**Лабораторная работа № 7**

**Минимизация функции многих переменных методом градиента с дроблением шага**

# Введение

**Цель работы:** изучить метод градиента с дроблением шага для решения задачи минимизации функции многих переменных и получить практические навыки его применения.

Вариант № 15

# Ход работы

*1. Найти точные значения координат точки минимума и минимальное значение функции для функции соответствующего варианта задания, используя необходимые и достаточные условия локального минимума.*

**Необходимое условие локального минимума:**

Если дифференцируемая функция имеет экстремум в точке , то обе [частные производные 1-го порядка](http://mathprofi.ru/chastnye_proizvodnye_primery.html) в данной точке равны нулю:

**Достаточное условие локального минимума:**

Если точка M0 критическая и гессиан положительно определён в точке , то M0 – точка локального минимума функции.

Найдем критические точки, для этого вычислим первые частные производные функции И приравняем их к нулю

Из системы уравнений находим, что

Таким образом точка (5, 2 ) является критической

Вычислим вторые производные

Тогда гессиан примет вид:

Подставив в H, получим

Так как 4/5>0 и 4/5\*5-2\*1=2>0, то гессиан положительно определен и в точке (5, 2) имеется локальный минимум.

*2. Выполнить вручную вычисление приближенного значения точки минимума целевой функции методом градиента с дроблением шага, начиная с произвольно выбранного начального приближения . Точность решения ε=0,01. Вручную подробно достаточно выполнить первый шаг метода градиента с дроблением шага. Параметры метода выбрать самостоятельно.*

Выберем следующие параметры метода градиентного спуска с дроблением шага:

|  |  |
| --- | --- |
| Целевая функция |  |
| Градиент целевой функции |  |
| Начальное приближение |  |
| Точность |  |
| Параметры метода градиента с дроблением шага |  |

Критерий останова алгоритма

*1 шаг*

Проверяем условие

Т.к условие не выполнено, дробим шаг

Снова проверяем условие

Т.к условие не выполнено, дробим шаг

Снова проверяем условие

Условие выполняется, проверяем условие останова

Условие останова не выполнено, переходим к следующему шагу

*2 шаг*

*3 шаг*

*4 шаг*

Условие останова выполнено, ответ () получен за 4 шага

*3. Реализовать логическую функцию для нахождения приближенного значения точки локального минимума и минимального значения целевой функции методом градиента с дроблением шага. Входными данными для логической функции являются:*

* *Целевая функция у = f (x1, х2);*
* *Градиент целевой функции grad (f(x1,х2));*
* *Начальное приближение М0(x10, x20) к точке локального минимума;*
* *Точность решения ɛ;*
* *Ограничение на максимальное число итераций n;*
* *Параметры метода градиента с дроблением шага α, β, γ.*

*Функция возвращает значение «истина», если приближенное решение с заданной точностью получено за число итераций, не превышающее n, и «ложь» - в противном случае.*

*Результат работы программы: приближенное значение точки локального минимума и минимальное значение целевой функции с заданной точностью, количество выполненных итераций.*

*Предусмотреть возможность сохранения пошаговых результатов реализации метода градиента с дроблением шага в файл.*

Расчет функции в заданной точке проходит в функции func(x), градиент расчитывается в grad(x). Расчет минимума осуществляется функцией grad\_desc(). Эта функция возвращает 1 если расчет произведен за заданное число шагов, иначе 0. Также функция печатает результат в консоль и сохраняет пошаговые результаты в файл “result.txt”

Пример вызова grad\_desc:

>> double y[2] = {1.0, 1.0};

>> int r = grad\_desc(func, grad, y0, 0.01, 1, 0.2, 0.1, 10);

Результат работы:

===== Result ====

x1\_min = 4.9792

x2\_min = 2.0094

fu\_min = 30.0002

num\_steps = 6

Answer 1

Листинг файла main.c

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #include <math.h>  typedef double\* (\*functype)(const double\*) ;  double\* func(const double\* x)  {  double\* f = (double\*)malloc(sizeof(double));  \*f = x[0]\*x[1]+50/x[0]+20/x[1];  return f;  }  double\* grad(const double \*x)  {  double\* g = (double\*)malloc(2 \* sizeof(double));  g[0] = x[1]-50/(x[0]\*x[0]);  g[1] = x[0]-20/(x[1]\*x[1]);  return g;  }  int grad\_desc(functype func, functype grad,  double\* y0, double e, double alpha, double beta, double gamma, int n)  {  double y[n+1][2];  double f\_arr[n+1];  double delta[2];  y[0][0] = y0[0];  y[0][1] = y0[1];  for(int i=0; i<=n; i++){  double\* f = func(y[i]);  double\* g = grad(y[i]);  delta[0] = alpha\*g[0];  delta[1] = alpha\*g[1];  y[i+1][0] = y[i][0]-delta[0];  y[i+1][1] = y[i][1]-delta[1];  // проверяем условие для величины шага  double\* f\_next = func(y[i+1]);  for(int j=0; j<=10; j++){  if(\*f\_next>\*f-gamma\*(delta[0]\*g[0]+delta[1]\*g[1])){  // дробим шаг  free(f\_next);  delta[0] \*= beta;  delta[1] \*= beta;  y[i+1][0] = y[i][0]-delta[0];  y[i+1][1] = y[i][1]-delta[1];  f\_next = func(y[i+1]);  if(sqrt(delta[0]\*delta[0]+delta[1]\*delta[1])<1e-9 ){  printf("Smth wrong!");  return 0;  }  }  else break;  }  // сохраняем результаты в массив  f\_arr[i] = \*f;  // проверяем условие останова  if (fabs(\*f\_next-\*f) <= e){  f\_arr[i+1] = \*f\_next;  FILE \*fl = fopen("results.txt", "w");  if (fl == NULL)  {  printf("Error opening file!\n");  exit(-1);  }  fprintf(fl, "N\tx1\t\tx2\t\tf\n");  for(int k=0;k<=i+1;k++){  fprintf(fl, "%d\t%.4f\t%.4f\t%.4f\n",k ,y[k][0] ,y[k][1], f\_arr[k]);  }  fclose(fl);  // выводим результаты на экран  printf("===== Result ====\n");  printf("\tx1\_min = %.4f\n\tx2\_min = %.4f\n", y[i+1][0], y[i+1][1]);  printf("\tfu\_min = %.4f\n", f\_arr[i+1]);  printf("\tnum\_steps = %d\n", i+1);  free(f\_next);  return 1;  }  free(f\_next);  free(f);  free(g);  }  printf("Tolerance do not satisfied\n");  return 0;  }  int main()  {  double y0[] = {1.0, 1.0};  int r = grad\_desc(func, grad, y0, 0.01, 1, 0.2, 0.1, 10);  printf("Answer %d", r);  return 0;  } |