3. 常微分方程式 (Ordinary Differential Equation, ODE)

常微分方程式(Ordinary Differential Equation, ODE)は物理のあらゆる場面で現れる。この微分方程式を数値計算で解いてみよう。

3.1 習うより慣れろ

微分方程式(ODE)のアルゴリズムには、オイラー法、Runge-Kutta 法、アダムス法などがある。もちろん、これらの解法の仕組みを知って、どういう利点、欠点があるか、などを知るべきだが、それは後回しにして、Octave に用意されている ODE を解く関数(**lsode**[1])を使って、数値計算してみることから始めよう。ちなみに ODE を解く関数のことを ODE ソルバーと言う。

（例題）微分方程式、$\frac{dx}{dt} = -x$ を初期条件 $x(0) = 1$ で解き、$0 \le t \le 10$ の範囲で図示せよ。

まずは、ODE を関数として定義する。(xdot.m として保存)

```
% xdot.m
function xdot = xdot(x,t)
xdot=-x;            %これが ODE
```

ODE ソルバーである関数 **lsode** は、次のような書式で使う。

```
lsode('func',x0,t)
```

func は、関数として定義した微分方程式の関数名。上の例では、**xdot**。関数は、''で囲むことに注意。**x0** は初期値、今の例では **1**。**t** は定義域ベクトルである。メインプログラムは、

```
% odetest1.m
clear
t=[0:0.1:10]';
x=lsode('xdot',1,t);
plot(t,x);xlabel('t');ylabel('x');
```

これを odetest1.m という名前で保存して実行すると、指数関数的に減少するグラフが書ける(図 3.1)。

このように、解きたい ODE を function で定義すれば、ODE を解くアルゴリズムを知らなくても、とりあえずは解ける。

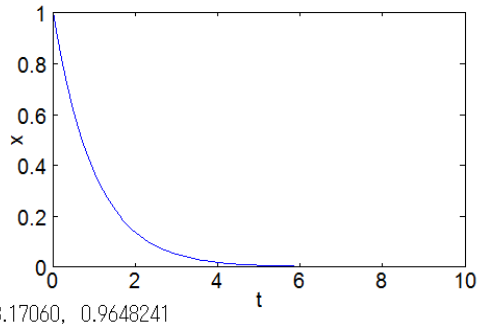

図 3.1

3.2. 二階の常微分方程式

二階の ODE の場合は、二元連立一階微分方程式に置き換えてやればいい。

（例題）$\frac{d^2x}{dt^2} + \frac{dx}{dt} + x = 0$ を初期値 $x(0) = 0$、$dx/dt\big|_0 = 1$ で解け。

$dx/dt = v$ とすれば、二元連立一階微分方程式に直せる。

$$\begin{cases} \frac{dx}{dt} = v \\ \frac{dv}{dt} = -v - x \end{cases} \qquad \begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v \\ -v - x \end{pmatrix}$$

今回も、**lsode** を使う。そのため、上の二元連立 ODE を関数として定義する必要がある。これを **xdot2** としよう。まず x を行列 **x** の 1 列目の要素 **x(1)** に代入し、x の微分 $v=dx/dt$ を行列 **x** の 2 列目の要素 **x(2)** に代入する。つまり、x と v をひとまとまりにして、行列 **x** で表すわけ。**x(1)** と **x(2)** を使えば、行列 **xdot2** の 1 列目の要素は **x(2)** となるし、2 列目の要素は、**-x(2)-x(1)** となる。このように、関数 **xdot2** は、**x** と **t** を入力すると、**xdot2** という $n \times 2$ 行列を出力するものとして定義する。(n は定義域ベクトル **t** のサイズ)　以上のように文章で説明をしようとするとかなり面倒だが、実際プログラムを書くと非常に簡単。

```
% xdot2.m
function xdot2 = xdot2(x,t)
xdot2=[x(2) -x(2)-x(1)];
```

