Politechnika Warszawska

WYDZIAŁ FIZYKI

Praca dyplomowa inżynierska

na kierunku Fizyka Techniczna
w specjalności Fizyka komputerowa

Zoptymalizowana symulacja Particle in Cell w języku Python

Dominik Stańczak
261604

promotor
dr Sławomir Jabłoński

WARSZAWA 2017

Streszczenie

Tytuł pracy: Zoptymalizowana symulacja Particle in Cell w języku Python

1 Streszczenie

Utworzono kod symulacyjny Particle-in-Cell mający modelować interakcję relatywistycznej plazmy wodorowej oraz impulsu laserowego. Kod zoptymalizowano w celu zademonstrowania możliwości użycia wysokopoziomowego języka programowania wywołującego niskopoziomowe procedury numeryczne do osiągnięcia wysokiej wydajności obliczeniowej. *Słowa kluczowe:*

python, plazma, particle in cell, symulacja, optymalizacja, elektrodynamika

(podpis opiekuna naukowego) (podpis dyplomanta)

Abstract

Title of the thesis: Optimised Particle in Cell simulation in Python

2 Abstract

A Python particle-in-cell plasma simulation code is developed to model the interaction between a hydrogen plasma target and a laser impulse. The code is then optimized to demonstrate the possibilities of using a high level programming language to call low level numerical procedures, thus achieving high computational efficiency. *Słowa kluczowe:*

<python, plasma, particle in cell, simulation, optimization, electrodynamics>

(podpis opiekuna naukowego) (podpis dyplomanta)

Oświadczenie o samodzielności wykonania pracy

Politechnika Warszawska
Wydział Fizyki

Ja niżej podpisany/a:

Dominik Stańczak, 261604

student/ka Wydziału Fizyki Politechniki Warszawskiej, świadomy/a odpowiedzialności prawnej przedłożoną do obrony pracę dyplomową inżynierską pt.:

Zoptymalizowana symulacja Particle in Cell w języku Python

wykonałem/am samodzielnie pod kierunkiem

dr Sławomira Jabłońskiego

Jednocześnie oświadczam, że:

- praca nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 o prawie autorskim i prawach pokrewnych, oraz dóbr osobistych chronionych prawem cywilnym,
- praca nie zawiera danych i informacji uzyskanych w sposób niezgodny z obowiązującymi przepisami,
- praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem dyplomu lub tytułu zawodowego w wyższej uczelni,
- promotor pracy jest jej współtwórcą w rozumieniu ustawy z dnia 4 lutego 1994 o prawie autorskim i prawach pokrewnych.

Oświadczam także, że treść pracy zapisanej na przekazanym nośniku elektronicznym jest zgodna z treścią zawartą w wydrukowanej wersji niniejszej pracy dyplomowej.

Warszawa, dnia 2017 (podpis dyplomanta)

Oświadczenie o udzieleniu Uczelni licencji do pracy

Politechnika Warszawska
Wydział Fizyki

Ja niżej podpisany/a:

Dominik Stańczak, 261604

student/ka Wydziału Fizyki Politechniki Warszawskiej, niniejszym oświadczam, że zachowując moje prawa autorskie udzielam Politechnice Warszawskiej nieograniczonej w czasie, nieodpłatnej licencji wyłącznej do korzystania z przedstawionej dokumentacji pracy dyplomowej pt.:

Zoptymalizowana symulacja Particle in Cell w języku Python

w zakresie jej publicznego udostępniania i rozpowszechniania w wersji drukowanej i elektronicznej*.

Warszawa, dnia 2017 (podpis dyplomanta)

---

*Na podstawie Ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (Dz.U. 2005 nr 164 poz. 1365) Art. 239. oraz Ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz.U. z 2000 r. Nr 80, poz. 904, z późn. zm.) Art. 15a. "Uczelni w rozumieniu przepisów o szkolnictwie wyższym przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli uczelnia nie opublikowała pracy dyplomowej w ciągu 6 miesięcy od jej obrony, student, który ją przygotował, może ją opublikować, chyba że praca dyplomowa jest częścią utworu zbiorowego."

Spis treści

1 Wstęp

Algorytmy Particle-in-Cell (cząstka w komórce) to jedne z najbardziej zbliżonych do fundamentalnej fizyki metod symulacji materii w stanie plazmy. Zastosowany w nich lagranżowski opis cząsteczek pozwala na dokładne odwzorowanie dynamiki ruchu elektronów i jonów. Jednocześnie, ewolucja pola elektromagnetycznego na Eulerowskiej siatce dokonywana zamiast bezpośredniego obliczania oddziaływań międzycząsteczkowych pozwala na znaczące przyspieszenie etapu obliczenia oddziaływań międzycząsteczkowych. W większości symulacji cząsteczkowych właśnie ten etap jest najbardziej krytyczny dla wydajności progamu.

