物理シミュレーションの手法と結果の検証

牧野淳一郎
東京大学

Version 0.0 — Document Created: June 1st, 2002
2002 年 10 月 4 日

はじめに

学生 A：赤木さん。どうもご無沙汰してます。相談ってなんです？

赤木：あ、こんにちは。今回はね、「シミュレーションの手法と結果の検証」という題で作者に代わって原稿を書いて下さいという依頼がきてるのよ。

学生 A： で、また僕がプログラムを書かされたあげく赤木さんにいじめられるんですか？

赤木： まあ、要するにそういう話なんだけど、君も前の本 [1] の時より少しは賢くなってるはずよね。もう D2(博士 2 年) なんだし。今回はページもないし、そんなに高度なことをするつもりじゃないのよね。

学生 A：赤木さんの高度じゃないってのは信用できないから、、、どんなことの話をするつもりなんですか？

赤木： 基本的には、前の君にやってもらった本とあんまり変わらないの。つまり、厳密解がある調和振動から始めて、非線型振動の話をして、後はハミルトン系のカオスか、あるいは硬い系[1]の話をしようと思って。

雑誌の性格からすると、ある程度応用的なこともしたいのね。応用的という時に一つの考え方は、例えば Octave[2] みたいな、あるいは極端な話 Mathematica とかを使うってことだけど、まあこれはそもそもよく知らないし、シミュレーションとかいうとあんまり使える気がしないわよね。で、そうでなくて問題というか応用分野を限ったシミュレーション用のパッケージみたいなのはいくらでもあるわけだけど、そういうのの使い方なんてのはパッケージによるわけでいちいちやってもしょうがないでしょ。だから、そういうパッケージを使う人にも、それから自分でプログラムを書く人にも役に立つような中身にしたいわけ。

[1] 定義については後述

[2] www.octave.org

学生 A：赤木さん硬い系なんかさわったことありましたっけ？
赤木： あんまりないんだけど、これからちょっとやってみようと思ってるの。それで自分が勉強するついでね。
学生 A：(また泥縄な、、、) えーと、まあ、いずれにしても最初のほうは僕ではいけないですね。 B 君とかにやってもらいますか？M1 (修士 1 年) でこれからその辺どうせ勉強しないといけないし。
赤木：そうね、それがいいかしら。じゃあ、 B 君に聞いておくわね。じゃあ、今日はそういうことで。
学生 A：(とりあえず逃げられたと安心して) わかりました。僕からもいっておきます。

1 調和振動

学生 B：赤木さんこんにちは。お世話になります。
赤木： あら、こちらこそ、引き受けてくれてありがとう。
学生 B：あ、いえ。これからシミュレーションのプログラムをなんとかしないといけないみたいなので、ちょうどいい機会だと思います。
赤木： そう。どこから始めるのがいいかしら？学部の講義で数値計算とかやってるわよね？ルンゲ・クッタとか線型多段階法ってわかる？
学生 B：ええと、すみません、どちらも聞いたことはあるんですが、、、
赤木： うーん、しょうがないわね。オイラー法は？
学生 B：あ、それはおぼえてます。
赤木： ちょっとどういうものか説明してみて。
学生 B：ええと、確か、

$$\frac{dx}{dt} = f(x,t) \tag{1}$$

みたいな常微分方程式があったとして、

$$x(t+\Delta t) = x(t) + \Delta t f(x(t),t) \tag{2}$$

というふうに、 時刻 t での解から時刻 $t+\Delta t$ の解を求める方法です。
赤木: なんだか記号の使い方が豪快だけど、基本的にはそういうことね。図 1 みたいに、時刻 t_i での近似解を x_i とすれば、次の時刻 $t_{i+1} = t_i + \Delta t$ での近似解が

$$x_{i+1} = x_i + \Delta t f(x_i, t_i) \tag{3}$$

で求められるということね。
図は x が 1 次元の場合だけど、多変数になっても原理は同じね。とにかく今のところでの真の解の接線を引くわけ。
でも、あれよね、図に細い線で書いてあるのが「本当の」解のつもりなわけね。そうすると、もちろんオイラー法で求まった解は本当の解とは違うわけじゃない。どれくらい違うのかしら？数学としてちゃんとやるには収束性とかが問題になるけど、これ割に面倒だからとりあえず簡単な例でプログラム書いて調べてみて。ええと、こうしましょう。

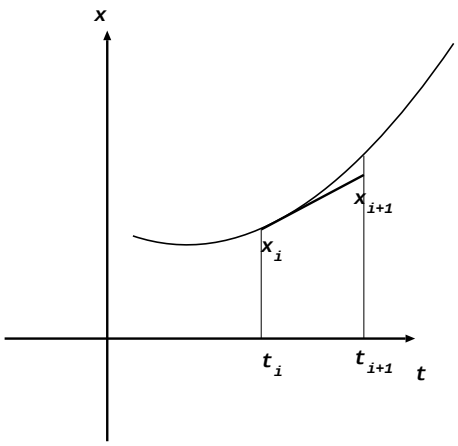

図 1: オイラー法の概念

問題:以下の連立微分方程式

$$\frac{dx}{dt} = v \tag{4}$$

$$\frac{dv}{dt} = -kx \tag{5}$$

を、初期条件 $x_0 = 1, v_0 = 0$ から 1 周期オイラー法で数値解を求めて、1 周期後の誤差を時間刻みの関数として求めてみよ。

学生 B：時間刻みはどれくらいの範囲でやるんですか

赤木： そういうのは自分で考えてみてね。

学生 B：(なんか適当な人だ、、、) はあ、、、

赤木： じゃあ、よろしく。結果はグラフに整理してね。

学生 B：わかりました、、、じゃあ、出来たらまたきますね。

しばらくたっても B が戻ってくる気配がないので、探しにいく赤木。

赤木： ねえ、まだ？

学生 B：あ、、、その、プログラム書いて答もでることはでるんですが、どうもおかしいんで、直そうとしてるんですが、、、

赤木： どうおかしいの？

学生 B：ええと、精度が良過ぎるような気がするんです。ちょっと実行結果をおみせします。

```
<yebisu:/home/makino/papers/Interface>euler
Enter k, nsteps :1 10
k, period, nsteps, dt = 1 6.28319 10 0.628319
0.628318530717958623 -0.041597661808345579 -0.113783383682847572
<yebisu:/home/makino/papers/Interface>!!
```

```
euler
Enter k, nsteps :1 100
k, period, nsteps, dt = 1 6.28319 100 0.0628319
0.0628318530717958679 -3.30347425558861194e-05 -0.00103451230852307213
<yebisu:/home/makino/papers/Interface>!!
euler
Enter k, nsteps :1 1000
k, period, nsteps, dt = 1 6.28319 1000 0.00628319
0.00628318530717958661 -3.25234124386782758e-08 -1.03355224671600188e-05
<yebisu:/home/makino/papers/Interface>
```

