# Flervariabelanalys och MATLAB
# Kapitel 2

Thomas Wernstål
Matematiska Vetenskaper

19 september 2012

## 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. Den som vill kan också läsa om metoden i avsnitt 13.6 i kursboken *Calculus* (av Adams och Essex). Låt oss t.ex. betrakta ett system 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 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 börja med att 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 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

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

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

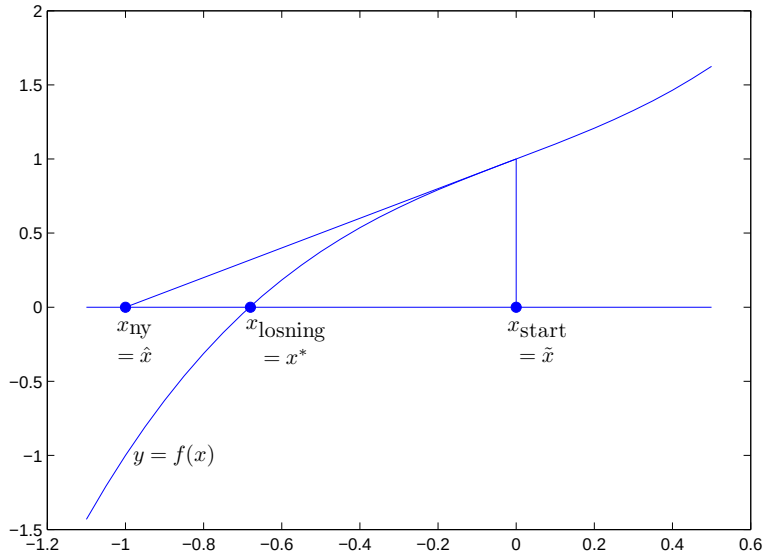
Om vi löser ut $x^*$ får vi att $$x^* \approx \tilde{x} - \frac{f(\tilde{x})}{f'(\tilde{x})}$$

Felet i denna approximation är av storleksordningen $|x^* - \tilde{x}|^2$ (ty nästa term i Taylorutvecklingen är av den storleksordningen), vilket är betydligt mindre än det ursprungliga felet $|x^* - \tilde{x}|$. Värdet på

$$\hat{x} = \tilde{x} - \frac{f(\tilde{x})}{f'(\tilde{x})}$$

ger därför en bättre approximation av $x^*$ än vad $\tilde{x}$ gav. 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 successivt 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

$$\begin{cases} f(x, y) = 0 \\ g(x, y) = 0 \end{cases} \tag{1}$$

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

$$\begin{bmatrix} f'_x(\tilde{x},\tilde{y}) & f'_y(\tilde{x},\tilde{y}) \\ g'_x(\tilde{x},\tilde{y}) & g'_y(\tilde{x},\tilde{y}) \end{bmatrix} \begin{bmatrix} \hat{x}-\tilde{x} \\ \hat{y}-\tilde{y} \end{bmatrix} = \begin{bmatrix} -f(\tilde{x},\tilde{y}) \\ -g(\tilde{x},\tilde{y}) \end{bmatrix} \tag{2}$$

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

Geometriskt innebär metoden följande: Lösningen på systemet (1) 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 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 $x^3y^2+x^2+y^4-3=0$ på något område. Låt oss pröva följande;

```
>> 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'), grid, hold off
```

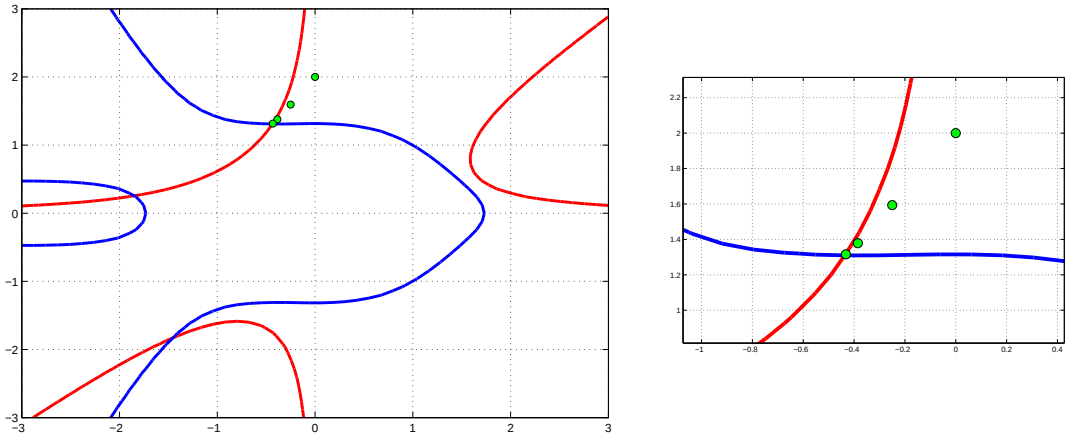
Vi inser då att det bl.a. finns en lösning (skärningspunkt) i närheten av $(0, 2)$ och använder Newtons metod för att förbättra denna (grova) approximation till lösningen.

```
>> 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,y1=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 (2) med hjälp av backslash-kommandot (ekvationsystem av typen $Ax = b$ kan lösas i MATLAB med kommandot `x=A\b`). Ett steg med Newtons metod ger (enligt ovan) att $x^* \approx -0.25$ och $y^* \approx 1.59$. Om vi vill stega oss närmare den exakta lösningen så är det bara att upprepa de tre sista kommandoraderna ovan.

