VII. Symulacje w czasie rzeczywistym

Ta część jest pierwszą z serii rozdziałów zaprojektowanych w celu dokładnego wprowadzenia do tematu symulacji w czasie rzeczywistym. Mówimy o wprowadzeniu, ponieważ temat jest zbyt obszerny i złożony, aby można go było odpowiednio potraktować w kilku rozdziałach; dokładamy jednak wszelkich starań, ponieważ zrobimy więcej niż symulacje w czasie rzeczywistym. W rzeczywistości przeprowadzimy Cię przez dwie proste symulacje, jedną w dwóch wymiarach, a drugą w trzech wymiarach. Mamy nadzieję, że uda nam się w wystarczającym stopniu zrozumieć ten temat, aby można było kontynuować go z większą pewnością. Innymi słowy, chcemy, abyś miał solidne zrozumienie podstaw, zanim wskoczysz do korzystania z czyjegoś silnika fizyki lub wyruszysz, aby napisać własną. W kontekście tego tekstu symulacja w czasie rzeczywistym jest procesem, w którym obliczysz stan obiektu (lub obiektów), który próbujesz reprezentować w locie. Nie polegasz na preskryptowanych sekwencjach ruchu, aby animować swój obiekt, ale zamiast tego polegasz na swoim modelu fizyki, równaniach ruchu i rozwiązaniu równania różniczkowego, aby zająć się ruchem obiektu w miarę postępów symulacji. Ten rodzaj symulacji może być użyty do modelowania sztywnych ciał, takich jak samolot w naszym przykładzie FlightSim, lub elastycznych ciał, takich jak szmatki i postacie ludzkie. Być może jednym z najbardziej fundamentalnych aspektów implementacji symulatora sztywnego ciała w czasie rzeczywistym jest rozwiązywanie równań ruchu za pomocą numerycznych technik integracyjnych. Z tego powodu całą tę część poświęcimy na wyjaśnienie liczbowych technik integracji, które wykorzystacie później w symulatorze 2D i 3D, który będziemy rozwijać. Jeśli przejdziesz przez chwilę do rozdziału 4, w którym przedstawiliśmy ogólną procedurę rozwiązywania problemów związanych z kinetyką, zobaczysz, że do tej pory omawialiśmy wiele kwestii. Poprzednie rozdziały pokazały, jak oszacować właściwości masy i rozwinąć rządzące równania ruchu. W tym rozdziale pokażemy, jak rozwiązać równania ruchu, aby określić przyspieszenie, prędkość i przemieszczenie. Będziemy śledzić tę część z kilkoma pokazującymi, jak wdrożyć symulacje sztywnego ciała 2D i 3D.

Całkowanie równań ruchu

Do tej pory powinieneś dokładnie zrozumieć dynamiczne równania ruchu cząstek i ciał sztywnych. Jeśli nie, możesz wrócić i przejrzeć część 1 do części 4 przed przeczytaniem tego. Kolejnym aspektem radzenia sobie z równaniami ruchu jest rzeczywiste rozwiązywanie ich w twojej symulacji. Równania ruchu, które omawialiśmy, można sklasyfikować jako równania różniczkowe zwyczajne. W części 2 i części 4 można było jednoznacznie rozwiązać te równania różniczkowe, ponieważ mieliśmy do czynienia z prostymi funkcjami przyspieszania, prędkości i przemieszczania. Nie będzie tak w przypadku twoich symulacji. Jak zobaczymy w dalszych rozdziałach, obliczenia siły i momentu dla twojego systemu mogą być dość skomplikowane i mogą nawet polegać na tabelarycznych danych empirycznych, które uniemożliwią Ci pisanie prostych funkcji matematycznych, które można łatwo zintegrować. Oznacza to, że musisz użyć technik całkowania numerycznego, aby w przybliżeniu zintegrować równania ruchu. Mówimy w przybliżeniu, ponieważ rozwiązania oparte na integracji numerycznej nie będą dokładne i będą miały pewną ilość błędów w zależności od wybranej metody. Zaczniemy od dość nieformalnego wyjaśnienia, w jaki sposób zastosujemy integrację numeryczną, ponieważ łatwiej będzie ją zrozumieć. Później przejdziemy do formalnej matematyki. Przyjrzyj się równaniu różniczkowemu ruchu liniowego dla cząstki (lub środka masy ciała sztywnego):

$$F = m \, dv/dt$$

Przypomnijmy, że to równanie jest wyrażeniem siły równym masowemu przyspieszeniu, gdzie F jest siłą, m jest masą, a dv / dt jest pochodną czasu prędkości, która jest przyspieszeniem. W prostych przykładach wcześniejszych części przepisaliśmy to równanie w następującej formie, aby można było je całkować jawnie:

$dv/dt = F/m$

$dv = (F/m)dt$

Jednym ze sposobów interpretacji tego równania jest to, że nieskończenie mała zmiana prędkości, dv, jest równa (F / m) razy nieskończenie mała zmiana czasu. We wcześniejszych przykładach zintegrowaliśmy się jednoznacznie, przyjmując określoną całkę po lewej stronie tego równania w odniesieniu do prędkości i prawej strony względem czasu. W integracji numerycznej musisz podejmować skończone kroki w czasie, tak więc dt przechodzi od bycia nieskończenie małym do pewnego przyrostu czasu dyskretnego, Δt, a kończysz z dyskretną zmianą prędkości, Δv:

$\Delta v = (F/m)\ \Delta t$

Ważne jest, aby zauważyć, że nie daje to wzoru na chwilową prędkość; zamiast tego daje tylko przybliżenie zmiany prędkości. Tak więc, aby zbliżyć rzeczywistą prędkość twojej cząstki (lub sztywnego ciała), musisz wiedzieć jaka była jej prędkość przed zmianą czasu Δt. Na początku twojej symulacji, w czasie 0, musisz znać początkową prędkość swojej cząstki. Jest to warunek początkowy i jest wymagany, aby jednoznacznie zdefiniować prędkość twojej cząstki podczas przechodzenia przez czas, używając tego równania:

$$v_{t+\Delta t} = v_t + (F/m)\ \Delta t$$

gdzie warunek początkowy to

$v_{t=0} = v_0$

Tutaj v_t jest prędkością w pewnym czasie t, $v_{t+\Delta t}$ jest prędkością w pewnym czasie plus krok czasu, Δt jest krokiem czasu, a v0 jest prędkością początkową w czasie 0. Możesz włączyć liniowe równanie ruchu jeszcze raz w w celu przybliżenia przemieszczenia (lub pozycji) twojej cząstki. Po ustaleniu nowej wartości prędkości, w czasie t + Δt, można przybliżone przesunięcie za pomocą:

$$s_{t+\Delta t} = s_t + \Delta t\ (v_{t+\Delta t})$$

gdzie początkowym warunkiem przemieszczenia jest:

$s_{t=0} = s_0$

Omawiana tutaj technika integracji jest znana jako metoda Eulera i jest najbardziej podstawową metodą integracji. Chociaż metoda Eulera jest łatwa do uchwycenia i dość łatwa do implementacji, niekoniecznie jest to najdokładniejsza metoda. Możesz rozumować, że im mniejszy robisz swój krok w czasie - to znaczy, gdy Δt zbliża się do dt - im bliżej dojdziesz do dokładnego rozwiązania. Występują jednak problemy obliczeniowe związane z używaniem bardzo małych kroków czasowych. W szczególności, będziesz potrzebował ogromnej liczby obliczeń przy bardzo małych Δt, a ponieważ twoje obliczenia nie będą dokładne (w zależności od dokładności liczbowej będziesz zaokrąglał i obcinania liczb), skończy się to nagromadzeniem błąd zaokrąglania. Oznacza to, że istnieje praktyczny limit określający, jak mały może być czas. Na szczęście istnieje kilka technik integracji numerycznej, które zostały zaprojektowane w celu zwiększenia dokładności w rozsądnych rozmiarach kroku. Nawet jeśli użyliśmy liniowego równania ruchu dla cząstki, ta technika integracji (i te, które pokażemy później) odnosi się równie dobrze do kątowych równań ruchu.

Metoda Eulera

Poprzednie wyjaśnienie metody Eulera było, jak powiedzieliśmy, nieformalne. Aby traktować metodę Eulera w bardziej matematycznie rygorystyczny sposób, przyjrzymy się rozszerzeniu serii Taylora o funkcji ogólnej, y (x). Twierdzenie Taylora umożliwia przybliżenie wartości funkcji w pewnym momencie, wiedząc coś o tej funkcji i jej pochodnych w jakimś innym punkcie. To przybliżenie jest wyrażone jako nieskończona seria wielomianowa postaci:

$$y(x + \Delta x) = y(x) + (\Delta x)\, y'(x) + ((\Delta x)^2 / 2!)\, y''(x) + ((\Delta x)^3 / 3!)\, y'''(x) + \cdots$$

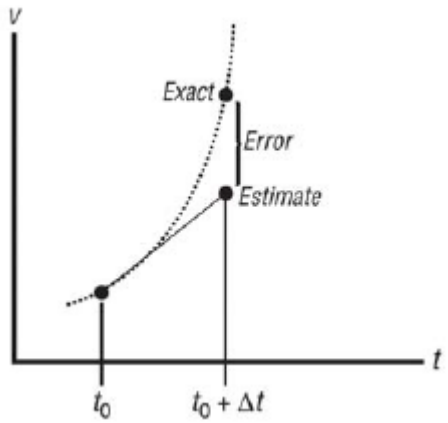
gdzie y jest jakąś funkcją x, (x + Δx) jest nową wartością x, przy której chcesz przybliżona y, y' jest pierwszą pochodną y, y'' jest drugą pochodną y, i tak dalej. W przypadku równania ruchu omawianego w poprzednim rozdziale, funkcja próbujesz przybliżyć to prędkość jako funkcję czasu. W ten sposób możesz napisz v (t) zamiast y (x), co daje rozszerzenie Taylora:

$$v(t + \Delta t) = v(t) + (\Delta t)\, v'(t) + ((\Delta t)^2 / 2!)\, v''(t) + ((\Delta t)^3 / 3!)\, v'''(t) + \cdots$$

