Flervariabelanalys och Matlab
Kapitel 2

Thomas Wernstål
Carl-Henrik Fant
Matematiska Vetenskaper

16 september 2009

2 Ickelinjära ekvationssystem och optimering

2.1 Newtons metod för system av ekvationer

I detta avsnitt skall vi studera Newtons metod för lösning av ekvationssystem av typen

$$\begin{cases} f(x,y) = 0 \\ g(x,y) = 0 \end{cases}$$

Om funktionerna $f(x,y)$ och $g(x,y)$ är linjära så kan vi använda oss av kända metoder från linjär algebra kursen för att lösa systemet. Situationen är dock (i allmänhet) betydligt mer komplicerad om funktionerna inte är linjära. Låt oss först titta på härledningen av Newtons metod i en variabel dvs för lösning av en ekvation med en obekant. Antag att $\tilde{x}$ ligger nära en lösning x^* till ekvationen $f(x) = 0$. Om vi Taylorutvecklar $f(x)$ kring $\tilde{x}$ t.o.m. första ordningen så får vi $f(x) \approx f(\tilde{x}) + f'(\tilde{x})(x - \tilde{x})$. Denna approximation stämmer bra i en nära omgivning av $\tilde{x}$ och då speciellt för $x = x^*$ vilket ger oss att

$$0 = f(x^*) \approx f(\tilde{x}) + f'(\tilde{x})(x^* - \tilde{x}) \quad \Leftrightarrow$$

$$f'(\tilde{x})(x^* - \tilde{x}) \approx -f(\tilde{x})$$

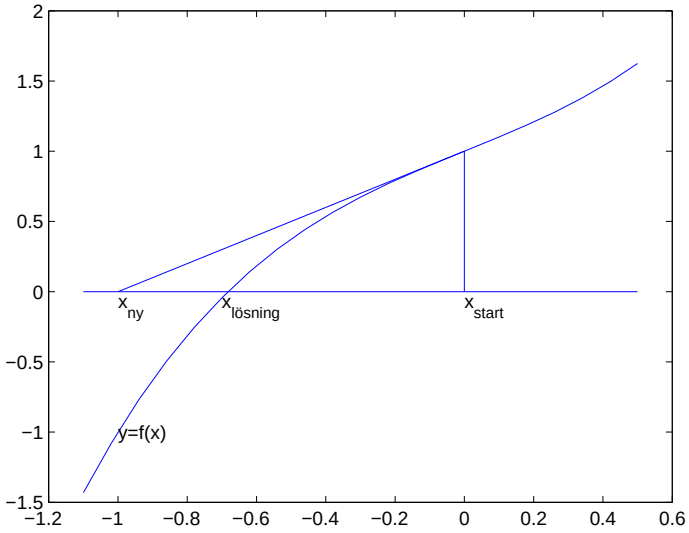
Om vi löser ut x^* så får vi att

$$x^* \approx \tilde{x} - \frac{f(\tilde{x})}{f'(\tilde{x})}$$

Felet i denna approximation är av storleksordningen $|x^* - \tilde{x}|^2$ (ty nästa term i Taylorutvecklingen är av den storleksordningen) dvs. betydligt mindre än det ursprungliga felet $|x^* - \tilde{x}|$. Så $\hat{x} = \tilde{x} - \frac{f(\tilde{x})}{f'(\tilde{x})}$ ger alltså en ännu bättre approximation av x^*. Såvida man inte känner sig nöjd med denna förbättrade approximation så är det högst naturligt att tänka sig upprepa proceduren för att ytterligare förfina approximationen. Newtons metod innebär att man på detta sätt succesivt itererar sig närmare en lösning på ekvationen.

Metoden kan också åskådliggöras geometriskt. Lösningen på ekvationen $f(x) = 0$ är det x-värde för vilket grafen $y = f(x)$ skär x-axeln. När vi ersätter $f(x)$ med dess Taylorutveckling i $\tilde{x}$ t.o.m. första graden i ekvationen så innebär det att vi istället undersöker var

tangenten till $y = f(x)$ i punkten $(\tilde{x}, f(\tilde{x}))$ skär x-axeln. Denna skärningspunkt blir sedan ny utgångspunkt för nästa iteration. Följande figur illustrerar ett steg i Newtons metod.



Vi kan nu på liknande sätt härleda en metod för att lösa system av ekvationer. Antag att vi vill lösa system av typen

$$\begin{cases} f(x,y) = 0 \\ g(x,y) = 0 \end{cases}$$

Om $(\tilde{x}, \tilde{y})$ ligger nära en lösning (x^*, y^*) till ekvationssystemet så har vi

$$\begin{cases} 0 = f(x^*, y^*) \approx f(\tilde{x}, \tilde{y}) + f'_x(\tilde{x}, \tilde{y})(x^* - \tilde{x}) + f'_y(\tilde{x}, \tilde{y})(y^* - \tilde{y}) \\ 0 = g(x^*, y^*) \approx g(\tilde{x}, \tilde{y}) + g'_x(\tilde{x}, \tilde{y})(x^* - \tilde{x}) + g'_y(\tilde{x}, \tilde{y})(y^* - \tilde{y}) \end{cases}$$

