Fizyka w szkole i komputery

Jerzy Karczmarczuk
Zakład Informatyki, Uniwersytet w Caen, Francja

Od odczytu danych i sterowania eksperymentami, poprzez symulację i wizualizację, aż do skomplikowanych obliczeń symboliczno-algebraicznych – fizyk bez komputera ani rusz. Wszystko to stanowi nieoddzielne elementy większości dziedzin naukowo-technicznych, z fizyką na czele. Co z tego *potrzebujemy* w szkole średniej?...

Potrzeba dydaktyczna zwykle bywa funkcją osobistej wizji uczącego i żaden program ministerialny tego nie zmieni. Pedagog, który nie lubi komputerów, nie będzie ich używał, a fascynaci mogą łatwo przesadzić. Niejeden wpada w pułapkę używania komputera jako symulowanego „świata", w którym może modelować i ilustrować zjawiska fizyczne, nie przejmując się kłopotami z wyposażeniem rzeczywistej pracowni fizycznej i przesłaniając różnice między światem realnym a artefaktami numerycznymi (i często nie dostrzegając ich, a tutaj łatwo o poważne błędy). Inni, zachęcając uczniów do korzystania z zasobów Internetu, zapominają, że rozstrzelona informacja, niepoparta w miarę spójną metodologią nauczania, rzadko zostaje na trwałe w pamięci. Komputery stają się znakomitą zabawką, i... często nią pozostają...

Sensowne wykorzystanie tego narzędzia wymaga pewnej konsekwencji. Chcieliśmy więc zaproponować Czytelnikom cykliczną rubrykę poświęconą zagadnieniom „fizyki komputerowej", obejmującą głównie techniki obliczeniowe i graficzne. Nie ma to być kącik pokazowy, lecz raczej zachęta do samodzielnej pracy, przeznaczona dla osób, które nie boją się programowania. Pewna znajomość technik obliczeniowych będzie konieczna, aby w pełni skorzystać z tekstów i programów, dołączonych lub dostępnych na stronicach internetowych działu, jednak nie mogą być one zawieszone w próżni, nauczyciele i uczniowie winni dysponować bogatą biblioteką oprogramowania pozwalającego na szybką realizację niewielkich projektów. Proponujemy więc zacząć od kryteriów, jakie winno ono spełniać, zdając sobie sprawę z faktu, że w tej dziedzinie nie ma panaceum.

- Po pierwsze, oprogramowanie winno być darmowe i w miarę możliwości dostępne zarówno pod Windows jak i pod Linuksem. Wtedy można myśleć o wspólnej bazie do pracy w szkole i w domu, a także o kontaktach z zainteresowanymi osobami pracującymi na wyższych uczelniach. Winno być stabilne, przetestowane przez instytucje dydaktyczne i dobrze udokumentowane (ze sporą liczbą przykładów). Oczywiście powinno być łatwe w instalacji.

- Winno być bogate i *zintegrowane*, dysponujące modułami numerycznymi i graficznymi, nadające się do pracy konwersacyjnej i wygodne w testowaniu. Te cechy są nieporównywalnie ważniejsze niż „szybkość" programów.
- Oprogramowanie winno ułatwiać pisanie programów krótkich i czytelnych. Takie będą nasze przykłady, a celem ich będzie nie tylko ich uruchomienie, ale przede wszystkim zrozumienie.

Ale co to jest właściwie „oprogramowanie" w niniejszym kontekście? Składają się nań następujące elementy:

- Język programowania, to oczywiste. Nie należy wpaść w pułapkę szukania najlepszego możliwego, bo go nie ma. Podstawowymi cechami będą dla nas łatwość opanowania i dysponowanie strukturami ułatwiającymi modelowanie naszej fizyki, a także dobra współpraca z modułami graficznymi. To może być (a na początku na pewno będzie) *niewielki* (w sensie struktur składniowych, deklaracji danych itp.) język, wbudowany w specjalistyczny pakiet.
- Interfejs użytkownika: łatwość redagowania i uruchamiania programów, wybór opcji, podręcznik *on-line*, operowanie okienkami graficznymi, poprawianie błędów itp.
- Bogata – jak już wspomnieliśmy – biblioteka z gotowymi podprogramami, modułami graficznymi, demonstracjami itp.

