TDP015 Tema n: Kompletterande Material

Victor Lagerkvist

Syftet med föreläsningen är att ge en introduktion till numeriska metoder. I denna kompletterande text försöker vi ge ytterligare motivation till *varför* numeriska metoder är viktiga.

Bakgrund

Numeriska metoder handlar i grund och botten om att ge rimliga uppskattningar av funktioner eller egenskaper hos funktioner. Exempelvis:

1. Uppskatta/approximera en funktion.
2. Uppskatta/approximera derivatan till en given funktion.
3. Uppskatta/approximera integralen till en funktion (arean under den resulterande kurvan).

I manuskriptet presenteras Newton-Raphson-metoden som försöker approximera ett nollställe (det vill säga, en lösning) till en funktion f genom följande:

1. Låt x_0 vara en "ursprungsgissning".
2. Ta den nuvarande bästa approximationen x_i.
3. Räkna ut en ny approximation x_{i+1} baserad på x_i som är bättre.
4. Detta görs genom att beräkna lutningen hos x_i och undersöka var den resulterande linjen har sitt nollställe.
5. Avbryt när nuvarande gissning är tillräckligt bra.

Man kan alltså tänka sig att man givet en gissning x_i gör en ny gissning genom att skjuta iväg en "linjärapproximering" (som bestäms av derivatan med avseende på x_i). En vanlig användning av Newton-Raphson-metoden är att beräkna $\sqrt{n}$ för ett givet n, det vill säga, försök hitta nollstället till $x^2 = n$. Det känns som en helt rimlig, om än inte särskilt upphetsande tillämpning, men är det värt besväret? När vill vi egentligen räkna ut det?

Motivation 1

En vanlig tillämpning inom datorgrafik är att beräkna den *inversa roten ur* (inverse square root). Denna operation dyker nämligen upp naturligt när man normaliserar en vektor (tryck ihop vektorn till

en vektor i intervallet $[0, 1]$), vilket man i sin tur gör för att det blir lättare att räkna ut exempelvis vinklar och reflektioner. I modern hårdvara är detta något som vanligtvis är hårdkodat och ingenting man behöver implementera själv, men innan hårdvarustöd för detta var allmänt tillgängligt fanns ett stort behov av att räkna ut inverse square root så effektivt som möjligt. I källkoden till Quake III finns följande funktion för detta syfte (vanligtvis tillskrivet John Carmack, men metoden är äldre än så).

```
float Q_rsqrt( float number )
{
   long i;
   float x2, y;
   const float threehalfs = 1.5F;

   x2 = number * 0.5F;
   y = number;
   i = * ( long * ) &y; // evil floating point bit level hacking
   i = 0x5f3759df - ( i >> 1 ); // what the fuck?
   y = * ( float * ) &i;
   y = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
   return y;
}
```

Ovanstående kod är egentligen inget annat än en enda iteration av Newton-Raphson-metoden med en bra initial gissning. I de smått bisarra raderna

```
i = * ( long * ) &y;
i = 0x5f3759df - ( i >> 1 );
```

För detaljer, se exempelvis följande.

gör vi (grovt förenklat) först om flyttalet y till ett heltal i, som sedan får sin initiala gissning med hjälp av flyttalsmagi.

Motivation 2

I maskininlärning är ett vanligt förekommande problem att försöka approximera en funktion (exempelvis en funktion som returnerar en sannolikhet inom $[0, 1]$) baserat på stora mängder testdata. Ett steg i denna process kan vara att försöka beskriva funktionen med hjälp av dess koefficienter (exempelvis, tror vi att det går att approximera datan med en kvadratisk funktion så behöver vi "bara" hitta lämpliga koefficienter). En metod för detta är så kallad *gradientsökning* som fungerar på ett väldigt likartat sätt som Newton-Raphson-metoden: börja med att gissa ett värde till koefficienten och förbättra gradvis gissningen genom linjärapproximationer. På ett liknande sätt används också gradientsökning för att förbättra och träna neurala nätverk.

Sammanfattingsvis: idén bakom Newton-Raphson-metoden är en viktig komponent inom datorgrafik, maskininlärning, och neurala nätverk. Behövs det egentligen mer motivering till varför det är en bra metod att förstå?