Feltételnélküli és feltételes optimalizálás SCILAB programcsomag segítségével

1. Feltételnélküli optimalizálás

1.1. Gradiens használata

Tekintsünk egy $f(x)$ többváltozó függvényt, amelynek minimumát szeretnénk meghatározni. Ennek a feladatnak a végrehajtására a Scilab az **optim()** függvényt használja. A parancs legegyszerűbb formája az alábbi, a későbbiekben visszatérünk a részletekre is.

[fopt,xopt]=optim(fvgr,x0)

ahol **fopt**, **xopt** változók a parancs kimenő paraméterei, amelyek rendre a függvény optimális (minimális) értékét és az optimális megoldásvektort jelentik. A parancs első bemenő paramétere egy, a felhasználó által definiálandó függvény, amely a minimalizálandó függvényt és annak gradiensét adja vissza, az **x0** bemenő paraméter pedig egy tetszőleges induló vektor.

1. példa

Legyen adott az

$$f(x) = x_1^2 + x_2^2 - 4x_1 + 6x_2 \rightarrow \min!$$

feltételnélküli optimalizációs feladat. A függvény gradiensvektora

$$\nabla f(x) = \begin{bmatrix} 2x_1 - 4 \\ 2x_2 + 6 \end{bmatrix}$$

Ezen a példán fogjuk bemutatni az **optim()** függvény használatát. Az optimalizálást végző **optim()** parancs kiadása előtt három függvényt kell definiálnunk: az optimalizálandó függvényt, annak gradiensét és az **optim()** parancs első paramétereként megadandó függvényt. A következőkben ezeket végezzük el, a függvények definiálásához a **function - endfunction** keretet használjuk.

A függvény definiálása (a függvény neve legyen f):

```
–>function z=f(x)
–>z=x(1)^2+x(2)^2-4*x(1)+6*x(2)
–>endfunction
```

A gradiensvektor mint függvény definiálása (a függvény neve legyen g). A gradiensvektort definiálhatjuk sorvektorként vagy oszlopvektorként is:

```
–>function z=g(x)
–>z=[2*x(1)-4,2*x(2)+6]
–>endfunction
```

Az együttes függvény definiálása (a függvény neve legyen *fvgr*). Itt meg kell jegyezni, hogy az **optim()** rutin használ egy skalár változót, amelyet a függvény-definicióban szerepeltetni kell, a változó neve legyen *ind*.

```
–>function [fv,gr,ind]=fvgr(x,ind)
–>fv=f(x)
–>gr=g(x)
–>endfunction
```

A függvények definiálása után már meghívhatjuk az **optim()** parancsot valamilyen induló vektorral. Amilyen típusú az induló vektor (sorvektor vagy oszlopvektor) ugyanolyan típusú lesz az optimális megoldásvektor is.

```
–>x0=[5;2]
x0 =
5.
2.
```

–>[fopt,xopt]=optim(fvgr,x0)
xopt =
2.
- 3.
fopt =
- 13.

–>x0=[5,2]
x0 =
5. 2.

–>[fopt,xopt]=optim(fvgr,x0)
xopt =
2. - 3.
fopt =
- 13.

Az alábbiakban megmutatjuk, hogy az **optim()** parancsnak milyen fontosabb paraméterei lehetnek. A parancs általánosabb alakja a következő. Nem minden paramétert vettünk fel, csupán a fontosabbakat.

[fopt,xopt,gopt]=optim(fvgr,korlatok,x0,algo)

ahol **gopt** kimenő paraméter jelenti az optimális megoldáshoz tartozó gradiens vektor értékét (amilyen típusú az induló vektor (sorvektor vagy oszlopvektor) ugyanolyan típusú lesz az gradiens vektor is), a bemenő paraméterek pedig rendre a következők lehetnek:

korlatok: itt adhatjuk meg az x döntési változóra vonatkozó egyedi alsó és felső korlátokat, három paramétert kell megadni, rendre: **'b',xa,xf**, a bevezető paraméter kötelezően **'b'** (bound=korlát), a második és a harmadik paraméter az egyedi alsó és felső korlát, amelyek típusa meg kell egyezzen az induló vektor típusával.

algo: ez a paraméter egy sztring, amellyel a megoldandó algoritmus fajtáját adhatjuk meg, az alábbi három paraméter lehet:

'qn' : kvázi Newton módszer használata esetén, ez a Scilab alapbeállítása
'gc' : konjugált gradiens módszer használata esetén

Az alábbiakban az egyes paraméterek használatára mutatunk be példákat, a fenti optimalizálási feladatot oldjuk meg változatlanul.

A gradiens kimenő paraméter használata:
–>[fopt,xopt,gopt]=optim(fvgr,x0)
gopt =
1.0D-14 *
- 0.3552714
- 0.1776357
xopt =
2.
- 3.
fopt =
- 13.

A gradiens vektor értéke az optimális pontban zérusvektornak tekinthető, azaz $\nabla f(xopt) = (0, 0)$, az optimalitás szükséges feltételét kaptuk meg.

A megoldó algoritmus típusának beállítása:
–>[fopt,xopt]=optim(fvgr,x0,'gc')
xopt =
2.
- 3.
fopt =

- 13.

Egyedi alsó és felső korlátok használata:
A megoldandó optimalizálási feladat legyen a következő:

$$\begin{aligned} f(x) &= x_1^2 + x_2^2 - 4x_1 + 6x_2 \rightarrow \min! \\ 3 &\leqq x_1 \leqq 5 \\ 1 &\leqq x_2 \leqq 4 \end{aligned}$$

–>x=[7;2]
x =
7.
2.