```
>> 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 $x^* \approx -0.38$ och $y^* \approx 1.38$. Man kan sedan upprepa iterationen ytterligare några gånger tills man känner sig nöjd (t.ex. med en liten snurra). Nedanstående figurer illustrerar hur Newtons metod ger punkter som successivt närmar sig lösningen (den högra bilden är en uppförstorning av den vänstra).



. □

Om bara startapproximationen är tillräckligt bra kommer Newtons metod att konvergera mot en lösning väldigt snabbt ty konvergenshastigheten är ”kvadratisk”. Det skall dock påpekas att en dålig startapproximation också kan leda till att Newtons metod divergerar dvs. inte närmar sig någon lösning alls.

Låt oss nu titta lite mer allmänt på Newtons metod för system med $n$ ekvationer och $n$ obekanta;

$$\begin{cases} f_1(x_1, ..., x_n) = 0 \\ \quad\vdots \\ f_n(x_1, ..., x_n) = 0 \end{cases} \tag{3}$$

Om vi sätter

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}, \quad \mathbf{0} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

så kan systemet (3) kortare skrivas

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{4}$$

Antag att vi på något sätt erhållit en approximativ lösning $\mathbf{x}_k$ till detta system, som vi vill förbättra. I analogi med ovan ersätter vi då $\mathbf{f}$ med sin linjärisering i $\mathbf{x}_k$ (motsvarar Taylorutveckling av komponentfunktionerna t.o.m. ordning 1) dvs.

$$\mathbf{L}(\mathbf{x}) = \mathbf{f}(\mathbf{x}_k) + \mathbf{Df}(\mathbf{x}_k)(\mathbf{x} - \mathbf{x}_k)$$

där $\mathbf{Df}(\mathbf{x})$ är Jacobimatrisen för funktionen $\mathbf{f}(\mathbf{x})$, innehållande de partiella derivatorna (se avsnitt 12.6 i *Calculus*, av Adams och Essex). Lösningen till det erhållna linjära ekvationssystemet

$$\mathbf{L}(\mathbf{x}) = \mathbf{0} \tag{5}$$

ger oss då en ny (och förhoppningsvis bättre) approximation $\mathbf{x}_{k+1}$ av lösningen till det ursprungliga systemet (4).

Om vi inför $\mathbf{h} = \mathbf{x} - \mathbf{x}_k$ kan (5) skrivas

$$\mathbf{Df}(\mathbf{x}_k)\mathbf{h} = -\mathbf{f}(\mathbf{x}_k)$$

Detta är ett linjärt ekvationssystem som vi löser med avseende på $\mathbf{h}$ (förslagsvis med hjälp av backslash-kommandot i MATLAB). Lösningen $\mathbf{h}_k$ på detta system ger den vektor med vilket vi skall stega oss fram till nästa punkt dvs.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{h}_k$$

Vi kan sedan upprepa processen, och förbättra approximationen ytterligare, genom att istället utgå från $\mathbf{x}_{k+1}$ som startapproximation.

Vi behöver nödvändigtvis inte för hand exakt bestämma de partiella derivatorna i Jacobimatrisen $\mathbf{Df}(\mathbf{x})$ utan det duger (för våra ändamål) bra med approximationer med hjälp av differenskvoter. Den $j$:te kolonnen i $\mathbf{Df}(\mathbf{x})$ kan t.ex. ersättas med

$$\frac{\mathbf{f}(\mathbf{x} + \delta\mathbf{e}_j) - \mathbf{f}(\mathbf{x} - \delta\mathbf{e}_j)}{2\delta}$$

för något lämpligt litet tal $\delta$ (förslagsvis $10^{-5}$), och där $\mathbf{e}_j$ är $j$:te enhetsvektorn.

