МАТЕМАТИЧНІ МЕТОДИ МОДЕЛЮВАННЯ ТА ОПТИМІЗАЦІЇ ПРОЦЕСІВ

Практичне заняття № 8

Тема. Прямі методи безумовної оптимізації нульового порядку.

План проведення заняття
Вступ.
1. Метод Гауса .
2. Метод Нелдера – Міда.
Заключення.

Завдання на СРС:
Виконати приклад 1 і приклад 2 із точністю ε=0,01.

Вступ

При реалізації прямих методів істотно скорочується етап підготовки рішення задачі, так як немає необхідності у визначенні перших і других похідних. До прямих методів відноситься цілий ряд алгоритмів, які відрізняються за своєю ефективністю. Такі методи носять в основному евристичний характер.

Прямі методи призначені для вирішення безумовних завдань оптимізації виду:

$$\min_{\bar{x} \in E^n} f(\bar{x})$$

1. Метод Гауса.

Це найпростіший алгоритм , що полягає в тому, що на кожному кроці (кожної ітерації) мінімізація здійснюється тільки по одній компоненті вектора змінних $\bar{x}$.

Нехай нам дано початкове наближення $\bar{x}^0 = (x_1^0, x_2^0, \dots, x_n^0)^T$. На першій ітерації знаходимо значення мінімуму функції при змінній першій координаті і фіксованих інших компонентах, тобто

$$x_1^1 = \arg\min_{x_1} f(x_1, x_2^0, \dots, x_n^0)$$

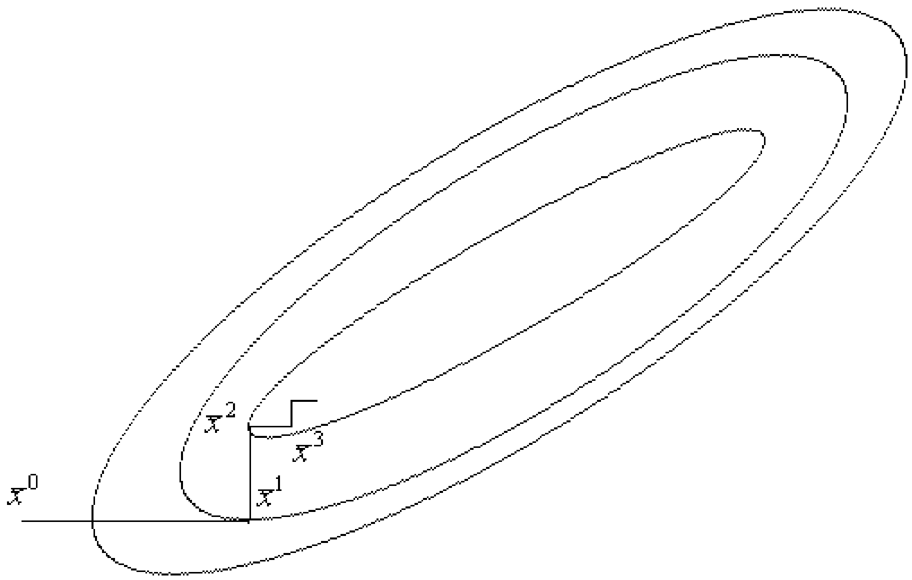
В результаті отримуємо нову точку $\bar{x}^1 = (x_1^1, x_2^0, \dots, x_n^0)$. Далі з точки $\bar{x}^1$ шукаємо мінімум функції, змінюючи другу координату і вважаючи фіксованими всі інші координати. В результаті отримуємо значення

$$x_2^1 = \arg\min_{x_2} f(x_1^1, x_2, x_3^0, \dots, x_n^0)$$

і нову точку $\bar{x}^2 = (x_1^1, x_2^2, x_3^0, \dots, x_n^0)$.. Продовжуючи процес, після n кроків отримуємо точку $\bar{x}^n = (x_1^1, x_2^2, \dots, x_n^n)$.., починаючи з якої процес пошуку поновлюється по першій змінній.

В якості умов припинення пошуку можна використовувати наступні два критерії:

1) $\left\| f(\bar{x}^{k+1}) - f(\bar{x}^k) \right\| \le \varepsilon_0$
2) $\left| x_i^{k+1} - x_i^k \right| \le \varepsilon_1 \ \forall i$



Приклад траєкторії спуску в алгоритмі Гауса

Метод дуже простий, але не дуже ефективний. Проблеми можуть виникнути, коли лінії рівня сильно витягнуті і "еліпсоїди" орієнтовані, наприклад, уздовж прямих виду $x_1 = x_2$. У подібній ситуації пошук швидко застряє на дні такого яру, а якщо початкове наближення виявляється на осі "еліпсоїда", то процес так і залишиться в цій точці.