–>xa=[3;1]
xa =
3.
1.

–>xf=[5;4]
xf =
5.
4.

–>[fopt,xopt]=optim(fvgr,'b',xa,xf,x0)
xopt =
3.
1.
fopt =
4.

–>[fopt,xopt,gopt]=optim(fvgr,'b',xa,xf,x0)
gopt =
2.
8.
xopt =
3.
1.
fopt =
4.

Mint látható, az egyedi korlátok befolyásolták az optimális értéket, ez a gradiensvektor nemzérus voltából is következik.

Megjegyezzük, hogy az optim() függvény legegyszerűbb hívása: **[fopt]=optim(fvgr,x0)**

Végezetül az alábbi megjegyzést tesszük. A szükséges függvények definiálását végezhettük volna a **deff()** függvénnyel is, sőt egyetlen paranccsal is lehet definiálni a három szükséges függvényt az alábbi módon:

deff('fvgr *fv. def.***','f** *fv. def.***, g** *fv. def.***')**

A megvalósítás a következő:

–> deff('[f,g,ind]=fvgr(x,ind)','f=x(1)^2+x(2)^2-4*x(1)+6*x(2),g=[2*x(1)-4,2*x(2)+6]')

–>x0=[4,6]
x0 =
4. 6.

–>[fopt,xopt]=optim(fvgr,x0)
xopt =
2. - 3.
fopt =
- 13.

2. példa

A következőkben tekintsük az $f(x) = (x_1 - 2)^4 + (x_1 - 2x_2)^2 \rightarrow \min!$ optimalizációs feladatot. A függvény gradiensvektora

$$\nabla f(x) = \begin{bmatrix} 4(x_1 - 2)^3 + 2(x_1 - 2x_2) \\ -4(x_1 - 2x_2) \end{bmatrix}$$

A megvalósítás a következőképpen történik:

–>deff('[f,g,ind]=fvgr(x,ind)','f=(x(1)-2)^4+(x(1)-2*x(2))^2,g=[4*(x(1)-2)^3+2*(x(1)-2*x(2)),-4*(x(1)-2*x(2))]')

–>x0=[12;5]
x0 =
12.
5.

–>[fopt,xopt]=optim(fvgr,x0)
xopt =
2.
1.
fopt =
2.601D-43

–>[fopt,xopt,gopt]=optim(fvgr,x0)
gopt =
1.0D-31 *
- 0.4606887
0.
xopt =
2.
1.
fopt =
2.601D-43

–>xa=[3;2]
xa =
3.
2.

–>xf=[5;6]
xf =
5.
6.

–>[fopt,xopt,gopt]=optim(fvgr,'b',xa,xf,x0)
gopt =
2.
4.
xopt =
3.
2.
fopt =
2.

3. példa

A következőkben tekintsük $f(x) = x_1^3 + x_2^3 - 27x_1 - 3x_2 \rightarrow \min!$ optimalizációs feladatot. A függvény gradiensvektora

$$\nabla f(x) = \begin{bmatrix} 3x_1^2 - 27 \\ 3x_2^2 - 3 \end{bmatrix}$$

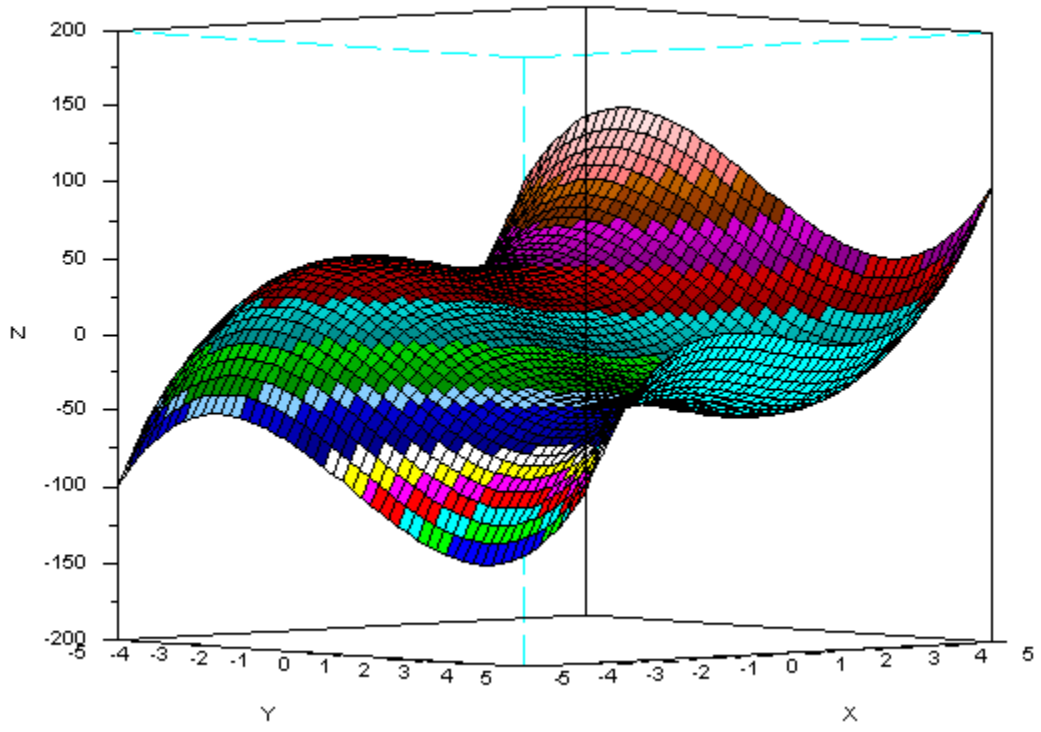
A megvalósítás a következőképpen történik:

```
–> deff('[f,g,ind]=fvgr(x,ind)','f=x(1)^3+x(2)^3-27*x(1)-3*x(2),g=[3*x(1)^2-27,3*x(2)^2-3]')

–>x0=[5;2]
x0 =
5.
2.

–>[fopt,xopt]=optim(fvgr,x0)
xopt =
3.
1.
fopt =
- 56.
```