学生B：最初にいれてるのは k で、これはいつでも 1 です。で、次にいれてるのが 1 周期を何ステップでやるかで、これを $10, 100, 1000$ と 10 倍ずつ変えてみたんですが、、、 x の誤差は 3 桁づつ減るし、v の誤差も 2 桁づつ減ってるんです。これって変じゃないですか？でも、プログラムいくらみてもおかしいところはないみたいだし、、、

赤木： まあ、とりあえずプログラム見せてくれる？

学生B：ええと、これです。

```
//
// Euler.C
//
// J. Makino 2002/06/02
#include  <stdlib.h>
#include  <math.h>
#include  <iostream>

using namespace std;

int main()
{
    double x,v,t;
    double k;
    int nsteps;
    cerr << "Enter k, nsteps :";
    cin >> k >> nsteps;
    double period = sqrt(1.0/k)*M_PI*2;
    double dt = period/nsteps;
    cerr << "k, period, nsteps, dt = " << k << " "
         << period << " "
         << nsteps << " "
         << dt << endl;

    x=1;
    v=0;
    t=0;
    for(int i = 0;i<nsteps;i++){
        v += -dt*k*x;
```

```
        x += dt*v;
        t += dt;
        cout.precision(18);
        cout << t << " " << x << " " << v << endl;
    }
    cout << dt << " " << x-1 << " " << v  << endl;
}
```

赤木： ちょっと見せて。(数十秒後) ああ、なるほどね。それはそうなるわね。
学生 B：えええっ？なんかおかしいことしてます？
赤木： (こいつもなかなか天然ではないか。。。) そうねえ、あら、ステップ毎に結果を書き出すこともできるのね。まず、最初のステップがちゃんとできてるかどうかってみた？
学生 B：え、1 周期でちゃんと戻ってきてるし、グラフにも書いてみてコサインの形になっているのは確認しましたけど。
赤木： そうじゃなくて、数字自体がちゃんとでてるかどうかよ。オイラー法の式なんか簡単なんだから、電卓でも bc でも使って計算できるでしょ？
学生 B：えーと、じゃあ、 $\Delta t = \pi/5$ の時でやってみます。最初は $x_0 = 1, v_0 = 0$ ですから、ええと、$x_1 = 1, v_1 = -\pi/5 = .62831853$ くらいですね。
赤木：「くらい」ってのはなによ？まあいいわ、で、プログラムの答は？
学生 B：ちょっと待って下さい、、、ええと、

```
euler
Enter k, nsteps :1 10
k, period, nsteps, dt = 1 6.28319 10 0.628319
0.628318530717958623 0.605215823956425703 -0.628318530717958623
......
```

あれ、x が 1 じゃない、、、え、どうしてですか？
赤木： どうしてって、あなたのプログラムがそうなってるからそういう答になるんじゃないの？プログラムではどう計算してるの？
学生 B：ええと、まず v を計算して、それから x を、、、あ、そういうことか、 新しい x を計算するのに、v の新しい値を使っているのが違うわけですね。 とりあえず直して、、、

```
double dv = -dt*k*x;
double dx = dt*v;
v += dv;
x += dx;
```

こんなのでどうでしょう。
赤木： まあ、よさそうよね。動かしてみたら？
学生 B：

```
<yebisu:/home/makino/papers/Interface>euler1
Enter k, nsteps :1 10
k, period, nsteps, dt = 1 6.28319 10 0.628319
0.628318530717958623 1 -0.628318530717958623
......
```

と、最初は今度はいいみたいです。
赤木： じゃあ、1 周期やってみて。
学生 B：では、ステップ毎に書くのをやめて、、、

```
<delaware:/usr2/makino/papers/Interface>!eu
euler1
Enter k, nsteps :1 10
k, period, nsteps, dt = 1 6.28319 10 0.628319
0.628318530717958623 3.12658511680165985 3.29196073410105283
<delaware:/usr2/makino/papers/Interface>euler1
Enter k, nsteps :1 100
k, period, nsteps, dt = 1 6.28319 100 0.0628319
0.0628318530717958679 0.217706841984230459 0.0100448605046154719
<delaware:/usr2/makino/papers/Interface>euler1
Enter k, nsteps :1 1000
k, period, nsteps, dt = 1 6.28319 1000 0.00628319
0.00628318530717958661 0.0199349143076453517 8.4329693743111499e-05
<delaware:/usr2/makino/papers/Interface>euler1
Enter k, nsteps :1 10000
k, period, nsteps, dt = 1 6.28319 10000 0.000628319
0.000628318530717958618 0.00197586995377499117 8.28467565412249203e-07
<delaware:/usr2/makino/papers/Interface>euler1
Enter k, nsteps :1 100000
k, period, nsteps, dt = 1 6.28319 100000 6.28319e-05
6.28318530717958564e-05 0.00019741157073215628 9 8.26997353232364789e-09
<delaware:/usr2/makino/papers/Interface>
```

x のほうはステップを 10 倍にすると大体 $1/10$ になってますが、 v は 2 桁づつ減ってます。これで合っているんでしょうか？
赤木： 「合ってるんでしょうか」って、あなたはどう思うの？
学生 B：え、そう言われても、、、オイラー法は一次精度ですよね。一次精度ということは、同じ時間積分した時の誤差がステップサイズに比例するということだから、少なくとも x のほうはそうなってるし、それでいいと思います。
赤木： まあ、そうなんだけど、折角線型の問題を持って来たから、もうちょっとやってみて。(x, v) をベクトル $\mathbf{x}$ と書くと、 $\mathbf{x}_i$ から $\mathbf{x}_{i+1}$ への変換は線型変換だから行列で書けるわね。面倒だから $k = 1$ にしましょ、周期で時間と速度をスケールすれば同じことだから。
学生 B：ええと、

$$\mathbf{x}_{i+1} = \begin{pmatrix} 1 & \Delta t \\ -\Delta t & 1 \end{pmatrix} \mathbf{x}_i \tag{6}$$