Wybór jest spory i wymaga przeanalizowania wielu możliwości. W ogóle nasz dział będzie miał raczej charakter zaawansowany i przeznaczony będzie dla Czytelników ambitnych i aktywnych. Uzupełnieniem materiałów drukowanych będą nasze stronice internetowe, gdzie Czytelnik znajdzie instrukcje używania niektórych pakietów i języków programowania i będzie mógł korespondencyjnie wyjaśnić swoje wątpliwości. Ich utrzymywanie na bieżąco i rozwój będą jednak zależały w dużej mierze właśnie od echa ze strony Czytelników.

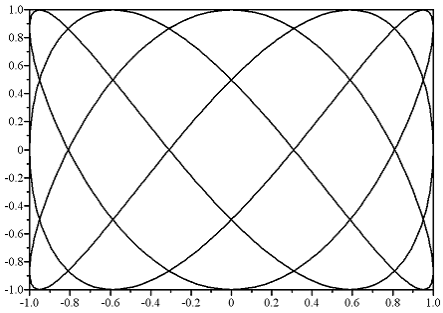
Proponujemy zacząć od zapoznania się ze **Scilabem**, uniwersalnym systemem do obliczeń naukowych i prezentacji, będącym dziełem instytutu INRIA (Rocquencourt pod Paryżem). Jest to jedna z akademickich realizacji zbliżonych do komercjalnego pakietu **Matlab**, bardzo popularnego w środowisku naukowo-technicznym. Więcej można się dowiedzieć ze stronic internetowych http://pauillac.inria.fr/cdrom/www/scilab/eng.htm.

Scilab dysponuje własnym językiem programowania, dość klasycznym (zbliżonym do Pascala czy C), ale posiadającym użyteczną możliwość operowania na raz całymi tablicami w wyrażeniach algebraicznych, a także wygodne instrukcje tworzenia tych tablic. Te tablice (zbiory liczb: wektory, macierze) mogą reprezentować na raz całe trajektorie fizyczne, pola sił albo proste własności, ale dużej liczby cząstek. To pozwala unikać pisania pętli programowych, które wykonują te

same operacje wiele razy na kolejnych zmiennych. Jeśli `x` jest taką tablicą, to „sinus tablicy”, `sin(x)`, staje się tablicą sinusów poszczególnych elementów.

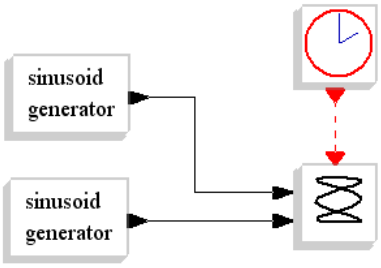
Napiszmy więc w tym stylu program, który rysuje na ekranie wykres Lissajoux, złożenie dwóch prostopadłych okresowych ruchów. Wystarczą trzy wiersze kodu:

```
t=2*%pi*(0:1000)/1000;
x=sin(3*t); y=cos(5*t);
plot2d(x,y);
```



Zauważmy prostotę konstrukcji. `t` jest ciągiem liczb od 0 do 2π co 0.001. Definicje tablic `x` i `y` są tak samo zwarte, jak gdybyśmy chcieli sobie je po prostu zanotować na papierze. Procedura `plot2d` jest jedną z setek instrukcji dotyczących grafiki; w niniejszym kontekście działa jak „oscylograf”, wykreślając `y` względem `x`. Oczywiście mamy również do dyspozycji wiele procedur służących do generowania grafiki trójwymiarowej. Scilab dysponuje także modułami do tworzenia animacji, generatorami liczb losowych, niezbędnymi do symulowania zjawisk w fizyce statystycznej itp. Możemy wykonywać mnożenie tablic (odpowiadających sobie elementów): (.*) i mnożenie macierzowe (*), użyteczne w składaniu obrotów. Większość tych elementów języka i towarzyszących bibliotek można opanować na prostych przykładach wbudowanych w pakiet albo towarzyszących dokumentacji.

Interesującym modułem wchodzącym w skład Scilabu jest **Scicos**, służący głównie do symulacji układów dynamicznych i pozwalający „rysować programy”, a właściwie składać je z cegiełek graficznych, połączonych liniami, którymi wędrują dane, oraz sygnały taktujące, definiujące „zdarzenia”, tj. z grubsza „momenty czasu, w których coś się dzieje”. Spróbujmy „narysować” program generujący takie same krzywe Lissajoux jak poprzednio.



