Newtons metode for system av ligninger

Arne Morten Kvarving
http://www.math.ntnu.no/∼arnemort/m4-itersys.pdf

Department of Mathematical Sciences
Norwegian University of Science and Technology

15. Oktober 2009

Problem

- Gitt problemet

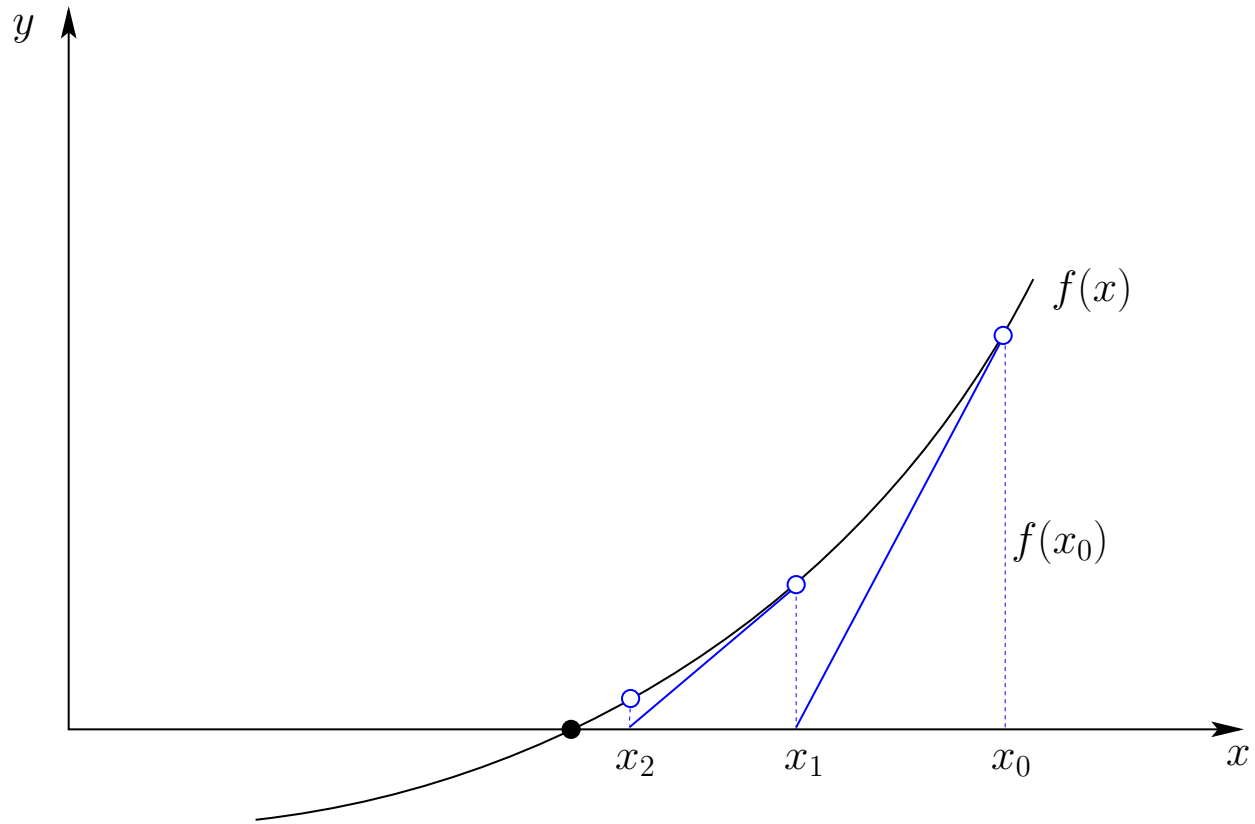
$$\underline{f}\,(\underline{x}\,) = \underline{0}$$

  for et eller annet system av funksjoner f.
- Et eksempel på slike problem kan være

$$\begin{aligned} f_1(x_1, x_2) = x_1^2 + x_2^2 - 9 &= 0 \\ f_2(x_1, x_2) = x_1 x_2 - 1 &= 0 \end{aligned}$$

Repetisjon - skalar Newton

- Idé: Vi approksimerer løsningen ved å følge tangenter.