Detta är ett linjärt ekvationssystem i de obekanta x^* och y^* (om vi betraktar $(\tilde{x}, \tilde{y})$ som känd). Sambanden kan också uttryckas på matrisform enl.

$$\begin{bmatrix} f'_x(\tilde{x}, \tilde{y}) & f'_y(\tilde{x}, \tilde{y}) \\ g'_x(\tilde{x}, \tilde{y}) & g'_y(\tilde{x}, \tilde{y}) \end{bmatrix} \begin{bmatrix} x^* - \tilde{x} \\ y^* - \tilde{y} \end{bmatrix} \approx \begin{bmatrix} -f(\tilde{x}, \tilde{y}) \\ -g(\tilde{x}, \tilde{y}) \end{bmatrix}$$

Felet i denna approximation är av storleksordningen $||(x^*, y^*) - (\tilde{x}, \tilde{y})||^2$, vilket är avsevärt mindre än $||(x^*, y^*) - (\tilde{x}, \tilde{y})||$ som vi hade från början. Lösningen $(\hat{x}, \hat{y})$ på ekvationen

$$(*) \qquad \begin{bmatrix} f'_x(\tilde{x}, \tilde{y}) & f'_y(\tilde{x}, \tilde{y}) \\ g'_x(\tilde{x}, \tilde{y}) & g'_y(\tilde{x}, \tilde{y}) \end{bmatrix} \begin{bmatrix} \hat{x} - \tilde{x} \\ \hat{y} - \tilde{y} \end{bmatrix} = \begin{bmatrix} -f(\tilde{x}, \tilde{y}) \\ -g(\tilde{x}, \tilde{y}) \end{bmatrix}$$

bör alltså ge en bättre approximation till lösningen (x^*, y^*) än vad $(\tilde{x}, \tilde{y})$ gav.

Geometriskt innebär metoden följande. Lösningen på systemet $\begin{cases} f(x,y)=0 \\ g(x,y)=0 \end{cases}$ är den punkt (x^*, y^*) i vilket funktionsytorna $z = f(x,y)$ och $z = g(x,y)$ skär varandra i xy-planet. När vi ersätter $f(x,y)$ och $g(x,y)$ med respektive Taylorutveckling i $(\tilde{x}, \tilde{y})$ t.o.m. första graden i systemet så innebär det att vi istället undersöker var tangentplanen till $z = f(x,y)$ och $z = g(x,y)$ där $(x,y) = (\tilde{x}, \tilde{y})$ skär varandra i xy-planet. Denna skärningspunkt blir sedan ny utgångspunkt för nästa iteration och man upprepar proceduren tills båda funktionsvärdena är tillräckligt små.

Låt oss nu titta på ett exempel

Exempel. Antag att vi vill hitta en lösning (x^*, y^*) till ekvationssystemet

$$\begin{cases} xy(x-y) = 1 \\ x^3y^2 + x^2 + y^4 = 3 \end{cases}$$

Ett sätt att försöka hitta lämpliga startvärden för Newtons metod är att plotta nivåkurvorna $xy(x-y) - 1 = 0$ och $x^3y^2 + x^2 + y^4 - 3 = 0$

```
>> f=@(x,y) x.*y.*(x-y)-1; g=@(x,y) x.^3.*y.^2+x.^2+y.^4-3;
>> x=linspace(-3,3,30);y=linspace(-3,3,31);
>> [X,Y]=meshgrid(x,y);
>> contour(X,Y,f(X,Y),[0 0],'r'), hold on
>> contour(X,Y,g(X,Y),[0 0],'b'), hold off
```

Vi ser då att det finns en lösning (skärningspunkt) i närheten av $(-0.5, 1.2)$. Vi använder sedan Newtons metod för att förbättra denna (grova) approximation av lösningen.

```
>> fx=@(x,y) 2*x.*y-y.^2; fy=@(x,y) x.^2-2*x.*y;
>> gx=@(x,y) 3*x.^2.*y.^2+2*x; gy=@(x,y) 2*x.^3.*y+4*y.^3;
>> x1=-0.5,y1=1.2
>> J=[fx(x1,y1) fy(x1,y1);gx(x1,y1) gy(x1,y1)];
>> ny=[x1;y1]+J\[-f(x1,y1);-g(x1,y1)]
>> x1=ny(1); y1=ny(2);
```

Här har vi löst det linjära ekvationssystemet $(*)$ med hjälp av backslash-kommandot (ett ekvationsystem av typen $Ax = b$ kan lösas i Matlab med kommandot `x=A\b`). Ett steg med Newtons metod ger (enligt ovan) att $x* \approx -0.42$ och $y* \approx 1.33$. Vi kan upprepa de tre sista kommandoraderna ovan för att förbättra dessa närmevärden till lösningen ytterligare.

```
>> J=[fx(x1,y1) fy(x1,y1);gx(x1,y1) gy(x1,y1)];
>> ny=[x1;y1]+J\[-f(x1,y1);-g(x1,y1)]
>> x1=ny(1); y1=ny(2);
```

vilket ger att $x* \approx -0.44$ och $y* \approx 1.31$. Man kan naturligtvis upprepa iterationen ytterligare några gånger tills man känner sig nöjd men notera att konvergensen är kvadratisk så närmevärdena blir snabbt väldigt bra. □

2.1.1 Övningar

2.1.1 Använd Newtons metod för att hitta närmevärden till minst en lösning på följande system av ekvationer. Plotta ingående nivåkurvor för att hitta lämpliga startvärden.