Låt oss illustrera detta med ett exempel.

**Exempel.** Antag att vi vill hitta en lösning till ekvationssystemet

$$\begin{cases} y^2 + z^2 &= 3 \\ x^2 + z^2 &= 2 \\ x^2 - z &= 0 \end{cases}$$

Vi börjar då med att skapa följande funktion;

```
>> f=@(x) [x(2).^2+x(3).^2-3; x(1).^2+x(3).^2-2; x(1).^2-x(3)];
```

Låt oss utgå från $\mathbf{x}_1 = (2, 1, 0)$ som startapproximation i Newtons metod. Vi börjar med att beräkna Jacobimatrisen för $\mathbf{f}$ i punkten $\mathbf{x}_1$. Med en liten snurra beräknar vi Jacobimatrisen (`J`), kolonn för kolonn, med differenskvoter;

```
>> x=[2;1;0];
>> delta=10^(-5);
>> J=zeros(3,3);
>> for j=1:3 ...
       ej=zeros(3,1);
       ej(j,1)=1;
       J(:,j)=(f(x+delta*ej)-f(x-delta*ej))/(2*delta);
   end
```

Sedan stegar vi oss fram till nästa approximation med följande kommandon;

```
>> h=J\(-f(x));
>> x=x+h
```

Notera att vi här tilldelar variabeln `x` den nya approximationen och därmed skriver över den föregående approximationen.

Man kan naturligtvis upprepa processen ovan (t.ex. med en liten snurra) för att förbättra approximationen ytterligare. Utför ytterligare några steg och försök avgöra vilken lösning vi närmar oss. □

## 2.2 Andra metoder för ekvationslösning

I MATLAB löser man normalt linjära ekvationssystem med s.k. vänsterdivision (kommandot \ ). Det finns även andra kommandon som kan vara användbara vid lösning av sådana system t.ex. `rref, inv, det, rank, lu, qr, chol`.

Om ekvationssystemet inte är linjärt så kan man istället använda kommandot `fsolve`. Kommandot finns inte 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). Chalmers har en MATLAB-licens för denna toolbox (och en mängd andra), så om du jobbar på en Chalmersdator bör det inte vara några några problem att använda kommandot `fsolve` och andra funktioner i detta programpaket.

För linjära ekvationssytem vet vi att det antingen saknas lösningar, finns en enda lösning eller så har systemet oändligt många lösningar. Till ickelinjära ekvationssystem kan det dock finnas fler än en lösning utan att det nödvändigtvis finns oändligt många t.ex. två, tre eller fler lösningar.

Programmet `fsolve` kräver som indata (precis som Newtons metod) en startgissning dvs. en första approximation av lösningen. För $2 \times 2$-system kan vi (som i föregående avsnitt) ev. bilda oss en uppfattning om hur många lösningar det finns (och vilka dessa är), på ett visst område, genom att plotta nivåkurvor. Varje skärningspunkt mellan kurvorna motsvarar en lösning till ekvationssystemet. Om man inte har någon ytterligare information (om t.ex. det bakomliggande problemet och vad variablerna representerar) kan det dock vara svårt att uppskatta i vilket område man skall börja söka efter lösningar (endast genom att studera ekvationerna).

**Exempel.** Betrakta ekvationssystemet $\begin{cases} x^2 + y = 2 \\ x - y^3 = 1 \end{cases}$

Låt oss undersöka om det måhända kan finnas några lösningar 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],'r'), hold on
>> contour(X,Y,g(X,Y),[0 0],'b'), grid, hold off
```

Vi inser 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 $(1, 0.2)$ kan vi nu ge följande kommandon;

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

Observera här att vi måste skriva `x(1)` och `x(2)` istället för $x$ och $y$ när vi definierar den anonyma funktionen.

Om man vill kan man få MATLAB att redovisa detaljer om de olika stegen som `fsolve` utför för att hitta lösningen (eller snarare en förbättrad approximation av lösningen). Ge i så fall följande kommando;