W ostatnich czasach symulacje Particle-in-Cell zostały wykorzystane między innymi do

dło, meta

symulacji przewidywanej turbulencji plazmy w reaktorze termojądrowym ITER

modelowania rekonekcji linii magnetycznych w polu gwiazdy

dło

projektowania silników jonowych (Halla)

dło

badania interakcji laserów z plazmą w kontekście tworzenia niewielkich, wysokowydajnych akceleratorów cząstek

dło

Należy zauważyć, że w świetle rosnącej dostępności silnie równoległej mocy obliczeniowej w postaci kart graficznych możliwości algorytmów Particle-in-Cell będą rosły współmiernie, co może pozwolić na rozszerzenie zakresu ich zastosowań. Przykładem takiego projektu jest PIConGPU

dło

Inżynieria oprogramowania zorientowanego na wykorzystanie możliwości kart graficznych, jak również w ogólności nowoczesnych symulacji wykorzystujących dobrodziejstwa nowych technologii jest jednak utrudniona poprzez niskopoziomowość istniejących języków klasycznie kojarzonych z symulacją numeryczną (C, FORTRAN) oraz istniejących technologii zrównoleglania algorytmów (MPI, CUDA, OpenCL).

To sprawia, że pisanie złożonych programów symulacyjnych, zwłaszcza przez osoby zajmujące się głównie pracą badawczą (na przykład fizyką eksperymentalną), a nie tylko programowaniem, jest znacznie utrudnione. Należy też zauważyć, że programy takie często są trudne, jeżeli nie niemożliwe do weryfikacji działania, ponownego wykorzystania i modyfikacji przez osoby niezwiązane z oryginalnym autorem z powodów takich jak

- brak dostępności kodu źródłowego
- niedostateczna dokumentacja
- brak jasno postawionych testów pokazujących, kiedy algorytm działa zgodnie z zamiarami twórców
- zależność działania kodu od wersji zastosowanych bibliotek, sprzętu i kompilatorów

Niniejsza praca ma na celu utworzenie kodu symulacyjnego wykorzystującego metodę Particle-in-Cell do symulacji oddziaływania wiązki laserowej z tarczą wodorową w popularnym języku wysokopoziomowym Python, przy użyciu najlepszych praktyk tworzenia reprodukowalnego, otwartego oprogramowania i zoptymalizowanie go w celu osiągania maksymalnej wydajności i sprawności obliczeniowej. Może to też oczywiście pozwolić na dalsze zastosowanie kodu w celach badawczych i jego dalszy rozwój, potencjalnie z użyciem kart graficznych. Ostatecznie, jest to

również test wydajnościowy możliwości Pythona w symulacjach numerycznych.

TODO: s
ostateczn

2 Część analityczno-teoretyczna

2.1 Plazma - czwarty stan materii

Plazma, powszechnie nazywana czwartym stanem materii, to zbiór zjonizowanych cząstek oraz elektronów przejawiających jako grupa globalną obojętność elektryczną. Innymi słowy, od gazu plazmy odróżnia fakt, że cząstki są zjonizowane, więc oddziałują kolektywnie między sobą na odległość, ale ich pola elektryczne wzajemnie się neutralizują na długich dystansach.

formalna
plazmy

Plazmy występują w całym wszechświecie, od materii międzygwiezdnej po błyskawice. Ich istnienie uwarunkowane jest obecnością wysokich energii, wystarczających do zjonizowania atomów gazu.

Fizyka plazmy jest stosunkowo młodą nauką, której rozwój nastąpił dopiero w ostatnim stuleciu, zaczynając od badań Langmuira (1928), który eksperymentował z jonizowaniem gazów w szklanych rurach zwanych rurami Crookesa.

TODO re

Globalny wzrost zainteresowania fizyką plazmy na arenie geopoliycznej ozpoczął się w latach ’50 ubiegłego wieku, gdy uświadomiono sobie, że można zastosować ją do przeprowadzania kontrolowanych reakcji syntezy jądrowej, które mogą mieć zastosowania w energetyce jako następny etap rozwoju po reakcjach rozpadu wykorzystywanych w "klasycznychęlektrowniach jądrowych. Był to jeden z elementów zimnowojennego wyścigu technologicznego międzu Stanami Zjednoczonymi a ZSRR, jak również jeden z projektów mających na celu ponowne nawiązanie współpracy naukowej między supermocarstwami po zakończeniu tego konfliktu.

TODO -
zweryfiko

TODO: re
fusion in
history of

TODO re

Poza tym ogromnym projektem plazmy mają szerokie zastosowania w obecnym przemyśle, na przykład:

- metalurgicznym - przecinaki plazmowe
- elektronicznym - nacinanie powierzchni urządzeń półprzewodnikowych
- materiałowym - powierzchniowa obróbka materiałów, CVD
- kosmicznym - silniki plazmowe, interakcja z rozgrzanym powietrzem podczas powtórnego wchodzenia w atmosferę
- użytkowym - ekrany telewizorów, oświetlenie (świetlówki)
- utylizacja odpadów
- czyszczenie powierzchni

TODO re
fusion for
history of

TODO: s
nazwę - ł
coś?

TODO:
przeformu

Należy też zwrócić uwagę, że ze względu na złożoność układów plazmowych pre-komputerowa fizyka miała ogromne problemy z merytorycznymi badaniami zachowania plazmy poza wybranymi, mocno uproszczonymi reżimami. Postęp w badaniach plazmy, jak sugeruje rozwój technologii fuzyjnej, jest silnie skorelowany z rozwojem mocy obliczeniowej oraz algorytmów symulacyjnych.

