Newtons metod

1 Inledning

Vi skall lösa system av icke-linjära ekvationer. Som exempel kan vi ta,

$$\begin{cases} x_1\,(1+x_2^2)-1=0 \\ x_2\,(1+x_1^2)-2=0 \end{cases}$$

som är ett system av två ekvationer i två obekanta (Exempel 1, Adams 13.6). Om vi inför de två funktionerna

$$f_1(x_1,x_2) = x_1\,(1+x_2^2)-1$$
$$f_2(x_1,x_2) = x_2\,(1+x_1^2)-2$$

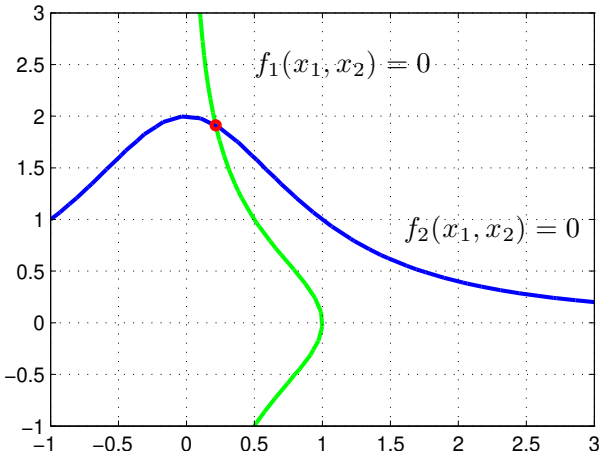
kan ekvationssystemet skrivas

$$\begin{cases} f_1(x_1,x_2)=0 \\ f_2(x_1,x_2)=0 \end{cases}$$

Med vektorbeteckningar $\boldsymbol{f}=(f_1,f_2)$, $\boldsymbol{x}=(x_1,x_2)$, och $\mathbf{0}=(0,0)$, ser vi att $\boldsymbol{f}:\mathbb{R}^2\to\mathbb{R}^2$ och vi kan skriva ekvationerna på den kompakta formen

$$\boldsymbol{f}(\boldsymbol{x})=\mathbf{0}.$$

Vi skall använda Newtons metod för att lösa sådana ekvationer. Men vi börjar med att rita upp noll-nivåkurvorna till f_1 respektive f_2.



Vi ser lösningen till $\boldsymbol{f}(\boldsymbol{x})=\mathbf{0}$ som den punkt där noll-nivåkurvorna till f_1 och f_2 skär varandra.

En första approximation av lösningen kan vi läsa av i grafiken för att sedan förbättra denna med Newtons metod. Vi skall beskriva Newtons metod, men först skall vi se hur vi ritar noll-nivåkurvor.

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

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
```

Koden ovan ger bilden på förra sidan. När vi skall använda Newtons metod vill 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.

2 Newtons metod

Låt $f : \mathbb{R} \to \mathbb{R}$ vara en deriverbar funktion. Vi skall lösa ekvationen $f(x) = 0$ med Newtons metod. Det här gjorde vi i envariabelanalysen så detta blir lite repetition.

Antag att vi har en approximativ lösning x_k och vi vill hitta en bättre approximation x_{k+1}. Vi bildar linjäriseringen av f i x_k:

$$L(x) = f(x_k) + f'(x_k)(x - x_k)$$

och löser $L(x) = 0$ istället för $f(x) = 0$.

$$f(x_k) + f'(x_k)(x - x_k) = 0 \tag{1}$$

Lösningen får bli nästa approximation:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

Detta är Newtons metod. Geometriskt betyder (1) att vi följer tangenten och hittar x_{k+1} där denna skär x-axeln.

Som stoppvillkor för iterationen tar vi

$$|x_{k+1} - x_k| \leq tol$$

Det betyder att vi accepterar x_{k+1} om ändringen i sista iteration är mindre än toleransen tol. Vi tillåter maximalt k_{max} iterationer ($k_{max} = 10$ är rimligt).

Som ett litet exempel tar vi $f(x) = \cos(x) - x$ och vi skall lösa $f(x) = 0$. En graf (rita den gärna) visar att vi har ett nollställe och vi tar $x_0 = 1$ som startapproximation.

```
>> f=@(x)cos(x)-x; Df=@(x)-sin(x)-1;
>> x=1;
>> kmax=10; tol=0.5e-8;
>> for k=1:kmax
       h=-f(x)/Df(x);
       x=x+h;
       disp([x h])
       if abs(h)<tol, break, end
   end