Az alábbiakban bemutatjuk a függvény képét, ehhez az alábbi utasításokat adtuk ki:

```
–>deff('z=fv(x)','z=x^3+y^3-27*x-3*y')

–>x=[-5:0.1:5];
–>y=x;

–>fplot3d1(x,y,fv)
```

A függvény képe az alábbi ábrán látható. Megjegyezzük, hogy ennek a fügvénynek a (3,1) pontban minimuma, a (-3,-1) pontban maximuma, a (-3,1) és a (3,-1) pontokban nyeregpontja van, amelyek az ábrán is elég jól láthatók. A függvényértékek a fenti pontokban rendre -56, 56, 52, -52.



Az optimalizálásnál körültekintően kell eljárni, főleg az induló vektor megválasztásánál. Javasoljuk az olvasónak, hogy adja ki az optim() parancsot különböző induló vektorral. Nem igaz az, hogy bármilyen induló vektor esetén megkapjuk az optimális megoldást. Általában igaz, hogy az optimális megoldás közelében kell megválasztani az induló vektort. Ez egy kicsit ellentmondásnak tűnik, hiszen éppen az optimumot keressük, viszont a megoldandó gyakorlati feladatoknál általában van valamilyen előzetes elképzelésünk az optimum nagyságrendjéről.

4. példa

Végül tekintsük az $f(x) = 100 * (x_2 - x_1^2)^2 + (1 - x_1)^2 \rightarrow \min!$ optimalizációs feladatot. A függvényt Rosenbrock függvénynek nevezzük, amelyet algoritmusok tesztfüggvényeként szoktak használni. Szokásos elnevezése még a völgyfüggvény (térbeli alakja miatt) vagy a banánfüggvény (szinvonalai miatt). A függvény gradiensvektora

$$\nabla f(x) = \begin{bmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1) \\ 200(x_2 - x_1^2) \end{bmatrix}$$

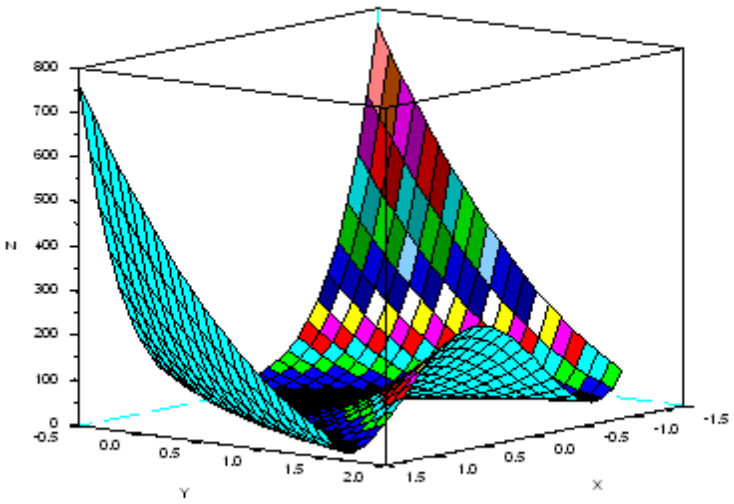
A megvalósítás a következőképpen történik:

```
–>deff('[f,g,ind]=fvgr(x,ind)','f=100*(x(2)-x(1)^2)^2+(1-x(1))^2,g=[-400*x(1)*(x(2)-x(1)^2)-2*(1-x(1)),
200*(x(2)-x(1)^2)]')

–>x0=[3;4]
x0 =
3.
4.

–>[fopt,xopt,gopt]=optim(fvgr,x0)
gopt =
0.
0.
xopt =
1.
1.
fopt =
0.

–>[fopt,xopt,gopt]=optim(fvgr,[56;-102],'gc')
gopt =
0.
0.
xopt =
1.
1.
fopt =
0.
```