(a) $\begin{cases} x^2 + y^2 = 5 \\ xy = 2 \end{cases}$

(b) $\begin{cases} xy + \arcsin(x+y) = 1 \\ x - y + \sin(xy) = 0 \end{cases}$

Några avsnitt ur kursboken som anknyter till detta avsnitt: 13.1,13.3 & 13.6

2.2 Gradientmetoden för optimering

Vi skall börja detta avsnitt med att undersöka den geometriska betydelsen av gradienten till en funktion. Låt oss t.ex. betrakta $f(x,y) = (60 - 2y^2 - 4xy - x^4)/20$. Vi börjar med att generera två koordinatmatriser `X` och `Y`

```
>> x=linspace(-2,2,20);y=linspace(-2,2,21);
>> [X,Y]=meshgrid(x,y);
```

och beräknar sedan funktionsvärdena

```
>> f=@(x,y) (60-2*y.^2-4*x.*y-x.^4)/20;
>> Z=f(X,Y);
```

Eftersom $f'_x(x,y) = (-y - x^3)/5$ och $f'_y(x,y) = (-y - x)/5$ så kan vi beräkna gradienten i varje punkt i rutnätet med följande kommandon

```
>> G1=@(x,y) (-y-x.^3)/5; G2=@(x,y) (-y-x)/5;
>> Gx=G1(X,Y);Gy=G2(X,Y);
```

Alternativt kan vi beräkna de partiella derivatorna (approximativt) med differenskvoter;

```
>> h=10^(-9);
>> G1=@(x,y) (f(x+h,y)-f(x-h,y))/(2*h);
>> G2=@(x,y) (f(x,y+h)-f(x,y-h))/(2*h);
>> Gx=G1(X,Y);Gy=G2(X,Y);
```

Vi kan nu rita nivåkurvor och gradientvektorerna i samma figur med kommandot

```
>> contour(X,Y,Z), hold on, quiver(X,Y,Gx,Gy), hold off
```

Om man ger kommandot `axis('equal')` efter `quiver` så blir vinklarna korrekta och man ser att gradientvektorerna är vinkelräta mot nivåkurvorna. Gradienten är en vektor som pekar i den riktning funktionsvärdena ökar mest. Kraftigare ökning betyder längre gradientvektor. Om vi söker en funktions lokala maximum kan vi alltså:

(1) Välja en startpunkt (x_1, y_1).

(2) Beräkna gradienten i denna punkt.

(3) Sätta $(x_2, y_2) = (x_1, y_1) + r \cdot grad(f(x_1, y_1))$ där r är ett lämpligt (litet) tal.
(Söker vi minimum så skall vi istället gå i riktningen $-grad(f(x_1, y_1))$)

(4) Beräkna differensen $f(x_2, y_2) - f(x_1, y_1)$.
(Troligen kommer $f(x_2, y_2)$ att vara större än $f(x_1, y_1)$ om inte r är för stort. Faktorn r kan behövas eftersom funktionen kan växa väldigt brant. Med $r = 1$ kan man missa maxpunkten helt och i värsta fall hoppa fram och tillbaka. Tyvärr kan man inte veta i förväg vad r bör vara)

(5) Upprepa 3) och 4) tills ändringen i funktionsvärde är tillräckligt liten.

Denna metod för att hitta lokala max/min-punkter kallas gradientmetoden.
Låt oss nu utföra några steg i gradientmetoden på funktionen $f(x, y) = (60 - 2y^2 - 4xy - x^4)/20$, och se hur vi närmar oss en maxpunkt. För att tydligt följa vad som händer i varje steg så illustrerar vi stegen geometriskt. Vi börjar med att plotta grafen till funktionen

```
>> f=@(x,y) (60-2*y.^2-4*x.*y-x.^4)/20;
>> x=linspace(-2,2,40);y=linspace(-1,3,41);
>> [X,Y]=meshgrid(x,y);Z=f(X,Y);
>> surf(X,Y,Z),alpha(0.2),shading interp,hold on
```

Vi har här valt att hålla kvar figurfönstret med kommandot `hold on` så att vi kan plotta fler saker i samma figur (se nedan). Vi har också valt att göra ytan lite transparant så att kommande plottar kommer att synas lite bättre. Vi väljer nu en startpunkt $p = (x_1, y_1)$, t.ex.

```
>> p=[1,2];
```

Av illustrativa skäl så plottar vi nivåkurvan till f genom $(0, 1)$ och markerar punkten $(0, 1, f(0, 1)$ med en stjärna.

```
>> c=f(p(1),p(2));
>> contour(X,Y,Z,[c c]),axis equal
>> plot3(p(1),p(2),c,'r*')
>> plot3([p(1) p(1)],[p(2) p(2)],[0 c])
```