二階の ODE なので、初期値が二個 $(x_0,v_0)=(0,1)$ ある。これを **x0=[0 1]** というベクトルで表現しよう。**lsode** を実行すると、1 列目が $x(t)$、2 列目が $v(t)$ である $n \times 2$ の行列の形で解が出力される。つまり、**R=lsode(...)** として、行列 **R** の 1 列目を取

[1] **lsode** は、MATLAB では使えない。そのかわり、**ode45**, **ode23**, **ode113** の他、さまざまなソルバーが用意されている。

り出せば $x(t)$、2 列目を取り出せ $v(t)$が得られる。具体的には、**x=R(:,1)**、**v=R(:,2)**とすればいい。ここで使われているコロン「:」は、すべての要素を代表するという意味の記号。具体的なメインプログラムは、

```
% odetest2.m
t=[0:0.1:10]';
x0=[0 1];                %初期値は行ベクトルとして入力
R=lsode('xdot2',x0,t);
x=R(:,1);  %R の 1 列目の要素すべてを x とおく
v=R(:,2);  %R の 2 列目の要素すべてを v とおく
plot(t,x,t,v) ;xlabel('t'); ylabel('x');
title('odetest2.m')
```

このように、n 階の ODE は、n 元連立に直すことで解くことができる。

3.3. ODE の可視化

ODE について少し理解したところで、ODE の意味とアルゴリズムを考えるのに役立つ ODE の可視化をしてみよう。例として、上で用いた $dx/dt = -x$ を考える。

(例) $-2 \leqq t \leqq 2$, $-2 \leqq x \leqq 2$ の範囲で $dx/dt = -x$ を可視化する。

t-x 座標上の点(t,x)に注目し、それぞれの点でのベクトル(dt,dx)を図に書き込んでいく、という作業をしよう。ODE では、dx と dt の比$(= -x)$だけしか与えられていないので、ベクトルの長さは決められない。そこで仮に $dt=1$ とおけば、ベクトル(dt,dx) は $(1,-x)$ とすることができる。あとは、$-2 \leqq t \leqq 2$, $-2 \leqq x \leqq 2$ で刻みを適当に設定して、この範囲内のすべての点に対して、ベクトル (dt,dx) をプロットすればいい。**[t,x]=meshgrid(範囲を指定)**という関数を使うと、範囲内のすべての点が行列の形で作ることが出来る。矢印プロットには、**quiver** という関数を使う。

```
% visual.m
[t,x]=meshgrid(-2:0.2:2, -2:0.2:2);
[m n]=size(t);
dt=ones(m); %サイズが m×m で全要素が 1 の行列
dx=-x;    %これが ODE
quiver(t,x,dt,dx) % ベクトルを矢印でプロットする関数
xlabel('t');ylabel('x');
title('visual.m');
```

これ(visual.m)を実行すると、図 3.3.1 のような、矢印だらけの図がかける。仮に、初期値の点(t_0,x_0)を適当にとり(手書きの○印)、ベクトルの向きに線を延ばしていくと、その線が ODE の解になる。ベクトル(dt,dx)は、もともとODE の解の集合なので、初期値からスタートしてベクトルをつなげて作った線も ODE の解になっているというわけだ。

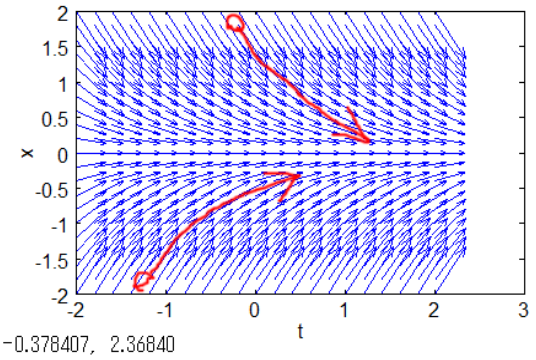

図 3.3.1

もう少し複雑な $\frac{dx}{dt} = t^2 x$ ではどうだろう。**dx=-x** の代わりに、**dx=t.^2 .*x** にすると(visual2.m)、図 3.3.2 のようになる。この例では、初期値をどこにとるかによって、解がどちら方向に発展していくか、ひと目で分かる。

