FYS4150 - Computational Physics

PROSJEKT 3

Numerisk integrasjon, en odyssé om Gaussisk kvadratur og Monte Carlo

Jon Vegard Sparre
jonvsp@uio.no

Dato: 21. oktober 2015

Sammendrag

I dette prosjektet ser vi på numerisk integrasjon på tre (fire) måter. Vi starter med Gaussisk kvadratur (GK) i form av Gauss-Legendre og Gauss-Laguerre kombinert med Gauss-Legendre, deretter ser vi på rett fram-Monte Carlo og en litt forbedra Monte Carlo-metode. Vi har vist at GK-metoder ikke er spesielt godt egna til integrering i mange dimensjoner da feilen skalerer med antall datapunkter N opphøyd i antall dimensjoner, mens feilen for Monte Carlo-metoder skalerer feilen alltid som $1/\sqrt{N}$. Vi har også sett at Monte Carlo-metoder krever utregninger og dermed kortere tid for å oppnå høyere presisjon. Monte Carlo-metoder ga oss fire desimalers presisjon på et halv minutt, mens GK ga oss to desimalers presisjon på to minutter.

Lenke til Jon Vegards GitHub-domene:
https://github.com/jonvegards/FYS4150

Introduksjon

Numerisk integrasjon kan gjøres på mange måter, vi ser her nærmere på metoder som baserer seg på Gaussisk kvadratur og Monte Carlo-teknikker. Førstnevnte er den eldste metoden og ble utviklet før datamaskiner fantes, mens sistnevnte er en litt nyere metode som kom i første halvdel av 1900-tallet. Integralet som vi skal teste disse metodene på er,

$$\left\langle \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \right\rangle = \int_{-\infty}^{\infty} d\mathbf{r}_1\, d\mathbf{r}_2\, e^{-2\alpha(r_1+r_2)} \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}. \quad (1)$$

Dette integralet har en analytisk løsning som er $5\pi^2/16^2$, så vi kan lett sjekke om de numeriske resultatene stemmer.

Gaussisk kvadratur er en integrasjonsmetode som baserer seg på å summere opp funksjonsverdier ganget med en vektfunksjon som vekter leddene i summen forskjellig. De forskjellige vektene blir bestemt av ortogonale polynomer som Legendre- eller Laguerre-polynomer. Vi ser nærmere på utledninga av dette i Teori-delen.

Monte Carlo-metoder baserer også seg på summasjon av funksjonsverdier av uniformt tilfeldig genererte tall. Vi har her brukt en rett-fram-metode som integrerer integralet som det er i kartesiske koordinater i en seks-dimensjonal hyperkube og en litt forbedret metode der vi bruker sfæriske koordinater og eksponentialfordelinga som sørger for at de genererte datapunktene passer bedre med integranden vår, det gjør at resultatet vil konvergere raskere mot den analytiske løsninga.

Teori

For å skjønne hvordan integrasjonsmetodene virker så kan det lønne seg å se litt på utledninga av de. Som nevnt i introduksjonen så baserer GK seg på summasjon av funksjonsverdier som er vektet forskjellig. I metoder som Simpsons metode så vektes alle integrasjonspunkt likt, og det er ikke alltid like fornuftig. Hvis integranden vår ikke varierer mye over et større intervall, så vil metoder som den ovennevnte konvergere sakte og sørge for at vi bruker unødig mye regnekraft.

Vi kan derfor introdusere GK som kort og godt skrives,

$$I = \int_a^b f(x)\, dx \approx \sum_{i=1}^{N} \omega_i f(x_i). \quad (2)$$

Hvor vi da har ω_i som vektene og x_i er integrasjonspunktene. Det er flere måter å velge vektene på, i dette prosjektet ser vi først på Gauss-Legendre. Den bruker ortogonale polynomer i intervallet $x \in [-1, 1]$ med vektfunksjonen $W(x) = 1$. Men for å komme dit må vi først se litt på hvilken rolle vektfunksjonen har. Det viser seg at en integrand som ikke er glatt i integrasjonsintervallet kan gjøres glatt ved å dra ut en vektfunksjon fra den, vi får,

$$I = \int_a^b f(x)\, dx = \int_a^b W(x) g(x)\, dx \approx \sum_{i=1}^{N} \omega_i g(x_i). \quad (3)$$

