Tanker om objekt-orientert partikkelsporingsmodellering

author: Bjørn Ådlandsvik
date: 2014-01-17

Prøver å tenke generelt, dekke både drivbaner ogv partikkelfordelinger. Tenker ikke mye på effektivitet i implementering (ennå).

Grunnleggende begreper og klasser

Point

Dette er den grunnleggende klassen, med attributter for horisontal posisjon, dyp, tid og partikkel-id med muligheter for alle mulige subklasser + eventuelt et flagg for aktiv/inaktiv.

Navnet **point** tar jeg fra CF-1.6 appendix H1. Jeg er ikke helt fornøyd med det, i mitt hode høres det ut som noe som ligger fast, pluss at punkter i rommet er punkter selv om der aldri kommer en partikkel forbi. Jeg har av og til sagt *particle realization* (abstrakt og tungvint), fysikere kaller et punkt i rom og tid for *event* (som er noe annet i datahandsaming). Ordet *LagrangianElement* kunne også ha vært brukt (men det er teknisk og Knut-Frode bruker det i annen betydning under) Bruker begrepet punkt videre i mangel av noe bedre.

Trajectory (drivbane)

Med en **trajectory** menes hele "livshistorien" til en partikkel, det vil si en sekvens av points med samme particle-id. Dette er vel det samme som Knut-Frode kaller et *LagrangianElement*. CF-appendikset bruker betegnelsen *trajectory*. Jeg er fristet til å kalle det en *particle* siden en partikkel beholder sin identitet mens tiden går.

Particle distribution

En **particle distribution** er en mengde points med samme tid. I modellsammenheng har jeg kalt dette *state*, siden det representerer modell-tilstanden ved et gitt tidspunkt.

Particle load (partikkelbelastning)

Dette er integralet av antall punkter som er i et område i et tidsintervall. En partikkel som er i området lenge kan gi samme belasting som mange partikler som driver raskt gjennom. Vi bruker f.eks. dette for lakselus. Som veldig unge er de ufarlig for laks. Deretter kan de smitte en laks i en viss periode. Hvis de ikke har funnet en vert innen

den tid dør de. Det som er interessant for oss er nettopp partikkelbelastningen, hvor mange aktive lus det er i området over tid.

Også her strever vi litt med nomenklaturen, både for teknisk og for å kommunisere med andre. Belastning er greitt for lakselus, men blir for negativt når det gjelder fiskelarver i nærheten av et fuglefjell. Engelsk *load* er bedre, kan kanskje oversettes med *last* istedet for belastning. Jeg vil tro at lignende begrep brukes i oljedriftsammenheng.

Tid

Jeg går utfra at modellen har et fast tidskritt (ser bort fra muligheten å bruke kortere tidskritt nær land og i områder med høyere variabilitet). Tiden kan da representeres som antall tidskritt, altså som et heltall. Det er vel naturlig å bruke null for simuleringens start slik at klokken går likt for alle partikler. Alternativt kan hver drivbane ha sin egen tidsregning fra sitt starttidspunkt, med metoder for omregning til/fra absolutt tid.

Partikkel-id (pid)

I motsetning til tid som går framover i en ordnet sekvens, kan partikkel-id være hva som helst (personnummer, strek-kode, ...). Jeg foreslår imidlertid en grei konvensjon. Partikkel-id er et heltall, starter mest naturlig med 1 (men 0 i python?). Partiklene numereres etter når de "fødes", med tilfeldig rekkefølge for de som har samme starttidspunkt.

Ved enhver tid er høyeste pid antall drivbaner som enten er aktive ved eller har vært aktive før dette tidspunkt (max(pid)+1 i python).

Punkt-array

Den store samlende datastrukturen er mengden av alle points. Med konvensjonene over kan dette organiseres som et 2D array av punkter, indeksert av pid og tid. (pid og tid er da unødvendige som attributter, siden de er indekser). Mer presist vil det være et maskert array, hvor inaktive partikler (ikke sluppet ut ennå, strandet, dødd, drevet ut av området ...) er maskert vekk. Dette arrayet må kunne vokse i begge dimensjoner ved at tiden går og eventuelt nye partikler slippes ut. Python har ikke problem med en slik struktur, mens Fortran må ha maksimaldimensjonene for arrayet spesifisert på forhånd (det kan gjøres, simuleringsperiode og utslippstrategi er normalt gitt på forhånd).

Ved relativt få partikler og kort simuleringstid kan hele strukturen holdes i minne. Dette gir større frihet i hvordan resultatene skrives ut. Det er f.eks. mulig å simulere hele drivbanen til en partikkel før en starter på neste. Det mest normale er vel likevel en "state space"-tankegang. Tilstanden er her partikkelfordelingen, alle aktive punkter ved gitt tidspunkt, og modellen beregner neste tilstand utfra nåværende (og evt. noen nylige) tilstander og input. Etter periodevis utskrift kan gamle tilstander slettes og minnet frigjøres.

Output-format

Med relativt få partikler og kort simuleringstid kan mange format brukes, bl.a. tekstbaserte. Med mange punkter er det naturlig å bruke netCDF som ellers er vanlig i oseanografi og meteorologi. CF-standarden nevner (fra versjon 1.6, kap. 9) flere måter å lagre slike data.

Hvis alle partikler er aktive hele tiden er strukturen et vanlig 2D array, som representeres naturlig i NetCDFs standard struktur "orthogonal multidimensional array representation". ROMS sin on-line partikkeltracking gjør dette. En enkel generalisering er "incomplete multidimensional array representation" hvor det brukes en Fill_value for inaktive punkter. NetCDF4 gjør dette enda bedre ved at både pid og tid kan være ubegrensete dimensjoner og inline komprimering effektivt tar hånd om overheaden i datastrukturen.

Ladim bruker i dag en "contiguous ragged array representation" hvor partikkelfordelingene ved gitt tid er sammenhengende (contiguous) med supplerende count-array for å finne partikkelfordelingene. Dette er optimalisert for å studere fordelingene, men siden pid lagres kan også drivbanene gjenfinnes. Fordelingen ved gitt tid er sortert etter økende pid, og dersom ingen tidligere partikler er blitt inaktiv beholder en partikkel sin indeks (og den blir ihvertfall ikke høyere). Dette betyr at en kan bruke binær søking med godt utgangspunkt.

CF-standarden beskriver dessverre ikke en slik struktur, men gjør mer av den *transponerte* i appendix H.4.3 "contiguous ragged array representation of trajectories". Denne optimaliserer analyse av drivbanene og gjør det tungvint (men ikke umulig) å få ut partikkelfordelingene. Med lange simuleringer kan det være vanskelig å skrive ut dette formatet, siden drivbanene må holdes i minnet til de er ferdige.

NetCDF4 åpner for nyere strukturer, som "variable length" arrays. Dette kan være et godt alternativ til "ragged" arrays. Hverken eksisterende (1.6) og draft versjon (1.7) av CF-standarden snakker om NetCDF4.

Det er relativt greitt å kunne håndtere flere utskriftsversjoner, men en standardisering gjør det greiere å lage skript og programmer for plotting og analyse. Vi bør og se på hva andre miljøer gjør. Det krever ofte at en må sette seg ned å sjekke programvaren, dokumentasjonen kan være så som så på dette området.