Newtoniterasjoner i Matlab - runde 3

- Implementer $f(x)$ og $f'(x)$ som seperate funksjoner

```
function y = test(x)
    y = x^2-3*x+1;

function y = testd(x)
    y = 2*x-3;
```

Newtoniterasjoner i Matlab - runde 3 kont

- Selve Newtonløseren ser nå ut som

```
function x = newton3(x0,n,gamma,f,fd)
    x = x0;
    for i=1:n
        x_old = x;
        eps = -feval(f,x)/feval(fd,x);
        x = x + eps;
        if abs(x-x_old) <= gamma*abs(x_old)
            break;
        end
    end

    if i==n
        disp('warning: did not converge');
    end
```

Taylorutvikling i flere dimensjoner

- Vi betrakter

$$\underline{f}\left(\underline{x}+\underline{\Delta x}\right).$$

- Vi vil taylorutvikle denne langs hver komponent x_i.
- På komponentform er dette

$$f_1\left(x_1+\delta_1, x_2+\delta_2\right)$$
$$f_2\left(x_1+\delta_1, x_2+\delta_2\right)$$

Taylorutvikling i flere dimensjoner

- Taylorutvikler først langs den første retningen:

$$f_1(x_1+\delta_1, x_2+\delta_2) = f_1(x_1, x_2+\delta_2) + \delta_1 \frac{\partial f_1}{\partial x_1}(x_1, x_2+\delta_2) + \mathcal{O}(\delta_1^2).$$

- Taylorutvikler så denne i den andre retningen

$$f_1(x_1, x_2+\delta_2) = f_1(x_1, x_2) + \delta_2 \frac{\partial f_1}{\partial x_2}(x_1, x_2) + \mathcal{O}(\delta_2^2)$$

$$\delta_1 \frac{\partial f_1}{\partial x_1}(x_1, x_2+\delta_2) \approx \delta_1 \frac{\partial f_1}{\partial x_1}(x_1, x_2) + \delta_1\delta_2 \frac{\partial^2 f_1}{\partial x_1 \partial x_2}(x_1, x_2).$$

Taylorutvikling i flere dimensjoner

- På samme måte for den andre funksjonen får vi

$$f_2(x_1+\delta_1, x_2+\delta_2) = f_2(x_1, x_2+\delta_2) + \delta_1 \frac{\partial f_2}{\partial x_1}(x_1, x_2+\delta_2) + \mathcal{O}(\delta_1^2)$$

$$f_2(x_1, x_2+\delta_2) = f_2(x_1, x_2) + \delta_2 \frac{\partial f_2}{\partial x_2}(x_1, x_2) + \mathcal{O}(\delta_2^2)$$

$$\delta_1 \frac{\partial f_2}{\partial x_1}(x_1, x_2+\delta_2) \approx \delta_1 \frac{\partial f_2}{\partial x_1}(x_1, x_2) + \delta_1\delta_2 \frac{\partial^2 f_2}{\partial x_1 \partial x_2}(x_1, x_2).$$

- Trunkerer så disse og betrakter kun ledd som er $\mathcal{O}(\delta_1)$ og $\mathcal{O}(\delta_2)$.

Taylorutvikling i flere dimensjoner

- Samlet og skrevet ved hjelp av matriser og vektorer får vi

$$\underline{f}\left(\underline{x}+\underline{\Delta x}\right) \approx \underline{f}\left(\underline{x}\right)+\underline{J}\left(\underline{x}^{k}\right)\underline{\Delta x}$$

  hvor $\underline{J}$ er *Jakobimatrisen*

$$\underline{J}=\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

- Legg merke til at matrisen $\underline{J}$ er en funksjon av $\underline{x}^{k}$.