TODO re

TODO re

TODO: to
potrzebuj

TODO re

TODO re

TODO co

2.2 Modelowanie i symulacja plazmy

Zjawiska z zakresu fizyki plazmy są jednymi z bardziej złożonych problemów modelowanie komputerowej. Głównym, koncepcyjnie, powodem uniemożliwiającym zastosowanie prostych metod symulacji znanych z newtonowskiej dynamiki molekularnej jest mnogość oddziaływań - każda cząstka oddziałowuje z każdą inną nawzajem poprzez niepomijalne oddziaływania kulombowskie, skalujące się z odległością jak $\approx r^{-2}$.

Z powodu dużej liczby cząstek w układach plazmowych, jedynymi praktycznymi podejściami fundamentalnymi (jako opierającymi się na fundamentalnej fizyce) są opisy statystyczne, opierające się na modelach kinetycznych. Wielkością opisującą plazmę jest tu funkcja dystrybucji zdefiniowana jako

$$\int \int f(\vec{x}, \vec{v}, t) d\vec{x} d\vec{v} = 1 \tag{1}$$

opisująca gęstość prawdopodobieństwa rozkładu plazmy w N-sześciowymiarowej przestrzeni fazowej (po trzy wymiary na położenia oraz prędkości, powielone dla każdej z N cząstek).

Podstawowym równaniem statystycznym opisującym plazmę jest równanie Vlasova

$$\frac{d}{dt} f_\alpha(\vec{x}, \vec{v}, t) - \nabla f - \nabla_{\vec{v}}(\vec{v} \times \vec{B} + \vec{E}) = f_{coll} \tag{2}$$

W praktyce jest ono również nierozwiązywalne poza trywialnymi przypadkami o ułatwiających problem symetriach. Jednym z powodów jest konieczność uzyskania dobrej rozdzielczości prędkości przy jednoczesnym zachowaniu zakresów obejmujących prędkości relatywistyczne. Należy zauważyć, że skalowanie liczby punktów na siatce tego typu jest proporcjonalne do $N_r^3 N_v^3$, gdzie N_r to liczba punktów przestrzennych, zaś N_v to liczba punktów na siatce prędkości. Jest to więc niepraktyczne obliczeniowo, między innymi ze względu na istotne w plazmach zjawisko "uciekających elektronów" o dużych prędkościach.

W modelowaniu komputerowym plazmy stosuje się dwa główne podejścia:

1. modele kinetyczne
2. modele płynowe oparte na ciągłym opisie plazmy poprzez uśrednienie po dystrybucji wielkości termodynamicznych, co daje modele takie jak magnetohydrodynamikę
3. modele dyskretne oparte na samplowaniu dystrybucji plazmy przy użyciu dyskretnych cząstek (matematycznie jest to równanie Klimontowicza przybliżające równanie Vlasova)

Prawdopodobnie najpopoularniejszym modelem z tej drugiej kategorii są modele Particle-in-cell.

2.3 Modele Particle-in-cell

Idea modelu particle-in-cell jest wyjątkowo prosta i opiera się na idei przyspieszenia najbardziej złożonego obliczeniowo kroku symulacji dynamiki molekularnej, czyli obliczania sił

międzycząsteczkowych. Cząstki poruszają się w ciągłej, Lagrange'owskiej przestrzeni. Ich ruch wykorzystywany jest do zebrania informacji dotyczącej gęstości ładunku i prądu na dyskretną, Eulerowską siatkę. Na siatce rozwiązane są (jako równania różniczkowe cząstkowe) równania Maxwella, dzięki którym otrzymuje się pola elektryczne i magnetyczne, które z powrotem są przekazane do położeń cząstek. Obliczeniowo, uwzględniając koszty odpowiednich interpolacji, pozwala to zredukować złożoność kroku obliczenia sił międzycząsteczkowych do $n \log n$ z n^2

Algorytm particle-in-cell składa się z czterech elementów

TODO: w złożonoś przez roz

GRAFIKA cykliczny schemac

TODO: d wzór

2.3.1 GATHER

Depozycja ładunku oraz prądu z położeń cząstek do lokacji na dyskretnej siatce poprzez interpolację, co pozwala na sprawne rozwiązanie na tej siatce równań Maxwella jako układu różnicowych równań cząstkowych zamiast obliczania skalujących się kwadratowo w liczbie cząstek oddziaływań kulombowskich między nimi. $1 = \sum_i S_i$ W naszym przypadku najbardziej istotnym

2.3.2 SOLVE

Sprawne rozwiązanie równań Maxwella na dyskretnej, Eulerowskiej siatce; znalezienie pól elektrycznego i magnetycznego na podstawie gęstości ładunku i prądu na siatce. Istnieją dwie główne szkoły rozwiązywania tych równań: metody globalne i lokalne. Metody globalne wykorzystują zazwyczaj równania dywergencyjne, rozwiązywane iteracyjnie lub spektralnie, zaś lokalne - równania rotacyjne. Metody globalne nadają się do modeli elektrostatycznych, nierelatywistycznych. Metody lokalne pozwalają na ograniczenie szybkości propagacji zaburzeń do prędkości światła, co przybliża metodę numeryczną do fizyki zachodzącej w rzeczywistym układzie tego typu.

