УДК 510.67

А.Е.Сланбекова, Л.С.Фазылова

Карагандинский государственный университет им. Е.А.Букетова (E-mail: SLanbekovaAE@mail.ru)

Программирование методов минимизации для многомерных функций

В работе рассмотрены многомерные методы безусловной минимизации: метод градиентного спуска, метод наискорейшего спуска, метод сопряженных направлений. Приведена программа вычисления точек минимума многомерной функции рассмотренными методами. Проводится сравнительный анализ данных методов.

Ключевые слова: минимизация, градиентный спуск, метод наискорейшего спуска, сопряженные направления, TurboRascal.

Постановка задачи. Рассмотрим задачу безусловной минимизации

$$J(u) \to \min;\ u \in U \equiv E^n, \tag{1}$$

где E^n — евклидово пространство n-мерных векторов, $J(u)$ — выпуклая функция на заданном множестве и $J(u) \in C^1(E^n)$.

В данной работе рассматриваются следующие численные методы решения задачи (1):

- метод градиентного спуска с дроблением шага [1];
- метод наискорейшего спуска [2];
- метод сопряженных направлений [2].

Опишем *метод градиентного спуска* [1]. Будем считать, что некоторая начальная точка $u^{(0)}$ уже выбрана. Тогда метод градиентного спуска заключается в построении последовательности $\{u^{(k)}\}$ по правилу

$$u^{(k+1)} = u^{(k)} - \alpha^{(k)} J'(u^{(k)}),\ \alpha^{(k)} > 0,\ k = 0,1,2,.... \tag{2}$$

Число $\alpha^{(k)}$ из (2) называют длиной шага или шагом метода градиентного спуска. Существуют различные способы выбора $\alpha^{(k)}$ в методе (2). В зависимости от способа выбора $\alpha^{(k)}$ можно получить различные варианты метода градиентного спуска. Укажем способы выбора $\alpha^{(k)}$, которые были рассмотрены в данной работе.

1) *Метод с дроблением шага* [1]. Достаточно малое $\alpha^{(k)} > 0$ часто выбирают так, чтобы выполнялось условие монотонного убывания $J(u^{(k+1)}) < J(u^{(k)}),\ k = 0,1,2,....$. В случае нарушения этого условия шаг $\alpha^{(k)}$ дробят до тех пор, пока не восстановится условие монотонного убывания. Затем переходят к вычислению следующей итерации.

2) *Метод наискорейшего спуска* [2]. Приведем пример, когда величина $\alpha^{(k)}$ существует и может быть выписана в явном виде.

Пример. Пусть дана квадратичная функция

$$J(u) = \frac{1}{2} < Au,\ u > - < b,\ u >,\ u \in E^n, \tag{3}$$

где A — симметричная неотрицательно определенная матрица порядка $n \times n$; b, u — векторы из E^n, $< b,\ u >$ — скалярное произведение из E^n.

Для функции (3) шаг метода наискорейшего спуска вычисляется по формуле

$$\alpha^{(k)} = \frac{|\ Au^{(k)} - b\ |^2}{< A(Au^{(k)} - b), Au^{(k)} - b) >} > 0. \tag{4}$$

Таким образом, метод наискорейшего спуска для квадратичной функции (3) заключается в построении последовательности точек $\{u^{(k)}\}$ по формулам (2), (4).

Метод сопряженных направлений [2] состоит в построении последовательных приближений $\{u^{(k)}\}$ к точке минимума функции $J(u)$ следующим образом:

$$u^{(k+1)} = u^{(k)} - \alpha^{(k)} p^{(k)},\ k = 0,1,...,\ u^{(0)} \in E^n, \tag{5}$$

где $u^{(0)}$ — заранее выбранное начальное приближение, шаг $\alpha^{(k)}$ выбирается как в методе наискорейшего спуска. А направление спуска в (5) — $p^{(k)}$ определяется по формуле $p^{(k)} = J'(u^{(k)}) + \beta_k p^{(k-1)},\ k = 0,1,...,\ p^{(0)} = J'(u^{(0)})$, где

$$\beta_k = \frac{\| J'(u^{(k)}) \|^2}{\| J'(u^{(k-1)}) \|^2} = \frac{\sum_{i=1}^{n} \left(\frac{\partial J(u^{(k)})}{\partial u_i} \right)^2}{\sum_{i=1}^{n} \left(\frac{\partial J(u^{(k-1)})}{\partial u_i} \right)^2}.$$

В качестве условия окончания рассмотренных методов обычно используется близость к нулю градиента $J'(u^{(k)})$, т.е. выполнение неравенства

$$\| J'(u^{(k)}) \| = \sqrt{\sum_{i=1}^{n} \left[\frac{\partial J(u^{(k)})}{\partial u_i} \right]} \le \varepsilon,$$

где ε — заданная точность.