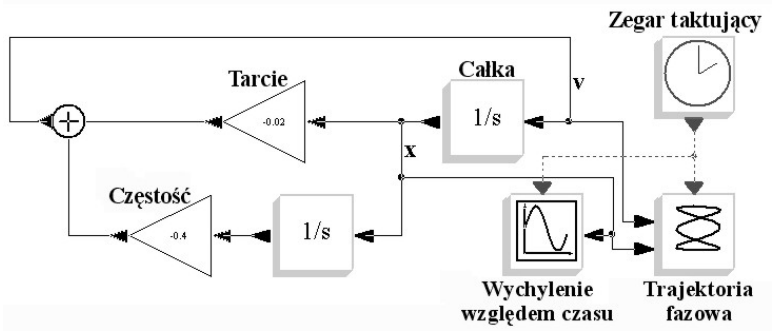
Otwieramy puste okienko i kilka okienek „palet”, zawierających rysunki takich elementów obwodu, jak zegar, generatory sinusoid, oscyloskopy, obwody całkujące itp. Wprowadzamy myszą te elementy do naszego okienka i łączymy obwód jak na rysunku. Teraz należy go sparametryzować.

W dalszym ciągu operując myszą, „otwieramy pudełka” i interakcyjnie definiujemy takie parametry, jak częstość sinusoid, fazy początkowe (aby druga z nich była raczej cosinusoidą, a nie sinusoidą), całkowity czas symulacji, czułość oscyloskopu itp. Następnie każemy program wykonać i dostajemy znany już nam wykres.

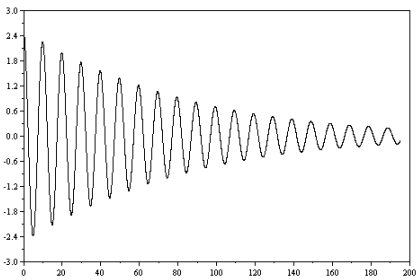
Scicos dostarcza nam dziesiątek różnych elementów: elementy mnożące sygnał przez stałą albo przez inny sygnał, całkowanie, dodawanie sygnałów, obcinanie powyżej progu, przełączniki pozwalające wybierać drogi przesyłania danych (które zastępują instrukcje warunkowe w programach tekstowych), „pudełka” programowalne pozwalające zadać dowolną funkcją napisaną w języku Scilabu, szereg wariantów obwodów do wizualizacji, statycznej i animowanej (gdzie czas biegnie rzeczywiście i plamka na ekranie porusza się) itp. Można także definiować własne elementy złożone z prostszych i zachować je na później w prywatnej palecie podukładów, „makrobloków”.

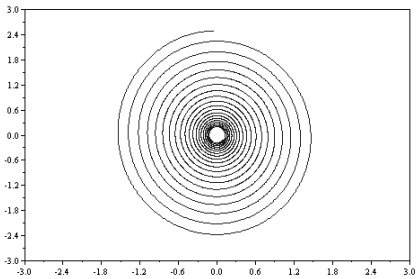
W tym odcinku proponujemy jeszcze jeden prosty (ale nie najprostszy) przykład na zaostrzenie apetytu, później czekają nas inne modele, bardziej wyrafinowane. Pragniemy podkreślić, że naszym celem nie jest po prostu symulacja „prawdziwej fizyki”, ale potraktowanie komputera jako urządzenia ułatwiającego obserwację pewnych zjawisk, nie zawsze prostych w realizacji materialnej. A diagramy są czytelniejsze od programów tekstowych...

Naszym przykładem jest po prostu tłumiony oscylator harmoniczny w jednym wymiarze. Prędkość v jest pochodną czasową położenia, $v = \dot{x}$. Zapiszemy to symbolicznie odwrotnie, wyrażając położenie jako całkę z prędkości: $x = \int v$. Przyspieszenie $\dot{v}$ spełnia równanie $\dot{v} + fv + kx = 0$, gdzie f jest współczynnikiem tarcia, a k określa kwadrat częstości drgań; masę przyjęliśmy równą 1. Prędkość jest więc symbolicznie określona przez $v = -fx - k\int x$. Definicje x i v są dwoma rekurencyjnymi, związanymi ze sobą równaniami. A oto program, który je rozwiązuje iteracyjnie, punkt po punkcie:



Linie ciągłe przenoszą „sygnały”, dane liczbowe, takie jak prędkość czy położenie, próbkowane w czasie. Linie przerywane są sygnałami taktującymi, łączącymi zegar z urządzeniami wykreślającymi wyniki. Trójkąty obrazują mnożenie przez stałe, tu: przez *f* lub *k*. Węzeł oznaczony jako **1/s** jest obwodem całkującym (to symboliczne oznaczenie bierze się z reprezentacji całkowania przez transformatę Laplace’a w teorii obwodów liniowych), i to jest cały program. Całkowanie jest sumowaniem: sygnał wyjściowy jest (w przybliżeniu) sumą wszystkich wartości do tej pory odebranych na wejściu i akumulowanych. Zauważmy, jak zapętlona linia obrazuje równanie rekurencyjne spełniane przez prędkość: jest ona sumą wkładów otrzymanych po pojedynczym i podwójnym całkowaniu tejże prędkości. Doceńmy również, jak prosto jest dołączyć do „układu fizycznego” urządzenie obserwujące, tu: dwa warianty oscyloskopu. Parę sekund wystarczy, aby wprząc inne urządzenie, np. obwód przesyłający dane do tablicy, która może być analizowana po zakończeniu symulacji. Uruchomienie programu generuje dwa (animowane albo statyczne) wykresy, wychylenie *x* w funkcji czasu oraz wykres fazowy: położenie *x* (na osi odciętych) względem prędkości *v* (oś rzędnych), parametryzowane czasem. A oto ich zdjęcia migawkowe:





Oczywiście, powyższy opis programu jest niekompletny, nie napisaliśmy, jak parametryzować pudełka (np. ustalać mnożniki, częstość taktowania zegara itp.). Jest to bardzo łatwe, możliwe w trybie konwersacyjnym, ale stanowi szczegół techniczny. Naszym celem było tylko pokazanie, że modelowanie i wizualizacja niekoniecznie muszą być związane z długimi i nieczytelnymi tekstami programów (oczywiście powyższy diagram posiada odpowiednik tekstowy, 240 wierszy kodu, którego lepiej na oczy nie oglądać..., ale nie musi nas to interesować). Zintegrowane pakiety w rodzaju Scilabu zmniejszają dystans między konceptualnymi modelami zjawisk fizycznych a ich realizacją na komputerze i pozwalają nie tylko na symulacje, ale i na projektowanie i analizę rzeczywistych doświadczeń. Ale pomysłów na ciekawe doświadczenia i ich wizualizację nie dostarczy żaden program, a składanie programu z graficznych klocków nie zmniejsza roli dyscypliny obliczeniowej programującego ani jego kompetencji w dziedzinie fizyki i mate-

matyki. W szczególności dobrze jest wiedzieć, że prosty oscylator tłumiony nie wymaga żadnego specjalnego programu do symulacji, gdyż rozwiązanie analityczne tego problemu jest dobrze znane.

Jednak, oprócz aspektów wizualnych, które są po prostu zabawne, pakiety w rodzaju Scicosu (czy innych, takich jak Labview, Simulink, Khoros albo Modelica) pozwalają popatrzyć nieco inaczej na model świata w komputerze: zamiast bloku programowego, który *kontroluje* zachowanie całego modelowanego „świata", mamy zestaw podukładów, które widzą wzajemnie swoje własności i synchronizują swoje zachowania w sposób autonomiczny, trochę podobnie jak układy fizyczne w świecie realnym. Oczywiście nie wszystko daje się programować w ten sposób, ale pedagogiczne walory takiego wizualnego podejścia do symulacji zostały sprawdzone już wielokrotnie.

Nie chodzi tu zresztą wyłącznie o symulację fizyki, ale o konstrukcję oprogramowania naukowego i dydaktycznego w ogóle. Problemy obliczeniowe stają się coraz bardziej złożone i opanowanie techniki składania dużych aplikacji z małych modułów, które sobie wzajemnie przesyłają dane, jest rzeczą pierwszoplanową. W znacznej liczbie szkół (nie tylko w Polsce), i to nawet takich, gdzie naukę programowania traktuje się serio, popełnia się jednak zbyt często istotny błąd metodologiczny: poświęca się dużo czasu na opanowanie jakiegoś języka programowania i przerabia się małe, niezależne od siebie ćwiczenia, zapominając o tym, że istotny postęp w zastosowaniach komputerów bierze się nie ze sprawności programowania w jakimś języku, ani nawet z kodowania nowych algorytmów, ale z integracji istniejących rozwiązań, z umiejętności synchronizowania osobno programowanych modułów i współpracy wielu pakietów pisanych przez różne osoby w różnych językach.

Jak wspomnieliśmy na początku, zamierzamy poświęcić fizyce komputerowej cały cykl artykułów, nie ograniczając się zresztą do przykładów w Scilabie. Zachęcamy do współpracy, np. do nadsyłania propozycji tematów, które wydają się interesujące dla nauczycieli lub grup uczniowskich.