ですね。
赤木： そうそう、で、これ回転と定数倍に分かれるでしょ？
学生 B：あ、そうですね

$$\begin{pmatrix} 1 & \Delta t \\ -\Delta t & 1 \end{pmatrix} = \sqrt{1+\Delta t^2} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \tag{7}$$

で、θ は $\tan\theta = \Delta t$ の解です。あ、これなら数値解がどうなるか分かりますね、n ステップ後の解は初期条件の $(1,0)$ に上の行列の n 乗を掛けるだけだから、 $(1+\Delta t^2)^{n/2}(\cos n\theta, \sin n\theta)$ ですね。
赤木： そうね、後、$n\Delta t = 2\pi$ として、それから $\tan^{-1}$ のテイラー展開

$$\tan^{-1} x = x - x^3/3 + x^5/5 \cdots \tag{8}$$

を使って展開して、$(1+\Delta t^2)^{n/2}$ も展開して最初の項だけ残してみて。
学生 B：ええと、、、、 $(1 + 2\pi^2/n, 8\pi^3/3n^2)$ となりますね。これと上の数値解を比べると、ああ、 n が大きいとよく合いますね。
赤木： そういうことね。調和振動をオイラー法で解く時には、真の解は回転なわけで数値解の誤差は回転角の誤差とベクトルの長さというか絶対値の誤差になるわけね。1 ステップでの回転角の誤差は $\Delta t^3/3$ で、絶対値の誤差は $\Delta t^2/2$ だから、1 周期後は絶対値は $1/n$、角度は $1/n^2$ に比例した誤差が見えるわけ。で、これがそれぞれ x と v の誤差になってるのよ。
学生 B：そうですね、でも、結局全部手計算で答がわかるんならプログラム書かなくてもよかったんでは、、、
赤木： まあ、この問題についてはもちろんそうなわけ。数値積分するのは、解析的に解けない問題をやるためだから。でも、実際あなたの最初のプログラムは解析的に出せる答も出せてなかったわけよね。で、解析解と誤差の大きさまで合ってればプログラムが正しいのはさすがに確かでしょ。
学生 B：それはまあそうなんですが、、、

2　シンプレクティック法

学生 B：ところで、僕が書いた最初の間違ったプログラムですが、どうしてあれのほうが正しいオイラー法よりもいいんですか？
赤木：そうねえ、まあ偉そうにいうと、あれは「一次のシンプレクティック法」というものにたまたまなっていたからうまくいくのね。ハミルトン力学系でハミルトニアンが分離可能な時、まあ、普通は要するに運動方程式が

$$\frac{d^2\mathbf{x}}{dt^2} = -\nabla\Phi(\mathbf{x}) \tag{9}$$

と書ける場合ね、そういう時に、一次のシンプレクティック法っていうのは、例えば

$$\mathbf{v}_{i+1} = \mathbf{v}_i - \Delta t \nabla \Phi(\mathbf{x}_i) \tag{10}$$
$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta t \mathbf{v}_{i+1} \tag{11}$$

というふうに、最初に速度を更新して、更新した速度を使って位置を更新するというふうに位置と速度で違うやり方で計算する方法なわけ。多変数だったら、もちろんまず速度の全要素を更新してから位置を更新しないといけないわけね。

これがどうしてうまくいくかってことだけど、理屈はいろいろあるけど、詳しくは吉田さんの解説 [2] でもみて。とりあえずさっきのオイラー法みたいに行列で書いてみて。

学生 B：あ、はい、、、って、x_{i+1} の式に v_{i+1} があるから行列にならないですけど、、、

赤木： え、でも、v_{i+1} は x_i と v_i で書けてるんだから、それを代入すればいいでしょ。

学生 B：あ、なるほど、そうすると、

$$\mathbf{x}_{i+1} = \begin{pmatrix} 1 - \Delta t^2 & \Delta t \\ -\Delta t & 1 \end{pmatrix} \mathbf{x}_i \tag{12}$$

です。左上の要素に $-\Delta t^2$ がつくのが違いますね。

赤木： そうね。形を見ればわかるけど、これは回転+定数倍にはなってないわけね。 まあ、固有値をまずは求めてみましょ。

学生 B：えーと、固有方程式が

$$(1 - \Delta t^2 - \lambda)(1 - \lambda) + t^2 = 0 \tag{13}$$

で、整理すれば

$$\lambda^2 - (2 - \Delta t^2)\lambda + 1 = 0 \tag{14}$$

あれ、複素解ですよ。 $\Delta t < 2$ なら複素解で、絶対値が 1 ですね。

赤木： そういうこと。だから、固有ベクトルの空間で考えると厳密に円軌道をとるわけね。もとの空間ではどうなるかしら？

学生 B：えーと、なんか楕円ですよね。

赤木： そういうことなわけ。つまり、時間刻みを有限にとっているのに、数値解は何周回っても必ずある楕円の上にのるのよ。

学生 B：えーと、それで x の誤差が 3 次になることが説明できます？

赤木： できるわよ。でも、それあんまり難しくないし、あとで考えてみて。調和振動だけで話が終わっちゃうと書くほうは楽でいいけど、さすがにそれだけですますわけにもいかないから話を進めましょう。次は簡単な非線型問題がいいんだけど、どんなのにしましょ？