```
>> options=optimset('Display','iter');
>> losn=fsolve(fun,[1,0.2],options)
```

Algoritmen innehåller villkor för när iterationen skall avbrytas, och om man vill kan man förändra dessa villkor. Den som vill kan läsa mer om detta t.ex. genom att ge kommandot `help fsolve`. □

Vi kan även lösa system med fler än två ekvationer och obekanta med `fsolve`. Om ekvationssytemet består av tre ekvationer och tre obekanta så finns (precis som för $2 \times 2$-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 ekvationssystemet

$$\begin{cases} 2x - 6x^2z &= 0 \\ 2y - 3y^2z &= 0 \\ 2x^3 + y^3 &= 10 \end{cases}$$

För att få en uppfattning om ev. lösningar till systemet prövar vi med att 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);
>> clf
>> 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 se till att de tre ytorna plottas med olika färg (olika nivåer i samma figur ger olika färg). Av figuren kan det dock vara svårt att avgöra antalet skärningspunkter och därmed hur många lösningar som ekvationssystemet har och det kan vara ännu knepigare att grafiskt bestämma närmevärden till lösningarna. Genom att vända och vrida på figuren inser man dock i detta fall att det bör finnas tre lösningar; nära punkterna $(2, 0, 0), (0, 2, 0)$ resp. $(1, 2, 0)$. Låt oss bestämma dessa lite bättre med hjälp av `fsolve`;

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

□

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

## 2.3 Gradientmetoden för optimering

Vi skall börja detta avsnitt med att undersöka den geometriska betydelsen av gradienten till en funktion. Låt oss t.ex. betrakta funktionen

$$f(x,y) = \frac{60 - 2y^2 - 4xy - x^4}{20}$$

Vi börjar med att generera två koordinatmatriser `X` och `Y`;

```
>> x=linspace(-2,2,20);y=linspace(-1,3,21);
>> [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 $f'_y(x,y) = (-y - x)/5$ kan vi beräkna gradienten i varje punkt i rutnätet med följande kommandon;

```
>> fx=@(x,y) (-y-x.^3)/5; fy=@(x,y) (-y-x)/5;
>> GX=fx(X,Y);GY=fy(X,Y);
```

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

```
>> h=10^(-5);
>> fx=@(x,y) (f(x+h,y)-f(x-h,y))/(2*h);
>> fy=@(x,y) (f(x,y+h)-f(x,y-h))/(2*h);
>> GX=fx(X,Y);GY=fy(X,Y);
```

Vi kan sedan plotta nivåkurvor och gradientvektorer i samma figur med kommandot;

```
>> contour(X,Y,Z), hold on, quiver(X,Y,GX,GY), hold off
```

Om man dessutom ger kommandot `axis equal` blir vinklarna korrekta och man ser att gradientvektorerna är vinkelräta mot nivåkurvorna. Om man använder `quiver` som ovan så skalar MATLAB om vektorernas längd för att ge en bättre illustration av hur de varierar på området. Vektorernas längd relativt varandra stämmer dock. Om man trots allt vill plotta gradientvektorernas faktiska längd så kan man ge kommandot `quiver(X,Y,GX,GY,0)` (pröva gärna och jämför).

Gradienten är en vektor som pekar i den riktning funktionsvärdena ökar mest (se t.ex. avsnitt 12.7 i *Calculus*, av Adams och Essex). Kraftigare ökning betyder längre gradientvektor. Vi kan använda denna observation för stega oss fram mot större och större funktionsvärden tills vi ev. hittar ett maximum. Följande schema beskriver denna idé;

**(1)** Välja en startpunkt $(x_1, y_1)$.

**(2)** Beräkna gradienten $\nabla f(x_1, y_1)$.

**(3)** Stega fram till nästa punkt $(x_2, y_2)$ genom;

$$(x_2, y_2) = (x_1, y_1) + r \cdot \nabla f(x_1, y_1)$$

där $r$ är ett lämpligt (litet) tal.

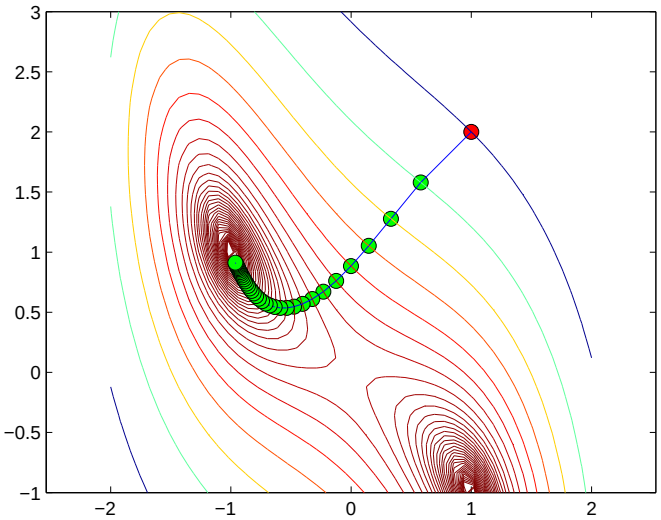
(Söker vi minimum skall vi istället gå i riktningen $-\nabla f(x_1, y_1)$ )