Denne vektfunksjonen må være positiv i hele integrasjonsintervallet slik at $\int_a^b |x|^n W(x)\, dx$ er integrerbar. Likn. (3) kalles en GK hvis den kan integrere alle polynomer p eksakt,

$$I = \int_a^b W(x) p(x)\, dx = \sum_{i=1}^{N} \omega_i p(x_i).$$

Vi får av dette $2N$ likninger, N for integreringspunktene og N for vektene, dette impliserer at vi kan tilnærme integranden vår $f(x) \approx P_{2N-1}(x)$, en dobling fra metoder som for eksempel Simpons metode. Men hvordan ser polynomet vi tilnærmer integranden med, ut? Vi definerer det til å være,

$$P_{2N-1}(x) = L_N(x)P_{N-1}(x) + Q_{N-1}(x), \tag{4}$$

hvor $L_N(x)$ er Legendre-polynomer av N-grad, og $P_{N-1}(x)$ og $Q_{N-1}(x)$ er polynomer av grad $N-1$ eller lavere. Vi forutsetter nå at integrasjonsintervallet er $[-1,1]$ siden Legendrepolynomer er definert i det intervallet, vi må altså huske på å endre variablene våre slik at integrasjonsgrensene passer med intervallet til Legendrepolynomene. Likninga ovenfor kan settes inn i integralet vårt og vi kan bruke ortogonalitetsegenskapene til Legendrepolynomene slik at vi får

$$\begin{aligned}\int_{-1}^{1} P_{2N-1}(x)\,\mathrm{d}x &= \int_{-1}^{1} L_N(x)P_{N-1}(x) + Q_{N-1}(x)\,\mathrm{d}x \\ &= \int_{-1}^{1} Q_{N-1}(x)\,\mathrm{d}x\,.\end{aligned}$$

Vi kan nå bruke at integrasjonspunktene x_k, for $K = 0,\ldots,N-1$, er valgt slik at de også er nullpunktene til Legendre-polynomene,

$$P_{2N-1}(x_k) = Q_{N-1}(x_k).$$

Dette kan vi bruke til å definere Q_{N-1} på den måten hele integralet. Vi kan så skrive Q_{N-1} som en sum Legendrepolynomer,

$$Q_{N-1}(x) = \sum_{i=0}^{N-1} \alpha_i L_i(x).$$

Ortogonalitetsegenskapene til Legendrepolynomene gir,

$$\int_{-1}^{1} Q_{N-1}(x) = \sum_{i=0}^{N-1} \alpha_i \int_{-1}^{1} L_0(x)L_i(x)\,\mathrm{d}x = 2\alpha_0. \tag{5}$$

Vi bruker så at x_k er nullpunktene til Legendrepolynomene, vi får da,

$$Q_{N-1}(x_k) = \sum_{i=0}^{N_1} \alpha_i L_{ik}, \tag{6}$$

hvor $L_{ik} = L_i(x_k)$. Nå har vi fått en matrise L som har som kolonnevektorer Legendrepolynomene $L_i(x_k)$ der i er kolonnen og graden av Legendrepolynomet, mens x_k er nullpunktet til Legendrepolynomet av k-te grad, diagonalen er med andre ord null. Kolonnene er også lineært uavhengig på grunn av ortogonaliteten til Legendrepolynomene, slik at vi har $L^{-1}L = \mathbf{1}$. Vi ganger (6) med L^{-1} og får

$$(L^{-1})_{ki}Q_{N-1}(x_i) = \alpha_k. \tag{7}$$

Vi kan bruke (7) til å finne α_0 og sette det inn i (5),

$$\int_{-1}^{1} Q_{N-1}(x) = 2\sum_{i=0}^{N-1} (L^{-1})_{0i}P_{2N-1}(x_i).$$

Vektene er da $\omega_i = (L^{-1})_{0i}$, og vi får til slutt

$$\int_{-1}^{1} f(x) \approx \sum_{i=0}^{N-1} \omega_i P_{2N-1}(x_i).$$

Til slutt vil vi nevne hvordan man transformerer koordinater fra intervallet $x \in [a,b] \to t \in [-1,1]$. Man gjør det ganske enkelt ved

$$t = \frac{a-b}{2}x + \frac{b+a}{2}.$$

For tilfellet $x \in [0,\infty)$ bruker vi,

$$\tilde{x}_i = \tan\left(\frac{\pi}{4}(1+x_i)\right),$$

$$\tilde{\omega}_i = \frac{\pi}{4}\frac{\omega_i}{\cos^2(\frac{\pi}{2}(1+x_i))}.$$

Biblioteket `lib.h` gjør jobben med å regne ut vektene for begge GK-metodene vi bruker.