Sedan beräknar vi gradienten till f i $(0, 1)$ med kommandot

```
>> gx=G1(p(1),p(2));gy=G2(p(1),p(2));
```

Gradientmetodens ide är nu att stega fram i gradientens riktning till en ny punkt q (som ligger närmare den punkt som ger maximum)

```
>> r=0.7;
>> q=p+r*[gx,gy]
```

Låt oss nu dra en linje mellan startpunkten p och den nya punkten q så att det tydligt framgår att vi stegat oss fram vinkelrät mot nivåkurvan (i den riktning för vilket funktionsvärdena växer snabbast).

```
>> d=f(q(1),q(2));
>> plot3(q(1),q(2),d,'r*')
>> plot3([p(1) q(1) q(1)],[p(2) q(2) q(2)],[0 0 d])
```

Analysera gärna närmare genom att förstora och rotera bilden.

Eftersom skillnaden i funktionsvärdena

```
>> diff=d-c
```

är relativt stor så väljer vi att upprepa steg 3) och 4), och eftersom q är ny utgångspunkt för våra beräkningar så sätter vi

```
>> p=q;
```

Då är det bara att upprepa våra kommandon ovan

```
>> c=f(p(1),p(2));
>> contour(X,Y,Z,[c c])
>> gx=G1(p(1),p(2));gy=G2(p(1),p(2));
>> q=p+r*[gx,gy];
>> d=f(q(1),q(2));
>> diff=d-c
>> plot3(q(1),q(2),d,'r*')
>> plot3([p(1) q(1) q(1)],[p(2) q(2) q(2)],[0 0 d])
>> p=q
```

Vi vill fortsätta upprepa dessa kommandon tills skillnaden mellan funktionsvärdena i två efterföljande steg (`diff`) är tillräckligt liten (vilket indikerar att vi kommit nära ett maximum). Upprepningen underlättas om vi gör en snurra.

```
while diff>10^(-2)
    c=f(p(1),p(2));
    contour(X,Y,Z,[c c])
    gx=G1(p(1),p(2));gy=G2(p(1),p(2));
    q=p+r*[gx,gy]
    d=f(q(1),q(2));
    diff=d-c
    plot3(q(1),q(2),d,'r*')
    plot3([p(1) q(1) q(1)],[p(2) q(2) q(2)],[0 0 d])
    p=q;
    pause(0.5)
end
```

Vilken lokal maximipunkt närmar vi oss i detta fall ?

2.2.1 Övningar

2.2.1 **(a)** Lägg in alla kommandon i texten ovan i en scriptfil. Testkör sedan filen/programmet för att kontrollera att den gör vad som förväntas.

(b) Testa programmet på några andra startpunkter (ändra startpunkten $p = (1, 2)$ i programmet)

(c) Ta bort alla plottkommandon i programmet och se till att det finns semikolon efter alla återstående kommandon. Ta också bort pausen i snurran och ändra villkoret så att snurran pågår så länge som `diff>10^(-6)`. Testkör sedan det modifierade programmet med startpunkten $p = (1, 0)$. Finner programmet någon lokal maxpunkt och vilken är i så fall denna?

2.3 Andra metoder för ekvationslösning

I Matlab löser man normalt linjära ekvationssystem med s.k. vänsterdivision dvs. kommandot \. Det finns även andra kommandon som kan vara användbara vid lösning av sådana linjära system t.ex. `rref, inv, det, rank, lu, qr, chol`. Om ekvationssystemet inte är linjärt så kan man istället använda kommandot `lsqnonlin` (least square nonlinear) som hittar minstakvadratlösningar till ekvationssystem, vilka även är äkta lösningar i de fall sådana existerar. Kommandot finns inte automatiskt inbyggt i Matlabs grundpaket

utan kräver ett speciellt programpaket, en sk. toolbox, som är till för att lösa optimeringsproblem (Optimization Toolbox, ge kommandot `help optim` för att se tillgängliga kommandon). Denna toolbox finns installerad på Chalmers studentdatorer. Till linjära ekvationssytem vet vi att det antingen saknas lösningar, finns en enda lösning eller finns oändligt många lösningar. Till ickelinjära ekvationssystem kan det emellertid också vara så att det finns flera, men ett begränsat antal, lösningar. Om man har ett ekvationssystem bestående av två ekvationer och två obekanta så kan man först få en uppfattning om hur många lösningar det finns (och vilka dessa är) genom att plotta de nivåkurvor som ekvationerna representerar. Varje skärningspunkt mellan de två kurvorna representerar en lösning till ekvationssystemet.