学生 B：ええと、ケプラー問題とかはどうでしょう？非線型だけど解析解があるから、とりあえずプログラム書くほうとしては安心できます、、、

赤木： そうね、最終的にはもちろん解析解がないものを考えるんだけど、調和振動からいきなりなにもわからない世界にいくのもつらいかしらね。でも、3 変数か、平面問題としても 2 変数よね。プログラムどんなふうに書くのがいいかな、、、まあ、とりあえずおまかせするわ。

学生 B：はい、、、

3　ケプラー問題

(とても長い時間のあと)
一応作ってみました。積分法は、折角なのでシンプレクティックというののほうです。こっちのほうがプログラム簡単ですし。

```
//
// kepler.C
//
// J. Makino 2002/06/03
#include  <stdlib.h>
#include  <math.h>
#include  <iostream>

using namespace std;

const int ndim=2;

int main()
{
    double x[ndim],v[ndim],t;
    double k;
    int norbits, nsteps, ndiag;
    cerr << "Enter norbits, nsteps, ndiag :";
    cin >> norbits >> nsteps >> ndiag;
    double dt = 2*M_PI/nsteps;
    cerr << "Enter vx,vy (normalized to 1):";
    cin >> v[0]>>v[1];
    double vabs = 0;
    for (int k = 0;k< ndim;k++) vabs +=v[k]*v[k];
    vabs = sqrt(vabs);
    for(int k = 0;k<ndim;k++) v[k]/= vabs;
    cerr << "v = " << v[0] << " " << v[1] << endl;
    x[0] = 1 ; x[1] = 0;
    t = 0;
    for(int i = 0 ; i<norbits*nsteps;i++){
        double a[ndim];
        double r2 = 0;
        for (int k = 0;k< ndim;k++) r2 += x[k]*x[k];
        double r = sqrt(r2);
        double r3inv = 1.0/(r*r2);
        t += dt;
        for (int k = 0;k< ndim;k++) a[k] = -x[k]*r3inv;
        for (int k = 0;k< ndim;k++) v[k] += a[k]*dt;
        for (int k = 0;k< ndim;k++) x[k] += v[k]*dt;
        //
        // calculate energy etc...
        //
        if ( ((i+1) % ndiag) == 0){
```

```
            double v2 =0;
            for (int k = 0;k< ndim;k++) v2 += v[k]*v[k];
            double etot = 0.5*v2 - 1.0/r;
            cout.precision(12);
            cout << "T, E,x = " << t <<" " << etot << " " << -0.5 -etot
                 << " " << x[0] << " " << x[1]<<endl;
        }
    }
}
```

とりあえず周期は 2π になるような初期条件から始めることにしてます。円軌道の場合で時間刻みをいろいろ変えてみたんですが、なんか調和振動の場合と変わんないみたいなんですけど、、、

赤木： 変わらないっていうと？

学生 B：ええと、 $(1,0)$ からの円軌道で、x と y の 1 周期後の誤差をみるとやっぱり時間刻みの 3 乗や 2 乗になってます。

赤木： 円軌道でない場合はやってみた？

学生 B：やってないですけど、おんなじじゃないんですか？

赤木： まあ、折角プログラム書いたんだから、確かめてみたら。

学生 B：はあ、、、

(調べる)

あれ、ほんの少しでも円軌道からずれると、いきなり全然駄目っていうか、誤差が時間刻みの 1 乗になっちゃいます。え、どうしてなんですか？

赤木： これもちゃんと説明すると長いんだけど、要するに円軌道っていうのは特別な場合になってるのよ。と、これでは全然説明になってないわね、まあちゃんと計算すればすぐわかるはずよ。つまり、円軌道の場合には本当の解自体は調和振動と同じでしょ、で、誤差が小さくとすれば、誤差の従う差分方程式を書き下せるわけね。これは線型の方程式になって、前の調和振動の場合と同じ振舞いをするわけ。

学生 B：え、でもそれなんか変じゃないですか。誤差が小さければ線型化した方程式を書けるのは円軌道でなくても同じじゃないですか。それに方程式自体同じようなものにならないですか？

赤木：それはねえ、本当はそうなの。ではなにが違うかってのは、まあ計算してみればすぐにわかるわ。こういうところの理屈がわかって、ちゃんと計算できるのは大事なことよ。そこまでいちいち私が説明したんではしょうがないでしょ。

学生 B：ええと、それはそうなんですが、でも自分でやってできなかったら困るじゃないですか。僕は後で赤木さんに聞けるけど、読者はどうするんですか？

赤木： あーもう、細かいわね。問い合わせが殺到するようなら作者が Web[3] に説明をだすでしょ。それはいいんだけど、誤差が一周のあいだにどう変わるかって調べてみた？例えばエネルギーとか。

学生 B：みてないですけど、一次の公式なんだから単調に増えるか減るかするだけじゃないんですか？

[3] http://grape.astron.s.u-tokyo.ac.jp/pub/people/makino/interface-simulation/ の予定。本誌からは省略したプログラム等も収録する。

赤木：ちょっとやってみてよ。
学生 B：やってみました、、、え、一周期でエネルギーがもとに戻る、、、
赤木： そういうことね。これは調和振動の場合もエネルギーを書いてみれば同じだったはずね。円軌道になるはずのが楕円になったんだから、周期的にエネルギーが変わるわけ。もとが楕円のケプラー軌道なら、数値解もまあ楕円みたいなものがでてきてその周期でエネルギーも変わるわけ。ただ、数値解は厳密な楕円じゃないし、そもそももとのところに戻らないから、エネルギーが変わらなければいいというものでもないけど。
こういうふうになるというのがシンプレクティック法の特徴の 1 つなの。真の解が周期解なら、数値解も周期解になるのね。もちろん数値誤差があるから同じ周期解にはならないわけで、ずれの大きさが誤差で決まるわけ。
学生 B：ということは、シンプレクティック法でも 1 次ではなくてもっと高次の方法を使えば誤差を簡単に小さくできるわけですよね。そういう高次の方法っていうのはあるんですか？
赤木： もちろんいくらでもあるの。ここ 15 年くらいでいろいろ研究が進んだし。 まず 2 次からいきましょう。2 次はこういうの

$$
\begin{aligned}
v_{i+1/2} &= v_i + \Delta t a(x_i)/2 && (15)\\
x_{i+1} &= x_i + \Delta t v_{i+1/2} && (16)\\
v_{i+1} &= v_{i+1/2} + \Delta t a(x_{i+1})/2 && (17)
\end{aligned}
$$

