Chapter 8

運動方程式の数値的解法

今回の内容

/// レポート回収

/// 前回の復習と補足

§8 運動方程式の数値解法

前回の復習

- 点変換
- 運動方程式の共変性
- 電磁場中の荷電粒子のラグランジアン $L = m\dot{\boldsymbol{x}}^2/2 + e\boldsymbol{A}\cdot\boldsymbol{x} - e\phi$

補足：一様磁場のベクトルポテンシャル

円筒座標での curl の公式

$$(\nabla\times\boldsymbol{A})_r = \frac{1}{r}\frac{\partial A_z}{\partial\phi} - \frac{\partial A_\phi}{\partial z} \tag{8.1}$$

$$(\nabla\times\boldsymbol{A})_\phi = \frac{\partial A_r}{\partial z} - \frac{\partial A_z}{\partial r} \tag{8.2}$$

$$(\nabla\times\boldsymbol{A})_z = \frac{1}{r}\frac{\partial}{\partial r}(rA_\phi) - \frac{1}{r}\frac{\partial A_r}{\partial\phi} \tag{8.3}$$

から、

$$\boldsymbol{A} = (A_r, A_\phi, A_z) = (0, rB_0/2, 0) \qquad (B_0 \text{ は定数})$$

ならば

$$\boldsymbol{B} = (B_r, B_\phi, B_z) = (0, 0, B_0)$$

注意１： 同じ $\boldsymbol{B}$ を持つ $\boldsymbol{A}$ は一つだけではない。任意のスカラー場 ψ に対して、$\boldsymbol{A}' = \boldsymbol{A} + \nabla\psi$ だけの不定性がある。

注意２： 同じ z 方向の一様磁場をカーテシアン座標で書くと、

$$\boldsymbol{B} = (B_x, B_y, B_z) = (0, 0, B_0)$$

これを導くベクトルポテンシャル（の一つ）はカーテシアン座標系では

$$\boldsymbol{A} = (A_x, A_y, A_z) = (0, xB_0, 0)$$

である。これも $\boldsymbol{A}' = \boldsymbol{A} + \nabla\psi$ だけの不定性がある。

レポート解答

$$L(r,\phi) = \frac{m}{2}\left(\dot{r}^2 + r^2\dot{\phi}^2\right) + \frac{eB_0}{2}r^2\dot{\phi} + \frac{eQ}{r} \tag{8.4}$$

なので、運動方程式は

$$\ddot{r} = r\dot{\phi}^2 + \frac{eB_0}{m}r\dot{\phi} - \frac{eQ}{m}\frac{1}{r^2} \tag{8.5}$$

$$\ddot{\phi} = -2\frac{\dot{r}}{r}\dot{\phi} - \frac{eB_0}{m}\frac{\dot{r}}{r} \tag{8.6}$$

である。

原点に置かれた電荷から引力を受けつつ、磁場中を運動する荷電粒子の方程式を導くことはできたが、この微分方程式系を解析的に解くのは大変である。そこで今日はこのような 2 階の微分方程式系を数値的に解く方法を学ぼう。

8.1 数値積分

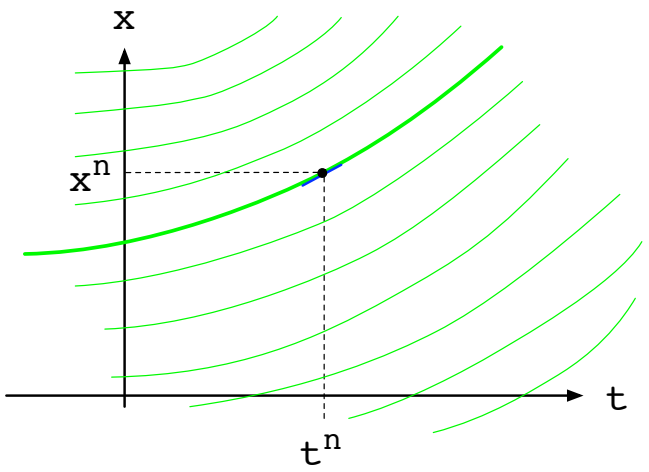
ラグランジュの運動方程式は時間の関数としての一般化座標に対する微分方程式である。ここでは数値積分法を使って、その運動方程式を解く方法について学ぶ。