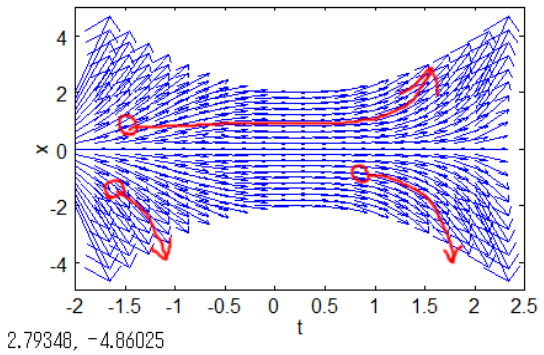

図 3.3.2

3.4. オイラー法と Runge-Kutta 法

次に ODE を解くアルゴリズムについて考えてみよう。ODE の数値解法は、突き詰めるとかなり奥が深い問題なので、ここですべてを紹介することはできないが、最も教科書的で基本的なアルゴリズムである、オイラー法と Runge-Kutta 法を紹介する。

まずはオイラー法。一般的な ODE を $\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}(t),t)$ とする。数値計算で求めたい解は、時間の関数である N 次元ベクトル

$\mathbf{x}(t)=[x_1(t)\ \ldots\ x_N(t)]$である。一般化すると分からない! という人のために 1 次元の関数 $x(t)$で考えよう。ODE の可視化の際に見たように、ある時間 t における点 x が与えられたとき、ODE の解は、矢印の方向に進むことで求まった。つまり、$t=t_0$における $x(t_0)$が初期値として与えられたとき、ある時間τだけ後の点 $x(t_0+\tau)$を(近似解で)求めることが出来たとすると、その次の点 $x(t_0+2\tau)$、さらに次の点 $x(t_0+3\tau)$...を求めれば、それらはODE の解の集合になっているはずだ。イメージとしては、図3.4.2。

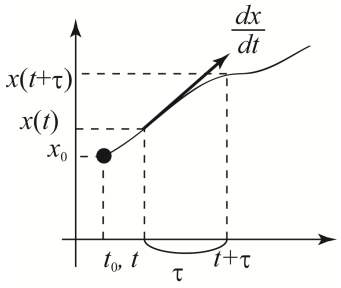

図 3.4.1

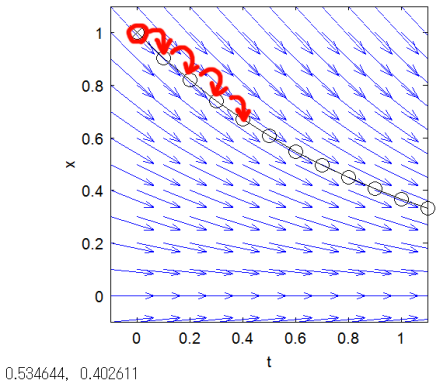

図 3.4.2

したがって、$x(t)$が与えられたとき、まずは次の点、つまり時間τ後の点 $x(t+\tau)$を求めたい。そのために、$x(t+\tau)$をテイラー展開して二次以降の項を無視する近似を行う。

$$x(t+\tau)=x(t)+\tau\frac{dx(\zeta)}{dt}\quad(\text{ただし}t<\zeta<t+\tau)$$
$$=x(t)+\tau f(x(\zeta),\zeta)$$

$\zeta=t$とおいて、

$$x(t+\tau)=x(t)+\tau f(x,t)$$

x が二次元以上の場合は、x をベクトルにすることで

$$\boldsymbol{x}(t+\tau)=\boldsymbol{x}(t)+\tau\boldsymbol{f}(\boldsymbol{x},t)$$

というオイラー法の公式が得られる。配列で表現すると、

$$x_{k+1}=x_k+\tau f_k$$

となる。繰り返しになるが、初期値 x_0 を与えると上の式から時間τ後の点 x_1 が求まり、さらにその次の x_2...が与えられるので、x_k が時間の関数として求まる。初期値(例えば図 3.4.2 で、(0,1)の〇印)を決めて矢印の方に作図すれば解が求まるのと同じように、$x_{k+1}=x_k+\tau f_k$ を使って、初期値から隣の点を次々と求めて並べていけば、解が求まるというわけだ。