Если на каком-то шаге k это неравенство выполнилось, то полагают $u^* \approx u^{(k)}$, $J^* \approx J(u^{(k)})$, где u^* — точка минимума, J^* — минимальное значение функции $J(u)$.

Приведем программу вычисления точек безусловного минимума многомерной квадратичной функции рассмотренными методами. Программа реализована на языке TurboRascal и состоит из трех модулей: основного, вспомогательного (коды алгоритмов 3 методов) и модуля матричного представления квадратичной функции.

Основной модуль

```
uses GradMet,CRT,u_matrix;
var
  i,k: Integer;
begin
  ClrScr;
  WriteLn ('Программа минимизации квадратичных функций');
  WriteLn ('градиентными методами');
  WriteLn ('Введите квадратичную функцию:');
  WriteLn ('1. В скалярном виде.');
  WriteLn ('2. В матричном виде.');
  Write ('Вы выбрали способ ввода №');
  ReadLn (k);
  GotoXY (1,WhereY - 4);
  for i := 1 to 4 do DelLine;
  case k of
    1: VvodF;
    2: VvodMatrixF;
  end;
  Write ('Точность: eps = ');
  ReadLn (eps);
  Write ('Введите начальный вектор (', n,' координат): ');
  for i := 1 to N do
    Read (x[i]);
  Write ('Начальный шаг для метода с дроблением шага: alpha = ');
  ReadLn (alpha);
  ClrScr;
  WriteLn ('Программа минимизации квадратичных функций');
  WriteLn ('градиентными методами');
  WriteLn ('Метод с дроблением шага:');
  MetodGradienta (alpha,eps,x);
  PrintV (x,1,N);
```

```
  WriteLn;
  WriteLn;
  WriteLn ('Метод наискорейшего спуска:');
  Spusk (eps,x);
  PrintV (x,1,N);
  WriteLn;
  WriteLn;
  WriteLn ('Метод сопряженных направлений:');
  Naprav (eps,x);
  PrintV (x,1,N);
  WriteLn;
  ReadLn;
end.
```