まずは数値積分について簡単に復習しよう。t の関数 $x = x(t)$ に対する常微分方程式を考える。

$$\frac{dx}{dt} = f(x,t) \tag{8.7}$$

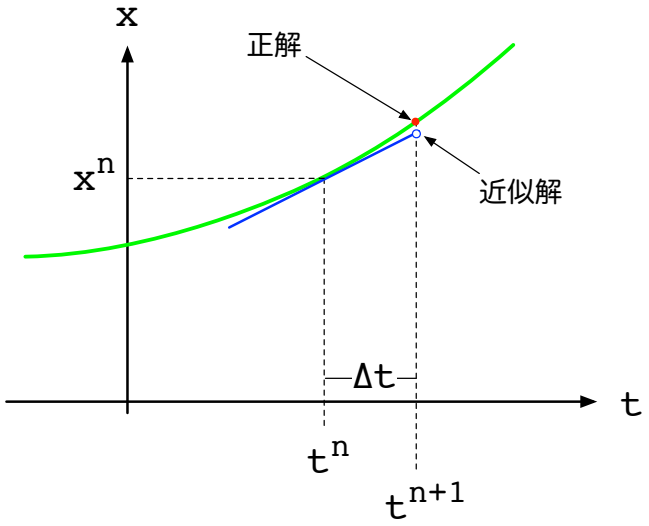
ある時刻 t^n での解 x^n は既知とする。時間が Δt だけ進んだ未来の時刻 $t^{n+1} := t^n + \Delta t$ での解 x^{n+1} を求める。

まず始めに、元の微分方程式 (8.7) のイメージを得ておこう。右辺の $f(x,t)$ は、2 次元 x-t 空間の場である。この微分方程式のある初期条件に対する一つ解は、この空間上のある曲線 $x = x(t)$ である。初期条件が異なれば異なる解をもつ。二つの異なる解の曲線が交わることはない。だから、この x-t 空間は、（異なる初期条件に対応する）無数の解曲線がびっしりと埋め尽くしている。各点での関数 $f(x,t)$ の値は、その位置を通る解曲線 $x = x(t)$ の接線の傾きを表している。