Zauważ tutaj, że v'(t) jest równe dv/dt, co równa się F/m w przykładowym równaniu ruchu omówionym w poprzedniej sekcji. Zauważ także, że znasz wartość v w czasie t. To, co chcesz znaleźć, to wartość v w czasie t + Δt wiedząca v w czasie t i jego pochodna w czasie t. W pierwszym przybliżeniu, a ponieważ nie wiesz nic o drugiej, trzeciej lub wyższej pochodnej v, możesz obciąć serię wielomianową po terminie (Δt) v '(t), co daje:

$$v(t + \Delta t) = v(t) + (\Delta t)\, v'(t)$$

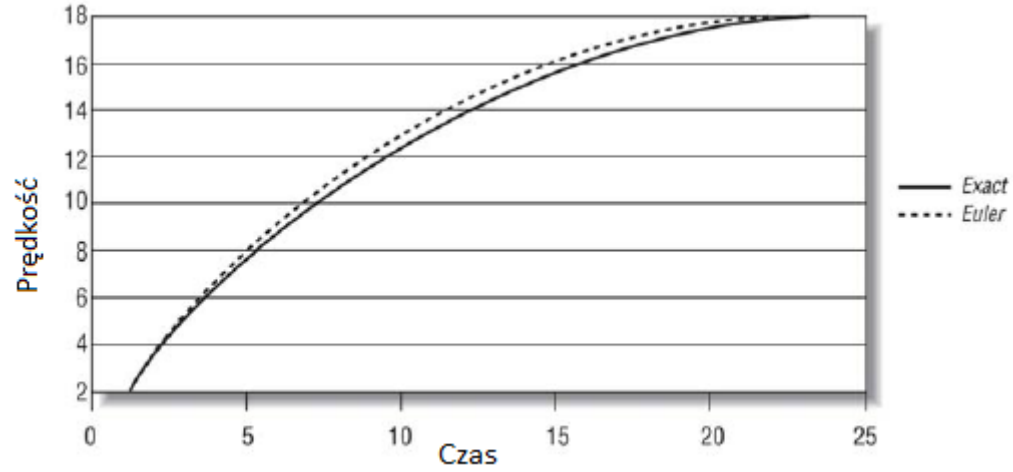
Oto wzór całkowania Eulera, który zobaczyłeś w ostatniej sekcji. Ponieważ formuła Eulera wychodzi tylko do terminu zawierającego pierwszą pochodną, resztą serii, która została przerwana, jest błąd obcięcia. Te terminy, które zostały pominięte, nazywane są terminami wyższego rzędu, a pozbycie się ich skutkuje aproksymacją pierwszego rzędu. Przesłanką tego zbliżenia jest to, że im dalej idziesz w szeregu, tym mniejsze są warunki i mniejszy wpływ na przybliżenie. Ponieważ zakłada się, że Δt jest małą liczbą, Δt2 jest jeszcze mniejsza, Δt3 jeszcze mniejsza, i tak dalej, i ponieważ te Δt warunki pojawiają się w licznikach, każde kolejne wyrażenie wyższego rzędu staje się coraz mniejsze. W tym przypadku pierwsze skrócone określenie $((\Delta t)^2/2!)\, v''(t)$ dominuje błąd obcięcia, a metoda ta ma błąd rzędu $(\Delta t)^2$. Geometrycznie, metoda Eulera przybliża nową wartość, w bieżącym kroku, dla rozważanej funkcji przez ekstrapolację w kierunku pochodnej funkcji z poprzedniego kroku. Zostało to zilustrowane na rysunku 7-1



Rysunek 7-1 ilustruje błąd skracania i pokazuje, że wynikiem będzie wielokątne przybliżenie rozważanej funkcji gładkiej. Oczywiście, jeśli zmniejszysz rozmiar kroku, zwiększasz liczbę segmentów wielokątnych i lepiej przybliżasz funkcję. Jak już powiedzieliśmy wcześniej, nie zawsze jest to skuteczne, ponieważ liczba obliczeń w twojej symulacji wzrośnie, a błąd zaokrąglania narasta szybciej. Aby zilustrować metodę Eulera w praktyce, przyjrzyjmy się liniowemu równaniu ruchu dla przykładu statku z Części 4:

T - (C v) = ma

gdzie T jest kursem śmigła, C jest współczynnikiem oporu, v jest prędkością statku, m jego masą i przyspieszeniem. Rysunek 7-2 przedstawia rozwiązanie całkujące Eulera, z krokiem 0,5 s, nałożone na dokładne rozwiązanie otrzymane w części 4 dla prędkości statku w czasie



Powiększenie tego wykresu pozwala zobaczyć błąd w przybliżeniu Eulera. To pokazano na rysunku 7-3.