   0.750363867840244  -0.249636132159756
   0.739112890911362  -0.011250976928882
   0.739085133385284  -0.000027757526078
   0.739085133215161  -0.000000000170123
```

System av ekvationer

Nu är det dags för system av n ekvationer i n obekanta

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ \qquad\vdots \\ f_n(x_1, \ldots, x_n) = 0 \end{cases}$$

Om vi låter

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

så har vi $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$, och vi kan skriva systemet på formen

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}.$$

Antag att vi har en approximativ lösning $\boldsymbol{x}_k$ och vi vill hitta en bättre approximation $\boldsymbol{x}_{k+1}$. Vi bildar linjäriseringen av $\boldsymbol{f}$ i $\boldsymbol{x}_k$:

$$\boldsymbol{L}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}_k) + \boldsymbol{Df}(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) = \boldsymbol{0}.$$

och löser $\boldsymbol{L}(\boldsymbol{x}) = \boldsymbol{0}$ istället för $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$.

Om vi låter $\boldsymbol{h} = \boldsymbol{x} - \boldsymbol{x}_k$ så kan $\boldsymbol{L}(\boldsymbol{x}) = \boldsymbol{0}$ skrivas som ekvationen

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

Detta är ett linjärt ekvationssystem, Jacobimatrisen $\boldsymbol{Df}(\boldsymbol{x}_k)$ är en $n \times n$-matris, som vi löser med avseende på $\boldsymbol{h}$. Nu bildar vi nästa approximation av lösningen till $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$:

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

Som stoppvillkor för iterationen tar vi

$$||\boldsymbol{x}_{k+1} - \boldsymbol{x}_k|| \leq tol$$

Vi avbryter när vi har tillräcklig noggrannhet och vi tillåter maximalt k_{max} iterationer ($k_{max} = 10$ är rimligt).

Nu ser vi åter på exemplet från inledningen (Exempel 1, Adams 13.6). Vi har alltså

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} x_1(1+x_2^2) - 1 \\ x_2(1+x_1^2) - 2 \end{bmatrix}, \quad \boldsymbol{Df}(\boldsymbol{x}) = \begin{bmatrix} 1+x_2^2 & 2x_1x_2 \\ 2x_1x_2 & 1+x_1^2 \end{bmatrix},$$

och skall lösa $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$. Vi har ju redan beskrivit funktionen och dess komponenter. Nu måste vi även beskriva Jacobimatrisen

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

Från bilden med noll-nivåkurvorna ser vi att $\boldsymbol{x}_0 = (0.25, 2)$ nog är en bra startapproximation av nollstället och vi bestämmer det noggrant med Newtons metod enligt

```
>> x=[0.25;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

   0.217391304347826   1.913043478260870   0.092869605927364
   0.214829670172721   1.911781803315968   0.002855484777347
   0.214829232694196   1.911768811990568   0.000012998689285
   0.214829232680284   1.911768811998807   0.000000000016168
```

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.

Uppgift 1. Låt $\boldsymbol{f}(\boldsymbol{x}) = (x_1^3 + x_2^2 - 1, \mathrm{e}^{x_1x_2} + x_1 + x_2 - 2)$. Lös ekvationssystemet $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$. Rita upp noll-nivåkurvorna till f_1 och f_2 för att se var ungefär lösningarna (skärningspunkterna) ligger. Hur många lösningar finns det? Läs av i grafiken en första approximation av en lösning för att sedan förbättra denna med Newtons metod. Rita ut lösningen med en liten ring. Upprepa tills du beräknat alla lösningar till systemet.

Uppgift 2. På studiohemsidan finns funktionen `newton` som bygger på `min_newton` från envariabelkursen. Ladda ner och studera programkoden samt hjälptexten. Använd `newton` för att lösa ekvationssystemet $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, där $\boldsymbol{f}(\boldsymbol{x}) = (\sin(x_1) - x_2, x_1 - \cos(x_2))$.

Newtons metod utnyttjar funktions- och derivatavärden för att lokalt med linjärisering approximera den funktion vi söker nollställe till. Varje ny approximation ger en bättre bestämning av nollstället. Det finns dock en svag punkt, en linjärisering är inte alltid en bra approximation i

ett stort område. Vi måste därför alltid ge Newtons metod en bra första gissning av nollstället. Men det är inget större problem om vi har två ekvationer i två obekanta för då kan vi rita noll-nivåkurvor för att får grepp om hur många nollställen det finns, var ungefär de ligger och vilka som är av intresse för oss. Har vi fler obekanta (och fler ekvationer) får vi med kunskap om tillämpningen söka startapproximationer.

3 Ekvationslösning i MATLAB

I MATLAB OPTIMIZATION TOOLBOX finner vi `fsolve` som använder en modifierad version av Newtons metod. Derivatan av funktionen (som vi söker nollställe till) approximeras i programmet så vi behöver inte ge den. För en sista gång ser vi på exemplet från inledningen. Vi har alltså

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} x_1(1+x_2^2)-1 \\ x_2(1+x_1^2)-2 \end{bmatrix} = \mathbf{0},$$

och i MATLAB löser vi med (samma startapproximation som tidigare)

```
>> f=@(x)[x(1)*(1+x(2)^2)-1
          x(2)*(1+x(1)^2)-2];
>> x0=[0.25;2];
>> x=fsolve(f,x0)
x =
   0.214829232694215
   1.911768811990538
```

Uppgift 3. Låt $\boldsymbol{f}(\boldsymbol{x}) = (\sin(x_1) - x_2, x_1 - \cos(x_2))$. Lös ekvationssystemet $\boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}$. Rita upp noll-nivåkurvorna till f_1 och f_2 för att se var ungefär lösningarna ligger. Hur många lösningar finns det? Läs av en första approximation av en lösning och förbättra den sedan med `fsolve`. Rita ut lösningen med en liten ring. Upprepa tills du beräknat alla lösningar till systemet. Använd `ginput` för att läsa av startapproximation i grafiken.