8.1.1 陽的一次オイラー法

さて、この微分方程式 (8.7) を数値的に解く（積分する）ことを考えよう。図



から

$$x^{n+1} = x^n + \left(\frac{dx}{dt}\right)^n \Delta t$$

と近似するのが（陽的）1 次オイラー法である。式 (8.7) より

$$\left(\frac{dx}{dt}\right)^n = f(x^n, t^n)$$

だから一次オイラー法は

$$x^{n+1} = x^n + f(x^n, t^n)\,\Delta t$$

と書ける。

図から明らかに $x = x(t)$ の曲線がまっすぐ（t の一次関数）でない限り、一次オイラー法には誤差がある。微分方程式 (8.7) を積分すると

$$[\,x(t)\,]_{t^n}^{t^{n+1}} = \int_{t^n}^{t^{n+1}} f(x,\tau)\,d\tau$$

つまり

$$x^{n+1} = x^n + \int_{t^n}^{t^{n+1}} f(x,\tau)\,d\tau$$

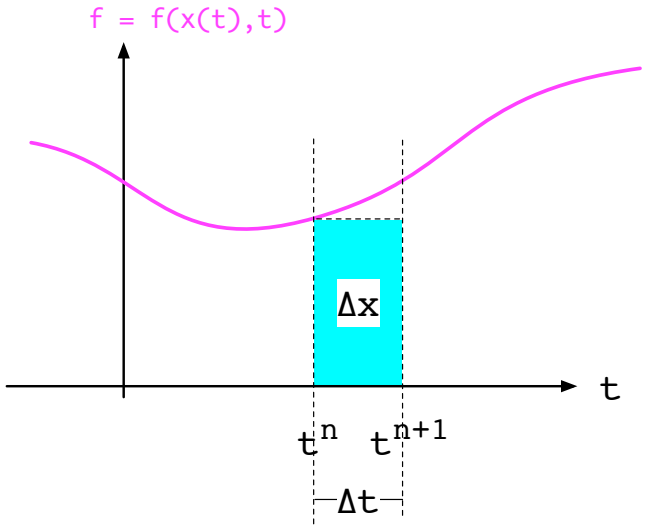
なので、

$$\Delta x := \int_{t^n}^{t^{n+1}} f(x,\tau)\,d\tau$$

を定義すると、一次オイラー法は

$$\Delta x = \int_{t^n}^{t^{n+1}} f(x,\tau)\,d\tau \sim f(x^n, t^n)\,\Delta t$$

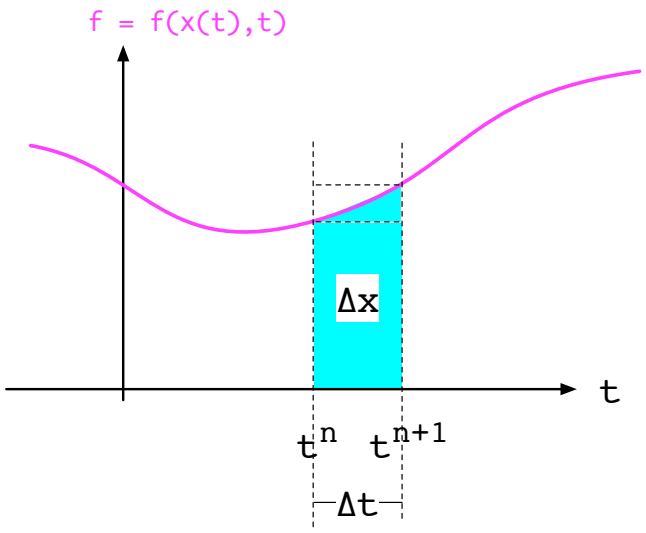
と近似していることを意味する。



8.1.2 2 次ルンゲ=クッタ法

積分

$$\Delta x = \int_{t^n}^{t^{n+1}} f(x,\tau)\,d\tau$$



の精度を上げるためには台形公式

$$\Delta x = \frac{1}{2}\left\{f(x^n, t^n)\,\Delta t + f(x^{n+1}, t^{n+1})\,\Delta t\right\}$$

を使えば良い。つまり

$$x^{n+1} = x^n + \frac{1}{2}\left\{f(x^n, t^n)\,\Delta t + f(x^{n+1}, t^{n+1})\,\Delta t\right\}$$

であるが、右辺の最後の項には、今求めようとしている x^{n+1} が入っているので、このままでは使えない。そこで、右辺の最後の項に出てくる x^{n+1} を、一次オイラー法を使って近似した値 x^* を使う。

$$x^* = x^n + f(x^n, t^n)\,\Delta t$$
$$x^{n+1} = x^n + \frac{1}{2}\left\{f(x^n, t^n)\,\Delta t + f(x^*, t^{n+1})\,\Delta t\right\}$$