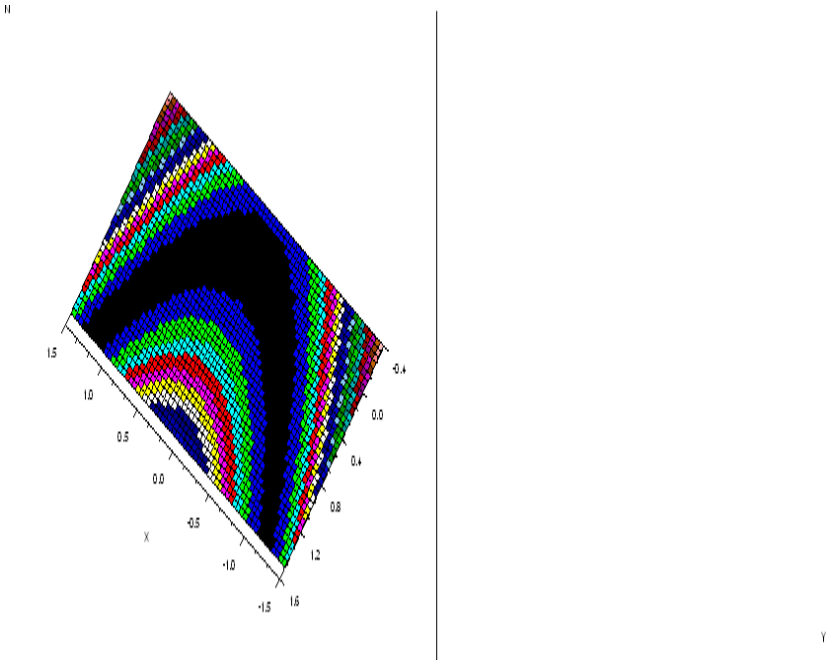
Az alábbiakban bemutatjuk a függvény képét, ehhez az alábbi utasításokat adtuk ki:

```
–>deff('z=f(x,y)','z=100*(y-x^2)^2+(1-x)^2')

–>x=[-1.5:0.1:1.5];

–>y=[-0.5:0.1:1.5];

–>fplot3d1(x,y,f)
```

Az alábbi ábrán látható a völgy, amelynek (1,1) pontjában helyezkedik el a minimumpont.



A következő ábrán pedig a szintvonalakat lehet látni, amelynél a fekete jelzi a mély völgyet, egy "banán"-ra emlékeztet.



1.2. Gradiens helyettesítése véges differenciákkal

Az előzőekben használt **optim()** függvény használata igényelte a gradiens vektor megadását. Elképzelhető olyan optimalizálási feladat, amelynél a célfüggvény nem differenciálható vagy differenciálható ugyan, de túl bonyolult a gradiens megadása. Ezekre az esetekre szolgál az **optim()** függvény egy másik változata, amelynek legegyszerűbb formája az alábbi:

[fopt,xopt,gopt]=optim(list(NDcost,celfv),x0)

ahol **fopt**, **xopt, gopt** változók a parancs kimenő paraméterei, amelyek rendre a függvény optimális (minimális) értékét, az optimális megoldásvektort és az optimális megoldáshoz tartozó gradiens vektor értékét jelentik. A parancs első bemenő paramétere egy **list()** nevű Scilab függvény, amelynek két paramétere van, az első egy **NDcost** nevű szintén Scilab függvény, ennek segítségével számítja ki a program a gradienst, a második paraméter pedig a minimalizálandó függvény, az **x0** bemenő paraméter pedig egy tetszőleges induló vektor. Tehát a felhasználónak csak a célfüggvényt kell előállítani.

5. példa

A következőkben tekintsük az $f(x) = (x_1 - 2)^4 + (x_1 - 2x_2)^2 \rightarrow \min!$ optimalizációs feladatot. A függvénynek számítható a gradiense, de most anélkül végezzük el az optimalizálást. Definiáljuk a célfüggvényt, legyen a neve *cf*.

```
–>function z=cf(x)
–>z=(x(1)-2)^4+(x(1)-2*x(2))^2
–>endfunction

–>x0=[10;20];

–>[fopt,xopt,gopt]=optim(list(NDcost,cf),x0)
gopt =
1.0D-13 *
0.0487434
- 0.1021405
xopt =
1.9999999
1.0000000
fopt =
3.363D-29
```

Láthatjuk, hogy a gradiensvektor is kiszámítódik, de nem a tényleges értéket kapjuk, hanem véges differencia módszerrel történő közelítő értéket számol a gép.

6. példa

Most tekintsük a már megismert Rosenbrock függvényt, de módosítsuk egy kicsit, a második tagban az x_1 változó együtthatója legyen p. A feladat tehát

$$f(x) = 100 * (x_2 - x_1^2)^2 + (1 - px_1)^2 \rightarrow \min!$$

A függvény definíciója a paramétert is figyelembe véve:

```
–>function z=rf(x,p)
–>z=(x(2)-x(1)^2)^2+(1-p*x(1))^2
–>endfunction
```

Az **optim()** függvény paraméterek használatát is lehetővé teszi, ezeket a **list()** függvény harmadik bemenő paramétereként kell megadni, ekkor a függvény hívása:

[fopt,xopt,gopt]=optim(list(NDcost,celfv,par),x0)

Több paraméter is elképzelhető, természetesen vektorokat is használhatunk. Ez a lehetőség hasznos lesz a feltételes optimalizálásnál, amint később látni fogjuk.

Az eredeti Rosenbrock függvény optimalizálása a $p = 1$ paraméterértékkel történik:

```
–>[fopt,xopt,gopt]=optim(list(NDcost,rf,1),x0)
gopt =
1.0D-10 *
- 0.3415834
0.4950063
xopt =
1.
1.
fopt =
2.287D-21
```

Egy másik hívás, ha a paraméter értéke 0.5 .

```
–>[fopt,xopt,gopt]=optim(list(NDcost,rf,0.5),x0)
gopt =
1.0D-09 *
- 0.2102988
0.0550089
xopt =
2.
4.
fopt =
8.119D-20
```

2. Feltételes optimalizálás

A feltételes optimalizálási feladat általános alakja a következő:

$$\begin{array}{ll} f(\mathbf{x}) \rightarrow \min! & \\ g_i(\mathbf{x}) \leqq 0 & i = 1, 2, \ldots, m \\ h_i(\mathbf{x}) = 0 & i = 1, 2, \ldots, k \end{array}$$