オイラー法を使った ODE ソルバーを作ってみよう。

```
% euler.m
function x=euler(F,x,t)
[m n]=size(t); %t のサイズ(=ステップ数)を抽出
for k=1:m-1  %t のサイズ-1 回繰り返す。
  tau=t(k+1)-t(k);
  x(k+1,:)=x(k,:)+tau.*feval(F,x(k,:),t(k));
    %上の式がオイラー法の公式
end
```

関数$f(x(t),t)$の具体的な形を指定することなく、任意の関数で使えるようにするために、f_kを `feval(F,x,t)` という関数で表現している。これは、`F` という関数に、`x`、`t` を代入すると、$\mathbf{F}(x,t)$を出力するという関数である。オイラー法の公式

$x_{k+1}=x_k+\tau f_k$ は、

```
x(k+1,:) = x(k,:) + tau.*feval(F,x(k,:),t(k))
```

と表現できる。なお、`x(k,:)` の「:」は、「すべて」を表していて、`x(k,:)` は、「行列 `x` の `k` 番目の行の要素をすべてを取り出したベクトル」を表すことになる。こうすることで、`x` の次元を指定せずに済むので、この ODE ソルバーは、n 元連立 ODE でも使える。なお、`tau` は、`tau=t(k+1)-t(k)` によって、ループ内で毎回計算されているので、時間ベクトルの間隔 `tau` は `t` に依存する変数であってもかまわない[2]。

オイラー法は分かりやすいので、教育的な意味でよく取り上げられるが、テイラー級数の 2 次以降の項を無視しているため、途中で計算を打ち切ることによる誤差、打ち切り誤差($O(\tau^2)$)が生まれ、ステップを経るごとに、徐々に誤差が積み重なっていくので、実用上はあまり使われない。

ODE を解くためによく使われる解法は、4 次の Runge-Kutta(RK)法で、公式は、

$$\boldsymbol{x}(t+\tau)=\boldsymbol{x}(t)+\frac{1}{6}\tau[\boldsymbol{F_1}+\boldsymbol{2F_2}+\boldsymbol{2F_3}+\boldsymbol{F_4}]$$

[2] 刻み `tau` を一定にせず、変化の大きい領域で刻みを細かくとると、解の精度が上がる。

$$\boldsymbol{F}_1 = \boldsymbol{f}(\boldsymbol{x}, t)$$

$$\boldsymbol{F}_2 = \boldsymbol{f}\left(\boldsymbol{x} + \frac{1}{2}\tau\boldsymbol{F}_1, t + \frac{1}{2}\tau\right)$$

$$\boldsymbol{F}_3 = \boldsymbol{f}\left(\boldsymbol{x} + \frac{1}{2}\tau\boldsymbol{F}_2, t + \frac{1}{2}\tau\right)$$

$$\boldsymbol{F}_4 = \boldsymbol{f}(\boldsymbol{x} + \tau\boldsymbol{F}_3, t + \tau)$$

である。4 つの点を使い、それぞれの重み付けを変えることで誤差を減らして、次の点をより正確に求めようという方法だ。4 次の RK の打ち切り誤差は、$O(\tau^5)$でオイラー法より精度が高いことが知られている。euler.m と同じように、4 次の RK のソルバーを作ってみよう。

```
% rk4.m
function x=rk4(F,x,t)
[m n]=size(t);
for k=1:m-1  %t のサイズ-1 だけステップを繰り返す。
 tau = t(k+1)-t(k); %刻み幅
 F1=feval(F,x(k,:),t(k));
 F2=feval(F,x(k,:)+1/2*tau.*F1,t(k)+1/2*tau);
 F3=feval(F,x(k,:)+1/2*tau.*F2,t(k)+1/2*tau);
 F4=feval(F,x(k,:)+tau*F3,t(k)+tau);
 x(k+1,:)= x(k,:)+tau/6*(F1 +2*F2 +2*F3 +F4);
      %以上が 4 次の RK 法の公式
end
```