これを 2 次ルンゲ=クッタ法という。

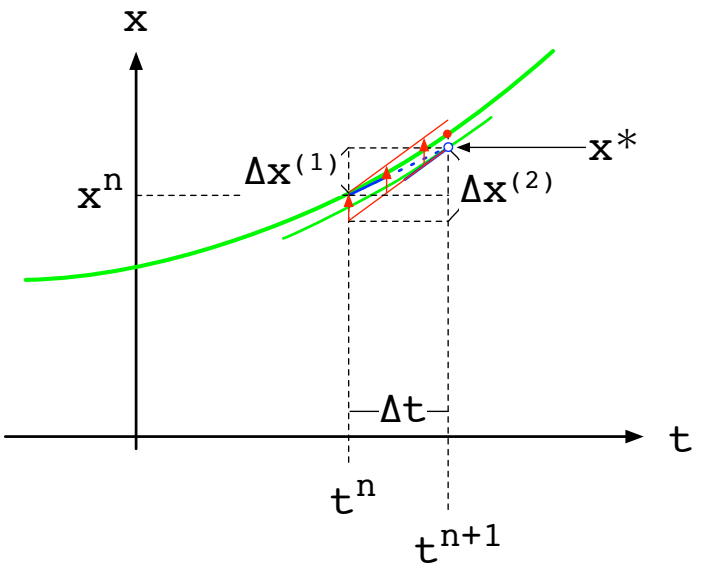
$$\Delta x^{(1)} := f(x^n, t^n)\,\Delta t$$

$$\Delta x^{(2)} := f(x^*, t^{n+1})\,\Delta t$$

と定義すると、

$$x^{n+1} = x^n + \frac{1}{2}\left(\Delta x^{(1)} + \Delta x^{(2)}\right)$$

と書ける。f のグラフではなく、解のグラフ $x(t)$ で描いた次のような図でも理解できる。



8.1.3 4 次ルンゲ=クッタ法

2 次のルンゲ=クッタ法よりもさらに精度を上げるためには、Δx の積分を計算するのに、台形公式ではなく、シンプソンの公式を使えばよい。そうして得られる数値積分法が 4 次のルンゲ=クッタ法である。

上に述べた 1 次オイラー法や 2 次ルンゲ=クッタ法は、

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

という微分方程式を既知量 x^n から未知量 x^{n+1} を求めるために

$$x^{n+1} = x^n + \Delta x$$

という計算をする。ここで増分 Δx は

$$\Delta x = \sum_{k=1}^{K} w^{(k)}\,\Delta x^{(k)} \qquad (w^{(k)} \text{ は定数})$$

という線形和である。1 次オイラー法の場合には $K=1$ で

$$\Delta x^{(1)} = f(x^n, t^n)\,\Delta t$$

であり、2 次ルンゲ=クッタ法では $K=2$ で

$$\Delta x^{(1)} = f(x^n, t^n)\,\Delta t$$

$$\Delta x^{(2)} = f(x^*, t^{n+1})\,\Delta t$$

であった。

4 次ルンゲ=クッタ法も手続きは同様である。この場合 $K = 4$ で、

$$\Delta x^{(k)} = f(x^\dagger, t^\dagger)\,\Delta t \qquad (x^n \le x^\dagger, t^n \le t^\dagger)$$

という 4 つの増分の線形和で x^{n+1} を近似する。4 次ルンゲ=クッタ法の詳しい式の導出法は省略することにして、最終的な形だけ下に書く。

式 (8.7) を解くのに、初期条件を

$$x(t_0) = x_0 \tag{8.8}$$

として、

$$x_{n+1} = x_n + \frac{\Delta t}{6}k_1 + \frac{\Delta t}{3}k_2 + \frac{\Delta t}{3}k_3 + \frac{\Delta t}{6}k_4 \tag{8.9}$$

が 4 次ルンゲ=クッタ法の公式（の一つ）である。ここで

$$k_1 = f(x_n, t_n) \tag{8.10}$$

$$k_2 = f(x_n + \frac{\Delta t}{2}k_1, t_n + \frac{\Delta t}{2}) \tag{8.11}$$

$$k_3 = f(x_n + \frac{\Delta t}{2}k_2, t_n + \frac{\Delta t}{2}) \tag{8.12}$$

$$k_4 = f(x_n + \Delta t k_3, t_n + \Delta t) \tag{8.13}$$

である。これは 2 次のルンゲ=クッタ法と 1 次のオイラー法を組み合わせたものであり、グラフを使った図形的な解釈もできる。

解くべき微分方程式が

$$\frac{dx(t)}{dt} = f(x) \tag{8.14}$$

というタイプの場合、つまり右辺が陽には t に依存しない場合、4 次ルンゲ=クッタ法の公式は

$$x(t_0) = x_0 \tag{8.15}$$

として、

$$x_{n+1} = x_n + \frac{\Delta t}{6}k_1 + \frac{\Delta t}{3}k_2 + \frac{\Delta t}{3}k_3 + \frac{\Delta t}{6}k_4 \tag{8.16}$$