A feladat megoldására szolgáló módszer alapgondolata az, hogy a feltételes optimalizálási feladat megoldását visszavezetjük feltételnélküli optimalizálási feladatok sorozatos megoldására. Ezt a megoldási eljárást szokás SUMT (Sequential Unconstrained Minimization Technique) módszer néven is emlegetni. Ennek egyik változata a büntetőfüggvényes eljárás. A módszer lényege tehát az, hogy a feltételeket beépítjük a célfüggvénybe, így kapunk egy feltételnélküli optimalizálási feladatot, amelyet segédfeladatnak szokás nevezni. Az alábbi segédfüggvényt használják leggyakrabban:

$$\phi(\mathbf{x},\mu) = f(\mathbf{x}) + \mu \left[\sum_{i=1}^{m} \left[\max \{0, g_i(\mathbf{x})\} \right]^2 + \sum_{i=1}^{k} \left[h_i(\mathbf{x}) \right]^2 \right]$$

A segédfüggvény tartalmaz egy pozitív értékű ún. büntetőparamétert (μ). A segédfüggvényben a célfüggvény utáni tagot büntetőtagnak nevezzük. A büntetőfüggvényes eljárás algoritmusa röviden az alábbi:

Kiindulunk valamilyen kezdő vektorral és egy kezdő büntetőparaméterrel.

A további lépésekben az alábbi két feladatot ismételjük:

– Meghatározzuk a segédfüggvény minimumát feltételnélküli optimalizálással.

– Növeljük a büntetőparaméter értékét.

Tehát sorozatban oldunk meg egy-egy feltételnélküli optimalizálási feladatot, növekvő büntetőparaméter értékekkel. Ahogy a büntetőparaméterek tartanak a végtelenhez, úgy tartanak a feltételnélküli optimális megoldások a feltételes optimális megoldáshoz.

Általában a kezdő büntetőparaméter értékét 1-re választjuk, a következő lépésekben pedig értéke: 10, 100, 1000, ...

Azért, hogy hamarabb eljussunk az optimális megoldáshoz, célszerű a feltételnélküli optimalizálásoknál a kezdő vektort az előző feltételnélküli optimalizálási feladat optimális megoldásának választani. Természetesen ezek nem kötelező érvényűek.

7. példa

A következőkben tekintsük az alábbi feltételes optimalizációs feladatot, amely egyetlen egy egyenlőséges feltételt tartalmaz:.

$$\begin{array}{rcl} f(x) & = & (x_1 - 2)^4 + (x_1 - 2x_2)^2 \rightarrow \min! \\ h(x) & = & x_1^2 - x_2 = 0 \end{array}$$

A feladathoz tartozó segédfüggvény:

$$\phi(\mathbf{x},\mu) = (x_1 - 2)^4 + (x_1 - 2x_2)^2 + \mu(x_1^2 - x_2)^2$$

A segédfüggvény neve legyen *segedfv*. A függvénydefinicióban paraméterként a büntetőparamétert fogjuk használni.

```
-->function z=segedfv(x,bp)
-->z=(x(1)-2)^4+(x(1)-2*x(2))^2+bp*(x(1)^2-x(2))^2
-->endfunction

-->x0=[10;20];
```

A fenti előkészületek után meghívjuk az **optim()** parancsot, amelyben a büntetőparaméter értéke 1. Majd újra meghívjuk a növelt büntetőparaméterekkel.

```
-->[fopt,xopt,gopt]=optim(list(NDcost,segedfv,1),x0)
gopt =
1.0D-09 *
- 0.0183343
0.1925098
xopt =
1.1687246
0.7406733
fopt =
0.9661694

-->[fopt,xopt,gopt]=optim(list(NDcost,segedfv,10),x0)
gopt =
- 1.903D-08
9.350D-09
xopt =
0.9906151
0.8424581
fopt =
1.7129469

-->[fopt,xopt,gopt]=optim(list(NDcost,segedfv,100),x0)
gopt =
0.0000002
- 6.556D-08
xopt =
0.9507637
0.8874682
fopt =
1.9184047

-->[fopt,xopt,gopt]=optim(list(NDcost,segedfv,1000),x0)
gopt =
1.0D-10 *
- 0.1833426
- 0.3666853
xopt =
0.9461094
0.8934415
fopt =
1.9433497
```

A növekvő büntetőparaméterrel történő hívások sorozatából is szépen látható, hogy a feltételnélküli optimalizálási feladatok optimális megoldásai konvergálnak, mégpedig a feltételes optimalizálási feladat optimális megoldásához.

Az előző hívásokban mindig ugyanazzal a kezdővektorral dolgoztunk, az alábbiakban az előző optimális megoldást használjuk kezdővektornak, ehhez az x0=xopt értékadást kell kiadni minden hívás előtt.

```
-->x0=[10;20];
-->[fopt,xopt,gopt]=optim(list(NDcost,segedfv,1),x0)
gopt =
1.0D-09 *
- 0.0183343
0.1925098
xopt =
1.1687246
0.7406733
fopt =
0.9661694

-->x0=xopt;
-->[fopt,xopt,gopt]=optim(list(NDcost,segedfv,10),x0)
gopt =
- 5.564D-08
3.522D-08
xopt =
0.9906151
0.8424581
fopt =
1.7129469

-->x0=xopt;
-->[fopt,xopt,gopt]=optim(list(NDcost,segedfv,100),x0)
gopt =
1.0D-09 *
0.4950251
- 0.2016769
xopt =
0.9507637
0.8874682
fopt =
1.9184047

-->x0=xopt;
-->[fopt,xopt,gopt]=optim(list(NDcost,segedfv,1000),x0)
gopt =
1.0D-09 *
- 0.1100056
0.0733371
xopt =
0.9461094
0.8934415
fopt =
1.9433497
```