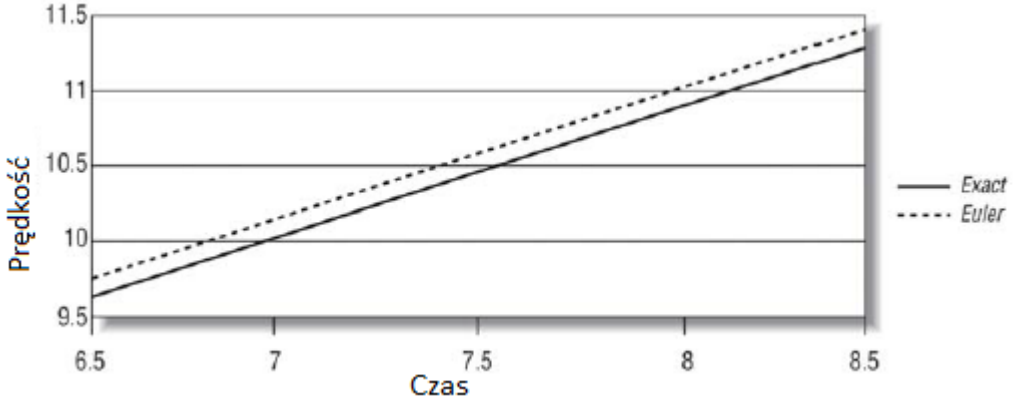
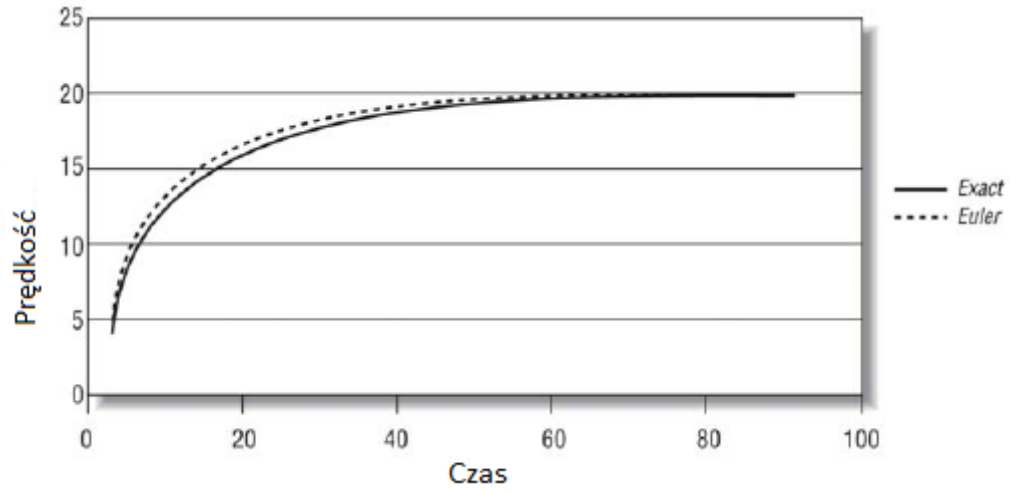


Tabela 7-1 pokazuje wartości liczbowe prędkości względem czasu dla zakresu pokazanego na rysunku 7-3. W tabeli 7-1 pokazano również różnicę procentową, błąd między dokładnym rozwiązaniem a roztworem Eulera na każdym etapie czasowym.

Czas (s)	Prędkość, dokładność (m / s)	Prędkość, Euler (m / s)	Błąd
6,5	9,559084	9,733158	1,82%
7	10,06829	10,2465	1,77%
7,5	10,55267	10,73418	1,72%

8	11.01342	11.19747	1.67%
8,5	11,4517	11 63759	1,62%

Jak widać, błąd obcięcia w tym przykładzie nie jest zły. Mogłoby być lepiej, jednak, za chwilę pokażemy Ci kilka dokładniejszych metod. Przed tym, należy jednak zauważyć, że w tym przykładzie metoda Eulera jest również stabilna - to znaczy, że jest zbieżna z dokładnym rozwiązaniem, jak pokazano na rysunku 7-4, gdzie przesunęliśmy przedział czasu dalej