Гарні результати виходять в тих випадках, коли цільова функція являє собою опуклу сепарабельну функцію виду

$$f(\bar{x}) = \sum_{i=1}^{n} f_i(x_i)$$

2. Метод Нелдера – Міда.

У цьому методі в процесі пошуку здійснюється робота з регулярними симплексами. Регулярні багатогранники в просторі E^n називаються ***симплексами.*** Для n = 2 регулярний симплекс представляє собою рівносторонній трикутник, при n = 3 – тетраедр і т.д.

Координати вершин регулярного симплекса в n-вимірному просторі можуть бути визначені наступною матрицею D, в якій стовпці являють собою вершини симплекса, пронумеровані від 1 до (n +1), а рядки - координати вершин, $i = \overline{1, n}$. Матриця має розмірність $n \times (n+1)$:

$$D = \begin{bmatrix} 0 & d_1 & d_2 & \dots & d_2 \\ 0 & d_2 & d_1 & \dots & d_2 \\ 0 & d_2 & d_2 & \dots & d_2 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & d_2 & d_2 & \dots & d_1 \end{bmatrix}_{n \times (n+1)}$$

де: $d_1 = \frac{t}{n\sqrt{2}}\left(\sqrt{n+1} + n - 1\right)$; $d_2 = \frac{t}{n\sqrt{2}(\sqrt{n+1} - 1)}$

t - відстань між вершинами.

У найпростішому вигляді симплексний алгоритм полягає в наступному. Будується регулярний симплекс. З вершини, в якій $f\ (\bar{x})$ максимальна (точка 1, див. рис. 2.4) проводиться проектуюча пряма через центр ваги симплекса. Потім точка 1 виключається і будується новий ***відбитий симплекс*** з решти старих точок і однієї нової, що розміщена на проектуючій прямій на належній відстані від центру ваги.

Продовження цієї процедури, в якій кожен раз виключається вершина, де цільова функція максимальна, а також використання правил зменшення розміру симплекса і запобігання циклічного руху в околиці екстремуму, дозволяє досить ефективно визначати мінімум для "хороших" функцій. Але для "яружних" функцій такий пошук неефективний.

Уявлення про ідею алгоритму дає малюнок 2.4.

У симплексному алгоритмі Нелдера і Міда мінімізація функцій n змінних здійснюється з використанням деформованого багатогранника.