A jobb közelítő optimális megoldás eléréséhez több büntetőparaméterrel szoktuk meghívni a rutint. Látható, hogy ez fáradságosabb munka. Szerencsére a Scilab programcsomagban programozási lehetőség is van. Az alábbiakban ezt mutatjuk be röviden. Nem célunk, hogy megismertessük az olvasót a programozás minden lehetőségével, csupán az optimalizálási feladatunk megoldásához szükséges alapvető programelemekre koncentrálunk. A többszörös hívást az új büntetőparaméterrel és a kezdővektor beállítást ciklusban végezhetjük, ezt egyszerűen megvalósíthatjuk a Scilab **for** ciklusutasítása segítségével. A **for** ciklusutasítás első sorában (ciklusfej) a ciklusváltozó értékeit adjuk meg, majd ezt követi a ciklusmag (itt írjuk le az ismételten végrehajtandó utasításokat), majd a ciklust az **end** alapszó zárja le. Az alábbiakban a ciklussal történő megvalósítást mutatjuk be.

Megjegyezzük, hogy a továbbiakban helytakarékosság miatt nem íratjuk ki a gradiensvektor értékét.

```
-->x0=[10;20];
```

```
–>for i=1:4
–>bp=10^(i-1)
–>[fopt,xopt,gopt]=optim(list(NDcost,segedfv,bp),x0)
–>x0=xopt;
–>end
```

A fenti parancs beírása után azonnal lefut a ciklus és az alábbiak kerülnek kiíratásra.

```
bp =
1.
xopt =
1.1687246
0.7406733
fopt =
0.9661694

bp =
10.
xopt =
0.9906151
0.8424581
fopt =
1.7129469

bp =
100.
xopt =
0.9507637
0.8874682
fopt =
1.9184047

bp =
1000.
xopt =
0.9461094
[More (y or n ) ?]
0.8934415
fopt =
1.9433497
```

Amennyiben sok sor íródna ki, a Scilab a
[More (y or n) ?]
kérdéssel megszakítja a kiiratást, amelyre az '**y**' válaszra folytatja, '**n**' válaszra pedig befejezi a kiiratást, ahogy ezt a fenti sorok is mutatják.

A programozási lehetőség, mint láttuk megkönnyíti a munkánkat, de a fenti eljárás sem a legelegánsabb megoldás. A Scilab lehetőséget kínál igazi programozásra is, amikor a parancssorainkat egy file-ba írhatjuk bele és azt egy végrehajtó utasítással elindíthatjuk. Az alábbiakban a programírás ezen változát mutatjuk be.

Első lépésként elkészítjük a file-t. Ezt valamilyen egyszerű szövegszerkesztővel is elvégezhetjük, de a Scilabnak is van egy **Editora**, amelynek neve **Scipad**. Ez az **Editor** menüből érhető el és megjelenik a Scilab szövegszerkesztő ablaka. Itt megírhatjuk a programunkat. A programunk legyen a következő:

```
function f=segedfv(x,bp)
f=(x(1)-2)^4+(x(1)-2*x(2))^2+bp*(x(1)^2-x(2))^2
endfunction

x0=[10;20];
```

```
for i=0:5
disp('==============================')
disp(i+1)
disp('iteráció')
disp('----------------------')
bp=10^i
[fopt,xopt]=optim(list(NDcost,segedfv,bp),x0)
x0=xopt;
end
```

A programot mentsük el a **save** paranccsal, legyen a neve *progr_ bf1.sce* , a Scilab szerint a kiterjeszése '**sce**' legyen. Ezután a prompt után meghívhatjuk a file-t, ennek parancsa a következő:

exec('file azonosítól', mod)

ahol az első paraméter egy sztring, amely a file-t azonosítja, tehát az elérési útat is meg kell adni, amennyiben nem az aktuális könyvtárban van a file, a második paraméter a futás módját szabályozza, el is hagyható.

A file futtattása:

```
->exec('c:\Documents and Settings\Rendszergazda\Dokumentumok\progr_bf1.sce')
```

Ennek kiadása után az alábbi futási eredmények jelennek meg a képernyőn

```
->function f=segedfv(x,bp)
-> f=(x(1)-2)^4+(x(1)-2*x(2))^2+bp*(x(1)^2-x(2))^2
->endfunction
->x0=[10;20];
->for i=0:5
-> disp('==============================')
-> disp(i+1)
-> disp('iteráció')
-> disp('----------------------')
-> bp=10^i
-> [fopt,xopt]=optim(list(NDcost,segedfv,bp),x0)
-> x0=xopt;
->end
==============================
1.
iteráció
----------------------
bp =
1.
xopt =
1.1687246
0.7406733
fopt =
0.9661694
==============================
2.
iteráció
----------------------
bp =
10.
xopt =
0.9906151
0.8424581
fopt =
```

1.7129469
==========
3.
iteráció
---
bp =
100.
xopt =
0.9507637
0.8874682
fopt =
1.9184047
==========
4.
iteráció
---
bp =
1000.
xopt =
0.9461094
0.8934415
fopt =
1.9433497
==========
5.
iteráció
---
bp =
10000.
xopt =
0.9456357
0.8940583
fopt =
1.9458997
==========
6.
iteráció
---
bp =
100000.
xopt =
0.9455879
0.8941196
fopt =
1.9461553