Vi kan nå se litt på hvordan Monte Carlo-metoder virker! Disse metodene baserer seg på uniformt tilfeldig genererte tall og funksjonsverdiene av de. Siden vi jobber med en uniform fordeling må integrasjonsgrensene være fra 0 til 1, det er lett å ordne ved å substituere variable. Vi kan generelt skrive integralet som en forventingsverdi,

$$I = \langle f \rangle = \int_0^1 f(x)p(x)\,\mathrm{d}x\,, \tag{8}$$

hvor vi da har at $p(x) = 1$ for $x \in [0,1]$ og null ellers, altså den uniforme distribusjonen. Men vi må huske på at

vi jobber numerisk, så integralet blir diskretisert (tilbake) til en sum,

$$I = \sum_{i=1}^{N} f(x_i).$$

I dette prosjektet skal vi bruke noe som kalles vektingsutvelging[1]. Det er en teknikk som sørger for at visse deler av integrasjonsintervallet veier tyngre enn andre, det betyr at vi kan bruke mer regnekraft på de områdene av intervallet som faktisk ligger innenfor området til integranden. Vi har integralet,

$$I = \int_a^b f(x)\,\mathrm{d}x\,,$$

som vi gjerne vil bruke vektingsutvelging på. Vi finner da en passene sannsynlighetsdistribusjonsfunksjon $p(x)$ som oppfyller,

$$\int_a^b p(x)\,\mathrm{d}x\, = 1,$$

samt at den er positiv definitt, integrerbar og at integralet av den er invertibelt. Vi setter $p(x)$ inn i I og får,

$$I = \int_a^b p(x)\frac{f(x)}{p(x)}\,\mathrm{d}x\,.$$

Vi er nå interessert i å gjøre i et variableskifte siden tallgeneratoren vår gir oss tall $y \in [0,1]$. Vi får da

$$y(x) = \int_a^x p(x')\,\mathrm{d}x',$$

siden vi kan finne $y(x)$, så kan vi også finne $x(y)$, vi kan skrive,

$$I = \int_{\tilde{a}}^{\tilde{b}} \frac{f(x(y))}{p(x(y))}\,\mathrm{d}y \approx \frac{1}{N}\sum_{i=0}^{N}\frac{f(x(y_i))}{p(x(y_i))}\,\mathrm{d}, \tag{9}$$

hvor vi brukte at $p(x)\,\mathrm{d}x\, = \,\mathrm{d}y$. Når vi skal implementere dette i programmet vårt, så må ikke glemme at grensene endres!

[1] Eng.: importance sampling

En annen viktig størrelse er variansen til resultatet, det forteller noe om nøyaktigheten til utregningene våre, den er definert som

$$\sigma_f^2 = \left\langle f^2 \right\rangle - \langle f \rangle^2\,.$$

Men vi er mer interessert i hvordan variansen varierer med antall datapunkter N, så vi skriver

$$\sigma_N^2 = \frac{\sigma_f^2}{N}.$$

Ettersom N øker så vil forhåpentligvis σ_N^2 avta, hvilket betyr at resultatet vårt blir mer nøyaktig.

Vi ser nå litt mer på hvordan integralet blir seende ut for de forskjellige integrasjonsmetodene. Først ut er Gauss-Legendre. Der er vektfunksjonen $W(x) = 1$, så den er triviell å faktorisere ut. Integralet som skal kodes inn blir da enkelt og greit,

$$\int_{-\infty}^{\infty} \frac{\mathrm{d}x_1\,\mathrm{d}x_2\,\mathrm{d}x_3\,\mathrm{d}x_4\,\mathrm{d}x_5\,\mathrm{d}x_6\,e^{-2\alpha(\sqrt{x_1^2+x_2^2+x_3^2}+\sqrt{x_4^2+x_5^2+x_6^2})}}{\sqrt{(x_1-x_4)^2+(x_2-x_5)^2+(x_3-x_6)^2}}. \tag{10}$$

For å finne grensene vi skal sette i programmet så plotter vi simpelthen den eksponentielle funksjonen i integranden for å se hvor fort den faller slik at vi kan bestemme hva som er "uendelig" hos oss, vi finner det til å være ved cirka $x = 2$.