2.3.3 SCATTER

Interpolacja pól z siatki do lokacji cząstek, co pozwala określić siły elektromagnetyczne działające na cząstki. Należy przy tym zauważyć, że jako że interpolacja sił wymaga jedynie lokalnej informacji co do pól elektromagnetycznych w okolicy cząstki, ta część algorytmu sprawia, że algorytmy Particle-in-cell doskonale nadają się do zrównoleglania (problem jest w bardzo dobrym przybliżeniu "trywialnie paralelizowalny"). Z tego powodu algorytmy Particle-in-cell nadają się doskonale do wykorzystania rosnącej mocy kart graficznych i architektur GPGPU.

2.3.4 PUSH

iteracja równań ruchu cząstek na podstawie ich prędkości (aktualizacja położeń) oraz działających na nie sił elektromagnetycznych (aktualizacja prędkości).

2.3.5 Makrocząstki

Należy zauważyć, że obecnie nie jest możliwe dokładne odwzorowanie dynamiki układów plazmowych w sensie interakcji między poszczególnymi cząstkami ze względu na liczbę cząstek rzędu liczby Avogadro $\approx 10^{23}$. W tym kontekście bardzo szczęśliwym jest fakt, że wszystkie istotne wielkości zależą nie od ładunku ani masy, ale od stosunku q/m. W praktyce stosuje się więc *makrocząstki*, obdarzone ładunkiem i masą będące wielokrotnościami tych wielkości dla cząstek występujących w naturze (jak jony i elektrony, pozwalając jednocześnie zachować gęstości cząstek i ładunku zbliżone do rzeczywistych.

vielkości

Zazwyczaj (“tradycyjnie”) stosuje się gęstości cząstek (rzeczywistych) rzędu jednej dziesiątej gęstości krytycznej plazmy, która jest opisana wzorem

$$n_c = m_e \varepsilon_0 * (\frac{2\pi c}{e\lambda})^2 \tag{3}$$

2.4 Problem testowy

Problemem testowym, jakiego używamy do przetestowania wydajności działania algorytmu jest interakcja impulsu laserowego z tarczą składającą się ze zjonizowanego wodoru i elektronów.

Układ ten modelowany jest jako jednowymiarowy. Jest to tak zwany w literaturze model 1D-3D. O ile położenia cząstek są jednowymiarowe ze względu na znaczną symetrię cylindryczną układu, cząstki mają prędkości w pełnych trzech wymiarach. Jest to konieczne ze względu na oddziaływania cząstek z polem elektromagnetycznym propagującym się wzdłuż osi układu.

2.5 Python

Python jest wysokopoziomowym, interpretowanym językiem programowania, którego atutami są szybkie prototypowanie,

Python znajduje zastosowania w analizie danych, uczeniu maszynowym (zwłaszcza w astronomii). W zakresie symulacji w ostatnich czasach powstały kody skalujące się nawet w zakres superkomputerów, na przykład w mechanice płynów

FR

Atutem Pythona w wysokowydajnych obliczeniach jest łatwość wywoływania w nim zewnętrznych bibliotek napisanych na przykład w C lub Fortranie, co pozwala na osiągnięcie podobnych rezultatów wydajnościowych jak dla kodów napisanych w C.

3 Implementacja

3.1 Zastosowane algorytmy

3.1.1 Leapfrog oraz Borys

Każda symulacja cząstek wymaga zastosowania integratora równań ruchu. Tradycyjnym

przykładem takiego integratora jest integrator Rungego-Kutty czwartego rzędu, znajdujący zastosowanie w wielorakich symulacjach. TODO: re

Niestety, w bieżącym kodzie nie można go zastosować ze względu na jego niesymplektyczność: mimo ogromnej dokładności jest on niestabilny pod względem energii cząstek. W symulacjach typu Particle-in-cell konieczne jest zastosowanie innych algorytmów. Dobrym algorytmem symplektycznym jest na przykład powszechnie znany *leapfrog*, polegający na przesunięciu prędkości o połowę iteracji czasowej względem położeń.Mimo tego, że energie cząstek w ruchu obliczonym tym integratorem nie są lokalnie stałe na krótkich skalach czasowych, to jednak zachowują energię na skali globalnej. TODO re TODO: re

W przypadku ruchu w polu magnetycznym nie wystarczy, niestety, użyć zwykłego algorytmu *leapfrog*. Używa się tutaj specjalnej adaptacji tego algorytmu na potrzeby ruchu w zmiennym polu elektromagnetycznym, tak zwanego integratora Borysa, który rozbija pole elektryczne na dwa impulsy, między którymi następują dwie rotacje polem magnetycznym. Algorytm jest dzięki temu symplektyczny i długofalowo zachowuje energię cząstek. TODO IM chyba mi na coldpl TODO RI TODO REFERE TODO CI

$$bo = ris \tag{4}$$

W naszym przypadku dochodzi jeszcze jedno utrudnienie związane z relatywistycznością symulacji. Przed obliczeniem korekty prędkości konieczne jest przetransformowanie prędkości z układu "laboratoryjnego" $\vec{v}$ na prędkość w układzie poruszającym się z cząstką $\vec{u}$, czego dokonuje się poprzez parę transformacji: TODO: b pusher e TODO st

$$\vec{u} = \vec{v}\gamma \tag{5}$$

TODO: fi eq

3.1.2 Depozycja gęstości ładunku i prądu

Depozycja ładunku odbywa się w prosty sposób, przy następujących założeniach: TODO pi