学生 B：なんか変な格好ですね。 $v_{i+1/2}$ ってのは中間で定義されるってことですか？でも、それ消去して

$$
\begin{aligned}
x_{i+1} &= x_i + \Delta t v_i + \Delta t^2 a(x_i)/2 && (18)\\
v_{i+1} &= v_i + \Delta t[a(x_i) + a(x_{i+1})]/2 && (19)
\end{aligned}
$$

としてもおんなじじゃないですか？
赤木： あ、これはもちろんおんなじよ。私の書いた形のほうが足し算一つ少ないから速いかもしれないけど、普通は加速度計算するほうに時間かかるから関係ないわね。いじましいことをいえば、計算量を減らすには逆に $v_{i+1/2}$ だけを残して

$$
\begin{aligned}
v_{i+1/2} &= v_{i-1/2} + \Delta t a(x_i) && (20)\\
x_{i+1} &= x_i + \Delta t v_{i+1/2} && (21)
\end{aligned}
$$

とすればいいわね。最初は上の式で v_i から計算するわけ。この形のことをリープフロッグ公式ということが多いわね。ここで書いたのはみんな同じだから、どれもリープフロッグと呼んで間違いではないけど。
学生 B：あれ、これだと一次の公式とおんなじじゃないですか。どうしてこれで 2 次になるんですか？
赤木： 2 番目の形をみれば 2 次でしょ、x はテイラー展開の 2 次まで入ってるし、 v は台形公式だから。台形公式って知ってるわよね？17 式

が定義。普通は陰公式で、 $i+1$ での値を求めるのに代数方程式を解かないといけないけど、この場合は加速度が速度に依存しないから陽的に計算できるわけ。最後の形だと x と v がずれて定義されてるのが仕掛けということになるわね。

学生 B : えーと、式がそうだってのはわかるんですが、なんか変な気がするんですが、、、 大体 x と v で式が違うのが変じゃないですか。

$$x_{i+1} = x_i + \Delta t(v_i + v_{i+1})/2 \tag{22}$$
$$v_{i+1} = v_i + \Delta t[a(x_i) + a(x_{i+1})]/2 \tag{23}$$

とは、、、あ、これだと陰公式になっちゃうんですね。 x_{i+1} を計算するのに v_{i+1} が必要で、それには $a(x_{i+1})$ がいるのか。

赤木 : そうね。台形公式はいろいろいい性質があるけど、普通に使うとさっきいったように陰公式で代数方程式を解く必要がでてくるわけ。リープフロッグは、形式的には

$$x_{i+1} = x_i + \Delta t(v_i + v_{i+1})/2 - \Delta t^2[a(x_{i+1}) - a(x_i)]/4 \tag{24}$$
$$v_{i+1} = v_i + \Delta t[a(x_i) + a(x_{i+1})]/2 \tag{25}$$

と書けるでしょ。x の最後の項は $O(\Delta t^3)$ だから、公式の次数には関係ないのよ。

学生 B : ええと、それって変じゃないですか？ $[a(x_{i+1}) - a(x_i)]/\Delta t$ は加速度の微分の近似になってるから

$$x_{i+1} = x_i + \Delta t v_i + \Delta t^2 a(x_i)/2 + \Delta t^2[a(x_{i+1}) - a(x_i)]/6 \tag{26}$$

としたら 3 次になりませんか？陰公式だから計算面倒ですが、、、

赤木 : あら、なかなか冴えてるじゃない。まあ、そのへんも、自分で調べてみて納得しないと、本とか読んだだけじゃわからないのよね。というわけでそれは宿題ということにして、リープフロッグでプログラム書いてみて。で、メインプログラムだけってのでなくて、もうちょっと違う公式とかそういうのを使いやすいように書いてみてくれないかしら。

学生 B : やってみます。

(無限とも思える時間の後) こんなのでどうでしょう？

```
//
// kepler-leap.C
//
// J. Makino 2002/06/05
#include  <stdlib.h>
#include  <math.h>
#include  <iostream>

using namespace std;

const int ndim=2;
```

```
void accel(double x[ndim],
           double v[ndim],
           double a[ndim])
{
    double r2 = 0;
    for (int k = 0;k< ndim;k++) r2 += x[k]*x[k];
    double r = sqrt(r2);
    double r3inv = 1.0/(r*r2);
    for (int k = 0;k< ndim;k++) a[k] = -x[k]*r3inv;
}
void update(double y[ndim],
            double ydot[ndim],
            double dt)
{
    for (int k = 0;k< ndim;k++) y[k] += ydot[k]*dt;
}

void leapfrog(double x[ndim],
              double v[ndim],
              double a[ndim],
              double dt)
{
    double dthalf = dt/2;
    update(v,a,dthalf);
    update(x,v,dt);
    accel(x,v,a);
    update(v,a,dthalf);
}

void diag(double x[ndim],
          double v[ndim],
          double t)
{
    double v2 =0;
    double r = 0;
    for (int k = 0;k< ndim;k++) v2 += v[k]*v[k];
    for (int k = 0;k< ndim;k++) r += x[k]*x[k];
    r = sqrt(r);
    double etot = 0.5*v2 - 1.0/r;
    cout.precision(12);
    cout << "T, E,x = " << t <<" " << etot << " " << -0.5 -etot
         << " " << x[0] << " " << x[1]<<endl;
}

int main()
{
    double x[ndim],v[ndim],t;
    double k;
```

```
    int norbits, nsteps, ndiag;
    cerr << "Enter norbits, nsteps, ndiag :";
    cin >> norbits >> nsteps >> ndiag;
    double dt = 2*M_PI/nsteps;
    cerr << "Enter vx,vy (normalized to 1):";
    cin >> v[0]>>v[1];
    double vabs = 0;
    for (int k = 0;k< ndim;k++) vabs +=v[k]*v[k];
    vabs = sqrt(vabs);
    for(int k = 0;k<ndim;k++) v[k]/= vabs;
    cerr << "v = " << v[0] << " " << v[1] << endl;
    x[0] = 1 ; x[1] = 0;
    t = 0;
    double a[ndim];
    accel(x,v,a);
    for(int i = 0 ; i<norbits*nsteps;i++){
        leapfrog(x,v,a,dt);
        t+= dt;
        //
        // calculate energy etc...
        //
        if ( ((i+1) % ndiag) == 0){
            diag(x,v,t);
        }
    }
}
```