Når vi skal bruke Gauss-Laguerre-metoden der integralet er i sfæriske koordinater, så må vi faktorisere ut $W(r_i) = r_i^{\beta}e^{-r_i}$. Transformasjon fra kartesiske til sfæriske koordinater gir oss,

$$I = \int \mathrm{d}\gamma \frac{e^{-2\alpha(r_1+r_2)}r_1^2r_2^2\sin\theta_1\sin\theta_2 4\pi^2}{r_{12}}, \tag{11}$$

hvor vi har $\mathrm{d}\gamma = \mathrm{d}r_1\,\mathrm{d}r_2\,\mathrm{d}\theta_1\,\mathrm{d}\theta_2\,\mathrm{d}\phi_1\,\mathrm{d}\phi_2$, $4\pi^2$ er Jacobideterminanten, og

$$r_{12} = \sqrt{r_1^2+r_2^2+2r_1r_2(\cos\theta_1\cos\theta_2+\sin\theta_1\sin\theta_2\cos(\phi_1-\phi_2))}.$$

Ved å trekke ut $r_i^2e^{-r_i}$ får vi,

$$I = \int \mathrm{d}\gamma \frac{e^{-(2\alpha-1)(r_1+r_2)}\sin\theta_1\sin\theta_2 4\pi^2}{r_{12}}r_1^2r_2^2e^{-r_1-r_2}, \tag{12}$$

hvor vi da skal sende brøkdelen[2] av integranden til `gaulag`-funksjonen i `lib.h`-biblioteket. Den angulære delen av integralet blir integrert med Gauss-Legendre på samme måte som for kartesiske koordinater.

Hvordan tar dette seg så ut når vi skal integrere det med Monte Carlo-metoder? Vi skal først gjøre det rett fram og integrere i kartesiske koordinater. Vi må bruke $z = a - (b - a)x$, der $x \in [0, 1]$, da tryller vi på magisk vis tall fra en uniform distribusjon ut over hele integrasjonsintervallet vårt. Mer trenger vi ikke å gjøre med integranden når den skal integreres i kartesiske koordinater. Så langt så vel.

Vi kan imidlertid gjøre integrasjonen på en litt mer elegant måte, vi kan bytte til sfæriske koordinater og bruke den eksponentielle distribusjonen slik at vi vår et resultat som konvergerer raskere mot det analytiske resultatet. Integralet vårt blir i sfæriske koordinater fant i stad, se (12) Vi ser at vi kan trekke ut en faktor e^{-r_i}, $i = 1, 2$, og dermed bruke den eksponentielle distribusjonen. Variabelskifte av den eksponentielle distribusjonen gir oss $r = -\ln(1 - x)$, $x \in [0, 1]$. Integranden endrer seg litt når vi gjør prosedyren som i (9),

$$\int \mathrm{d}\gamma \frac{e^{-(2\alpha-1)(r_1+r_2)} \sin\theta_1 \sin\theta_2 r_1^2 r_2^2 4\pi^2}{r_{12}}.$$

Integrasjonsgrensene for r_i blir så fra o til ∞ ved at vi bruker $r_i = -\ln(1 - x)$ for $x \in [0, 1]$. Vi er nå klare til å gå løs på selve programmeringa!

Kort oppsummert: Det vi må gjøre når vi skal integrere med Monte Carlo er å velge antall genererte tall N, sørge for at integrasjonsgrensene går fra 0 til 1, summere opp alle verdier og så gange med Jacobideterminanten/-volumelementet og dele på antall punkter som er brukt.

