7 Ćwiczenie 6
Minimalizacja funkcji dwóch zmiennych

7.1 Wprowadzenie

Minimalizacja funkcji

$$f(x_1, x_2, \ldots, x_n)$$

$$f : \mathbb{R}^n \to \mathbb{R}$$

polega na szukaniu minimum funkcji w następujący sposób:

1.

$$\mathcal{D}f = \begin{pmatrix} \frac{\partial}{\partial x_1} \\ \frac{\partial}{\partial x_2} \\ \vdots \\ \frac{\partial}{\partial x_n} \end{pmatrix} f = 0 \Rightarrow \begin{pmatrix} \frac{\partial f}{\partial x_1} = 0 \\ \frac{\partial f}{\partial x_2} = 0 \\ \vdots \\ \frac{\partial f}{\partial x_n} = 0 \end{pmatrix} \Rightarrow X_0$$

2.

$$\mathcal{D}^2 f = \begin{pmatrix} \frac{\partial^2}{\partial x_1^2} & \frac{\partial^2}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} \\ \frac{\partial^2}{\partial x_1 \partial x_2} & \frac{\partial^2}{\partial x_2^2} & \cdots & \frac{\partial^2}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2}{\partial x_1 \partial x_n} & \frac{\partial^2}{\partial x_2 \partial x_n} & \cdots & \frac{\partial^2}{\partial x_n^2} \end{pmatrix} f$$

Jeżeli:

$$det(\mathcal{D}^2 f)|_{x \in X_0}$$

(a) $= 0$ to badany punkt nie jest ekstremum

(b) > 0 to badany punkt jest minimum

(c) < 0 to badany punkt jest maksimum

Cele ćwiczenia:

- minimalizacja funkcji dwóch zmiennych bez ograniczeń metodami:
  - naiwną
    (polega na wykonywaniu małych kroków w kierunku spadku, aż do momentu gdy wszystkie drogi prowadzą już tylko w górę)
  - większych kroków
    (podobna do poprzedniej, ale wykonujemy większe kroki w kierunku spadku, dodatkowo oceniając kształt funkcji)
  - **dfp** Davidon, Fletcher, Powell
    (wykorzystujemy informację o kształcie funkcji z poprzedniego kroku)
- poznanie nowych komend **Scilaba**

Minimalizowane zostaną trzy poniższe funkcje, które są określone na całej płaszczyźnie, ale nasze obserwacje dotyczą obszaru [-1,1]×[-1,1]:

1. $$f(x_1, x_2) = 3 \cdot x_1^2 + x_2^2$$

   funkcja ta posiada minimum w punkcie (0,0) o wartości $f(0,0) = 0$

2. $$f(x_1, x_2) = 3 \cdot (x_2 - x_1^2)^2 + 0.1 \cdot x_2^2$$

   funkcja ta posiada minimum w punkcie (0,0) o wartości $f(0,0) = 0$

3. $$f(x_1, x_2) = (0.25 - x_1^2 - x_2^2)^2 + 0.1 \cdot x_1 + 0.05$$

   funkcja ta posiada minimum w punkcie (-0.5,0) o wartości $f(-0.5, 0) = 0$

7.2 Tworzenie wykresów poziomicowych funkcji

1. tworzymy funkcję **function [f,Df]=cel(x1,x2)**, która każdemu wektorowi x=[x_1, x_2] przyporządkowuje wartość funkcji f oraz gradientu $\mathcal{D}f = [\partial/\partial x_1, \partial/\partial x_2]$:

```
function [f,Df]=cel(x1,x2)
f=3*x1^2+x2^2;
Df=[6*x1,2*x2];
//f=3*(x2-x1^2)^2+0.1*x2^2;
//Df=[-12*x1*x2+12*x1^3,6.2*x2-6*x1^2];
//f=(0.25-x1^2-x2^2)^2+0.1*x1+0.05;
//Df=[-4*(0.25-x1^2-x2^2)*x1+0.1,-4*(0.25-x1^2-x2^2)*x2];
```