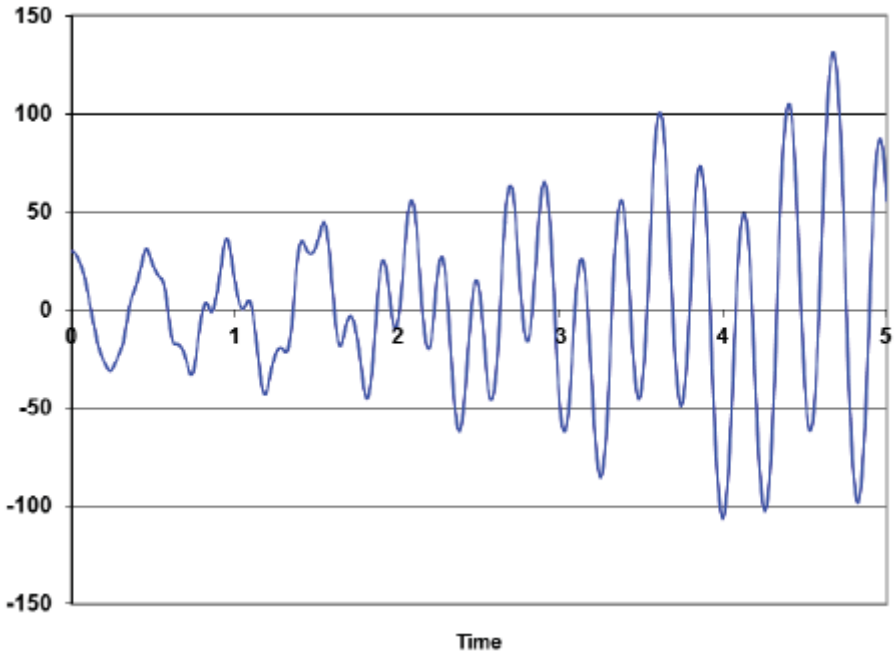


Oto fragment kodu implementujący metodę Eulera dla tego przykładu:

```
// Zmienne globalne
float T; // ciąg
float C; // współczynnik przeciągania
float  V; // prędkość
float M; // masa
float; // przemieszczenie
.
.
.
// Ta funkcja rozwija symulację o dt sekund przy użyciu
// Podstawowa metoda Eulera
void StepSimulation (float dt)
{
float F; // całkowita siła
```

```
float A; // przyspieszenie

float Vnew; // nowa prędkość w czasie t + dt

float Snew; // nowa pozycja w czasie t + dt

// Oblicz całkowitą siłę

F = (T - (C * V));

// Oblicz przyspieszenie

A = F / M;

// Oblicz nową prędkość w czasie t + dt

// gdzie V jest prędkością w czasie t

Vnew = V + A * dt;

// Oblicz nowe przemieszczenie w czasie t + dt

// gdzie S jest przesunięciem w czasie t

Snew = S + Vnew * dt;

// Zaktualizuj starą prędkość i przemieszczenie z nowymi

V = Vnew;

S = Snew;

}
```

Chociaż metoda Eulera jest stabilna w tym przykładzie, nie zawsze tak jest, w zależności od problemu, który próbujesz rozwiązać. Jest to coś, o czym należy pamiętać przy wdrażaniu dowolnego schematu integracji numerycznej. Rozumiemy przez to, że w tym przypadku rozwiązanie Eulera jest zgodne z dokładnym rozwiązaniem. Niestabilne rozwiązanie może ujawnić błędy na dwa sposoby. Po pierwsze, kolejne wartości mogły oscylować powyżej i poniżej dokładnego rozwiązania, nigdy się z nim nie zbliżając. Po drugie, kolejne wartości mogą odbiegać od dokładnego rozwiązania, tworząc coraz większy błąd w czasie. Spójrz na rysunek 7-5. Ta ilustracja pokazuje, jak metoda Eulera może stać się bardzo niestabilna. To, co widzisz na wykresie, reprezentuje ruch wibracyjny systemu sprężynowo-masowego. Jest to prosty układ dynamiczny, który powinien wykazywać regularny ruch sinusoidalny.



Z rysunku widać, że używanie metody Eulera daje strasznie niestabilne wyniki. Może zobaczyć, jak amplituda ruchu wciąż rośnie. Jeśli to była gra, powiedz, gdzie masz kilka obiektów połączonych ze sobą sprężynami, wtedy ten rodzaj niestabilności przejawia się w szalenie nierealistycznym ruchu tych obiektów. Co gorsza, symulacja może wybuchnąć, mówiąc liczbowo. Często wybór rozmiaru kroku wpływa na stabilność, gdy mniejsze rozmiary kroku mają tendencję do eliminowania lub minimalizowania niestabilności, a większe kroki pogłębiają problem. Jeśli pracujesz z wyjątkowo nieporęczną funkcją, może się okazać, że musisz znacznie zmniejszyć swój rozmiar kroku, aby osiągnąć stabilność. To jednak zwiększa liczbę obliczenia, które musisz wykonać. Jednym ze sposobów obejścia tej trudności jest zastosowanie tego, co jest nazwana adaptacyjną metodą wielkości kroku, w której zmienia się rozmiar kroku w locie w zależności od wielkości przewidywanej ilości błędu obcięcia od jednego kroku do drugiego. Jeśli błąd obcięcia jest zbyt duży, wykonaj kopię zapasową kroku, zmniejsz swój rozmiar kroku i spróbuj ponownie. Jednym ze sposobów wdrożenia tego w przypadku metody Eulera jest najpierw wykonanie kroku o wielkości At, aby uzyskać oszacowanie w czasie t + Δt, a następnie wykonanie dwóch kroków (począwszy od czasu t ponownie) o wielkości Δt/2 w celu uzyskania innego oszacowania w czasie t + Δt. Ponieważ mówiliśmy o prędkości w przykładach do tej pory, nazwijmy pierwszą ocenę v_1, a drugą ocenę v_2. Miarą błędu obcięcia jest zatem:

$$e_t = |v_1 - v_2|$$

Jeśli chcesz zachować błąd obcięcia w określonym limicie, e_{to}, możesz go użyć następującą formułę, aby dowiedzieć się, jaki powinien być twój rozmiar kroku, aby zachować pożądaną dokładność:

$$\Delta t_{new} = \Delta t_{old} \, (e_{to} / e_t)^{(1/2)}$$

Tutaj Δ_{told} jest starszym krokiem, a Δ_{tnew} jest nowym, który powinieneś użyć, aby utrzymać pożądaną dokładność. Musisz sprawdzić to za każdym razem, a jeśli okaże się, że błąd gwarantuje mniejszy krok, musisz wykonać kopię zapasową kroku i powtórzyć go z nowym krokiem. Oto poprawiona funkcja StepSimulation, która implementuje tę technikę adaptacyjnego rozmiaru kroku, sprawdzając błąd obcięcia na integracji prędkości:

```
// Nowa zmienna globalna
float eto; // Tolerancja błędu skracania
// Ta funkcja rozwija symulację o dt sekund przy użyciu
// Podstawowa metoda Eulera z adaptacyjnym rozmiarem kroku
void StepSimulation (float dt)
{
float F; // całkowita siła
float A; // przyspieszenie
float Vnew; // nowa prędkość w czasie t + dt
float Snew; // nowa pozycja w czasie t + dt
float V1, V2; // tymczasowe zmienne prędkości
float dtnew; // nowy krok czasowy
float et; // błąd skracania
// Wykonaj jeden krok o wielkości dt, aby oszacować nową prędkość
F = (T - (C * V));
A = F / M;
V1 = V + A * dt;
// Wykonaj dwa kroki o wielkości dt / 2, aby oszacować nową prędkość
F = (T - (C * V));
A = F / M;
V2 = V + A * (dt / 2);
F = (T - (C * V2));
A = F / M;
V2 = V2 + A * (dt / 2);
// Oszacuj błąd obcięcia
et = absf (V1 - V2);
// Oszacuj nowy rozmiar kroku
dtnew = dt * SQRT (eto / et);
if (dtnew <dt)
{// podejmij krok przy nowym mniejszym rozmiarze kroku
F = (T - (C * V));
```

```
A = F / M;

Vnew = V + A * dtnew;

Snew = S + Vnew * dtnew;

} else

{// Oryginalny rozmiar kroku jest w porządku

Vnew = V1;

Snew = S + Vnew * dt;

}

// Zaktualizuj starą prędkość i przemieszczenie z nowymi

V = Vnew;

S = Snew;

}
```

Lepsze metody

W tym momencie możesz się zastanawiać, dlaczego nie możesz po prostu użyć więcej terminów w serii Taylora, aby zredukować błąd skracania w metodzie Eulera. W rzeczywistości jest to podstawą kilku metod integracji, które oferują większą dokładność niż podstawowa metoda Eulera dla danego rozmiaru kroku. Częścią trudności związanej z uzyskaniem większej liczby terminów w serii Taylora jest możliwość określenia drugiej, trzeciej, czwartej i wyższej pochodnej funkcji, którą próbujesz zintegrować. Sposób obejścia tego problemu polega na wykonaniu dodatkowych rozszerzeń serii Taylora w celu przybliżenia pochodnych rozważanej funkcji, a następnie zastąpienia tych wartości powrotem do pierwotnego rozszerzenia. Przyjmując to podejście, aby uwzględnić jeszcze jeden termin Taylora poza podstawową metodą Eulera, uzyskuje się tak zwaną ulepszoną metodę Eulera, która ma zmniejszony błąd skracania rzędu $(\Delta t)^3$ zamiast $(\Delta t)^2$. Formuły dla tej metody to:

$$k_1 = (\Delta t)\, y'(t, y)$$
$$k_2 = (\Delta t)\, y'(t + \Delta t, y + k_1)$$
$$y(t + \Delta t) = y(t) + ½\,(k_1 + k_2)$$

Tu y jest funkcją t, a y' jest pochodną w funkcji t i ewentualnie y w zależności od równań, które próbujesz rozwiązać, i Δt wielkości kroku. Aby uczynić to jaśniejszym dla Ciebie, pokażemy te formuły w odniesieniu do przykładowego równania ruchu statku z Części 4, tego samego przykładu, który omówiliśmy w poprzedniej sekcji. W tym przypadku prędkość jest przybliżana przez następujące formuły:

$$k_1 = \Delta t\,[1/m\,(T - C\,v_t)]$$
$$k_2 = \Delta t\,[1/m\,(T - C\,(v_t + k_1))]$$
$$v_{t+\Delta t} = v_t + ½(k_1 + k_2)$$

gdzie vt to prędkość w czasie t, a $v_{t+\Delta t}$ to nowa prędkość w czasie t + Δt. Oto poprawiona funkcja StepSimulation pokazująca, jak zaimplementować tę metodę w kodzie:

```
// Ta funkcja rozwija symulację o dt sekund przy użyciu
// "ulepszona" metoda Eulera
void StepSimulation (float dt)
{
float F; // całkowita siła
float A; // przyspieszenie
float Vnew; // nowa prędkość w czasie t + dt
float Snew; // nowa pozycja w czasie t + dt
float k1, k2;
F = (T - (C * V));
A = F / M;
k1 = dt * A;
F = (T - (C * (V + k1)));
A = F / M;
k2 = dt * A;
// Oblicz nową prędkość w czasie t + dt
// gdzie V jest prędkością w czasie t
Vnew = V + (k1 + k2) / 2;
// Oblicz nowe przemieszczenie w czasie t + dt
// gdzie S jest przesunięciem w czasie t
Snew = S + Vnew * dt;
// Zaktualizuj starą prędkość i przemieszczenie z nowymi
V = Vnew;
S = Snew;
}
```

Możemy przeprowadzić tę procedurę, aby uzyskać jeszcze więcej warunków Taylora. Popularna metoda Runge-Kutta przyjmuje takie podejście, aby zredukować błąd skracania do rzędu $(\Delta t)^5$. Formuły integracji dla tej metody są następujące:

$$
\begin{aligned}
k_1 &= (\Delta t)\, y'(t, y) \\
k_2 &= (\Delta t)\, y'(t + \Delta t/2, y + k_1/2) \\
k_3 &= (\Delta t)\, y'(t + \Delta t/2, y + k_2/2) \\
k_4 &= (\Delta t)\, y'(t + \Delta t, y + k_3) \\
y(t + \Delta t) &= y(t) + 1/6\,(k_1 + 2\,(k_2) + 2\,(k_3) + k_4)
\end{aligned}
$$