**(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 eller kanske hoppa fram och tillbaka. Det finns metoder med vilket man kan bestämma bra val på $r$ men vi fördjupar oss inte i det här)

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

Denna metod för att hitta lokala max/min-punkter kallas ofta för *gradientmetoden* (men benämns mer vanligen på engelska med "*method of steepest decent*"). Nedanstående figur illustrerar hur man med denna metod successivt kan närma sig en extrempunkt (startpunkten är markerad med rött och efterföljande steg med grönt).



Låt oss utföra några steg i gradientmetoden på funktionen ovan (med litet positivt värde på $r$) och se hur vi närmar oss en maxpunkt. För att tydligt följa vad som händer i varje steg så illustrerar vi stegen med plottar. Vi börjar med att plotta grafen till funktionen;

```
>> 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 skall synas lite bättre. Vi väljer nu en startpunkt $p = (x_1, y_1)$, t.ex.

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

Av illustrativa skäl 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 de partiella derivatorna till $f(x, y)$ i $(0, 1)$;

```
>> gx=fx(p(1),p(2));gy=fy(p(1),p(2));
```

Gradientmetodens ide är nu att stega fram i gradientens riktning till en ny punkt $q$, som förhoppningsvis ligger närmare en maximipunkt..

```
>> 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 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=fx(p(1),p(2));gy=fy(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
```

Antag nu att vi vill upprepa iterationen 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^(-4)
    c=f(p(1),p(2));
    contour(X,Y,Z,[c c])
    gx=fx(p(1),p(2));gy=fy(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
hold off
```

Vilken lokal maximipunkt närmar vi oss i detta fall ?

Gradientmetoden kan naturligtvis även användas för att hitta max/min av funktioner av fler än två variabler. Låt oss titta på ett exempel där funktionen beror på tre variabler och där vi med plottar försöker illustrera hur metoden stegar sig fram till ett maximum.

**Exempel.** Betrakta funktionen $f(x,y,z) = \dfrac{1+x^2+y^2+z^2}{1+x^2+2y^2+z^4}$.

Låt oss börja med att illustrera funktionsvärdena utefter några plan;

```
>> f=@(x,y,z) (1+x.^2+y.^2+z.^2)./(1+x.^2+2*y.^2+z.^4);
>> [X,Y,Z]=meshgrid(linspace(-3,3,20));
>> W=f(X,Y,Z);
>> slice(X,Y,Z,W,[-3 0 3],[-3 0 3],[])
>> shading interp
>> alpha(0.35)
>> axis equal
>> hold on
```

Följande snurra illustrerar sedan hur vi med gradientmetoden kan närma oss ett maximum för funktionen $f$;

```
p=[1,1.5,2.5];
diff=1;
h=1e-5;
while diff>0.001
   x=p(1);y=p(2);z=p(3);
   grad(1)=(f(x+h,y,z)-f(x-h,y,z))/(2*h);
   grad(2)=(f(x,y+h,z)-f(x,y-h,z))/(2*h);
   grad(3)=(f(x,y,z+h)-f(x,y,z-h))/(2*h);
   q=p+grad;
   diff=f(q(1),q(2),q(3))-f(x,y,z);
   quiver3(x,y,z,q(1)-x,q(2)-y,q(3)-z,'LineWidth',2)
   p=q;
   pause(0.2)
end
hold off
```

Här startade vi iterationen i punkten $(1, 1.5, 2.5)$. Pröva gärna lite andra startpunkter och se vad som händer. □

## 2.4 Andra optimeringsmetoder

Om man vill bestämma maximi- och minimipunkter till en reellvärd funktion av en variabel kan man t.ex. använda kommandot `fminbnd`. T.ex. ger;

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

en lokal minimipunkt (`xmin`) på intervallet $[1, 3]$ till funktionen $f(x) = x \sin x$, samt funktionsvärdet (`ymin`) i denna minimipunkt.