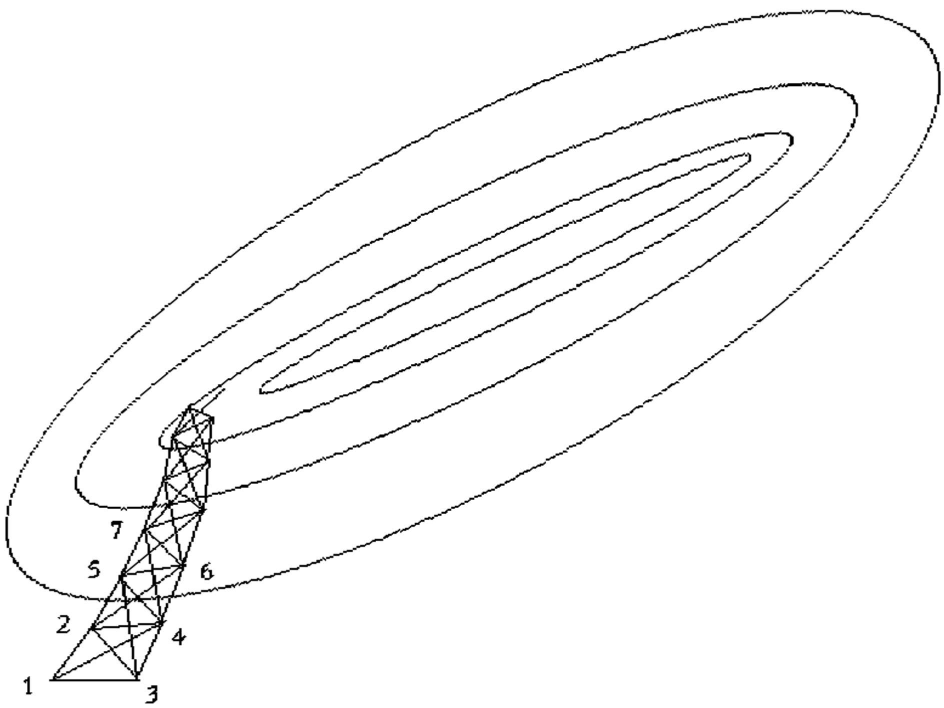


Рис. 2.4. Траєкторія спуску в найпростішому симплексному алгоритмі

Будемо розглядати k-у ітерацію алгоритму. Шлях

$\bar{x}_i^k = \left[x_{i1}^k, x_{i2}^k, \dots, x_{in}^k\right]^T$, $i = 1, \dots, (n+1)$, є i-ю вершиною в E^n на k-му етапі пошуку, k = 0,1,2, ..., і нехай значення цільової функції в цій вершині $f(\bar{x}_i^k)$. Відзначимо вершини з мінімальним і максимальним значеннями. І позначимо їх наступним чином:

$$f\left(\bar{x}_h^k\right) = \max\left\{f\left(x_1^k\right), \dots, f\left(x_{n+1}^k\right)\right\};$$

$$f\left(\bar{x}_l^k\right) = \min\left\{f\left(x_1^k\right), \dots, f\left(x_{n+1}^k\right)\right\}.$$

Багатогранник в E^n складається з $n+1$ вершин $\bar{x}_1^k, \bar{x}_2^k, \dots, \bar{x}_{n+1}^k$. Позначимо через $\bar{x}_{n+2}^k$ - центр ваги вершин без точки $\bar{x}_h^k$ з максимальним значенням функції. Координати цього центру обчислюються за формулою:

$$x_{n+2,j}^k = \frac{1}{n}\left[\sum_{i=1}^{n+1} x_{i,j}^k - x_{h,j}^k\right], j = 1, \dots, n.$$

Початковий багатогранник зазвичай вибирається у вигляді регулярного симплекса (з вершиною на початку координат). Можна початок координат помістити в центр ваги. Процедура відшукання вершин в $E^{n,}$ в яких $f(\bar{x})$ має краще значення, складається з наступних операцій: ***1) відображення; 2)розтягування; 3) стиснення; 4) редукції.***

1. Відображення. Відображення являє собою проектування точки $\bar{x}_h^k$ через центр ваги $\bar{x}_{n+2}^k$ у відповідності з наступним співвідношенням:

$$\bar{x}_{n+3}^k = \bar{x}_{n+2}^k + \alpha \cdot \left(\bar{x}_{n+2}^k - \bar{x}_h^k\right),$$

де $\alpha > 0$ - коефіцієнт відбиття.

Обчислюємо значення функції в знайденій точці $f(x_{n+3}^k)$. Якщо значення функції в даній точці $f\left(\bar{x}_{n+3}^k\right) \geq f\left(\bar{x}_h^k\right)$, то переходимо до четвертого пункту алгоритму - операції ***редукції.***

Якщо $f\left(\bar{x}_{n+3}^k\right) < f\left(\bar{x}_h^k\right) \wedge f\left(\bar{x}_{n+3}^k\right) < f\left(\bar{x}_l^k\right)$, то виконуємо операцію ***розтягування.***

В іншому випадку, якщо $f\left(\bar{x}_{n+3}^k\right) < f\left(\bar{x}_h^k\right) \wedge f\left(\bar{x}_{n+3}^k\right) \geq f\left(\bar{x}_l^k\right)$, то операція ***стиснення.***

2. **Розтягування.** Ця операція полягає в наступному.

Якщо $f\left(\bar{x}_{n+3}^k\right) < f\left(\bar{x}_l^k\right)$ (менше мінімального значення на k-му етапі), то вектор $(\bar{x}_{n+3}^k - \bar{x}_{n+2}^k)$ розтягується відповідно до співвідношення

$$\bar{x}_{n+4}^k = \bar{x}_{n+2}^k + \gamma \cdot (\bar{x}_{n+3}^k - \bar{x}_{n+2}^k),$$

де $\gamma > 0$ - коефіцієнт розтягування.

Якщо $f\left(\bar{x}_{n+4}^k\right) < f\left(\bar{x}_l^k\right)$, то $\bar{x}_l^k$ замінюється на $\bar{x}_{n+4}^k$ і процедура продовжується з операції ***відображення*** при $k = k + 1$. В іншому випадку $\bar{x}_l^k$ замінюється на $\bar{x}_{n+3}^k$ і також переходимо до операції ***відображення.***

3. Стиснення. Якщо $f\left(\bar{x}_{n+3}^{k}\right) > f\left(\bar{x}_{i}^{k}\right)$ для $\forall i, i \neq h$, то вектор $(\bar{x}_{h}^{k} - \bar{x}_{n+2}^{k})$ стискається відповідно до формули

$$\bar{x}_{n+5}^{k} = \bar{x}_{n+2}^{k} + \beta \cdot \left(\bar{x}_{h}^{k} - \bar{x}_{n+2}^{k}\right),$$

де $0 < \beta < 1$ - коефіцієнт стиснення. Після цього, точка $\bar{x}_{h}^{k}$ замінюється на $\bar{x}_{n+5}^{k}$, і переходимо до операції ***відображення*** з $k = k + 1$. Заново шукається $\bar{x}_{h}^{k+1}$.