Exempel. Betrakta ekvationssystemet $\begin{cases} x^2+y=2 \\ x-y^3=1 \end{cases}$. För att få en uppfattning om ev lösningar till ekvationssystemet så kan vi tex plotta de delar av nivåkurvorna $x^2+y=2$ och $x-y^3=1$ som ligger i kvadraten $-4 \leq x \leq 4, -4 \leq y \leq 4$.

```
>> f=@(x,y) x.^2+y-2; g=@(x,y) x-y.^3-1;
>> [X,Y]=meshgrid(-4:0.1:4);
>> contour(X,Y,f(X,Y),[0 0],'b'), hold on
>> contour(X,Y,g(X,Y),[0 0],'b'), grid, hold off
```

Vi ser att det finns två skärningspunkter och därmed två lösningar på ekvationssystemet i kvadraten. För att närmare bestämma den lösning som ligger i närheten av punkten $(1, 0.2)$ så kan vi naturligtvis även använda Newtons metod som beskrevs i föregående avsnitt. Men låt oss nu istället se hur kommandot `lsqnonlin` kan användas för att hitta lösningen.

```
>> fun=@(x) [f(x(1),x(2)),g(x(1),x(2))];
>> losn=lsqnonlin(fun,[1,0.2])
```

Observera här att `lsqnonlin` kräver att `fun` har en enda invariabel, men att denna är en radmatris. Därför måste vi skriva `x(1)` och `x(2)` istället för x och y när vi definierar den anonyma funktionen. □

Vi kan även lösa system med fler än två ekvationer och obekanta med `lsqnonlin`. Om ekvationssytemet består av tre ekvationer och tre obekanta så finns (precis som för 2×2-system) också en möjlighet att tolka lösningarna geometriskt. Varje ekvation representerar då en nivåyta och ev. lösningar motsvarar gemensamma skärningspunkter i de tre ytorna (dvs. punkter som tillhör alla de tre ytorna). Det kan dock vara svårt att lokalisera lösningarna grafiskt.

Exempel. Betrakta ES $\begin{cases} 2x-6x^2z=0 \\ 2y-3y^2z=0 \\ 2x^3+y^3=10 \end{cases}$.

För att få en uppfattning om ev. lösningar till ekvationssystemet så kan vi plotta de delar av nivåytorna $2x - 6x^2z = 0, 2y - 3y^2z = 0$ och $2x^3 + y^3 = 10$ som ligger i området $-4 \leq x \leq 4, -4 \leq y \leq 4, -4 \leq z \leq 4$.

```
>> f1=@(x,y,z) 2*x-6*x.^2.*z;
>> f2=@(x,y,z) 2*y-3*y.^2.*z;
>> f3=@(x,y,z) 2*x.^3+y.^3;
>> [X,Y,Z]=meshgrid(-4:0.3:4);
>> isosurface(X,Y,Z,f1(X,Y,Z),0)
>> isosurface(X,Y,Z,f2(X,Y,Z)+5,5)
>> isosurface(X,Y,Z,f3(X,Y,Z),10)
```

Här valde vi att addera en femma i den andra ekvationens båda led. Med detta lilla trick kunde vi på ett enkelt sätt till att de tre ytorna plottas med olika färg (olika nivåer i samma figur ger olika färg). Av figuren kan det vara svårt att avgöra antalet skärningspunkter och därmed hur många lösningar som ekvationssystemet har. Det kan vara ännu knepigare att grafiskt bestämma närmevärden till lösningarna. Genom att vända och vrida på figuren så inser man emellertid snart att det bör finnas tre lösningar; nära punkterna $(2, 0, 0), (0, 2, 0)$ resp. $(1, 2, 0)$. Låt oss bestämma dessa mer exakt med hjälp av kommandot `lsqnonlin`

```
>> f=@(x)[f1(x(1),x(2),x(3)), f2(x(1),x(2),x(3)),f3(x(1),x(2),x(3))];
>> losn1=lsqnonlin(f,[2,0,0])
>> losn2=lsqnonlin(f,[0,2,0])
>> losn3=lsqnonlin(f,[1,2,0])
```

□

Det går även bra att använda kommandot `lsqnonlin` för att lösa system med fler än tre ekvationer och obekanta.

2.3.1 Övningar

2.3.1 Plotta i samma figur de delar av kurvorna $xy - \arctan x = \sin y$ och $x + y + 3 = \cos xy$ som ligger i rektangeln $-4 \leq x \leq 2, -6 \leq y \leq 1$.
Hitta sedan alla lösningar till ekvationssystemet $\begin{cases} xy - \arctan x = \sin y \\ x + y + 3 = \cos xy \end{cases}$ som ligger i rektangeln. Undersök också hur väl de erhållna lösningarna satisfierar ekvationssytemet.

2.3.2 Plotta i samma figur på de delar av ytorna $xy - z = 1, xz + y = 2$ och $x^2 + y^2 - z^2 = 3$ som ligger i området $-5 \leq x \leq 5, -5 \leq y \leq 5, -5 \leq z \leq 5$.
Hitta sedan minst en lösning till ekvationssystemet $\begin{cases} xy - z = 1 \\ xz + y = 2 \\ x^2 + y^2 - z^2 = 3 \end{cases}$
(Tips: Det finns 4 lösningar i området).

2.3.3 Försök hitta en lösning till ekvationssystemet $\begin{cases} x+yz-w^2=3 \\ xy+z^2-w=0 \\ xyz+w=1 \\ x^2y+z^2w=0 \end{cases}$

(Tips: pröva olika startpunkter tills `lsqnonlin` hittar en lösning. Om kommandot ger svaret Maximum number of function evaluations exceededså betyder det att den inte hittar någon lösning. Fortsätt pröva tills det att kommandot svarar Öptimization terminated successfully".)

