

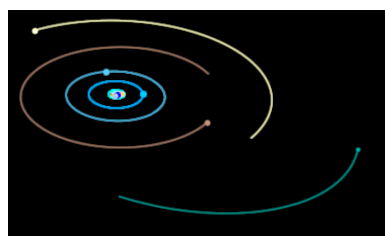
program w Pythonie do rysowania tych krzywych. Dodatkowo (choć nie prosiłem!) dołączył tabelkę z kilkoma sensownymi propozycjami zestawów parametrów prowadzących do różnych efektownych kształtów.

Znowu zapytałem Copilota.

Nie zdziwi nas w tym układzie, że Wielki Mag, zapytany o rozwiązania pewnego równania Riccattiego – w Δ_{21}^7 próbowaliśmy wspólnie z Grzegorzem Łukaszewiczem wykorzystać w tym celu metody grup Liego – z marszu wyprodukował szczegółowe rozwiązanie (powiedziałbym, typowe dla czwórkowego studenta). W dodatku elegancko sformatowane.

W przypadku poważniejszego wyzwania, jakim w Δ_{20}^9 była symulacja ruchu planet w Układzie Słonecznym na przestrzeni 50 000 lat, moi nowi Inteligentni Pomocnicy mieli trochę kłopotów. W zależności od humoru produkowali rozwiązania lepsze lub gorsze. Jednego razu model językowy Qwen3 wiedział, co prawda, że trzeba użyć zachowującego energię schematu symplektycznego (nawet wymienił prawidłowo dwa najbardziej popularne), ale następnie przygotował kod w Pythonie, korzystający nie z nich, tylko z najprostszego schematu Eulera – o którym powszechnie wiadomo, że wyjątkowo źle radzi sobie w tych okolicznościach. I rzeczywiście: symulacja zakończyła się spektakularną porażką, a planety szybko rozsypały się po Wszechświecie.

Więcej o tych schematach pisaliśmy w Δ_{20}^9 .

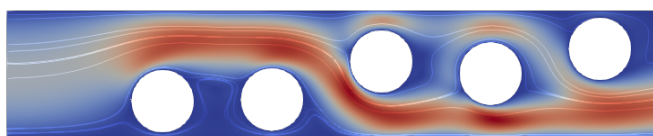


Lecz innego dnia ten sam model był w wyraźnie lepszej formie i poszedł na całość, proponując – zgodnie z duchem wspomnianej na początku rubryki *Pół szklanki mocnego kodu* – wykorzystanie gotowych, mocnych bibliotek: jednej do rozwiązywania równań ruchu, a drugiej do wyznaczenia początkowych położań i prędkości planet na podstawie danych NASA. Efekt był po prostu rewelacyjny – co widać na (zwodniczo niepozornym) rysunku obok, ukazującym finalny fragment symulacji.

Może więc faktycznie żyjemy już w epoce *vibe coding*, kiedy rolą człowieka jest tylko przedstawienie problemu do rozwiązania, a całą resztę – w tym nudne i żmudne kodowanie w Pythonie – ogarnie już tylko Sztuczny Inteligent? Nie odważę się odpowiedzieć na to pytanie, ale w zamian zrobię jeszcze jeden eksperyment.

Skorzystam z wolnodostępnego dużego modelu językowego ogólnego przeznaczenia GPT-OSS-120b. Wiadomo, że są lepsze – lecz ich nie próbowałem.

Zadanie przeprowadzenia komputerowej symulacji przepływu płynu w rurze, wzdłuż której przemieszczają się niewielkie przeszkody (powiedzmy, kulki; zob. rys. poniżej) jest jednym z tych zadań, które z powodzeniem mogłyby trafić do dawnej rubryki *o mocnym kodzie*: dysponujemy na tyle wyrafinowanymi bibliotekami, że rozwiązanie powinno dać się zapisać w zwięzły sposób w Pythonie.



Poprosiłem więc mojego Inteligentnego Pomocnika, aby samodzielnie przygotował taki właśnie kod, opisując mu (niektóre) fizyczne parametry zadania. W odpowiedzi poprawnie domyślił się, jakie brakujące warunki należy dodać, aby otrzymać to, o co mi chodziło, a następnie wyprodukował ładnie wyglądający program w Pythonie.

Nie od razu udało się uruchomić tę symulację. W kodzie było kilka drobnych błędów składniowych, które dopiero w mozolnym procesie kolejnych poprawek (wprowadzanych przez niego) i testów (wykonywanych przeze mnie) zostały w końcu szczęśliwie usunięte przez Sztucznego Programistę.

Jednak wówczas ujawniły się następne, tym razem poważniejsze kłopoty – związane z wybranym algorytmem – i tego już nie udało się przeskoczyć: choć Sztuczny Inteligent starał się, jak mógł, nie był w stanie pójść dalej bez merytorycznej pomocy z mojej strony. Lecz ja, kompletnie nie rozumiejąc, *co się dzieje i o co chodzi* w jego kodzie, miałem w głowie totalną pustkę.

A z pustego to, jak wiadomo, i Salomon nie należy.