- Każda makrocząstka ma własny (wspólny wewnątrz `Species`) ładunek q oraz parametr `scaling` (również) decydujący o tym, ile rzeczywistych cząstek reprezentuje. Sumaryczny ładunek makrocząstki wynosi więc `q*scaling` TODO S
- Każda makrocząstka ma szerokość jednej komórki siatki Δx. Cząstka zlokalizowana więc środkiem w połowie długości komórki będzie w niej całkowicie zawarata.
- W ten sposób możemy stwierdzić, TODO C

3.1.3 Interpolacja pól elektrycznego i magnetycznego

Interpolacja pól elektrycznego i magnetycznego odbywa się na bardzo podobnej zasadzie, co depozycja. Wartosci pol sa liniowo skalowane do pozycji czastek wedlug ich wzglednych położeń

wewnątrz komórek.

W celu przyśpieszenia działania programu stosuje się istniejącą metodę `RegularGridInterpolator` z biblioteki `scipy.interpolate`.

3.1.4 Field solver

Ewolucja pola elektromagnetycznego opisana jest poprzez równania Maxwella. Jak pokazują Buneman i Villasenor, numerycznie można zastosować dwa główne podejścia: 1. wykorzystać równania na dywergencję pola (prawa Gaussa) do rozwiązania pola na całej siatce. Niestety, jest to algorytm inherentnie globalny, w którym informacja o warunkach brzegowych jest konieczna w każdym oczku siatki 2. wykorzystać równania na rotację pola (prawa Ampera i Faradaya), opisujące ewolucję czasową pól. Jak łatwo pokazać (Buneman), dywergencja pola elektrycznego oraz magnetycznego nie zmienia się w czasie pod wpływem tak opisanej ewolucji czasowej:

Co za tym idzie, jeżeli rozpoczniemy symulację od znalezienia pola na podstawie warunków brzegowych i początkowych (gęstości ładunku), możemy już dalej iterować pole na podstawie równań rotacji. Ma to dwie znaczące zalety: * algorytm ewolucji pola staje się trywialny obliczeniowo, zwłaszcza w 1D - ogranicza się bowiem do elementarnych operacji lokalnego dodawania i mnożenia. * algorytm ewolucji pola staje się lokalny (do znalezienia wartości pola w danym oczku w kolejnej iteracji wykorzystujemy jedynie informacje zawarte w tym właśnie oczku i potencjalnie jego sąsiadach co zapobiega problemowi informacji przebiegającej w symulacji szybciej niż światło oraz zapewnia stabilność na podstawie warunku Couranta.

W 1D można dokonać dekompozycji składowych poprzecznych pola elektromagnetycznego (tutaj oznaczanych y, z) na propagujące się w przód (+) i w tył (−) obszaru symulacji. Składowe E_y, B_z są zebrane poprzez zamianę zmiennych w dwie wielkości elektrodynamiczne F^+, F^-.

Wychodzimy z rotacyjnych równań Maxwella:

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \nabla \times \vec{B} = \mu_0(\vec{j} + \epsilon_0 \frac{\partial \vec{E}}{\partial t}) \tag{6}$$

$$F^+ = E_y + cB_z F^- = E_y - cB_z \tag{7}$$

Analogicznie, dla składowych E_z, B_y:

$$G^+ = E_z - cB_y G^- = E_z + cB_y \tag{8}$$

Wyrazem “źródłowym” dla F, G jest prąd poprzeczny. Po dyskretyzacji równania, wyrażenie na ewolucję pól F, G między iteracjami przybiera postać:

$$F^{+\,n+1}_{\;i+1} = F^+_n + j \tag{9}$$

Z tego powodu bardzo istotnym dla dokładności i stabilności algorytmu staje się sposób depozycji ładunku - należy pilnować, aby był robiony w sposób który spełnia zachowanie ładunku. Inaczej koniecznym staje się aplikowanie tak zwanej poprawki Borysa, aby upewnić się, że warunek z równań Maxwella $\nabla\rho/\varepsilon_0 = \nabla \cdot \vec{E}$ jest wciąż spełniony.

Składowa podłużna pola jest obliczana poprzez wyrażenie

$$\frac{\partial E_x}{\partial t} = -\frac{j_x}{\varepsilon_0} \tag{10}$$

czy raczej jej dyskretny odpowiednik

$$E_i^{n+1} = E_i^n - \frac{\Delta t}{\varepsilon_0} j_{x,i}^{n+1/2} \tag{11}$$

3.2 Warunki początkowe dla cząstek

W celu dobrania warunków początkowych wykorzystuje się algorytm opisany w .Jego działanie można łatwo zilustrować na przykładzie początkowej funkcji gęstości cząstek zadanej dowolną funkcją analityczną. Używając funkcji dystrybucji w jednym wymiarze zależnej jedynie od położenia znormalizowanej do liczby cząstek N, można wykonać całkowanie kumulatywne po siatce gęstszej niż liczba cząstek na wybranym przedziale, po czym umieścić cząstki w miejscach, gdzie obliczona dystrybuanta funkcji przybiera kolejne większe całkowite wartości.