Stosowanie tych formuł na przykładach wydajności naszego statku:

$$k_1 = \Delta t\,[1/m\,(T - C\,v_t)]$$
$$k_2 = \Delta t\,[1/m\,(T - C\,(v_t + k_1/2))]$$
$$k_3 = \Delta t\,[1/m\,(T - C\,(v_t + k_2/2))]$$
$$k_4 = \Delta t\,[1/m\,(T - C\,(v_t + k_3))]$$
$$v_{t+\Delta t} = v_t + 1/6\,(k_1 + 2\,(k_2) + 2\,(k_3) + k_4)$$

W naszym przykładzie metoda Runge-Kutta została zaimplementowana w następujący sposób:

```
// Ta funkcja rozwija symulację o dt sekund przy użyciu
// metoda Runge-Kutta
void StepSimulation (float dt)
{
float  F; // całkowita siła
float A; // przyspieszenie
float Vnew; // nowa prędkość w czasie t + dt
float Snew; // nowa pozycja w czasie t + dt
float k1, k2, k3, k4;
F = (T - (C * V));
A = F / M;
k1 = dt * A;
F = (T - (C * (V + k1 / 2)));
A = F / M;
k2 = dt * A;
F = (T - (C * (V + k2 / 2)));
A = F / M;
k3 = dt * A;
F = (T - (C * (V + k3)));
A = F / M;
k4 = dt * A;
// Oblicz nową prędkość w czasie t + dt
// gdzie V jest prędkością w czasie t
Vnew = V + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
```

// Oblicz nowe przemieszczenie w czasie t + dt

// gdzie S jest przesunięciem w czasie t

Snew = S + Vnew * dt;

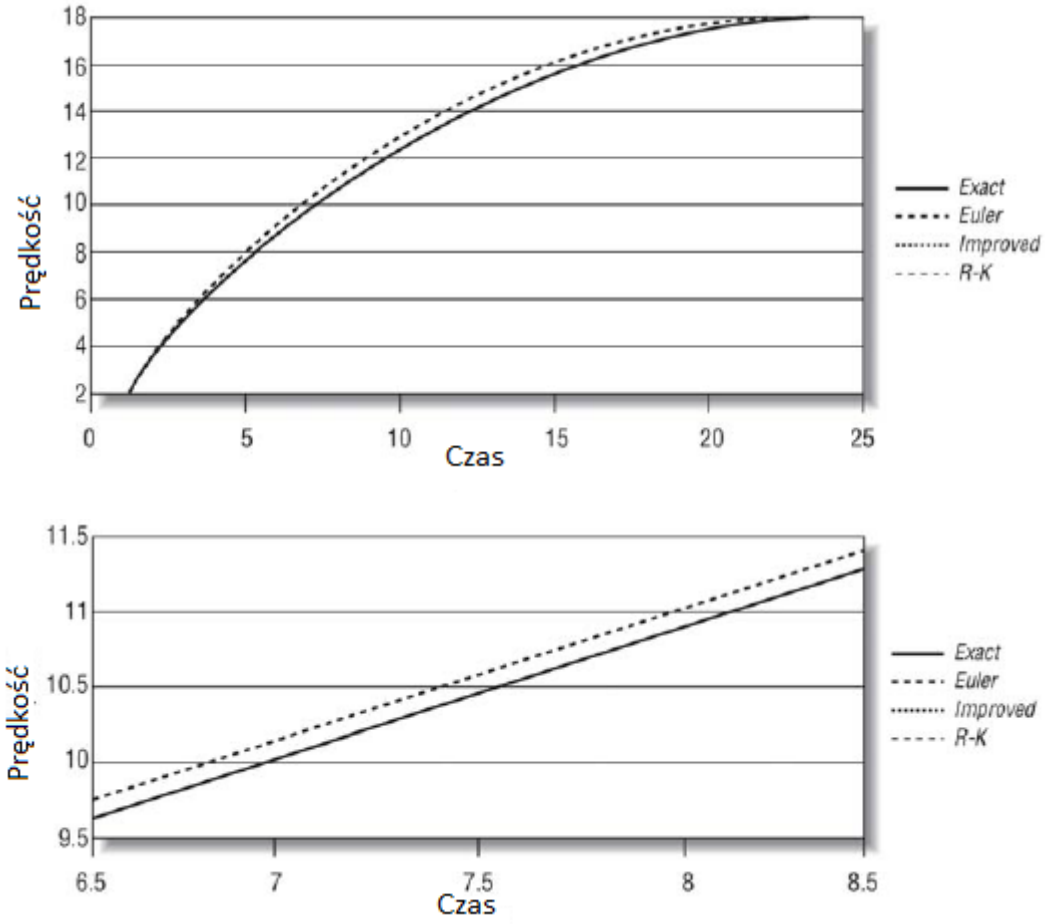
// Zaktualizuj starą prędkość i przemieszczenie z nowymi

V = Vnew;

S = Snew;