赤木： 初期設定がいま一つな感じもするけど、まあいいんじゃないかしら。精度とかは見た？

学生 B：ええ、ちゃんとエネルギーとかが 2 次になってるのは確認しました。これよりもっと精度をあげようと思うとどんな方法があるんですか？

赤木： そうね、一番簡単なのは、リープフロッグの組合せで次数をあげていく方法ね。図 2 みたいに、最初行き過ぎて、それからもどってまた行き過ぎるというふうに 3 回リープフロッグを使って、誤差がうちけしあうようにするわけ。 進む量をどうすればいいかっていうと、1 ステップでの誤差は Δt^3 に比例するわけだから、進む量と戻る量を $d_1\Delta t, d_2\Delta t$ とすれば、

$$2d_1 - d_2 = 1 \tag{27}$$

$$2d_1^3 = d_2^3 \tag{28}$$

を満たすように決めればいいわけ。

学生 B：あ、なるほど、そうすると、プログラムは例えば

```
void symp4(double x[ndim],
               double v[ndim],
```

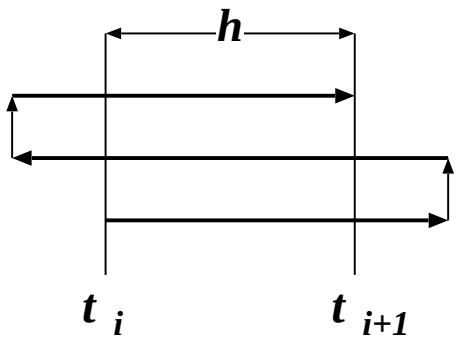

図 2: 4 次シンプレクティック法

```
                double a[ndim],
                double dt)
{
    const double d1 = 1.0/(2.0-pow(2.0,1.0/3));
    const double d2 = 1-2*d1;
    leapfrog(x,v,a,d1*dt);
    leapfrog(x,v,a,d2*dt);
    leapfrog(x,v,a,d1*dt);
}
```

というのでいいわけですね。

赤木： そういうことね。これは割合簡単でしょ？

学生 B : そうですね。3 体とか、粒子数が多いのをやってみたいんですが、、、解析解がある場合の話だけではつまんないですし。

赤木： そうね、まあ、なんか書いてみたら？

学生 B：はい、、、

4 多体問題

赤木： (無限^2 くらいの時間のあと) どう?

学生 B : ええと、その、一応出来たと思うんですけど、3 体でやってみるとなんか初期条件によっては時間刻みをものすごく小さくしないとうまくいかないみたいなんです、、、

赤木： あ、それはそうなんじゃない？3 体問題だと軌道がカオスになる場合があるでしょ、その時だと、運が悪い初期条件だと 2 つの粒子がすごく近づくことがあるから。粒子数をもっと増やして、星団とか銀河のシミュレーションをやろうとすれば運の問題ではなくて必ずそういうことは起きるわけね。これをどうやって解決するかというのはちゃんと話をすると本 1 冊では終わらないのよね、、、まあ、一番安直な解決は、物

理法則のほうをいじるってのね。つまり、ある適当な距離 ϵ くらいより近づいたらそれ以上引力大きくならないようにしちゃうわけ。式が簡単でよく使われているのはポテンシャルを $1/r$ の代わりに $1/\sqrt{r^2+\epsilon^2}$ にする方法ね。微分すればわかるけど力の式が簡単なのよ。

学生 B：あ、なるほど。でも、物理法則を変えて大丈夫なんですか？

赤木： それはもう時と場合によるわけよね。まあ、真の解ではなくて数値解をだすこと自体、物理法則を変えているみたいなものだから、同じような話ではあるわけ。つまり、物理法則を変えたためにみたいものに入ってくる誤差が、みたい精度に比べて小さくなっていればいいわけね。

学生 B：でも、どうすれば物理法則を変えたことによる誤差ってのがわかるんですか？

赤木： まあ、それはいろいろやってみる、例えば上の場合なら ϵ の値を変えた計算をしてみて、答がどうなるかみるくらいしかないわね。時々見かけるんだけど、これ変えた計算をして結果が変わらなかったら大丈夫とは限らないの。つまり、あまりにも絶望的に答が間違ってるので多少精度を変えても結果が変わらないこともあるのよ。

学生 B：えー、でも、そうかどうかってどうすれば、、、

赤木： 、、、わかるかってのは難しいのよね。まあ、理論的に見積もりが出来ればいいっていうか、理論的な見積もりが全然できなければ間違ってる可能性が高いと思ったほうがいいわね。それで、理論的な見積もりのほうは扱う対象の物理的な性質やみたい振舞いがなにかとかいったことに依存するから、あんまり一般的なことは言えないの。

あ、そうそう、3 体問題くらいだったら、時間刻みを可変にして 2 つが近づいたらそれに応じて時間刻みを短くして、さらにどうしようもなく近づいたら解析解でつなげばなんとかなるわ。ただ、可変刻みにするとシンプレクティック法はそのままでは使えないの。どうしてかっていうと、ほら、調和振動の時を思い出してみると、あれだと円が楕円になって、どんな楕円かは時間刻みによったわけ。とすると時間刻みをふらふら変えると楕円になってくれるとは限らないわけね。