Вспомогательный модуль

```
unit GradMet;
interface
uses Metod, U_Matrix, CRT;
var
  eps, alpha: Real;
  A: Matrix;
  b, p, x, Grad: Vector;
function Minimiz (f: TFunc) : Real;
procedure VvodF;
procedure VvodMatrixF;
function F (x: Vector) : Real;
procedure F1 (x: Vector);
procedure MetodGradienta (alpha,eps:Real; var x:Vector);
function PhiForSpusk (alpha:Real): Real; far;
procedure Spusk (eps:Real; var x1:Vector);
function PhiForNaprav (alpha:Real): Real; far;
procedure Naprav (eps:Real; var x1:Vector);
implementation
{----------}
function Minimiz (f: TFunc) : Real;
{Минимизация функции одной переменной}
var
  a,b, x, effect, u0,t: Real;
begin {Minimiz}
  u0 := 0;
  t := 0.01;
  while not (MetodSvenna (f,u0,t,a,b) and (a>0)) do
    u0 := u0 + t {1};
  MetodZolot (f,a,b,eps,x,effect);
  Minimiz := x
end; {Minimiz}
{---------}
procedure VvodF;
{Ввод квадратичной функции в виде многочлена}
var
  i,j: Integer;
  k, p: Real;
begin
  WriteLn ('Количество неизвестных (N <= 5):');
  repeat
    Write ('N = '); ReadLn (N);
```

```
  until (N <= 5) and (N > 0);
  Write ('f (x1');
  for i := 2 to N do
    Write (', x',i);
  WriteLn (') =');
  Write ('= a11 * x1*x1 ');
  for i:= 1 to N do
    begin
      if i <> 1 then
        GotoXY ((i - 1) * 14 + 1, WhereY);
      for j := i to N do
        if not ((i = 1) and (j = 1)) then
          Write ('+ a',i,j, ' * x',i,'*x',j,' ');
      WriteLn ('+');
    end;
  for i := 1 to N do
    Write ('+ b',i,' ');
  WriteLn;
  WriteLn ('Введите коэффициенты:');
  for i := 1 to N do
    begin
      WriteLn;
      for j := i to N do
        begin
          GotoXY (1 + (j - 1) * 16, WhereY - 1);
          Write ('a',i,j,' = ');
          ReadLn (k);
          if (i = j) then p := 2 * k
          else p := k;
          a [i,j] := p;
          a [j,i] := p;
        end;
    end;
  WriteLn;
  for i := 1 to N do
    begin
      GotoXY (1 + (i - 1) * 16, WhereY - 1);
      Write ('b',i,' = ');
      ReadLn (k);
      b [i] := -k;
    end;
end; {VvodF}
{--------}
procedure VvodMatrixF;
{Ввод квадратичной функциив вилде матрицы}
var
  i,j: Integer; k, p: Real; yes: Boolean;
begin
  WriteLn ('Количество неизвестных (N <= 5):');
  repeat
    Write ('N = ');
    ReadLn (N);
  until (N <= 5) and (N > 0);
  repeat
    yes := True;
```

```
      WriteLn ('Введите симметричную матрицу A (',n,' x ',n,'):');
      for i:= 1 to N do
for j:= 1 to N do
        Read (A [i,j]);
      {Проверим матрицу на симметричность}
      for i := 1 to N do
       begin
        for j := i to N do
         if not (a[i,j] = a[j,i]) then
          begin
           Write ('Ошибка! ');
           yes := False;
           break;
          end;
         if not yes then break;
       end;
    until yes;
    WriteLn ('Введите вектор B (',n,' компонент):');
    for i := 1 to N do
     Read (b[i]);
  end; {VvodF}
  {--------}
  function F (x: Vector) : Real;
  {Значение введенной квадратичной функции}
  var
    V: Vector;
  begin
    MultMV (A, x, V);
    F := ScalMult (V, x) * 1/2 - ScalMult (b, x);
  end;
  {--------}
  procedure F1 (x: Vector);
  {Первая производная квадратичной функции -> вектор Grad}
  begin
    MultMV (A,x, Grad);
    SubstVV (Grad, b, Grad);
  end;
  {--------}
  procedure MetodGradienta (alpha,eps:Real; var x:Vector);
  {Градиентный метод минимизации с дроблением шага}
  var
    x1: Vector;
  begin {MetodGradienta}
    F1 (x);
    while Norma1V (Grad) > eps do
     begin
      x1 := x;
      repeat
       F1 (x1);
       MultAV (alpha, Grad, x);
       SubstVV (x1, x, x);
       if f(x) >= f(x1) then
        alpha := alpha/2;
      until f(x) < f(x1);
      F1 (x);
```

```
    end;
end; {MetodGradienta}
{-------}
function PhiForSpusk (alpha:Real): Real;
{Функция шага для метода наискорейшего спуска}
var
  a, b: Vector;
begin
  MultAV (alpha, Grad, a);
  SubstVV (x, a, b);
  PhiForSpusk := F (b)
end;
{----------}
procedure Spusk (eps:Real; var x1:Vector);
{Метод градиентного наискорейшего спуска}
var
  alpha: Real;
  x: Vector;
  c:Integer;
begin {Spusk}
  F1 (x1);
  while Norma1V (Grad) > eps do
    begin
      x := x1;
      alpha := Minimiz (PhiForSpusk);
      MultAV (alpha, Grad, x1);
      SubstVV (x, x1, x1);
      F1 (x1);
    end;
end; {Spusk}
{-------}
function PhiForNaprav (alpha:Real): Real;
{Функция шага для метода сопряженных направлений}
var
  a, b: Vector;
begin
  MultAV (alpha, p, a);
  SubstVV (x, a, b);
  PhiForNaprav := F (b)
end;
{----------}
procedure Naprav (eps:Real; var x1:Vector);
var
  alpha, beta: Real;
  x: Vector;
begin {Naprav}
  F1 (x1);
  p := Grad;
  while Norma1V (Grad) > eps do
    begin
      x := x1;
      alpha := Minimiz (PhiForNaprav);
      MultAV (alpha, p, x1);
      SubstVV (x, x1, x1);
      beta := sqr (Norma3V (Grad));
```

```
      F1 (x1);
      beta := sqr (Norma3V (Grad)) / beta;
      MultAV (beta, p, p);
      AddVV (Grad, p, p)
    end;
end; {Naprav}
{-------}
end.
```