2.4 Andra optimeringsmetoder

Ett enkelt sätt att hitta ett närmevärde till en funktions största eller minsta värde är helt enkelt att beräkna ett antal funktionsvärden och ta ut det största resp. minsta av dessa med kommandona `max` resp. `min`. Låt oss först se hur detta kan göras för en funktion av en variabel. Om vi t.ex. vill hitta ett närmevärde till det största värdet för funktionen $f(x) = xsin(x)$ på intervallet $[1, 3]$ så kan vi utföra följande beräkningar

```
>> x=1:0.01:3; y=x.*sin(x); ymax=max(y)
```

Av dessa kalkyler framgår bara att funktionens största värde är ≈ 1.82. Om vi vill veta ungefär för vilket (eller vilka) x som detta maximum antas så kan vi ge följande kommando

```
>> xmax=x(find(y==ymax))
```

På ett liknande sätt kan vi hitta max/min till funktioner av flera variabler. T.ex. hittar vi det minsta värdet (och motsvarande minpunkt) till funktionen $f(x, y) = x \sin y + y - \cos x$ på området $2 \leq x \leq 5$, $-1 \leq y \leq 2$ med kommandona

```
>> [X,Y]=meshgrid(2:0.01:5,-1:0.01:2);
>> Z=X.*sin(Y)+Y-cos(X);
>> zmin=min(min(Z)), xmin=X(find(Z==zmin)), ymin=Y(find(Z==zmin))
```

Metoden ovan är enkel att förstå men har flera nackdelar. Metoden innebär t.ex. att man beräknar onödigt många funktionsvärden. Om man vill ha stor noggrannhet så kräver den dessutom oerhört många beräkningar. Det är också svårt att kontrollera hur bra närmevärdena är. Istället kan man med fördel använda kommandona `fminbnd` resp. `fminsearch` för att bestämma minpunkter till funktioner. Kommandot `fminbnd` används för att hitta minimum av funktioner av en variabel på angivet intervall t.ex.

```
>> f=@(x) x.*sin(x);
>> xmin=fminbnd(f,1,3), ymin=f(xmin)
```

Det skall påpekas att kommandot `fminbnd` endast hittar ett lokalt minimum på intervallet. Detta behöver inte nödvändigtvis vara funktionens minsta värde på intervallet. Söker man det minsta värdet så bör man först plotta funktionen och grovt uppskatta var detta ligger. I exemplet ovan är `xmin` det x-värde som ger det minsta funktionsvärdet. Detta minsta funktionsvärde har vi sedan kallat för `ymin` (I just detta exempel så antas det minsta funktionsvärdet i den ena ändpunkten på intervallet). Det finns dessvärre inget kommando som på liknande sätt hittar maximum av funktioner. Söker vi det största värdet så skriver vi istället

```
>> g=@(x) -f(x);
>> xmax=fminbnd(g,1,3), ymax=f(xmax)
```

Här har vi använt oss av observationen att det största värdet till en funktion $f(x)$ antas i samma punkt som det minsta värdet till funktionen $-f(x)$.

För att hitta lokala minimipunkter till funktioner av flera variabler så använder vi istället kommandot `fminsearch` t.ex.

```
>> f=@(x) x(1).*sin(x(2))+x(2)-cos(x(1));
>> xymin=fminsearch(f,[7,-8]), zmin=f(xymin)
```

Observera att den anonyma funktionen här skall skrivas på ett lite annorlunda sätt än tidigare. Istället för `x` och `y` skriver vi `x(1)` resp. `x(2)`. Kommandot `fminsearch` använder en iterativ metod som innebär att den behöver en startgissning (i detta fall punkten $(7, -8)$) nära den sökta minimipunkten. Utifrån detta värde stegar sig metoden succesivt närmare minimipunkten. Iterationerna fortsätter tills dess att en approximation med tillräcklig god noggrannhet erhålles. Om ingen noggrannhet anges så fortsätter iterationen tills approximationen har ett relativt fel som är mindre än 10^{-4}. Om vi istället vill att det relativa felet skall vara mindre än 10^{-6} så ger vi följande kommandot

```
>> xymin=fminsearch(f,[7,-8],optimset('TolX',1e-6))
>> zmin=f(xymin)
```

Precis som i en variabel så finns det inget inbyggt kommando för att bestämma maximum av funktioner av flera variabler, utan man får istället bestämma minimum av funktionen $-f$. Det finns heller inget kommando för att hitta max/min under bivillkor t.ex. över ett kompakt område eller längs en kurva i xy-planet. Sådana optimeringsproblem kan vi behandla på lite olika sätt. Låt oss titta på några exempel.