8. példa

A következőkben tekintsük az alábbi feltételes optimalizációs feladatot.

$$\begin{aligned} f(x) &= (x_1-2)^4 + (x_1-2x_2)^2 \rightarrow \min! \\ g(x) &= x_1^2 - x_2 \leqq 0 \end{aligned}$$

A feladathoz tartozó segédfüggvény:

$$\phi(\mathbf{x},\mu) = (x_1-2)^4 + (x_1-2x_2)^2 + \mu(\max(0, x_1^2 - x_2))^2$$

Ennek a feladatnak a megoldását már file-ba írt programmal végeztük el, felhasználtuk a Scilab **max()** függvényét is. A file neve: *progr_ bf2.sce*

```
function f=segedfv(x,bp)
f=(x(1)-2)^4+(x(1)-2*x(2))^2+bp*(max(0,x(1)^2-x(2)))^2
endfunction
x0=[10;20];
for i=0:5
disp('================================')
disp(i+1)
disp('iteráció')
disp('———————————')
bp=10^i
[fopt,xopt]=optim(list(NDcost,segedfv,bp),x0)
x0=xopt;
end
```

A programírás után következik a file futtatása a megismrt paranccsal:

```
–>exec('c:\Documents and Settings\Rendszergazda\Dokumentumok\progr_bf2.sce')
```

Ezekután pedig a végrehajtás eredményei következnek a képernyőn:

```
–>function f=segedfv(x,bp)
–> f=(x(1)-2)^4+(x(1)-2*x(2))^2+bp*(max(0,x(1)^2-x(2)))^2
–>endfunction
–>x0=[10;20];
–>for i=0:5
–> disp('================================')
–> disp(i+1)
–> disp('iteráció')
–> disp('———————————')
–> bp=10^i
–> [fopt,xopt]=optim(list(NDcost,segedfv,bp),x0)
–> x0=xopt;
–>end
================================
1.
iteráció
———————————
bp =
1.
xopt =
1.1687246
0.7406733
fopt =
0.9661694
================================
2.
iteráció
———————————
bp =
10.
xopt =
0.9906151
0.8424581
fopt =
1.7129469
================================
3.
iteráció
```

---

bp =
100.
xopt =
0.9507637
0.8874682
fopt =
1.9184047
==========
4.
iteráció

---

bp =
1000.
xopt =
0.9461094
0.8934415
fopt =
1.9433497
==========
5.
iteráció

---

bp =
10000.
xopt =
0.9456357
0.8940583
fopt =
1.9458997
==========
6.
iteráció

---

bp =
100000.
xopt =
0.9455879
0.8941197
fopt =
1.9461553

Korábban már jeleztük, hogy az **exec()** parancsnál használhatunk paramétert is, amely a végrehajtás módját vezérli, ha a paraméter értéke 2, akkor a program nem íródik ki, ezt mutatjuk be az alábbiakban. A helyszüke miatt csak két iteráció kiiratása látható:

–>exec('c:\Documents and Settings\Rendszergazda\Dokumentumok\progr_bf2.sce',2)
==========
1.
iteráció

---

bp =
1.
xopt =
1.1687246
0.7406733
fopt =
0.9661694

```
==============================
2.
iteráció
_____

bp =
10.
xopt =
0.9906151
0.8424581
fopt =
1.7129469
```

Az optimalizálási feladatok megoldási algoritmusainál többféle kilépési (megállási) kritériumot alkalmaznak, az **optim()** eljárásnak is van egyfajta megállási kritériuma. A mi programunkban is volt ilyen, azt írtuk elő, hogy hány ciklusvégrehajtás után álljon meg a program. Ez nem igazi megállási kritérium, hiszen a pontosságot nem tudtuk ezzel szabályozni. A következőkben olyan programot írunk, amely akkor áll meg, ha két egymást követő közelítő megoldásvektor abszolut értéke egy hibahatáron belül van. Legyen ez a hibakorlát: 0.000005 . Ennek megvalósítását egy másik ciklusutasítással, nevezetesen a **while ... end** ciklusutasítással oldjuk meg. A program neve legyen *progr_ bf3.sce*

A következőkben a file-t, a file futtatási parancsát és néhány iterációs lépést közlünk:

```
function f=segedfv(x,bp)
f=(x(1)-2)^4+(x(1)-2*x(2))^2+bp*(max(0,x(1)^2-x(2)))^2
endfunction
x0=[10;20];
xopt=2*x0;
i=0;
while (abs(x0-xopt)>0.000005)
x0=xopt;
disp('==============================')
disp(i+1)
disp('iteráció')
disp('_____')
bp=10^i
[fopt,xopt]=optim(list(NDcost,segedfv,bp),x0)
i=i+1;
end

->exec('c:\Documents and Settings\Rendszergazda\Dokumentumok\progr_bf3.sce')

->function f=segedfv(x,bp)
-> f=(x(1)-2)^4+(x(1)-2*x(2))^2+bp*(max(0,x(1)^2-x(2)))^2
->endfunction
->x0=[10;20];
->xopt=2*x0;
->i=0;
->while (abs(x0-xopt)>0.000005)
-> x0=xopt;
-> disp('==============================')
-> disp(i+1)
-> disp('iteráció')
-> disp('_____')
-> bp=10^i
-> [fopt,xopt]=optim(list(NDcost,segedfv,bp),x0)
-> i=i+1;
->end
==============================
1.
```

```
iteráció
 ---------------
bp =
1.
xopt =
1.1687246
0.7406733
fopt =
0.9661694
 ===============================
2.
iteráció
 ---------------
bp =
10.
xopt =
0.9906151
0.8424581
fopt =
1.7129469
 ===============================
3.
iteráció
 ---------------
bp =
100.
xopt =
[More (y or n ) ?]
```