Zaimplementowany algorytm jest w stanie przyjąć dowolną funkcję analityczną i zrenormalizować ją tak, aby $\int_0^L f(x)dx = N$. W praktyce wykorzystuje się wartości marginalnie większe niż N, mianowicie $N+0.1$, co pozwala na uniknięcie problemów ze skończoną dokładnością obliczeń na liczbach zmiennoprzecinkowych.

Aby uniknąć problemu w przypadku dwóch `Species` cząstek o identycznej liczbie makrocząstek i przeciwnym znaku które według powyższego algorytmu zostałyby rozłożone w identycznych miejscach z powodu niezależnego stosowania algorytmu dla każdej grupy cząstek, co prowadziłoby do neutralizacji ładunku na całej symulacji, na położenia cząstek nakłada się dodatkowy gaussowski szum o niewielkiej intensywności.

Analogiczny algorytm znajduje zastosowanie w obliczaniu początkowych wartości prędkości dla cząstek. Wykorzystuje się relatywistyczny rozkład Maxwella

$$f(p) = \frac{N}{2\pi}\frac{mc^2}{T}\frac{1}{1+T/mc^2}\exp\Big(\frac{-mc^2}{T}(\gamma-1)\Big)\gamma = \sqrt{1+p^2} \tag{12}$$

Należy wspomnieć, że aby cząstki były prawidłowo ztermalizowane należy zadbać o zdekorelowanie ich prędkości między sobą. Naiwne zastosowanie algorytmu na położenia prowadzi zaś do rozłożenia cząstek rosnąco numeracją w kierunku rosnącego położenia x.

Rozwiązaniem tego problemu jest losowa zamiana prędkości między losowo wybranymi

cząstkami.

3.3 Opis i treść kodu

Cały kod programu w celu reprodukowalności wyników tworzony był i jest dostępny na platformie Github

3.4 Wykorzystane biblioteki i technologie

3.4.1 Numpy

`numpy` to biblioteka umożliwiająca wykonywanie złożonych obliczeń na n-wymiarowych macierzach bądź tablicach, utworzona w celu umożlwiienia zastąpienia operacjami wektorowymi iteracji po tablicach, powszechnie stosowanych w metodach numerycznych i będących znanym słabym punktem Pythona.

Pod zewnętrzną powłoką zawiera odwołania do znanych, wypróbowanych i sprawdzonych w numeryce modułów LAPACK, BLAS napisanych w szybkich, niskopoziomowych językach C oraz FORTRAN. Jest to *de facto* standard większości obliczeń numerycznych w Pythonie.

Numpy jest oprogramowaniem otwartym, udostępnianym na licencji BSD.

3.4.2 scipy

Kolejną podstawową biblioteką w numerycznym Pythonie jest `scipy`, biblioteka zawierająca wydajne implementacje wielu podstawowych algorytmów numerycznych służących między innymi całkowaniu, optymalizacji, algebrze liniowej czy transformatom Fouriera.

3.4.3 Numba

`numba` to biblioteka służąca do kompilacji just-in-time kodu. W wielu przypadkach pozwala na osiągnięcie kodem napisanym w czystym Pythonie wydajności marginalnie niższej bądź nawet równej do analogicznego programu w C. Jednocześnie należy zaznaczyć prostotę jej użycia:

3.4.4 HDF5

HDF5 jest wysokowydajnym formatem plikow służącym przechowywaniu danych liczbowych w drzewiastej, skompresowanej strukturze danych. W Pythonie implementuje go biblioteka h5py. Używa się go na przykład w

W bieżącej pracy wykorzystuje się go do przechowywania danych numerycznych dotyczących przebiegu symulacji, pozwalających na ich dalsze przetwarzanie i analizę poprzez wizualizację.

3.4.5 matplotlib

Do wizualizacji danych z symulacji (oraz tworzenia schematów w sekcji teoretycznej niniejszej pracy) użyto własnoręcznie napisanych skryptów w uniwersalnej bibliotece graficznej `matplotlib`. `matplotlib` zapewnie wsparcia zarówno dla grafik statycznych w różnych układach współrzędnych (w tym 3D), jak również dla dynamicznie generowanych animacji przedstawiających przebiegi czasowe symulacji.

Matplotlib również jest oprogramowaniem otwartym, udostępnianym na licencji

TODO m
license, r

3.4.6 py.test

Przy pracy nad kodem użyto frameworku testowego `py.test` Obsługa testów jest trywialna:

TODO re

przykład

Należy zaznaczyć, że w numeryce, gdzie błędne działanie programu nie objawia się zazwyczaj błędem wykonywania programu, a jedynie błędnymi wynikami, dobrze zautomatyzowane testy jednostkowe potrafią zaoszczędzić bardzo dużo czasu na debugowaniu poprzez automatyzację uruchamiania kolejnych partii kodu i lokalizację błędnie działających części algorytmu. Dobrze napisane testy są praktycznie koniecznością w dzisiejszych czasach, zaś każdy nowo powstały projekt numeryczno-symulacyjny powinien je wykorzystywać, najlepiej do weryfikacji każdej części algorytmu z osobna.

`py.test` jest oprogramowaniem otwartym, dostępnym na licencji

TODO

TODO: tr

TODO: s

3.5 Struktura i hierarchia kodu

Program ma obiektową strukturę zewnętrzną, którą w celu łatwości zrozumienia jego działania nakrywa wewnętrzną warstwę składającą się głównie z n-wymiarowych tablic `numpy.ndarray` oraz zwektoryzowanych operacji na nich.