数学的な話としては、シンプレクティック法というのはもともと公式自体が解析力学でいう正準変換になっている、つまり、変換が物理系の性質を変えないような公式ということになるんだけど、時間刻みを可変にすると正準変換ではなくなっちゃうのね。それでもうまいことやれば時間刻みを可変にしてシンプレクティック法の良い性質を保てるというような話もないわけではないけど、ちょっとそこまで話がいかないわよね。

5 床とボール

これまで扱ってきたのは全部ハミルトン力学系で、運動エネルギーが保存するような場合だったんだけど、最後にそうでない場合をちょっと考えてみましょ。こんなのどうかしら。

$$\frac{d^2\mathbf{x}}{dt^2} = \begin{cases} -g & (x > 0) \\ -g - kx - \eta v & (x \leq 0) \end{cases} \tag{29}$$

柔らかい床にボールかなんかを落とした時にどう跳ねかえるかっていうのの一番単純化した式ね。まあ、解析的に解けるんだけど、練習にはいいでしょう。

学生 B：えーと、ちょっとまって下さい、これ加速度が不連続じゃないですか。

赤木： そうよ。なんか問題？解はあるでしょ？一意性だってあるわよ。

学生 B：そりゃそうですけど、数値解は、、、 x の符号が切り替わるところで破綻しませんか？

赤木：まあ、だから、その辺やってみてって話よ。積分公式はとりあえずオイラー法と、2 次のルンゲクッタと台形公式くらいを比べてみたら。シンプレクティックはとりあえず使えないから。

学生 B：はあ、、、

赤木： 係数をどれくらいにするかも決めときましょ。$g = 1$ として、あと k は大きくないとそれらしくならないから、 10^4 とかにしておく？で、η は 0 と 10 くらいとか。後、台形公式は本当はこれはこれでいろいろ特別な性質があるんだけど、ちゃんとそういう性質がでるためには陰解法でててくる方程式をニュートン法とかのちゃんとした方法で解く必要があるの。これはちょっと大変だから、単純反復でいいわよ。

$$x = f(x) \tag{30}$$

の形の方程式を解くのに、

$$x_{new} = f(x_{old}) \tag{31}$$

という形で繰り返し計算して、収束してくれたら嬉しいなっていう方法ね。

学生 B：分かりました。

学生 B：えーと結果ですが、まずグラフでもおみせするのがいいですよね。オイラー法は図 3 の通りです。これはまあ全然だめで、抵抗が入ってる場合でも逆に振幅が増えています。ルンゲクッタや台形則では時間刻みを小さくすればまあまあなんとか発散しなくなります。台形則のほうが多少性質が良くて、大きい刻みがとれるみたいです。

赤木： あら、なかなかちゃんとできてるじゃない。

学生 B：でしょ。これは大変だったんです、、、

赤木： ところで、どう誤差がでるかってのは調べてみた？

学生 B：ええ、グラフ 6 上を見て下さい。ボールが空中にある時には全く誤差はなくて、地面で跳ね返る時にだけ集中して誤差が出ています。

赤木：あ、ちょっとまって、跳ね返る時にはどんなふうに誤差がでるの？

学生 B：ええと、普通につもっていくんではないんですか？

赤木： まあ、そこを拡大してみてよ。

学生 B：あ、はい、6 下ですが、、、ああ、台形公式では x の符号が変わるところでしか誤差がでないんですね。

赤木： そう。これは、実は台形公式にもシンプレクティック法と同じくようにハミルトン系ではエネルギーの誤差がある程度以上は成長しないという性質があるからね。台形公式はシンプレクティックではないんだけど、時間反転に対して対称である、つまり、時間を逆に進めていくと