4. Редукція. Якщо $f\left(\bar{x}_{n+3}^{k}\right) > f\left(\bar{x}_{h}^{k}\right)$, то всі вектори $(\bar{x}_{i}^{k} - \bar{x}_{l}^{k})$, де $i = \overline{1,(n+1)}$ зменшуються в два рази з відліком від точки $\bar{x}_{l}^{k}$ за формулою

$$\bar{x}_{i}^{k} = \bar{x}_{l}^{k} + 0.5 \cdot \left(\bar{x}_{i}^{k} - \bar{x}_{l}^{k}\right), i = \overline{1,(n+1)}$$

і здійснюється перехід до операції ***відображення*** (на початок алгоритму з $k = k + 1$).

В якості критерію зупину можуть бути взяті ті ж правила, що і в інших алгоритмах. Можна також використовувати критерій зупину такого вигляду:

$$\left\{\frac{1}{n+1} \cdot \sum_{i=1}^{n+1} \left[f\left(\bar{x}_{i}^{k}\right) - f\left(\bar{x}_{n+2}^{k}\right)\right]^{2}\right\}^{1/2} < \varepsilon.$$

Вибір коефіцієнтів α, β, γ зазвичай здійснюється емпірично. Після того, як багатогранник відповідним чином промасштабований, його розміри повинні підтримуватися незмінними поки зміни в топологіях завдання не будуть потребувати багатогранника іншої форми. Найчастіше рекомендують $\alpha = 1$, $0.4 \leq \beta \leq 0.6$, $2 \leq \gamma \leq 3$.

Приклад 1

Знайти мінімум функції Розенброка: $f(x) = 100 * (y - x^2)^2 + (1 - x)^2$, при ε=0,000001, α=1, β=0,5, γ=2.

Розв'язання:

Складемо алгоритм та знайдемо мінімум даної функції за допомогою мови програмування Pascal.

Реалізація алгоритму мовою Pascal матиме вигляд:

```
const
  eps = 0.000001;
  alfa  = 1.0;
  beta  = 0.5;
  gamma = 2.0;
  t = 0.2;

  n_max = 6;

var
  x, y: array[0 .. n_max] of real;

  fh, fl, f4, f5, f6: real;
```

```
  h, l, it: integer;



function f(k: integer): real;
var r1, r2: real;
begin
  r1 := y[k]-x[k]*x[k]; r2 := 1-x[k];
  f := 100*r1*r1+r2*r2;
end;


function maxf: real;
var f1, f2, r: real;
begin
  f1 := f(1); f2 := f(2); r := f(0);
  h := 0;
  if r < f1 then begin
    r := f1; h := 1;
  end;
  if r < f2 then begin
    r := f2; h := 2;
  end;
  maxf := r;
end;

function minf: real;
var f1, f2, r: real;
begin
  f1 := f(1); f2 := f(2); r := f(0);
  l := 0;
  if f1 < r then begin
    r := f1; l := 1;
  end;
  if f2 < r then begin
    r := f2; l := 2;
  end;
  minf := r;
end;


label TheEnd;
var
  i, flag: integer;
  r, r1, x0, y0, x1, y1: real;
begin
    x0 := -1.2; y0 := 1;
    x[0] := x0-0.5*t; y[0] := y0-t*sqrt(3)/6;
    x[1] := x0; y[1] := y0; r := f(1); y[1] := y0+t*sqrt(3)/3;
    x[2] := x0+0.5*t; y[2] := y[0];

    it := 0;
    writeln(' it=', it:5, ' x=', x0:8:4, ' y=', y0:8:4, ' f=', r:8:4);
    repeat
        fh := maxf; fl := minf;
        x[3] := 0.5*(x[0]+x[1]+x[2]-x[h]);
        y[3] := 0.5*(y[0]+y[1]+y[2]-y[h]);
```

```
    x[4] := (1+alfa)*x[3]-alfa*x[h];
    y[4] := (1+alfa)*y[3]-alfa*y[h];

    f4 := f(4);
    if (f4 < fl) then begin
      x[5]:=(1-gamma)*x[3]+gamma*x[4];
      y[5]:=(1-gamma)*y[3]+gamma*y[4];
      f5:=f(5);
      if (f5 < fl) then begin x[h]:=x[5]; y[h]:=y[5]; end
      else begin x[h]:=x[4]; y[h]:=y[4]; end;
      goto TheEnd;
    end;

    flag := 0;
    for i := 0 to pred(3) do begin
     if ((i <> h) and (f4 > f(i))) then inc(flag);
    end;
    if (flag = 2) then begin
      x[6]:=beta*x[h]+(1-beta)*x[3]; y[6]:=beta*y[h]+(1-beta)*y[3];
      x[h]:=x[6]; y[h]:=y[6];

      goto TheEnd;
    end;

    if (f(4) < fh) then begin
      for i := 0 to pred(3) do begin
        x[i]:=0.5*(x[i]+x[l]); y[i]:=0.5*(y[i]+y[l]);
      end;
    end
    else begin
      x[h]:=x[4]; y[h]:=y[4];
    end;

    TheEnd:;
    r := 0;
    for i := 0 to pred(3) do begin
     r1 := f(i)-f(3); r := r + r1*r1;
    end;
    r:=sqrt(r/3);
    inc(it);

    r1:=(f(0)+f(1)+f(2))/3;
    x0:=(x[0]+x[1]+x[2])/3;
    y0:=(y[0]+y[1]+y[2])/3;

    if it mod 20 = 0 then
     writeln(' it=', it:5, ' x=', x0:8:4, ' y=', y0:8:4, ' f=', r:8:4);
  until (r < eps);

  writeln(' it=', it:5, ' x=', x0:8:4, ' y=', y0:8:4, ' f=', r:8:4);
end.
```