Część symulacyjna kodu składa się z kilku prostych koncepcyjnie elementów:

3.5.1 Grid

Klasa reprezentująca dyskretną siatkę Eulera, na której dokonywane są obliczenia dotyczące pól elektromagnetycznych oraz gęstości ładunku i prądu. Zawiera:

- x_i - tablicę położeń lewych krawędzi komórek siatki
- N_G - liczbę komórek siatki
- T - sumaryczny czas trwania symulacji
- Δx - krok przestrzenny siatki - $N_G * \Delta x$ daje długość obszaru symulacji
- ρ_i - tablicę gęstości ładunku na siatce.
- $\vec{j}_{i,j}$ - tablicę gęstości prądu na siatce.
- $E_{i,j}$ - tablicę pola elektrycznego na siatce.
- $B_{i,j}$ - tablicę pola magnetycznego na siatce.

- c, ε_0 - stałe fizyczne - prędkość światła oraz przenikalność elektryczną próżni.
- Δt - krok czasowy symulacji, obliczony jako $\Delta t = \Delta x/c$.
- N_T - liczbę iteracji czasowych symulacji.
- `BC` - *Boundary Condition*, funkcję czasu określającą wartość warunku brzegowego dotyczącego natężenia fali elektromagnetycznej (laserowej) wchodzącej do pola symulacji z lewej strony.

Istotne metody klasy `Grid`, o których należy wspomnieć, to:

- `apply_bc` - aktualizuje krańcowe wartości tablic E, B w oparciu o podany warunek brzegowy.
- gather_current
- gather_charge
- solve
- field_solve
- electric_field_function, magnetic
- save_to_h5py

3.5.2 Species

Klasa reprezentująca pewną grupę makrocząstek o wspólnych cechach, takich jak ładunek bądź masa. Przykładowo, w symulacji oddziaływania lasera z tarczą wodorową jedną grupą są protony, zaś drugą - elektrony. Do zainicjalizowania wymaga instancji `Grid`, z której pobiera informacje takie jak stałe fizyczne c, ε_0, liczbę iteracji czasowych N_T i czas trwania iteracji Δt.

Zawiera skalary:

- N - liczba makrocząstek
- q - ładunek cząstki
- m - masa cząstki
- `scaling` - liczba rzeczywistych cząstek, jakie reprezentuje sobą makrocząstka. Jej sumaryczny ładunek wynosi $q*$`scaling`, masa $m*$`scaling`.
- `N_alive` - liczba cząstek obecnie aktywnych w symulacji. Zmniejsza się w miarę usuwania cząstek przez warunki brzegowe.

Poza skalarami zawiera tablice rozmiaru N:

- jednowymiarowych położeń makrocząstek x^n, zapisywanych w iteracjach $n, n+1, n+2...$
- trójwymiarowych prędkości makrocząstek $\vec{v}^{n+\frac{1}{2}}$, zapisywanych w iteracjach $n+\frac{1}{2}, n+32, n+52...$
- stanu makrocząstek (flagi boolowskie oznaczające cząstki aktywne bądź usunięte z obszaru symulacji)

Poza tym, zawiera też informacje dotyczące zbierania danych dot. cząstek:

- `name` - słowny identyfikator grupy cząstek, dla potrzeb legend wykresów
- N_T - liczbę iteracji czasowych w symulacji
- N_T^s - zmniejszoną liczbę iteracji, w których następuje pełne zapisanie położeń i prędkości cząstek. Dane te są wykorzystywane do tworzenia diagramów fazowych cząstek.
- odpowiadające poprzednio wymienionym tablice rozmiaru (N_T^s, N), $(N_T^s, N, 3)$.
- jedną tablicę rozmiaru (N_T, N_G) dotyczącą zebranym podczas depozycji ładunku informacjom

diagnostycznym o przestrzennej gęstości cząstek.
- trzy tablice rozmiaru (N_T) dotyczącą średnich prędkości, średnich kwadratów prędkości i odchyleń standardowych prędkości.

Jeżeli liczba makrocząstek lub iteracji przekracza pewną stałą, dane zapisywane są jedynie dla co n-tej cząstki, gdzie n jest najniższą liczbą całkowitą która pozwala na zmniejszenie tablic poniżej tej stałej.

Warto wspomnieć o metodach klasy `Species`:
- push

TODO: fi

3.5.3 Simulation

Klasa zbierająca w całość Grid oraz dowolną liczbę Species zawartych w symulacji, jak również pozwalająca w prosty sposób na wykonywanie iteracji algorytmu i analizy danych. Jest tworzona tak przy uruchamianiu symulacji, jak i przy wczytywaniu danych z plików .hdf5.

- Δt - krok czasowy
- N_T - liczba iteracji w symulacji
- `Grid` - obiekt siatki
- `list_species` - lista grup makrocząstek w symulacji

3.5.4 Pliki pomocnicze

Poza powyższymi program jest podzielony na pliki:
- algorithms_grid - zawiera algorytmy dot. rozwiązywania równań Maxwella na dyskretnej siatce
- algorithms_interpolation - zawiera algorytmy dot. interpolacji z cząstek na siatkę i odwrotnie
- algorithms_pusher - zawiera algorytmy integrujące numerycznie równania ruchu cząstek
- animation - tworzy animacje dla celów analizy danych
- static_plots - tworzy statyczne wykresy dla celów analizy danych
- Plotting - zawiera ustawienia dot. analizy danych

TODO cz
można p
do simul
gdzieś?