Notera att `fminbnd` nödvändigtvis inte ger den punkt där funktionen antar sitt minsta värdet på intervallet, utan bara en lokal minimipunkt, som i exemplet ovan (plotta grafen till funktionen och verifiera att `fminbnd` inte gav det minsta värdet på intervallet).

Det finns dessvärre inget kommando som på liknande sätt hittar maximum av funktioner. Detta är dock inget större bekymmer ty maximipunkterna till $f(x)$ är desamma som minimipunkterna till $-f(x)$. Ett lokalt maxima hittar vi därför med följande kommandon;

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

För att hitta lokala minimipunkter till funktioner av flera variabler kan vi istället använda `fminunc` (om funktionen är differentierbar) eller `fminsearch` (om ej differentierbar). Pröva t.ex.

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

Notera hur vi här definierat den anonyma funktionen med `x(1)` resp. `x(2)` istället för `x` och `y`. Som input i `fminunc` måste man ange en startgissning (i detta fall punkten $(7, -8)$) nära den sökta minimipunkten. Utifrån detta värde stegar sig metoden successivt närmare minimipunkten, tills tillräcklig noggrannhet uppnås. Om man vill kan man få MATLAB att redovisa detaljer om de olika stegen som `fminunc` utför för att hitta minimipunkten (eller snarare en förbättrad approximation av minmipunkten). Ge i så fall följande kommando;

```
>> options=optimset('Display','iter');
>> xymin=fminunc(f,[7,-8],options)
```

Algoritmen innehåller villkor för när iterationen skall avbrytas, och om man vill kan man ändra på dessa villkor. Den som vill kan läsa mer om detta t.ex. genom att ge kommandot `help fminunc`.

Precis som för funktioner i en variabel finns inget färdigt kommando för att bestämma maximum av funktioner av flera variabler, utan man studerar istället minimum av funktionen $-f$. T.ex. ger;

```
>> g=@(x) -f(x);
>> xymax=fminunc(g,[5,-9]), zmax=f(xymax)
```

en lokal maximipunkt till funktionen $f(x,y) = x\sin y + y - \cos x$ i närheten av punkten $(5,-9)$. Försök övertyga dig om att ovanstående kommandon verkligen ger lokala maximi- och minimipunkter, genom att plotta funktionsytan och/eller nivåkurvor nära punkterna.

Ett annat alternativ om man söker max/min till en differentierbar funktion av flera variabler är att utnyttja det faktum att en max/min-punkt också är en stationär punkt dvs. en punkt där $\nabla f = \mathbf{0}$. För att bestämma lösningar till ett sådant system av ekvationer kan vi sedan använda teknikerna från avsnitt 2.1 & 2.2. En fördel med denna metod är att den (förutom max/min) också ger oss ev. sadelpunkter till funktionen. Låt oss titta på ett exempel.

**Exempel.** Betrakta åter igen funktionen $f(x,y) = x\sin y + y - \cos x$. De stationära punkterna till $f(x,y)$ är lösningar till följande ekvationssystem;

$$\begin{cases} \sin y + \sin x &= 0 \\ x\cos y + 1 &= 0 \end{cases}$$

Lösningar kan vi sedan bestämma med t.ex. Newtons metod (avsnitt 2.1) men låt oss för enkelhets skull använda det färdiga kommandot `fsolve` (se avsnitt 2.2);

```
>> fun=@(x) [sin(x(2))+sin(x(1)), x(1).*cos(x(2))+1];
>> stat=fsolve(fun,[8,-8])
```

Försök verifiera att den punkt som erhålls är en sadelpunkt till $f(x,y)$. Plotta t.ex. nivåkurvor i en omgivning av punkten och undersök Hessianen till funktionen $f$ i den erhållna punkten. Pröva gärna också att byta startpunkt och se om du lyckas hitta samma max/min-punkter som med `fminunc` ovan. □

Ibland vill man bestämma max/min av en funktion $f(\mathbf{x})$ under ett eller flera bivillkor. Då kan man istället använda kommandot `fmincon`. Bivillkoren kan vara av lite olika typ t.ex. linjära bivillkor av typen $A\mathbf{x} \leq \mathbf{c}$ och/eller $B\mathbf{x} = \mathbf{d}$ där $A, B$ är matriser och $\mathbf{c}, \mathbf{d}$ är vektorer. Man kan också som bivillkor ange enkla gränser på $\mathbf{x}$ av typen $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$, för några vektorer $\mathbf{l}, \mathbf{u}$. Det är också möjligt med icke-linjära bivillkor av typen $\mathbf{g}(\mathbf{x}) \leq 0$ eller $\mathbf{h}(\mathbf{x}) = 0$. Om sådana icke-linjära bivillkor finns med så skall syntax för `fmincon` vara på formen;

```
>> x=fmincon(f,x0,A,c,B,d,l,u,@nonlcon)
```

där `f` är målfunktionen (angiven som anonym funktion), `x0` är en startapproximation och `nonlcon` är en funktionsfil för de icke-linjära funktionerna **g** och **h** som skall ha formen;

```
function [g,h]=nonlcon(x)
g=g(x(1),x(2),...,x(n));
h=h(x(1),x(2),...,x(n));
```

