1 Linjärisering och tangentplan

Vi skall linjärisera $f(\boldsymbol{x}) = \frac{1}{2}x_1^2(1+x_2^2) - x_1 + 1$ och rita upp tangentplan.

Det gäller att

$$\frac{\partial f}{\partial x_1}(\boldsymbol{x}) = x_1(1+x_2^2) - 1, \quad \frac{\partial f}{\partial x_2}(\boldsymbol{x}) = x_1^2 x_2$$

och därmed $\boldsymbol{\nabla} f(\boldsymbol{x})^T = \begin{bmatrix} x_1(1+x_2^2) - 1 & x_1^2 x_2 \end{bmatrix}$.

Linjäriseringen vid t.ex. $\boldsymbol{a} = (0.8, 0.1)$ blir

$$L(\boldsymbol{x}) = f(\boldsymbol{a}) + \boldsymbol{\nabla} f(\boldsymbol{a})^T(\boldsymbol{x} - \boldsymbol{a}) = 0.5232 + \begin{bmatrix} -0.1920 & 0.0640 \end{bmatrix} \begin{bmatrix} x_1 - 0.8 \\ x_2 - 0.1 \end{bmatrix} =$$

$$= 0.5232 - 0.1920(x_1 - 0.8) + 0.0640(x_2 - 0.1)$$

Nu skall vi beskriva linjäriseringen i MATLAB och rita upp tangentplan. Samtidigt passar vi på att rita normalen (se Adams 12.3).

Vi beskriver funktionen och dess derivator med

```
>> f=@(x,y)0.5*x.^2.*(1+y.^2)-x+1;      % Enklare skriva x,y istället för x1,x2
>> dfdx=@(x,y)x.*(1+y.^2)-1; dfdy=@(x,y)x.^2.*y;
```

Sedan bildar vi linjärisering och normalvektor med funktionerna

```
>> L=@(x,y,a,b)f(a,b)+dfdx(a,b)*(x-a)+dfdy(a,b)*(y-b);
>> n=@(x,y)[-dfdx(x,y);-dfdy(x,y);1];
```

Vi anger tangeringspunkt och beräknar normalvektor samt anger området vi skall rita över.

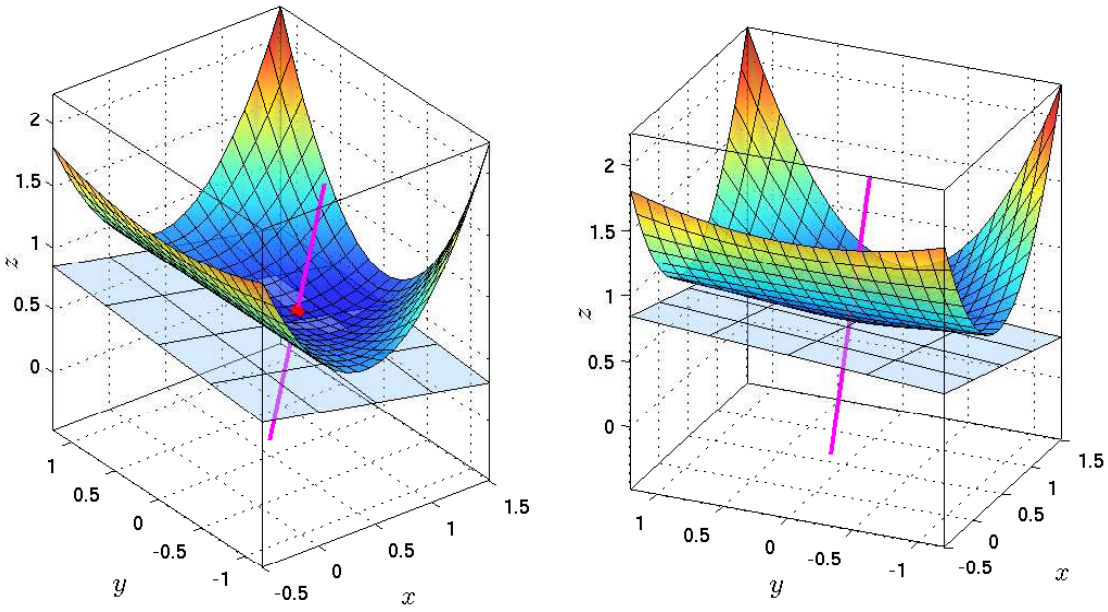
```
>> a=0.8; b=0.1;                        % Enklare skriva a,b istället för a1,a2
>> p0=[a;b;f(a,b)]; n0=n(a,b);          % Tangeringspunkt och normalvektor
>> x=linspace(-0.5,1.5,20); y=linspace(-1.2,1.2,20);
>> [X,Y]=meshgrid(x,y);                 % Koordinatmatriser för området
```

Ritar funktionsyta, tangentplan samt normal med

```
>> Z=f(X,Y); T=L(X,Y,a,b);
>> surf(x,y,Z), hold on                 % Funktionsytan
>> surf(x,y,T,'FaceColor','b','FaceAlpha',0.4)  % Tangentplanet
>> s=[-1 1];                            % För att rita normalen
>> plot3(p0(1)+s*n0(1),p0(2)+s*n0(2),p0(3)+s*n0(3),'m','linewidth',2), hold off
>> xlabel('x'), ylabel('y'), zlabel('z'), box on
>> axis equal, axis vis3d, rotate3d on
```

Vi använder `axis vis3d` så att skalan inte förändras då vi vrider och vänder på grafen, med `rotate3d on` blir det möjligt att ta tag i grafen och vrida den.