Процедуры из модуля U_Matrix

```
type
  Matrix = array [1..20,1..20] of Real;
  Vector = array [0..20] of Real;
var
  n: Integer;
{-------------}
procedure PrintV (x:Vector; m,n:Integer);
{Распечатка вектора длины N}
var
  i:Integer;
begin
  WriteLn ('Вектор:');
  for i:=m to n do
     Write (x[i]:10:5);
end;
{-------}
procedure MultMV (a:Matrix; x:Vector; var y:Vector);
{Умножить матрицу A на вектор X — результат: вектор У}
var
  k,i,j: Integer;
begin
  for i:=1 to n do
     begin
               y[i]:=0;
               for j:=1 to n do
                 y[i]:=y[i]+a[i,j]*x[j];
     end;
end;
{---------}
procedure AddVV (x,y:Vector; var z:Vector);
{Сложить вектор X и вектор У — результат: вектор Z}
var
  i: Integer;
begin
  for i:=1 to n do
     z[i]:=x[i]+y[i];
end;
{----------}
procedure SubstVV (x,y:Vector; var z:Vector);
{Вычесть вектор У из вектора X — результат: вектор Z}
var
  i: Integer;
begin
  for i:=1 to n do
     z[i]:=x[i]-y[i]
end;
```

```
{----------}
function Norma1V (x:Vector): Real;
{Первая норма вектора: max(модули компонент вектора)}
var
  i: Integer;
  norm: Real;
begin
  norm:=abs(x[1]);
  for i:=2 to n do
     if abs(x[i]) > norm then
              norm := abs (x[i]);
  Norma1V:=norm
end;
{---------------}
function ScalMult (a,b:Vector) : Real;
{Скалярное произведение векторов}
var
  i: Integer;
  c: Real;
begin
  c := 0;
  for i:=1 to n do
    c := c + a[i] * b[i];
  ScalMult := c
end;
{-----------}
procedure MultAV (a:Real; b:Vector; var c:Vector);
{Умножение вектора на число}
var  i: Integer;
begin
  for i:=1 to n do
    c[i] := a * b[i]
end;
```

Результат выполнения программы (выделены данные, вводимые пользователем; на экран выводятся точка минимума и минимальное значение функции *f**)

Пример 1. Функция вводится в скалярном виде:

$$f(x) = 3x_1^2 + 5x_2^2 - 2x_1x_2 + 4x_1 - 3x_2,\ x^{(0)} = (-1, 1).$$

Программа минимизации квадратичных функций.

Введите квадратичную функцию:
1. В скалярном виде.
2. В матричном виде.
Вы выбрали способ ввода № **1**
Количество неизвестных (N <= 5):
N = **2**
f (x1, x2) =
= a11 * x1*x1 + a12 * x1*x2 +
+ a22 * x2*x2 +
+ b1 + b2
Введите коэффициенты:
a11 = **3** a12 = **-2**
a22 = **5**
b1 = **4** b2 = **-3**
Точность: eps = **0.00001**
Введите начальный вектор (2 координат): **-1 1**
Начальный шаг для метода с дроблением шага: alpha = **0.1**

Вывод результата программы минимизации квадратичных функций.

Метод с дроблением шага:
Вектор:
-0.60714 0.17857, f* = -1.482143
Метод наискорейшего спуска:
Вектор:
-0.60714 0.17857, f* = -1.482143
Метод сопряженных направлений:
Вектор:
-0.60714 0.17857, f* = -1.482143

Пример 2. Функция вводится в матричном виде:

$$f(x) = \frac{1}{2}(Ax, x) - (b, x),$$

где

$$A = \begin{pmatrix} 4 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 4 \end{pmatrix}, b = \begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix}, x^0 = \begin{pmatrix} 0 \\ 1/2 \\ 1 \end{pmatrix}.$$

Программа минимизации квадратичных функций.
Введите квадратичную функцию:
1. В скалярном виде.
2. В матричном виде.
Вы выбрали способ ввода **№2**
Количество неизвестных (N <= 5):
N = **3**
Введите симметричную матрицу A (3 x 3):
3 1 -2
1 4 -1
-2 -1 5
Введите вектор B (3 компонент):
-1 2 1
Точность: eps = 0.0001
Введите начальный вектор (3 координат): 1 0 -2
Начальный шаг для метода с дроблением шага: alpha = 0.1

Вывод результата программы минимизации квадратичных функций.
Метод с дроблением шага:
Вектор:
-0.44995 0.64999 0.15003, f*=-0.950000
Метод наискорейшего спуска:
Вектор:
-0.44995 0.64999 0.15003, f*=-0.950000
Метод сопряженных направлений:
Вектор:
-0.44995 0.64999 0.15003, f*=-0.950000

При составлении алгоритма программы была использована следующая литература: [3], [4].