Exempel. Antag att man vill hitta det största och minsta värdet av funktionen $f(x, y) = x \sin y + y(\cos x - 1)$ på området $-2 \leq x \leq 12$, $-2 \leq y \leq 12$. Vi börjar med att plotta funktionens graf på angivet område.

```
>> f=@(x,y) x.*sin(y)+y.*(cos(x)-1);
>> [X,Y]=meshgrid(-2:0.2:12);
>> surf(X,Y,f(X,Y))
```

Genom att vrida figuren åt olika håll inser vi att det minsta värdet antas nära punkten $(x, y) = (10, 11)$. Med hjälp av `fminsearch` kan vi sedan bestämma minpunkten närmare.

```
>> fun=@(x) f(x(1),x(2));
>> xymin=fminsearch(fun,[10,11]), zmin=fun(xymin)
```

Av den plottade grafen avläser vi också att det största värdet på området tycks inträffa på randbiten $x = 12$, $-2 \leq y \leq 12$, där $y \approx 2$. Ett sätt att bestämma detta värde lite närmare är att bestämma maximum av funktionen $h(y) = f(12, y)$, för $0 \leq y \leq 4$, med hjälp av `fminbnd`.

```
>> h=@(y) f(12,y)
>> g=@(y) -h(y);
>> ymax=fminbnd(g,0,4), zmax=h(ymax)
```

Ett annat sätt att hitta det största värdet är att definiera om funktionen så att den är lika med 0 utanför området (obs! vi ser ju i den plottade grafen att det minsta värdet är mindre än 0 och det största värdet är större än 0). Detta gör vi enkelt med hjälp av de logiska operatorerna.

```
>> g=@(x) -fun(x).*(x(1)>=-2).*(x(1)<=12).*(x(2)>=-2).*(x(2)<=12);
>> xymax=fminsearch(g,[12,2]), zmax=fun(xymax)
```

Detta sätt fungerar även bra om området inte är en axelparallell rektangel. □

Exempel. Antag att vi söker det största och minsta värdet av samma funktion som i exemplet ovan dvs. $f(x, y) = x \sin y + y(\cos x - 1)$ fast på cirkelskivan $(x-4)^2+(y-4)^2 \leq 10$. En plott ger oss först en uppfattning var max- och minvärdet antas.

```
>> f=@(x,y) (x.*sin(y)+y.*(cos(x)-1)).*((x-4).^2+(y-4).^2<10);
>> [X,Y]=meshgrid(0:0.1:8);Z=f(X,Y);
>> surf(X,Y,Z)
```

Om vi vill kan vi justera denna plott så att bara funktionsvärdena på cirkelskivan plottas (utanför cirkelskivan, där funktionen antar värdet 0, är ju inte intressant här) genom att byta ut de värden i matrisen `Z` som är lika med 0 mot värdet `NaN` (Not a Number). Matlab plottar nämligen inga linjer till sådana värden.

```
>> Z(find(Z==0))=NaN; surf(X,Y,Z)
```

Vi ser att det minsta värdet antas då $(x, y) \approx (3, 5)$ och det största värdet då $(x, y) \approx (6, 2)$. Följande kommandon ger sedan bättre approximationer av funktionens minsta resp. största värde på cirkelskivan.

```
>> fun=@(x) f(x(1),x(2));
>> xymin=fminsearch(fun,[4,5]), zmin=fun(xymin)
>> g=@(x) -fun(x);
>> xymax=fminsearch(g,[6,2]), zmax=fun(xymax)
```

□

Ovanstående exempel illustrerar hur man kan hitta max/min till funktioner $f(x, y)$ under ett eller flera bivillkor av typen $g(x, y) \leq 0$. Att hitta största och minsta värde av f under bivillkor av typen $g(x, y) = 0$ dvs längs nivåkurvor i xy-planet kan vara lite mer komplicerat. I följande exempel beskrivs tre olika metoder som ev kan användas på sådana problem.

Exempel. Antag att vi söker det största och minsta funktionsvärdet som funktionen $f(x, y) = x \sin y + y(\cos x - 1)$ antar på cirkeln $(x-4)^2 + (y-4)^2 = 10$. Vi kan åskådliggöra problemet grafiskt genom att bara plotta de funktionsvärden som hör till punkter (x, y) på cirkeln dvs. plotta kurvan $\{(x, y, z) \in \mathbb{R}^3 : (x-4)^2+(y-4)^2 = 10, z = x \sin y+y(\cos x-1)\}$. Detta kan vi åstadkomma med följande kommandon.

```
>> f=@(x,y) x.*sin(y)+y.*(cos(x)-1);
>> g=@(x,y) (x-4).^2+(y-4).^2;
>> [X,Y]=meshgrid(0:0.1:8); Z=g(X,Y);
>> q=contour(X,Y,Z,[10,10]);
>> q=q(:,[2:length(q)]); x=q(1,:); y=q(2,:);
>> z=f(x,y); plot3(x,y,z)
```

En grov approximation av det största resp. minsta värdet kan vi naturligtvis få genom att välja ut det största resp. minsta värdet ur vektorn `z` (enligt den metod som beskrivs i början av detta avsnitt).

```
>> zmax=max(z), zmin=min(z)
```