2. piszemy i wykonujemy skrypt **mapa.sci**, który tworzy wykres poziomicowy funkcji f na prostokącie [-1,1]×[-1,1]:

```
xx0=-1:0.1:1;
yy0=xx0;
xx=ones(21,1)*xx0;
yy=yy0'*ones(1,21);
ff=3*xx.*xx+yy.*yy;
//ff=3*(yy-xx.*xx).*(yy-xx.*xx)+0.1*(xx.*xx);
//ff=(0.25-xx.*xx-yy.*yy).*(0.25-xx.*xx-yy.*yy)+0.1*xx;
xbasc()
contour2d(xx0,yy0,ff',[0.01,0.05,0.25,0.5,1,2,3,4]);
```

gdzie:
`xx` - macierz wartości x_1
`yy` - macierz wartości x_2
contour2d(X1,X2,F,Z) - komenda, która rysuje krzywe w dwóch wymiarach:
X1,X2 - wektory o długościach n_1, n_2 zawierające wartości x_1, x_2 w punktach podziału boków prostokąta
F - macierz $n_1 \times n_2$ wartości funkcji
Z - wektor określający wartości cięć poziomicowych

7.3 Algorytmy minimalizacji funkcji

Poznamy trzy algorytmy minimalizacji funkcji, każdy z nich deklarujemy w postaci:

```
function [x01,x02,f0,iter]=nazwa(x01,x02,maxiter,krok)
getf('cel.sci','c');
xbasc()
exec('mapa.sci',0);
⋮
```

gdzie:
Parametry wejściowe:
x0=[x01,x02] - wektor określający punkt startowy
maxiter - maksymalna ilość obliczeń funkcji celu
krok - współczynnik długości kroku, który jest dalej opisywany

Otrzymane wyniki:
x0=[x01,x02] - punkt końcowy, w którym jest spełnione kryterium stopu lub przekroczona jest maksymalna ilość iteracji
f0 - wartość funkcji celu w punkcie końcowym
iter - ilość iteracji wykonanych do momentu zatrzymania

1. tworzymy funkcję o nazwie **naive.sci**, która ma następujące własności:
   (a) bierze jako punkt początkowy kroku punkt startowy
   (b) zeruje licznik iteracji
   (c) oblicza wartość funkcji celu i jej gradientu w punkcie początkowym
   (d) oblicza normę gradientu funkcji celu za pomocą funkcji:

   norma wektora=norm(wektor)

   w następujący sposób:
   - gdy norma gradientu jest większa niż 0.001 oraz ilość iteracji nie przekracza **maxiter** wykonuje krok w kierunku minus gradientu:
     ```
     x11=x01-krok*g(1,1)
     x12=x02-krok*g(1,2)
     ```
   - w nowym miejscu oblicza funkcję celu i jej gradient
   - zwiększa licznik iteracji

- rysuje odcinek pomiędzy punktami $x0$=$[x0_1, x0_2]$ i $x1$=$[x1_1, x1_2]$
- przyjmuje punkt końcowy jako punkt początkowy
- uaktualnia bieżącą wartość funkcji celu i jej gradientu

(e) wykonujemy powyższą funkcję dla trzech badanych funkcji dwóch zmiennych z osobna za pomocą komend:
`--> getf('naive.sci');`

dla pierwszej funkcji
`--> [x01,x02,f0,iter]=naive(1,1,2000,0.001);`

dla drugiej funkcji
`--> [x01,x02,f0,iter]=naive(1,1,20000,0.01);`

dla trzeciej funkcji
`--> [x01,x02,f0,iter]=naive(1,1,20000,0.01);`

(f) eksperymentujemy z różnymi wartościami parametrów wejściowych dla badanych funkcji

2. tworzymy funkcję o nazwie **better.sci**, która ma własności podobne do funkcji **naive.sci** z następującymi wyjątkami:

(a) próbny krok wykonujemy do punktu $x1 = x0 - t \cdot g$

(b) $g = \mathcal{D}f(x0)$

$$\psi(t) = f(x0 - t \cdot g)$$

Zatem:

$$\frac{d}{dt}\psi(t)|_{(t=0)} = -\|g\|^2$$

(c) wykorzystujemy wiedzę o wartościach funkcji $f(0)$, $f(1)$

$$\psi(0) = f0$$

$$\psi(1) = f1$$

(d) dokonujemy aproksymacji funkcji $\psi(t)$ funkcją kwadratową

$$\psi(t) \approx \bar{\psi}(t) = a \cdot t^2 - b \cdot t + c$$

gdzie:
$b = \|g\|^2$
$a = f1 - f0 + b$

(e) dokonujemy analizy przebiegu funkcji dla różnych wartości a

- gdy $a \approx 0$ to $\psi(t)$ jest prawie funkcją liniową $\rightarrow$ wykonujemy krok o długości 1 za pomocą następujących instrukcji:
  ```
  x11=x01-g(1,1);
  x12=x02-g(1,2);
  ```

- gdy $a < 0$ to parabola leży pod prostą $\rightarrow$ wykonujemy krok o długości 1

- gdy $a > 0$ to próbujemy od razu wskoczyć do minimum paraboli $\rightarrow$ wykonujemy krok o długości $b/(2 \cdot a)$ za pomocą następujących instrukcji:
  ```
  if a>0.125*b then
  t=b/(2*a);
  else
  t=4;
  end
  ```

3. tworzymy funkcję o nazwie **dfp.sci** za pomocą następujących instrukcji:

```
function [x01,x02,f0,iter]=dfp(x01,x02,maxiter)
getf('cel.sci','c');
xbasc()
exec('mapa.sci',0);
[f0,Df0]=cel(x01,x02);
iter=0;
kryt=norm(Df0);
g=Df0;
D=eye(2,2);
dd=g*D';
t=1;
x11=x01-t*dd(1,1);
x12=x02-t*dd(1,2);
x1=[x11,x12];
[f1,Df1]=cel(x11,x12);
iter=iter+1;
while f1>f0 do
t=0.25*t;
x11=x01-t*dd(1,1);
x12=x02-t*dd(1,2);
x1=[x11,x12];
```

```
[f1,Df1]=cel(x11,x12);
iter=iter+1;
end
P1=[x01;x11]';
P2=[x02;x12]';
xsegs(P1,P2);
x0=[x01,x02];
x1=[x11,x12];
d=x1-x0;
y=Df1-Df0;
Df0=Df1;
x01=x11;
x02=x12;
x0=[x01,x02];
f0=f1;
kryt=norm(Df1);
while kryt>0.001 & iter<maxiter do
z=y*D';
if abs(d*y')<0.00001 | abs(y*z')<0.00001 then
D=eye(2,2);
else
a=1.0/(d*y');
b=-1.0/(y*z');
D=D+a*(d'*d)+b*(z'*z);
end
g=Df0;
dd=g*D';
t=1;
x11=x01-t*dd(1,1);
x12=x02-t*dd(1,2);
x1=[x11,x12];
[f1,Df1]=cel(x11,x12);
iter=iter+1;
while f1>f0 do
t=0.25*t;
x11=x01-t*dd(1,1);
x12=x02-t*dd(1,2);
x1=[x11,x12]; [f1,Df1]=cel(x11,x12);
iter=iter+1;
end
P1=[x01;x11]';
```

```
P2=[x02;x12]';
xsegs(P1,P2);
x0=[x01,x02];
x1=[x11,x12];
d=x1-x0;
y=Df1-Df0;
x01=x11;
x02=x12;
x0=[x01,x02];
f0=f1;
Df0=Df1;
kryt=norm(Df1);
end
```

4. dokonujemy analizy otrzymanych wyników dla poszczególnych algorytmów