となる。

これでオイラー法と RK 法を用いた自作 ODE ソルバーができたので、実際に使ってみよう。

(例題) $\frac{d^2x}{dt^2} + \frac{dx}{dt} + x = 0$ を初期値$x(0) = 0$、$\left.\frac{dx}{dt}\right|_0 = 1$で解け。

ODE の定義には、3.2 で使った**xdot2**をそのまま使う。比較のため、オイラー法、RK 法、**lsode** で求め、それぞれをプロットしてみる。

```
%rk4euler.m  %オイラー、RK、lsode の比較
t=[0:0.2:10]'; x0=[0 1]; %定義域、初期値の定義
R1=euler('xdot2',x0,t); x1=R1(:,1); %オイラー法
R2=  rk4('xdot2',x0,t); x2=R2(:,1);  %RK 法
R3=lsode('xdot2',x0,t); x3=R3(:,1);  %lsode
plot(t,x1,'rx-.',t,x2,'bx',t,x3,'ko');
xlabel('t');  ylabel('x');
legend('Euler','RK4','lsode');
title('rk4euler1.m');
```

これを実行すると、図 3.4.3 のようになる。RK4 と **lsode** による結果は、大体一致しているが、オイラー法は若干ずれていることが分かるだろう。

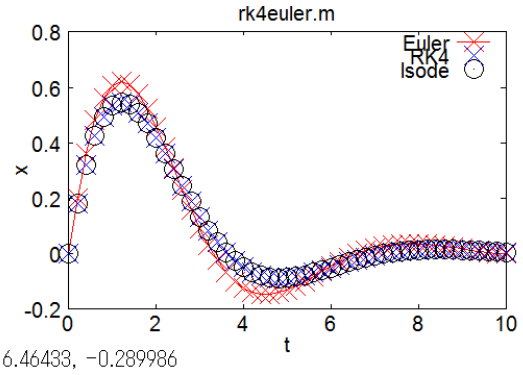

図 3.4.3

上の例題の場合は、オイラー法でも、RK 法でも **lsdoe** でも、t を大きくしていくと、誤差が次第に小さくなっていくように見える。これは、与えられた微分方程式が、初期値をどうとっても、$t \to \infty$のとき、$x \to 0$に収束していくことと関係がある。可視化したとき見たような、「初期値によって矢印の行き着く先が違う」ような解を持つ一般的な ODE の場合は、誤差が蓄積されて、次第に真の値からずれてしまうという点に注意してほしい。

オイラー法も、RK 法も、x_{k+1} を求めるのに、一つ前の x_k だけを使うという、一段解法だが、x_{k-1} や x_{k-2} といった、より過去の値を使って、x_{k+1} を求める方法もある。これを多段階法といい、**lsode** は、この多段階法を使っているそうだ。

3.5. 2 体問題

たとえば重力により引き合う二つの天体(物体)の運動は、解析的に解け、円や、楕円、双曲線で表すことができる。これまで学んだ ODE のアルゴリズムを使って、この 2 体問題、そして次に 3 体問題について考えてみよう。

重力で引き合う N 個の物体の運動方程式は、

$$m_j \frac{d^2\,\boldsymbol{x}_j}{dt^2} = -\sum_{j \neq i} G m_i m_j \frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{|\boldsymbol{x}_j - \boldsymbol{x}_i|^3}$$

である。いきなり 3 体以上を考えると難しいので、まずは 2 体で考える。簡単のため万有引力定数と質量を 1 とした単位系(GM=1)で、二つの天体(天体 0 と天体 1)の質量の差が大きいとする。重い方の天体 0 を原点に置いたとき、天体 1 の運動方