Som påpekats ovan så är detta ingen bra metod, men ger en första grov uppskattning av funktionens största och minsta värde på cirkeln. Om man vill få en bättre approximation så finns lite olika möjligheter. Antingen kan man formulera om problemet med hjälp av Lagrange multiplikatormetod (eller likande metoder). Då får man ett ekvationssystem som sedan kan lösas på det sätt som beskrivs i avsnitt 2.1 eller 2.3. I detta fall vill vi bestämma största och minsta värde på funktionen $f(x, y) = x \sin y + y(\cos x - 1)$ under bivillkoret $(x - 4)^2 + (y - 4)^2 = 10$. Detta leder till ekvationssytemet

$$\begin{cases} \sin y - y \sin x - \lambda \cdot 2(x - 4) = 0 \\ x \cos y + \cos x - 1 - \lambda \cdot 2(y - 4) = 0 \\ (x - 4)^2 + (y - 4)^2 = 10 \end{cases}$$

För att få en uppfattning om ev. lösningar till ekvationssystemet så plottar vi de delar av ekvationssystemets nivåytor som ligger i området $0 \leq x \leq 8, 0 \leq y \leq 8, -4 \leq \lambda \leq 4$

```
>> e1=@(x,y,z) sin(y)-y.*sin(x)-z.*2.*(x-4);
>> e2=@(x,y,z) x.*cos(y)+cos(x)-1-z.*2.*(y-4);
>> e3=@(x,y,z) (x-4).^2+(y-4).^2;
>> [X,Y,Z]=meshgrid(0:0.3:8,0:0.3:8,-4:0.3:4);
>> isosurface(X,Y,Z,e1(X,Y,Z),0)
>> isosurface(X,Y,Z,e2(X,Y,Z)+5,5)
>> isosurface(X,Y,Z,e3(X,Y,Z),10)
```

Vi ser att systemet verkar ha fyra lösningar nära punkterna $(6, 2, 0), (7, 4, 0), (6, 6, 1)$ resp. $(3, 7, 0)$. Detta stämmer även bra in på den tidigare plotten, då vi plottade funktionsvärdena utefter cirkeln $(x-4)^2+(y-4)^2 = 10$. För att bestämma lösningarna mer exakt så använder vi kommandot `lsqnonlin`

```
>> e=@(x) [e1(x(1),x(2),x(3)),
           e2(x(1),x(2),x(3)),
           e3(x(1),x(2),x(3))-10]
>> lsqnonlin(e,[6,2,0])
>> lsqnonlin(e,[7,5,-1])
>> lsqnonlin(e,[6,7,1])
>> lsqnonlin(e,[3,7,0])
```

Ett annat sätt att hitta min/max av funktionen $f(x, y) = x \sin y + y(\cos x - 1)$ på cirkeln $(x - 4)^2 + (y - 4)^2 = 10$ är att parametrisera cirkeln och sedan substituera uttrycken för x och y i uttrycket för f. Då övergår problemet i att hitta största och minsta värde av en funktion av en variabel (utan några bivillkor). Om vi t.ex. parametriserar cirkeln enl. $x = 4 + \sqrt{10}\cos t, y = 4 + \sqrt{10}\sin t$, $0 \leq t < 2\pi$, så studerar vi följdaktligen funktionen $g(t) = f(4+\sqrt{10}\cos t, 4+\sqrt{10}\sin t) = (4+\sqrt{10}\cos t)\sin(4+\sqrt{10}\sin t) + (4+\sqrt{10}\sin t)(\cos(4+\sqrt{10}\cos t) - 1)$ för $0 \leq t < 2\pi$. Minsta värdet får vi sedan med följande kommandon.

```
>> f=@(x,y) x.*sin(y)+y.*(cos(x)-1);
>> x=@(t) 4+sqrt(10)*cos(t); y=@(t) 4+sqrt(10)*sin(t);
>> fun=@(t) f(x(t),y(t));
>> tmin=fminbnd(fun,0,2*pi)
>> xmin=x(tmin), ymin=y(tmin), zmin=fun(tmin)
```

□

Kommandona `fminbnd` och `fminsearch` som beskrivits ovan använder båda en sk. simplexmetod för att hitta minimum.

2.4.1 Övningar

2.4.1 Använd Matlab för att bestämma största och minsta värde för funktionen $f(x, y) = (x^2 + y)e^{-x^2-y^2} + xy/10$ över cirkelområdet $x^2 + y^2 \leq 4$. Plotta även funktionens graf över detta område.

2.4.2 Använd Matlab för att bestämma största och minsta värde för funktionen $f(x, y) = x^2 - xy + y^2 - x - y + 1$ över området D som utgöres av triangelytan med hörn i punkterna $(0, 1), (0, -1)$ och $(2, 0)$. Plotta även funktionens graf på området.

2.4.3 Bestäm största och minsta värdet för funktionen $f(x, y) = (x+y)^2$ under bivillkoret $x^2+2y^2 = 1$, enligt de tre sätt som finns beskrivet i det avslutande exemplet ovan. Gör först en grov uppskattning genom att plotta den kurva som består av punkter (x, y, z) som är sådana att x, y tillhör cirkeln $x^2 + 2y^2 = 1$ och z antar funktionsvärdena $z = f(x, y)$. Lös sedan problemet mer exakt m.h.a. Lagrange multiplikatormetod. Ställ för hand upp det ekvationssytem som metoden beskriver och lös sedan detta med hjälp av kommandot `lsqnonlin`. Avslutningsvis lös problemet genom att parametrisera cirkeln $x^2 + 2y^2 = 1$.