Newtons metode for system

- Hvis vi nå går tilbake til Newtons metode husker vi at idéen er å beregne en lineær korreksjon for hver iterasjon.
- Dette betyr altså at vi for hver iterasjon må løse systemet

$$\underline{J}\,\underline{\Delta x}^{k+1} = -\underline{f}\left(\underline{x}^{k}\right).$$

- Når dette er gjort tar vi vår nye iteratverdi som

$$\underline{x}^{k+1} = \underline{x}^{k} + \underline{\Delta x}^{k+1}.$$

Newtons metode for system - forandringer

- Vi må løse et system istedet for en skalar ligning - så vi må bruke en løser for lineære system. I Matlab er dette heldigvis enkelt; Vi forandrer

```
eps = -feval(f,x)/feval(fd,x);
```

til

```
J = feval(fd,x);
F = feval(f,x);
eps = J\-F;
```

Newtons metode for system - forandringer

- Siden x nå er en vektor, må vi bruke en norm istedet for absoluttverdien. Vi forandrer

```
if abs(x-x_old) <= gamma*abs(x_old)
```

til

```
if norm(x-x_old) <= gamma*norm(x_old)
```

Newtons metode for system - Matlabkode

```
function y = simplesystem(x)
    y = zeros(2,1);
    y(1) = x(1)^2+x(2)^2-9;
    y(2) = x(1)*x(2)-1;

function J = simplesystemj(x)
    J = zeros(2,2);

    J(1,1) = 2*x(1);
    J(1,2) = 2*x(2);
    J(2,1) = x(2);
    J(2,2) = x(1);
```

Newtons metode for system - Matlabkode

- Selve Newtonløseren ser nå ut som

```
function x = newtonsystem(x0,n,gamma,f,fd)
    x = x0;
    for i=1:n
        x_old = x;
        J = feval(fd,x);
        F = feval(f,x);
        eps = J\-F;
        x = x + eps
        if norm(x-x_old) <= gamma*norm(x_old)
            break;
        end
    end
```

- Poeng: Denne virker også på skalare ligninger!

Forenklet Newton

- Vi må løse et nytt lineært system for hver iterasjon.
- I praksis betyr dette at vi må faktorisere en matrise per iterasjon.
- Som dere har sett tidligere er faktorisering av en matrise en kostbar operasjon (for store system).
- Men når vi først har faktorisert matrisen, er tilbake/forover substitusjoner derimot mye mindre kostbart.

Forenklet Newton

- Idé: Frys Jakobimatrisen, dvs, istedet for å bruke

$$\underline{J}\left(\underline{x}^{k}\right)$$

bruker vi istedet

$$\underline{J}\left(\underline{x}^{0}\right).$$

- Dermed inverterer vi samme matrise i alle iterasjoner.
- Dette lar oss *prefaktorisere* $\underline{J}$ for så å bare gjøre de kjappe substitusjonsoperasjonene per iterasjon.