Här ser vi ytan med tangentplanet och normalen. Vi tittar från två olika håll.



När vi ritar normalen ritar vi den som en rät linje från en punkt i rummet till en annan. Vi använder `plot3` och då måste koordinater för start- och slutpunkt separeras i tre vektorer för x-, y- respektive z-koordinaterna. Eftersom `s=[-1 1]` är en vektor så ger `p0+s*n0` en vektor (i MATLAB) motsvarande en linje med start i $\boldsymbol{a}-\boldsymbol{n}$ och slut i $\boldsymbol{a}+\boldsymbol{n}$. Vi separerar koordinaterna

```
x:      p0(1)+s*n0(1)
y:                    p0(2)+s*n0(2)
z:                                  p0(3)+s*n0(3)
```

och ritar linjen med

```
>> plot3(p0(1)+s*n0(1),p0(2)+s*n0(2),p0(3)+s*n0(3),'m','linewidth',2)
```

Vi ser en bit av normalen magentafärgad.

2 Icke-linjära ekvationer

Vi skall lösa det icke-linjära ekvationssystemet $\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{0}$, där

$$\boldsymbol{f}(\boldsymbol{x})=\begin{bmatrix} f_1(x_1,x_2)\\ f_2(x_1,x_2)\end{bmatrix}=\begin{bmatrix} x_1\,(1+x_1^2-x_2^2)+2\\ x_2\,(1+x_1^2)-5\end{bmatrix}$$

med Newtons metod.

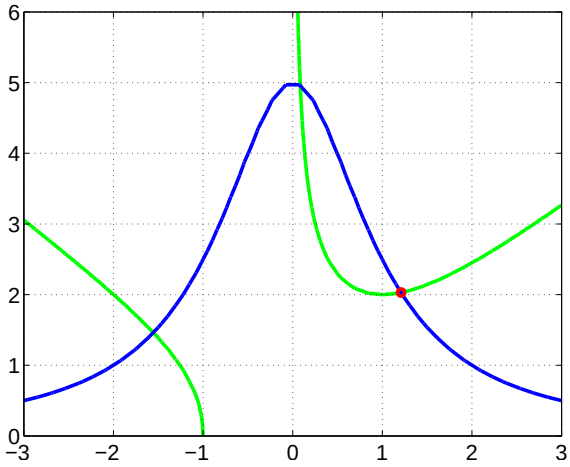
Först ritar vi upp noll-nivåkurvorna till f_1 respektive f_2 så att vi ser lösningen till $\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{0}$ som den punkt där dessa kurvor skär varandra.

Vi börjar med att beskriva de två komponentfunktionerna f_1 och f_2. För att vi skall kunna beräkna dessa med matriser av x_1- och x_2-värden låter vi dem bero av två separata variabler.

```
>> f1=@(x1,x2)x1.*(1+x1.^2-x2.^2)+2;
>> f2=@(x1,x2)x2.*(1+x1.^2)-5;
```

Nu kan vi använda `contour` på ett enkelt sätt. Eftersom `contour` kan ges antal nivåer eller en vektor med nivåvärden måste vi ge vektorn `[0 0]`, ger vi bara `0` tolkas det som ingen nivå.

```
>> x1=linspace(-1,3,30); x2=linspace(-1,3,30);
>> [X1,X2]=meshgrid(x1,x2);
>> Z1=f1(X1,X2); Z2=f2(X1,X2);
>> contour(x1,x2,Z1,[0 0],'g'), hold on
>> contour(x1,x2,Z2,[0 0],'b')
>> grid on
```



När vi använder Newtons metod skall vi beskriva problemet med den vektorvärda funktionen $\boldsymbol{f}$ och med vektorn $\boldsymbol{x}$ som variabel. Med koden

```
>> f=@(x)[f1(x(1),x(2))
          f2(x(1),x(2))];
```

bildar vi denna funktion. Lägg märke till hur vi använder komponentfunktionerna f_1 och f_2 samt hur vi plockar ut x_1- och x_2-värdena ur vektorn $\boldsymbol{x}$. Med `[...]` ser vi till att $\boldsymbol{f}$ blir en vektor.

Vi måste även beskriva Jacobimatrisen

```
>> Df=@(x)[1+x(1)^2-x(2)^2+2*x(1)^2 -2*x(1)*x(2)
           2*x(1)*x(2)                1+x(1)^2   ];
```

Från bilden med noll-nivåkurvorna ser vi att $\boldsymbol{x}_0 = (1, 2)$ nog är en startapproximation av ett av nollställena och vi bestämmer det noggrant med Newtons metod enligt

```
>> x=[1;2];
>> kmax=10; tol=0.5e-8;
>> for k=1:kmax
       h=-Df(x)\f(x);
       x=x+h;
       disp([x' norm(h)])
       if norm(h)<tol, break, end
   end
```

```
   1.250000000000000   2.000000000000000   0.250000000000000
   1.210936459304649   2.027441055015319   0.047738576769853
   1.210045042953554   2.029047914019690   0.001837557882334
   1.210044873897516   2.029049004374033   0.000001103382317
   1.210044873897639   2.029049004373946   0.000000000000151
```

Vi ritar ut nollstället med en liten röd ring med

```
>> plot(x(1),x(2),'ro')
```

så vi ser att det är rätt nollställe vi har bestämt.