (i == j + 1) s[i][j] = s[0][j] + d1;*

*else s[i][j] = s[0][j] + d2;*

*Fsi(x); // Вычисление значений целевой функции*

*do {*

*k = Knum("max");*

*for (j = 0; j < n; j++) {*

*t = 0;*

*for (i = 0; i <= n; i++)*

*if (i != k) t += s[i][j];*

*xc[j] = t / n;*

*}*

*for (j = 0; j < n; j++)*

*x[j] = 2 \* xc[j] - s[k][j];*

*t = f(x);*

*if (t < s[k][n]) {*

*for (j = 0; j < n; j++) s[k][j] = x[j];*

*s[k][n] = t;*

*goto M1;*

*}*

*k = Knum("min");*

*for (j = 0; j < n; j++)*

*for (i = 0; i <= n; i++)*

*if (i != k) s[i][j] = 0.5 \* (s[k][j] + s[i][j]);*

*Fsi(x);*

*M1:*

*for (j = 0; j < n; j++) {*

*t = 0;*

*for (i = 0; i <= n; i++) t += s[i][j];*

*xc[j] = t / (n + 1);*

*}*

*t = f(xc);*

*k = 0;*

*for (i = 0; i <= n; i++)*

*if (fabs(t - s[i][n]) < eps) k++;*

*} while (k != n + 1);*

*k = Knum("min");*

*printf("\nРезультаты оптимизации\n");*

*for (j = 0; j < n; j++)*

*printf("\nx[%d] = %lf", j + 1, s[k][j]);*

*printf("\nЗначение функции цели = %lf", s[k][n]);*

*free(x);*

*free(xc);*

*for (i = 0; i <= n; i++) {*

*free(s[i]);*

*}*

*free(s);*

*fflush(stdin);*

*getchar();*

*return 0;*

*}*

*// Целевая функция:*

*double f(double \* x){*

*return 4 \* pow(x[0], 2) + 2 \* x[0] \* x[1] + 5 \* pow(x[1], 2) + 9 \* x[0];*

*}*

*// Функция нахождения наибольшего ("max") или наименьшего ("min") значения целевой функции в вершинах симплекса.*

*int Knum(const char\* q) {*

*int i, j = 0;*

*double u;*

*u = s[0][n];*

*for (i = 1; i <= n; i++) {*

*switch (strcmp(q, "max")) {*

*case 0: // q == "max"*

*if (s[i][n] > u) {*

*u = s[i][n];*

*j = i;*

*}*

*break;*

*case 1: // q == "min"*

*if (s[i][n] < u) {*

*u = s[i][n];*

*j = i;*

*}*

*break;*

*}*

*}*

*return j;*

*}*

*// Функция вычисления значений целевой функции в вершинах симплекса.*

*void Fsi(double\* x) {*

*int i, j;*

*for (i = 0; i <= n; i++) {*

*for (j = 0; j < n; j++)*

*x[j] = s[i][j];*

*s[i][n] = f(x);*

*}*

*}*