References

1 *Panteleyev A.V., Letova T.A.* Optimization methods in examples and problems. — Moscow: High school, 2002. — 542 p.

2 *Attetkov A.V., Galkin S.V., Zarubin V.S.* Optimization methods. — Moscow: Publishing house of MGTU named N.E. Bauman, 2001. — 436 p.

3 *Vyrt N.* Algorithms and data structures / Translation from English. — Moscow: Mir, 1989. — 360 p.

4 *Kultin N.B.* Programming in Turbo Pascal 7.0 and Delphi. — 2-nd edition, revise and supplemented. St.-Petersburg: BHV — St.-Petersburg, 1999. — 416 p.

А.Е.Сланбекова, Л.С.Фазылова

Көп айнымалы функциялар үшін минимумдау әдістерді программалау

Мақалада көп өлшемді шартсыз минимумдау әдістері қарастырылған: градиенттік түсу әдісі, ең тез түсу әдісі, түйіндес бағыттар әдісі. Көп айнымалы квадраттық функциялардың минимум нүктелерін жоғарыда аталған әдістерімен есептеу программасы келтірілген. Сондай-ақ осы әдістердің салыстырмалы талдауы жасалған.

A.E.Slanbekova, L.S.Fazilova

Programming of minimization methods for multidimensional functions

Present work explains multidimensional unconditional minimization methods: method of gradient descent, the steepest descent method, the method of conjugate directions. Work contains program for calculation points of minimum in multidimensional function using considered methods and benchmark analysis of these methods.

ӘОЖ 338. 242: [338. 26. 015: 51:004]

Н.К.Сыздықова, Д.Р.Бейсенова

Е.А.Бөкетов атындағы Қарағанды мемлекеттік университеті (E-mail: dana_68_11@mail.ru*)*

MS Excel бағдарламасын қолданбалы есептерді шешуге қолдану

Мақалада қолданбалы есептерді шешуге Microsoft Excel электрондық редакторының қолданылуы қарастырылады. Мұнда тиімді жоспарды іздеу әдістерінің бірі көрсетілген. Негізгі міндеті ресурстар шектеулі болғанда шығынды минималдау болып табылады. Мақсатқа жету үшін Microsoft Excel бағдарламасының «Поиск решения» қондырмасын қолдандық. Мақалада есептің математикалық моделі көрсетілген, берілген есептің шешуі көрнекі түрде сипатталған.

Кілтті сөздер: сызықты бағдарламалау, электронды редактор, тиімді жоспар, есептің математикалық моделі.

Қолданбалы өндірістік есептерді шешу барысында заманауи ақпараттық технологияларды пайдалану қазіргі уақыттың көкейтесті талабы болып табылады. Мұндай арнайы курстардың бірінің тақырыбын — экономика мен басқарудың әр түрлі саласының есептерін Microsoft Excel электрондық кестесінің көмегімен сызықты бағдарламалау деп алуға болады.

Сызықты бағдарламалау — бұл ең үлкен және ең кіші мәндерді іздеп табу сияқты есептерді шешумен айналысатын математика бөлімі, олар үшін математикалық талдау әдістері жарамсыз болып қалады. Басқа сөзбен айтқанда, «сызықты бағдарламалау» термині нақты экономикалық объектінің оның элементтерінің арасындағы сызықты байланыстарды табу негізінде жұмыс жоспарын анықтауды білдіреді. Сызықты бағдарламалау есебі — берілген жүйеде шешуге шектемелер қойылған есептің тиімді, ең жақсы жоспарын табу [1].

Сызықты бағдарламалау есептер класына жоспарлау және басқарудың әр түрлі есептері жатады, мысалы:

– өнім шығарудың тиімді жоспарын табу (қорларды тиімді бөлу);
– салааралық ағындарды тиімділеу (салалар бойынша өнімдер түрін өндіруді жоспарлау);
– тиімді рационды анықтау (химиялық қоспаның құрамын тиімдендіру);
– транспорттық есеп (транспорттық желі бойынша тауар жеткізу ағынын тиімді бөлу);
– өндірісті орналастыру туралы есеп (өндіріске және өнімдерді тасымалдауға кететін шығынды есептей отырып жоспарлау);
– тағайындау туралы есеп (транспорттық құралдардың әр түрлі түрлерін тиімді бөлу) және т.б. [2].