Metode

Vi kan nå gå mer gjennom algoritmen og gangen i programmet `main.cpp`. Programmet er delt opp i fire, en for hver integrasjonsmetode. For Gauss-Legendre har vi to funksjoner, en som gjør integrasjonsrutinen og en som regner ut funksjonsverdien for et sett med punkter. Denne måten å organisere programmet på er valgt for enkelt å kunne kjøre bare én integrasjonsmetode av gangen. Inne i funksjonen for Gauss-Legendre-metoden så definerer vi integrasjonsgrensene, og `double`-objekter til å lagre integrasjonspunktene og vektingsfaktorene. Vi sender så inn dette til funksjonen `gauleg`, den returnerer to lister som er fylt med integrasjonspunkter og vektingsfaktorer. Vi går så inn i en seksdobbel `for`-løkke som summerer opp funksjonsverdien ganget med de seks vektingsfaktorene, en for hver dimensjon, og det er hele prosedyren! Det kan bemerkes at det kalles på `gauleg` kun en gang siden den seks-doble `for`-løkka kaller på seks forskjellige verdier fra de samme listene for hver itersjon, på den måten vil alle kombinasjonsmuligheter blir brukt slik at vi dermed integrerer over hele volumet vi har satt.

For Gauss-Laguerre er det derimot mer som må gjøres. På samme måte som i stad definerer vi variable som inneholder integrasjonsgrenser etc., men denne gangen gjør vi ett kall på `gauss_laguerre`, den gir oss oss integrasjonspunktene og vektingsfaktorene fra Gauss-Laguerre-metoden. Videre gjør vi to kall på `gauleg` siden integrasjonsgrensene ikke er like, *i.e.* $\theta \in [0, \pi]$ og $\phi \in [0, 2\pi]$. Vi har nå tre sett med integrasjonspunkter og vektfaktorer, vi gjenbruker den seksdoble `for`-løkka fra i stad og summerer sammen bidragene.

Vi kommer så til Monte Carlo-metodene. Når vi skal bruke rett-fram-metoden så må vi først transformere koordinatene slik at integrasjonsgrensene passer med den uniforme fordelinga og huske på å gange med Jacobideterminanten som blir

$$\prod_{i=0}^{D}(a_i - b_i),$$

for D dimensjoner. Siden vi her får $a_i = -b_i$, så blir Jacobideterminanten bare volumet vi integrerer over. `for`-løkka vil for denne metoden være dobbel, en som går over integrasjonspunktene og en som går over dimensjonene og genererer verdier. Etter N runder i løkka så har vi et tall som skal ganges med Jacobideterminanten og deles på N, og så har vi verdien av integralet. Vi finner også standardavviket og relativ feil for å se hvor korrekte resultatene er.

Til slutt har vi rosinen i pølsa, en forbedret Monte Carlo-metode der vi bruker sfæriske koordinater. Forskjellen fra forrige metode er heller liten, vi får en ny

[2] No pun intended.

Jacobideterminant på $4\pi^4$ og nye `for`-løkker for de forskjellige dimensjonene.

Hele dette programmet er en enhetstest, vi vet verdien av integralet vi skal regne ut, og det er derfor ikke lagt inn noen spesielle funksjoner for å teste programmet. Vi kunne sjekket om RNG-en vår fungerer ved å plotte de genererte tallene, ev. sjekket om de tilfredsstilte den uniforme fordelinga, men siden programmet gir korrekt resultat antar vi det er som en følge av at RNG-en fungerer som den skal.

Resultat

Programmet har kjørt med fire forskjellige metoder og hver av de med fem eller seks forskjellige antall integrasjonspunkter. Resultatene er listet opp i tabell (1) og (2)

I_{Legendre}	I_{Laguerre}	$\epsilon_{\text{Legendre}}$	$\epsilon_{\text{Laguerre}}$	n
0.129834	0.181567	0.326466	0.058093	10
0.199475	0.195887	0.034804	0.016192	15
0.177065	0.195636	0.081449	0.014892	20
0.189110	0.195240	0.018967	0.012837	25
0.185796	0.195070	0.036158	0.011955	30

Tabell 1: *Resultat fra kjøringer av begge GK-metodene. Ingen av de er spesielt gode sammenliknet med Monte Carlo-metodene. Når vi hadde ti integrasjonspunkter så betydde det 10^6 utregninger siden det var en seksdobbel løkke, en lite effektiv måte å angripe problemet på.*