Normalt har vi inte bivillkor av alla typer på samma gång i de problem vi vill lösa. Då ersätter vi motsvarande platser i anropet med `[]`.

Liksom `fminunc` behöver `fmincon` en startapproximation. Vi bör därför först bilda oss en uppfattning om det över huvudtaget finns några max/min-punkter och var vi i så fall kan förvänta oss att dessa finns. Ibland kan en plott vara till hjälp att undersöka detta. Låt oss titta på ett exempel.

**Exempel.** Antag att vi söker det största och minsta värdet på funktionen $f(x,y) = x\sin y + y(\cos x - 1)$ då $(x,y)$ ligger på cirkeln $(x-4)^2 + (y-4)^2 = 10$.

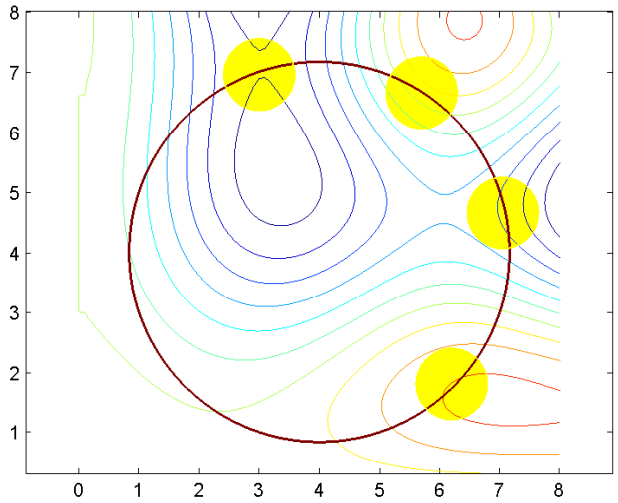
Låt oss börja med att plotta skärningskurvan mellan funktionsytan $z = f(x,y)$ och cylindern $(x-4)^2 + (y-4)^2 = 10$ (betraktad som en ekvation i $x, y, z$). Detta kan vi t.ex. åstadkomma med följande kommandon;

```
>> 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),grid
```

Av figuren framgår tydligt hur funktionsvärdena varierar och att det finns två maxima och två minima. Det kan dock vara lite svårt att avläsa var dessa ligger. En annan figur, som kanske bättre ger oss en uppfattning om var max/min-punkterna finns, får vi om vi plottar nivåkurvan $(x-4)^2 + (y-4)^2 = 10$ tillsammans med ett antal nivåkurvor till $f$;

```
>> contour(X,Y,f(X,Y)),grid,hold on
>> contour(X,Y,g(X,Y),[10,10]),hold off
```

Genom extrempunkterna vet vi att det går nivåkurvor till $f$ som tangerar cirkeln. Dessa nivåkurvor finns (naturligtvis) inte med i figuren men vi kan uppskatta var de skulle kunna tangera.




T.ex. ser det ut som det ligger en minimipunkt nära $(7, 5)$. Låt oss använda `fmincon` för att bestämma denna extrempunkt närmare. Vi börjar då med att skapa följande funktionsfil;

```
function [g,h]=nonlcon(x)
g=[];
h=(x(1)-4)^2+(x(2)-4)^2-10;
```

Här satte vi `g=[]` eftersom vi inte har något bivillkor av typen $\mathbf{g}(\mathbf{x}) \leq 0$.