程式は、二階の二元連立 ODE となる。

$$\frac{d^2x}{dt^2} = -\frac{x}{\sqrt{x^2+y^2}^3}$$

$$\frac{d^2y}{dt^2} = -\frac{y}{\sqrt{x^2+y^2}^3}$$

二階の ODE を一階の二元連立 ODE に直したことを思い出せば、二階の二元連立 ODE を 4 元連立 ODE に書き直せば解けそうだということに理解してもらえるだろう。あからさまに書き下すと

$$dx/dt = v_x$$

$$dy/dt = v_y$$

$$\frac{dv_x}{dt} = -\frac{x}{\sqrt{x^2+y^2}^3}$$

$$\frac{dv_y}{dt} = -\frac{y}{\sqrt{x^2+y^2}^3}$$

という四つの式に書き直せる。これをプログラムの形にしてみよう。まずは、ODE を関数(xdot2body.m)にする。

```
% xdot2body.m
function xdot2 = xdot2body(x,t)
x1=x(3);    %dx/dt=vx のこと;
x2=x(4);    %dy/dt=vy のこと;
r=(x(1).^2+x(2)^2).^(3/2);
x3=-x(1)./r;
x4=-x(2)./r;
xdot2=[x1 x2 x3 x4];
```

位置(x,y) = (1,0)から、初速度(v_x,v_y) = (0,1)でスタートした場合、RK 法と `lsode` の両方で解くメインプログラムは、つぎのようになる。

```
% orbit.m
clear; t=[0:0.01:1]'*2*pi;
x0=[1 0 0 1];    %初期値(x0,y0,vx0,vy0)を定義
R1=rk4('xdot2body',x0,t);       %RK4
R2=lsode('xdot2body',x0,t);    %lsode
x=R1(:,1); y=R1(:,2);       %RK4
x2=R2(:,1); y2=R2(:,2);    %lsode
plot(0,0,'ko',x,y,'b*',x2,y2,'r');
xlabel('x'); ylabel('y');title('orbit.m');
legend('body0','RK4','lsode');
axis equal          %図の縦横比を等倍にする。
axis([-1 1 -1 1]); %図の表示範囲を指定する。
```

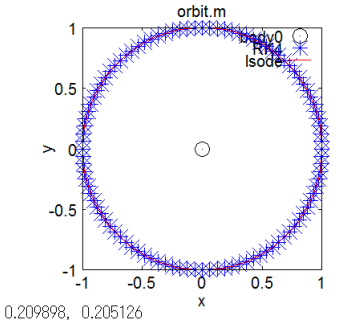

図 3.5.1

ODE の解は円軌道になる。初速度を少し遅くして、`x0=[1 0 0 0.7]`とすると、やや小さい楕円軌道になる。(orbit2.m)

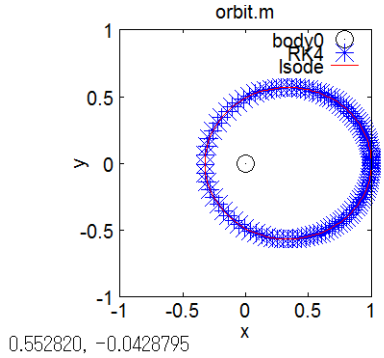

図 3.5.2

今、時間ベクトル `t` の刻みを等間隔にしているので、楕円の左端付近で*印が疎、つまり速度が速くなっていることが分かる。彗星が太陽に近づくとき速度が速くなることと同じだ。双曲線になる条件も各自調べてみてほしい。燃料を使わずに宇宙船の速度を加速減速させたり、軌道を変えたりするテクニックとして使われるスイングバイをシミュレーションしたり、彗星と太陽の距離を求めて彗星の尾を表示させたり、符号を逆にすれば、原子核とアルファ粒子のクーロン反発力によるラザフォード散乱など、いろいろなシミュレーションへ応用できて面白いかも。

3.6. 3 体問題

2 体に、もう一つの物体を加えて 3 体にしただけで、とたんに解が難しくなってしまうことが知られている。というのも、3 体問題は、一般的には厳密解が存在せず、摂動や数値計算でなければ解くことができないからだ。さらに、初期値によっては、3 体が衝突したり、無限遠方にはじき飛ばされたりする。安定して運動し続ける条件を決定する問題は、3 体問題(N 体問題と