Configi testowe są zawarte w plikach run_*:
- run_coldplasma
- run_twostream
- run_wave
- run_beam
- run_laser

TODO
przeform

Testy jednostkowe są zawarte w katalogu tests:

4 Część weryfikacyjna

Niniejsza analiza przeprowadzona została na “finalnej” w chwili pisania niniejszej pracy wersji programu. W repozytorium gita na Githubie jest to commit “placeholder” identyfikowany również jako wersja 1.0.

TODO: u
commita

4. WERYFIKACJA

4.1 Przypadki testowe

Kod przetestowano w dwojaki sposób. Pierwszym z nich są testy jednostkowe. Poszczególne algorytmy podlegały testom przy użyciu ogólnodostępnego pakietu pytest

Testy polegały na przeprowadzeniu fragmentu symulacji - w przypadku testów algorytmów było to na przykład wygenerowanie pojedynczej cząstki o jednostkowej prędkości oraz zdepozytowanie jej gęstości prądu na siatkę, co pozwala porównać otrzymany wynik z przewidywanym analitycznie dla danego rozmiaru siatki i położenia cząstki. Automatyczne testy jednostkowe uruchamiane po każdej wymiernej zmianie kodu pozwalają kontrolować działanie programu znacznie ułatwiają zapobieganie błędom.

- Gather
  (a) Depozycja prądu z pojedynczej cząstki na niewielką siatkę
  (b) Depozycja prądu z dwóch pojedynczych cząstek na niewielką siatkę i porównanie z sumą prądów dla obu pojedynczyczh cząstek
  (c) Depozycja prądu z dużej ilości równomiernie rozłożonych cząstek
- Solve
  - Symulacja fali sinusoidalnej, obwiedni impulsu i złożenia tych dwóch propagujących się w próżni
- Scatter
  -
- Push
  - Ruch w jednorodnym polu elektrycznym wzdłuż osi układu
  - Ruch w jednorodnym polu magnetycznym z polem magnetycznym

Aby zweryfikować działanie kodu, zastosowano kod do symulacji kilku znanych problemów w fizyce plazmy:

4.1.1 oscylacje zimnej plazmy

Jest to efektywnie fala stojąca. Jednorodne rozmieszczenie cząstek z zerową prędkością początkową (stąd określenie źimna plazma"jako nietermalna) jednego typu na okresowej siatce z jednoczesnym wysunięciem ich z położeń równowagi o $\Delta x = A\sin(kx)$, gdzie $k = n2\pi/L$, pozwala na obserwację oscylacji cząstek wokół ich stabilnych położeń równowagi. W przestrzeni fazowej x, V_x cząstki zataczają efektywnie elipsy, co pozwala wnioskować że ruch ten jest harmoniczny.

Jest to, oczywiście, spełnione jedynie dla niewielkich odchyleń; dla $A \to dx$ obserwuje się nieliniowy reżim

Jest to też łoże testowe dla innych przypadków, takich jak niestabilność Kaiser-Wilhelm oraz

4.1.2 niestabilność dwóch strumieni

Różnice między tym a poprzednim przypadkiem to obecność dwóch jednorodnie rozłożonych strumieni cząstek z przeciwnie skierowanymi prędkościami wzdłuż osi układu.

Dla niewielkich prędkości obserwuje się liniowy reżim

Dla dużych prędkości obserwuje się nieliniowe zachowanie cząstek, które zaczynają się mieszać ze sobą, zaś cały układ się termalizuje.

TODO

TODO: sparamet

TODO bu

4.2 Symulacja oddziaływania lasera z tarczą wodorową

TODO sp

TODO: o

TODO: p

Jako warunki początkowe przyjęto plazmę z liniowo narastającą funkcją rozkładu gęstości (jest to tak zwany obszar prejonizacji)

Gęstość rozkładu plazmy przyjęto jako

Początkowe prędkości cząstek przyjęto jako zerowe.

Za moc lasera przyjęto $10^{23} W/m^2$, zaś za jego długość fali 1.064 μm (jest to laser Nd:YAG)

Długość obszaru symulacji to

Prędkość światła c, stałą dielektryczną ε_0, ładunek elementarny e, masy protonu i elektronu m_p, m_e przyjęto według tablic, jak obrazuje następująca tabela:

TODO: w z relatyw rozkładu w kierunk

TODO AS nie jest z

4.3 Benchmarki - szybkość, zasobożerność

TODO

Do przeprowadzenia testów wydajności kodu użyto

TODO: z tabelkę n

4.4 Problemy napotkane w trakcie pisania kodu

fix

5 Zakończenie

TODO:

Utworzono kod symulacyjny implementacyjny algorytm particle-in-cell w Pythonie przy użyciu wszystkich dostępnych możliwości, jakie daje ekosystem open-source. Kod zoptymalizowano przy użyciu Otrzymane wyniki benchmarków pozwalają sądzić, że