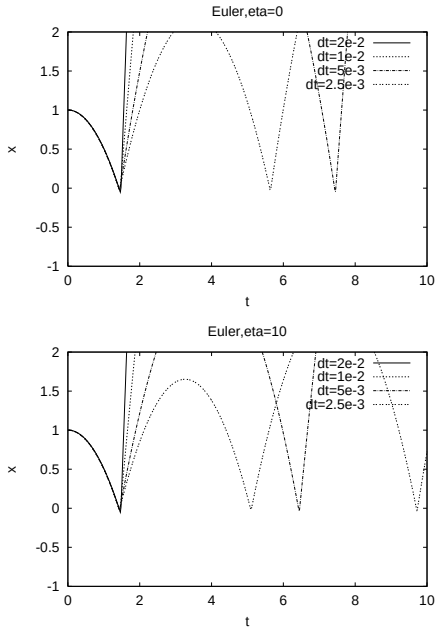

図 3: オイラー法での軌跡

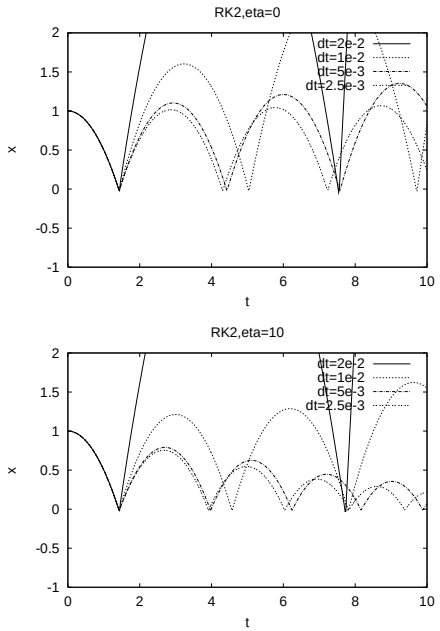

図 4: ルンゲクッタ法での軌跡

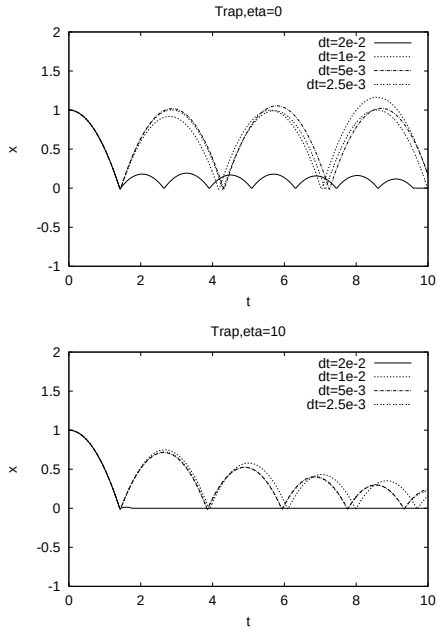

図 5: 台形則での軌跡

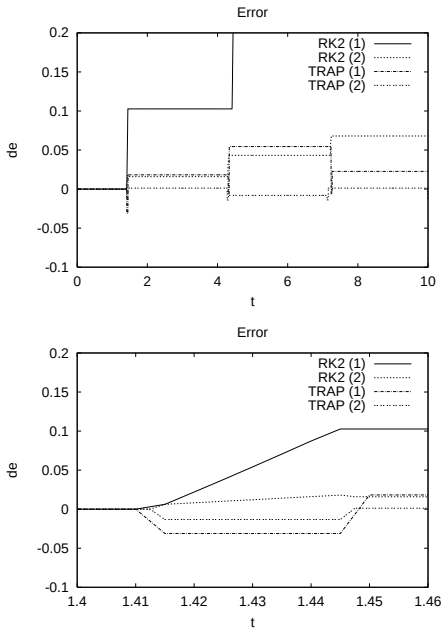

図 6: エネルギーの誤差

厳密にもとに戻るという性質があるの。こういう時間対称な公式はシンプレクティックと同じようにいろいろいいことがあるのね。

学生 B：なるほど。

赤木： まあ、でも、あれね、いいことがあるからそれを使うべきといえるかどうかというのは割合難しいところね。この例みたいに不連続性があるのはある意味一番ややこしい例なんだけど、この例そのものだったら本当は不連続になる時刻が求められるから、そこにくるように時間刻みを調節すればいいわけね。ただ、こんなふうにボール 1 つならそれでいいけど、沢山ボールがあってそれらがお互いにぶつかったり跳ね返ったりしてるとそう簡単にはいかないでしょ。

学生 B：そうですよね。そういう時はどうするんですか？

赤木： そうね、これはいろいろ調べてみたんだけど、なんかあんまりなにも考えていないみたい。オイラー法や 2 次の予測子・修正子法を使って、ただ時間ステップを十分短くしてすましているのが多いわね。

学生 B：それってなんか力任せっていうか、もうちょっとスマートなやりかたはないかって気もするんですけど。

赤木： そうね、原理的には、例えば粒子毎にバラバラに時間刻みを変えるようなやり方もあるし、分野によっては使われていないわけじゃないけど、そういうやり方の問題は、シンプレクティックとか時間対称とか、そういういい性質がいま知られているやり方では保たれないということね。シンプレクティックはそもそも時間刻みを変えることができないし、時間対称な公式は時間刻みを変えられないわけじゃないけど、バラバラには変えられないのよ。

まあ、割合原理的な問題として、どれくらい正確に計算する必要があるのか？ってこともあるけど。ボールっていうと遊びみたいだけど、粉体のシミュレーションでは実際に粒子同士がぶつかったり跳ね返ったりというのを計算するわけね。でも、そこで割合一般的にわかってきているらしいことは、いろんなことが衝突のモデルの詳細にはあんまり依存しないということなの。

学生 B：その、「らしい」ってのはなんです？

赤木： 私が確かめたわけじゃないからね。まあ、それはともかく、もちろん衝突して跳ね返った時にエネルギーが増えたりしたらだめだけど、減っていればどれくらい減るかは桁で違ってもあんまりなにが起きるかは変わらないとかね、そういうこともあるらしいの。まあ、そういう場合だと、そんなに真面目に計算してもしょうがないわけね。

まあ、そうはいっても、そういうのが本当かどうかを確かめるためにはちゃんと衝突した時に何が起きるかを制御できないといけないし、そのためにはもちろん精度がちゃんと制御できないといけないわけだからいい加減な計算でいいということには結局はならないはずだけど。

学生 B：はあ、そういうものですか、、、

赤木： そういうものよ。もうページもないみたいだから、とりあえず今回はこれくらいにしましょ。またなんかあったらよろしくね。

6　より深い理解のために

本文中で引用した文献の他に、有用なテキストをあげておく。
常微分方程式の数値解析についての、日本語で読める本としては
三井斌友:数値解析入門、朝倉書店 (1985)
三井斌友:微分方程式の数値解法 I、岩波講座応用数学、岩波書店 (1993)
がある。著者は同じで、内容もかなりオーバーラップしている。上の本の方が説明が丁寧でわかりやすいと思うが、絶版で入手困難かもしれない。下の本はより新しい成果まで取り上げられている。
シンプレクティック法については、
Sanz-Serna, J.M. & Calbo, M.P.: Numerical Hamiltonian Problems, 1994, Chapman & Hall
がある。
数値計算アルゴリズムの本としては
戸川隼人：新装版 UNIX ワークステーションによる科学技術計算ハンドブック、サイエンス社 (1998)
をあげておく。UNIX の基本的な使い方から、線形計算、非線形方程式、固有値問題、数値積分、補間、常微分方程式までの広範な分野をカバーする大著である。説明、プログラム例のどちらもきちんとしていると思う。同様な本としては、有名な
Press, W. H., Teukolski, S. A., Vetterling, W. T., & Flannery, B.P.: Numerical Recipes in C — The Art of Scientific Computing, 2nd edition, Cambridge University Press (1992)(丹慶勝市他訳：Numerical Recipes in C [日本語版]、技術評論社 (1993))
がある (翻訳は第一版) が、特に常微分方程式の数値解法については不適切な記述が多く、第一版はあまり勧められない。第二版では多少改善されている。
伊理正夫、藤野和建：数値計算の常識、共立出版 (1985)
は、読み物的なものだが、数値計算を実際に使う際に起きる奇妙な現象や問題点などをわかりやすく解説している。

参考文献

[1] 牧野淳一郎.「パソコン物理実地指導」、共立出版 1999

[2] 吉田春夫. シンプレクティック数値解法. 数理科学、384 1995