もいう)と呼ばれ数学や物理で重要な問題だ。ちなみに、3 体問題でも円軌道で近似し、第三の物体の質量が無視できるほど小さいときは、解析的に解けることがオイラーとラグランジュによって示されている。この場合、安定した軌道を描くことができる場所(安定点)は 5 点あり、ラグランジュ点(L 点)として知られている。太陽と木星の場合では、トロヤ群という小惑星群が L 点に存在しているし、太陽系観測衛星(SOHO)は、太陽と地球のラグランジュ点に配置されている。NASA の次期宇宙望遠鏡も L_2 点に配置される予定だそうだ。

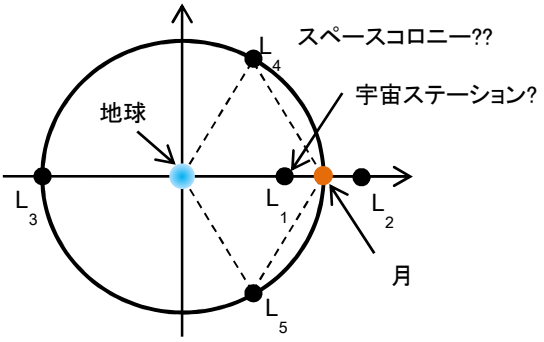

図 3.6.1 地球と月のラグランジュ点

太陽の質量が十分大きいとした 3 体問題を数値計算するプログラム自体は、8 元連立 ODE になるだけで、それほど難しくない。3 体問題の難しさ、そして面白さの本質は、初期値をどうとるかによって、結果が全く違ってくるという点にある。下に例を挙げよう。天体 0 (body0 例えば太陽)の位置を原点とする *xy* 座標系に、天体 1(body1)と天体 2(body2)を適当な初速度でおいて、軌道を計算した結果だ。

body1 を *xy* 座標表示で$(x,y)=(1,0)$におき、初速度$(v_x,v_y)=(0,1)$、body2 を$(-1,0)$、初速度 $(0,-1)$にした結果を図 3.6.2 に示す。(以降、初期条件を `x0=[1,0,0,1,-1,0,0,-1]` などと表記する。)

`x0=[1,0,0,1 -1,0,0,-1]`の場合(Nbody01.m)

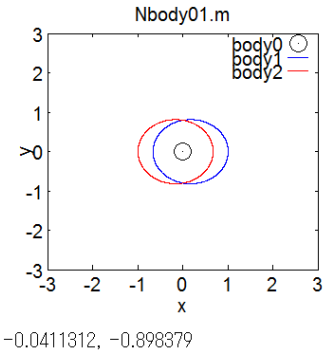

図 3.6.2

対称性があるので、body1, 2 が body0 に対して楕円軌道を描く安定解が得られる。次に body1 の初速度を 0.1%減らして対称性を崩してみるとどうなるだろう。

`x0=[1 0 0 0.999 -1 0 0 -1]`の場合(Nbody02.m)

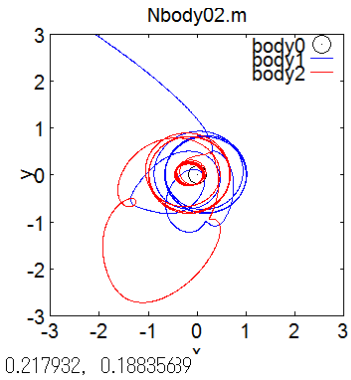

図 3.6.3

body1 と body2 は、相互に影響を及ぼしながら複雑な軌道を描き、最終的に body1 ははじき飛ばされている。

このように、初期値をわずかに変えると、全く違う振る舞いを示すところが、3 体問題の難しいところであり、研究者が興味を引かれるところでもある。

今回は重力を想定して話を進めたが、電磁気力やその他の力を使っても、(古典的ではあるが)ほとんど同じように扱うことができる。太陽系の惑星の軌道シミュレーションや、地球からロケットを打ち上げて、月を一周して地球に戻ってくるような NASA が実際に計算していそうな軌道計算など、いろいろな系を想定して計算してみてはどうだろうか。