Vi kan spørre oss hvorfor noen metoder fungerer bedre enn andre. I vårt tilfelle så har vi et seksdimensjonalt integral, det krever veldig mange punkter om man skal regne det ut. Vi integrerer altså over en seksdimensjonal hyperkube med sider L og dimensjon d, den inneholder $N = (L/h)^d$ datapunkter, hvor h er steglengden. Feilen på resultatet skalerer da med en faktor $N^{-k/d}$, hvor k er potensen som trunkeres, $\mathcal{O}(h^k)$. For Monte Carlo-metoder, som baserer seg på statistiske metoder, så vil feilen *alltid(!)* skalere som $\sigma \sim 1/\sqrt{N}$.

$I_{\text{brute force}}$	$I_{\text{spherical}}$	σ_{bf}	σ_{sph}	n
0.093212	0.181563	137.131	0.069448	10^3
0.150396	0.182732	67.442177	0.023418	10^4
0.222673	0.194373	27.162678	0.006958	10^5
0.190191	0.192976	8.009405	0.001937	10^6
0.179076	0.192717	2.021568	0.000792	10^7
0.192754	0.192789	0.436883	0.000230	10^8

Tabell 2: *Resultat fra kjøringer av begge Monte Carlo-metodene med forskjellige N. Vi ser at den forebedra metoden virkelig er forbedra, den gir et mer presist resultat for lavere N og konvergerer raskt, mens rett-fram-metoden er virkelig dårlig i starten, men blir litt bedre for veldig store N.*

Vi ser også litt på tidsbruken til de forskjellige metoden. I tabell (3) og (4) ser vi tidsbruken for hhv. GK og Monte Carlo-metodene. Ikke overraskende ser vi at Monte Carlo gir oss best resultat også med tanke på tidsbruk. Vi trenger bare et halvt minutt på å få et resultat med fire desimalers presisjon.

n	t_{Legendre}	t_{Laguerre}
10	0.20 s	0.16 s
15	1.88 s	1.77 s
20	9.88 s	9.98 s
25	38.42 s	38.14 s
30	115.48 s	114.97 s

Tabell 3: *Tidsbruken for de to GK-metodene er ganske lik, dette er ikke så overraskende siden de gjør så å si det samme, eneste forskjellen er at det blir regnet ut vektingsfaktorer og integrasjonspunkt tre ganger i stedet for én, men det er ikke den tidkrevende prosessen i integrasjonen. Det er den seksdoble for-løkka som tar opp tida vår.*

n	$t_{\text{MC rett-fram}}$	t_{MC}
10^3	0.000306 s	0.000333 s
10^4	0.003002 s	0.002974 s
10^5	0.030057 s	0.028833 s
10^6	0.3038 s	0.293207 s
10^7	3.13475 s	2.9413 s
10^8	31.2377 s	29.4754 s

Tabell 4: *Dette er en langt raskere metode enn GK sammenlignet med hvor nøyaktig resultatet blir. Vi trenger "bare" et halvt minutt på å få et resultat med fire desimalers presisjon, mens vi på nesten to minutter med GK fikk på det beste to desimalers presisjon.*

Numerisk stabilitet og presisjon

Vi ser av resultatene at den siste metoden var definitivt den beste. Den hadde et akseptabelt resultat for laveste verdien vi hadde av N og konvergerte raskt mot den analytiske løsningen. For vår største N hadde den det riktige svaret med fire desimalers nøyaktighet.

Programmet er stabilt, men stabilt dårlig for de dårlige metodene. Alle metoder konvergerer mot den samme løsningen, men med forskjellig hastighet og presisjon.

Konklusjon

Vi har i dette prosjektet sett på to ulike metoder å integrere på, den ene brukte ortogonale polynomer til å finne ut hvordan integralet skulle vektes, mens den andre genererte tilfeldige tall mellom 0 og 1 og brukte så en distribusjonsfunksjon til å distribuere disse i en fordeling som passet bedre til integranden. Vi har kjørt for forskjellig antall datapunkter og vist at Monte Carlo-metoder er den beste metoden i vårt tilfelle. Kjøretidene varierte i alt fra et brøkdels sekund til nesten to minutter.