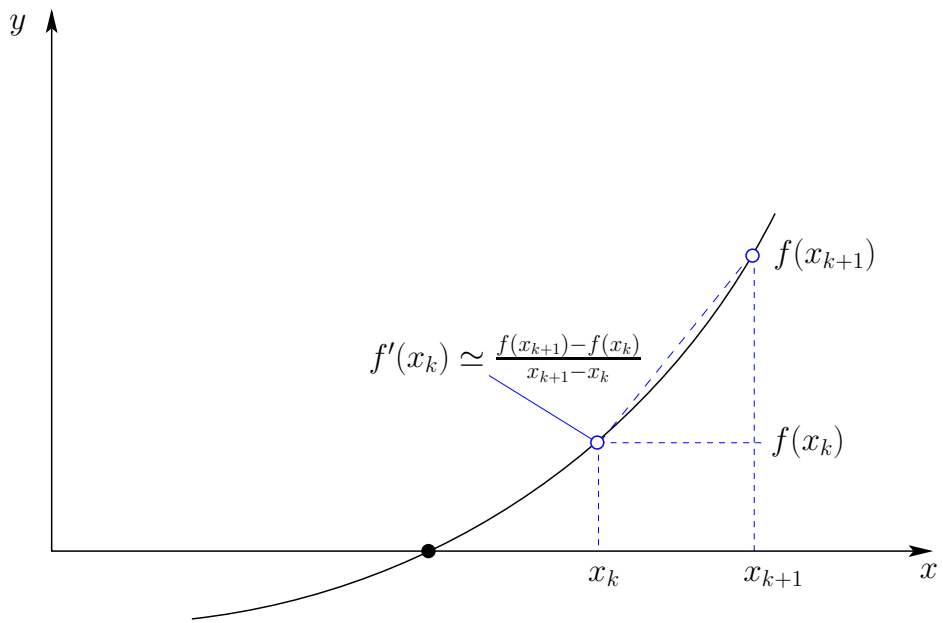
Forenkelt Newton - Matlabkode

- Selve Newtonløseren ser nå ut som

```
function x = sinewtonsystem(x0,n,gamma,f,fd)
    x = x0;
    J = feval(fd,x0);
    [L,U] = lu(J);
    for i=1:n
        x_old = x;
        F = -feval(f,x);
        v = L\F;
        eps = U\v;
        x = x + eps;
        if norm(x-x_old) <= gamma*norm(x_old)
            break;
        end
    end
```

Repetisjon - Sekantmetoden

- Noen ganger kan det være vanskelig å finne et uttrykk for den deriverte av funksjonen.
- Vi kan/må da ty til en approksimasjon til den deriverte av funksjonen.



Repetisjon - Sekantmetoden

- En måte å approksimere den deriverte av funksjonen på er å approksimere den deriverte som en *differanseoperator*.
- Vi bruker

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

- Setter inn og finner

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

Repetisjon - Sekantmetoden i Matlab

```
function x = sekant(x0,x1,n,gamma,f)
    fx_old = feval(f,x0);
    x_old = x0;
    x = x1;
    for i=1:n
        fx = feval(f,x);
        eps = -fx*(x-x_old)/(fx-fx_old);
        x_old = x;
        x = x + eps;
        fx_old = fx;
        if abs(x-x_old) <= gamma*abs(x_old)
            break;
        end
    end
```

Numerisk Jakobian

- Hvis det fins skalare funksjoner det er vanskelig å derivere, så fins det iallefall system av ligninger som er det.
- Idé: Approksimér alle de partiellderiverte ved hjelp av differanseoperatorer.

```
function J = numjac(f,f0,x,delta)
    n = length(x);
    In = delta*eye(n);
    J = zeros(n,n);
    for k=1:n
        fp = feval(f,x+In(:,k));
        J(:,k)=(fp-f0)/delta;
    end
```

Newton med numerisk Jakobian - Matlabkode

- Selve Newtonløseren ser nå ut som

```
function x = newtonsystem(x0,n,gamma,f)
    x = x0;
    delta = 1.e-8;
    for i=1:n
        x_old = x;
        F = feval(f,x);
        J = numjac(f,F,x,delta);
        eps = J\-F;
        x = x + eps;
        if norm(x-x_old) <= gamma*norm(x_old)
            break;
        end
    end
```

Forenklet Newton med numerisk Jakobian - Matlabkode

- Selve Newtonløseren ser nå ut som

```
function x = sinewtonsystemnum(x0,n,gamma,f)
    x = x0;
    J = numjac(f,feval(f,x0),x0,1.e-6);
    [L,U] = lu(J);
    for i=1:n
        x_old = x;
        F = feval(f,x);
        v = L\-F;
        eps = U\v;
        x = x + eps;
        if norm(x-x_old) <= gamma*norm(x_old)
            break;
        end
    end
```

Oppsummering

metode	jakobian	bruksområde
newton	eksakt	høy nøyaktighet
newton	numerisk	høy nøyaktighet, mangler jakobian
forenklet newton	eksakt	tung jakobian, moderat nøyaktighet
forenklet newton	numerisk	moderat nøyaktighet, mangler jakobian