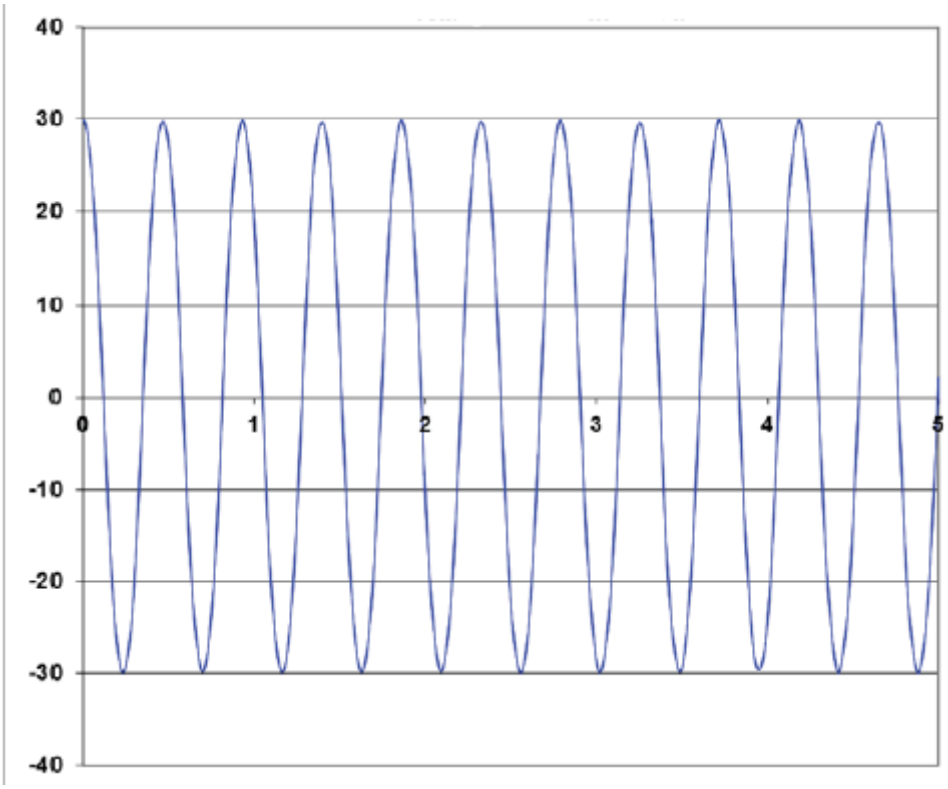
}

Aby pokazać, jak poprawiona jest dokładność w stosunku do podstawowej metody Eulera, dodaliśmy wyniki integracji dla przykładu statku, używając tych dwóch metod, niż te pokazane na Rysunku 7-2 i Rysunku 7-3. Rysunek 7-6 i Rysunek 7-7 przedstawiają wyniki, gdzie. 7-7 przedstawia powiększony widok 7-6.



Jak widać na tych rysunkach, nie można rozróżnić krzywych dla ulepszonych metod Eulera i Runge-Kutty od dokładnego rozwiązania, ponieważ są one prawie dokładnie na wierzchu. Wyniki te wyraźnie pokazują poprawę dokładności w stosunku do podstawowej metody Eulera, której krzywa różni się od pozostałych trzech. W zakresie od 6,5 do 8,5 s, średni błąd obcięcia wynosi 1,72%, 0,03% i 3,6 × 10-6% dla metody Eulera, ulepszonej metody Eulera, metody Runge-Kutta. Na podstawie tych wyników jest oczywiste, że dla tego problemu metoda Runge-Kutta daje znacznie lepsze wyniki dla danego rozmiaru kroku niż pozostałe dwie metody. Oczywiście płacisz za tę dokładność, ponieważ masz kilka

dodatkowych obliczeń na krok w metodzie Runge-Kutta. Obie te metody są na ogół bardziej stabilne niż metoda Eulera, co jest ogromną korzyścią w aplikacjach czasu rzeczywistego. Przypomnijmy wcześniej naszą dyskusję na temat stabilności metody Eulera. Rysunek 7-5 pokazuje wyniki zastosowania metody Eulera do oscylującego układu dynamicznego. Tam wyniki ruchu, które powinny być sinusoidalne, były szalenie nieregularne (to znaczy niestabilne). Zastosowanie ulepszonej metody Eulera lub metody Runge-Kutta do tego samego problemu daje stabilne wyniki, jak pokazano na rysunku 7-8.



Tutaj ruch oscylacyjny jest wyraźnie sinusoidalny, tak jak być powinien. Wyniki dla tego szczególny problem jest prawie identyczny, niezależnie od tego, czy używasz ulepszonej metody Eulera, czy metody Runge-Kutta. Ponieważ dla tego problemu wyniki obu metod są praktycznie takie same, można zaoszczędzić czas obliczeniowy i pamięć przy użyciu ulepszonej metody Eulera w porównaniu do metody Runge-Kutta. Może to być istotną zaletą w grach w czasie rzeczywistym. Pamiętaj, że metoda Runge-Kutta wymaga czterech obliczeń pochodnych na etap czasowy. Te metody nie są jedynymi, które są do Twojej dyspozycji, ale są najczęstsze. Metoda Runge-Kutta jest szczególnie popularna jako ogólny schemat integracji numerycznej. Inne metody próbują jeszcze bardziej zwiększyć wydajność obliczeniową, to znaczy, że są zaprojektowane tak, aby zminimalizować błąd skracania, a jednocześnie pozwalają na uzyskanie stosunkowo dużych rozmiarów kroków, aby zmniejszyć liczbę kroków, które musisz wykonać w swojej integracji. Jeszcze inne metody są specjalnie dostosowane do określonych typów problemów.