Sedan hittar vi extrempunkten med;

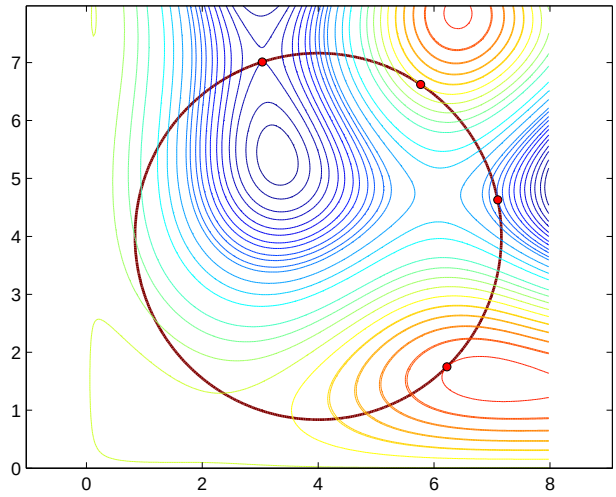
```
>> f=@(x) f(x(1),x(2));
>> x0=[7;5];
>> x=fmincon(f,x0,[],[],[],[],[],[],@nonlcon)
```

Med kommandot `ginput` kan man istället välja startpunkt genom att klicka på en punkt i figurfönstret. Det förenklar en del om man vill pröva sig fram och se vad som händer för olika startvärden. Ge i så fall följande kommandon (och upprepa dem t.ex. med en liten snurra);

```
>> x0=ginput(1)
>> x=fmincon(f,x0,[],[],[],[],[],[],@nonlcon)
```

Om man vill bestämma maximipunkterna så måste man som tidigare studera $-f$. Försök bestämma alla de fyra extrempunkterna med hjälp av `fmincon`. Gör

sedan en ny plott med nivåkurvor till $f$ och cirkeln i samma figur (i stil med ovan), men se till att också få med nivåkurvorna genom extrempunkterna. Det kan se ut ungefär så här;




Notera speciellt hur nivåkurvorna till $f$ genom extrempunkterna tangerar nivåkurvan $(x-4)^2+(y-4)^2=10$ (dvs. cirkeln). Eftersom gradienten till $f(x,y)$ och gradienten till $g(x,y)=(x-4)^2+(y-4)^2-10$ är vinkelräta mot resp. nivåkurvor måste de vara parallella i extrempunkterna dvs. $\nabla f=\lambda\nabla g$, för någon konstant $\lambda$. Denna observation ligger till grund för idén bakom *Lagrange multiplikatormetod* (se avsnitt 13.3 i *Calculus*, av Adams och Essex). Villkoret $\nabla f(x,y)=\lambda\nabla g(x,y)$, tillsammans med bivillkoret $g(x,y)=0$, ger oss ett ekvationssystem med tre ekvationer i tre obekanta ($x,y$ och $\lambda$). Systemen är i regel icke-linjära och kan vara knepiga (och oftast omöjliga) att lösa exakt för hand men kan lösas approximativt t.ex. med metoder från avsnitt 2.1 eller 2.3. I vårat exempel ovan erhålls följande ekvationssystem;

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

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

```
>> 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);
```

```
>> clf
>> 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 tidigare plottar. För att bestämma lösningarna lite närmare använder vi kommandot `fsolve`;

```
>> e=@(x) [e1(x(1),x(2),x(3)),
           e2(x(1),x(2),x(3)),
           e3(x(1),x(2),x(3))-10]
>> losn1=fsolve(e,[6,2,0])
>> losn2=fsolve(e,[7,5,-1])
>> losn3=fsolve(e,[6,7,1])
>> losn4=fsolve(e,[3,7,0])
```

Ovan har vi sett två olika sätt med vilket vi kan bestämma maximum och minimum av funktionen $f(x,y) = x\sin y + y(\cos x - 1)$ under bivillkoret $(x-4)^2 + (y-4)^2 = 10$. Metoderna är ganska generella och kan användas för andra liknande problem. I vissa fall (som ovan, och i de flesta andra liknande problem ni kommer stöta på i denna kurs), när bivillkoret beskriver en mängd som kan parametriseras, kan problemet dock förenklas. Om vi parametriserar cirkeln enl;

$$\begin{cases} x = 4 + \sqrt{10}\cos t \\ y = 4 + \sqrt{10}\sin t \end{cases}, \quad 0 \le t < 2\pi$$

och sätter;

$$g(t) = f(4 + \sqrt{10}\cos t, 4 + \sqrt{10}\sin t)$$

så reduceras problemet till att bestämma maximum och minimum av $g(t)$, för $0 \le t < 2\pi$. Detta är ett extremvärdesproblem i en variabel och utan några bivillkor, ett betydligt enklare problem än det ursprungliga. Ett lokalt minimum finner vi t.ex. 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)
```

Bestäm även de övriga tre extrempunkterna och jämför med vad tidigare metoder gav. □