Результати роботи програми наступні:

```
it=    0    x= -1.2000    y=  1.0000    f= 24.2000
it=   20    x= -0.5415    y=  0.2818    f=  0.0164
```

it= 40	x= -0.2912	y= 0.1231	f= 0.0291
it= 60	x= 0.0123	y= -0.0106	f= 0.0060
it= 80	x= 0.1233	y= -0.0075	f= 0.0039
it= 100	x= 0.4136	y= 0.1820	f= 0.1305
it= 120	x= 0.6089	y= 0.3575	f= 0.0019
it= 140	x= 0.6456	y= 0.4065	f= 0.0095
it= 160	x= 0.8838	y= 0.7762	f= 0.0017
it= 179	x= 1.0003	y= 1.0006	f= 0.0000.

Отже для розв'язання даної задачі нам знадобилося 179 ітерацій.

Приклад 2
знайти мінімум функції $f(x)=2x^2-12x$, при $\varepsilon=0,000001$, $\alpha=1$, $\beta=0,5$, $\gamma=2$.

Розв'язання:
Використаємо алгоритм із попереднього прикладу, внесемо деякі зміни:

1) Змінимо опис цільової функції:
```
function f(k: integer): real;
begin
  f := 2*sqr(x[k]) - 12*x[k] ;
end;
```
2) Змінимо крок виведення результатів ітерацій: if it mod 1 = 0 then
та форму виведення (приберемо з виводу змінну Y);

Отримуємо наступні результати:

it= 6	x= 2.1656	f=-17.9914
it= 7	x= 2.5500	f=-17.9914
it= 8	x= 2.7422	f=-17.9914
it= 9	x= 2.8996	f=-17.9935
it= 10	x= 2.9477	f=-17.9935
it= 11	x= 2.9717	f=-17.9935
it= 12	x= 2.9837	f=-17.9967
it= 13	x= 2.9960	f=-17.9984
it= 14	x= 2.9808	f=-17.9997
it= 15	x= 2.9960	f=-17.9998
it= 16	x= 2.9987	f=-17.9999
it= 17	x= 2.9955	f=-18.0000
it= 18	x= 2.9987	f=-18.0000
it= 19	x= 2.9992	f=-18.0000
it= 20	x= 2.9996	f=-18.0000
it= 21	x= 2.9999	f =-18.0000
it= 22	x= 2.9995	f=-18.0000
it= 23	x= 2.9996	f=-18.0000
it= 23	x= 2.9996	f=-18.0000

Отже для розв'язання даної задачі нам знадобилося 23 ітерацій.

Час виконання завдання 1 година.

Заключення

Прямі методи або методи нульового порядку не вимагають знання цільової функції в явному вигляді. Вони не вимагають регулярності та неперервності цільової функції та існування похідних. Це є суттєвою перевагою при вирішенні складних технічних і економічних завдань.

Завідувач кафедри вищої математики,
математичного моделювання та фізики
кандидат фізико-математичних наук, доцент І.В. Замрій