9. példa

Végezetül tekintsünk egy egyenlőséget és egy egyenlőtlenséget is tartalmazó feltételes optimalizációs feladatot:

$$\begin{aligned} x_1^2 - 12x_1 + x_2^2 - 4x_2 &\rightarrow \min! \\ x_1^2 + x_2^2 &\leqq 25 \\ x_1 + 4x_2 &= 16 \end{aligned}$$

A feladathoz tartozó segédfüggvény:

$$\phi(\mathbf{x},\mu) = x_1^2 - 12x_1 + x_2^2 - 4x_2 + \mu((\max(0, x_1^2 + x_2^2 - 25))^2 + (x_1 + 4x_2 - 16)^2)$$

Ennek a feladatnak a megoldását is file-ba írt programmal végeztük el, a file neve: *progr_ bf4.sce*

A *progr_ bf4.sce nevű* file, a program indítása és a futási eredmények az alábbiak:

```
function z=segedfv(x,bp)
z=x(1)^2-12*x(1)+x(2)^2-4*x(2) + bp*( ( max(0,x(1)^2+x(2)^2-25) )^2 + (x(1)+4*x(2)-16)^2 )
endfunction
x0=[10;20];
xopt=2*x0;
i=0;
while (abs(x0-xopt)>0.00005)
x0=xopt;
disp('===============================')
disp(i+1)
disp('iteráció')
disp('---------------')
bp=10^i
[fopt,xopt]=optim(list(NDcost,segedfv,bp),x0)
```

```
i=i+1;
end

->exec('c:\Documents and Settings\Rendszergazda\Dokumentumok\progr_bf4.sce')

->function z=segedfv(x,bp)
-> z=x(1)^2-12*x(1)+x(2)^2-4*x(2) + bp*( ( max(0,x(1)^2+x(2)^2-25) )^2 + (x(1)+4*x(2)-16)^2
)
->endfunction
->x0=[10;20];
->xopt=2*x0;
->i=0;
->while (abs(x0-xopt)>0.00005)
-> x0=xopt;
-> disp('================================')
-> disp(i+1)
-> disp('iteráció')
-> disp('———————————')
-> bp=10^i
-> [fopt,xopt]=optim(list(NDcost,segedfv,bp),x0)
-> i=i+1;
->end
================================
1.
iteráció
———————————
bp =
1.
xopt =
4.1749438
2.8038263
fopt =
- 35.566208
================================
2.
iteráció
———————————
bp =
10.
xopt =
4.0223085
2.9757214
fopt =
- 35.069264
================================
3.
iteráció
———————————
bp =
100.
xopt =
4.0022997
2.9975075
fopt =
- 35.007096
================================
4.
iteráció
```

```
 -------------------
 bp =
 1000.
 xopt =
 4.0002307
 2.9997501
 fopt =
 - 35.000711
 ================================
 5.
 iteráció
 -------------------
 bp =
 10000.
 xopt =
 4.000023
 2.9999751
 fopt =
 - 35.000071
 ================================
 6.
 iteráció
 -------------------
 bp =
 100000.
 xopt =
 3.9999981
 2.9999985
 fopt =
 - 34.999989
```

A programot módosíthatjuk, hogy csak a program lefutása után írja ki az eredményeket. Kiiratjuk az optimális megoldást, a célfüggvény optimális értékét, a feltételi függvények értékeit, a büntető paraméter értékét, valamint az iterációk számát.

A program, a futtatás és a kiiratás az alábbiakban látható. A program írásánál vigyázzunk a parancsot lezáró pontosvessző használatára és az **exec()** parancsot is pontosvessző zárja le.

```
function z=segedfv(x,bp)
z=x(1)^2-12*x(1)+x(2)^2-4*x(2) + bp*( ( max(0,x(1)^2+x(2)^2-25) )^2 + (x(1)+4*x(2)-16)^2 )
endfunction
x0=[10;20];
xopt=2*x0;
i=0;
while (abs(x0-xopt)>0.00005)
x0=xopt;
bp=10^i;
[fopt,xopt]=optim(list(NDcost,segedfv,bp),x0);
i=i+1;
end
x=xopt;
disp('Az optimális megoldásvektor:')
disp(x);
disp('Az f(x) célfüggvény értéke:')
disp(x(1)^2-12*x(1)+x(2)^2-4*x(2))
disp('A büntetőtag értéke:')
disp(bp*( ( max(0,x(1)^2+x(2)^2-25) )^2 + (x(1)+4*x(2)-16)^2 ))
disp('A g(x) feltételi függvény értéke:')
disp(x(1)^2+x(2)^2-25)
```

```
disp('A h(x) feltételi függvény értéke:')
disp(x(1)+4*x(2)-16)
disp('A büntetőparaméter értéke:')
disp(bp)
disp('Az iterációk száma:')
disp(i)

-->exec('c:\Documents and Settings\Rendszergazda\Dokumentumok\progr_bf5.sce');

Az optimális megoldásvektor:
3.9999981
2.9999985
Az f(x) célfüggvény értéke:
- 34.999996
A büntetőtag értéke:
0.0000063
A g(x) feltételi függvény értéke:
- 0.0000241
A h(x) feltételi függvény értéke:
- 0.0000079
A büntetőparaméter értéke:
100000.
Az iterációk száma:
6.
```