である。ここで

$$k_1 = f(x_n) \tag{8.17}$$

$$k_2 = f(x_n + \frac{\Delta t}{2}k_1) \tag{8.18}$$

$$k_3 = f(x_n + \frac{\Delta t}{2}k_2) \tag{8.19}$$

$$k_4 = f(x_n + \Delta t k_3) \tag{8.20}$$

8.1.4 連立微分方程式系

解くべき微分方程式が連立している場合も特に問題ない。たとえば t の関数 $\alpha = \alpha(t)$ と $\beta = \beta(t)$ が結合している系

$$\frac{d\alpha}{dt} = f(\alpha, \beta, t)$$
$$\frac{d\beta}{dt} = g(\alpha, \beta, t)$$

をルンゲ=クッタ法で数値積分するには、

$$\Delta\alpha^{(k)} = f(\alpha^\dagger, \beta^\dagger, t^\dagger)\,\Delta t$$
$$\Delta\beta^{(k)} = g(\alpha^\dagger, \beta^\dagger, t^\dagger)\,\Delta t$$

というそれぞれの変数の増分を計算し、その線形和をとればよい。

8.2 ラグランジュの運動方程式の数値積分

8.2.1 2 階微分方程式系の一階微分方程式系への変換

ラグランジアン $L(q_1, \ldots, q_N, \dot{q}_1, \ldots, \dot{q}_N)$ からラグランジュの運動方程式を作ると、q_i の 2 階微分が出てくる。例えば今日回収したレポート課題では、2 階の連立微分方程式 (8.5) と (8.6)

$$\ddot{r} = r\dot{\phi}^2 + \frac{eB_0}{m}r\dot{\phi} - \frac{eQ}{m}\frac{1}{r^2}$$
$$\ddot{\phi} = -2\frac{\dot{r}}{r}\dot{\phi} - \frac{eB_0}{m}\frac{\dot{r}}{r}$$

が得られた。このような 2 階の微分方程式を 4 次ルンゲ=クッタ法などを使って数値積分するのは簡単である。変数を増やして一階の微分方程式にすればよい。たとえば今の場合、

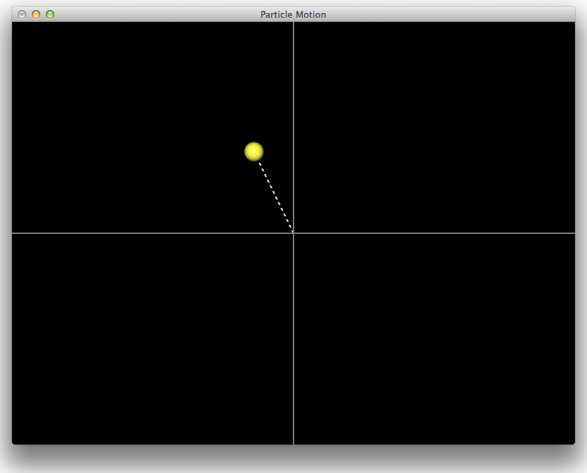
$$v_r = \dot{r}$$
$$v_\phi = \dot{\phi}$$

という新しい変数を定義すると、解くべき微分方程式系は

$$\dot{r} = v_r \tag{8.21}$$
$$\dot{v_r} = r v_\phi^2 + \frac{eB_0}{m} r v_\phi - \frac{eQ}{m}\frac{1}{r^2} \tag{8.22}$$
$$\dot{\phi} = v_\phi \tag{8.23}$$
$$\dot{v_\phi} = -2\frac{v_r}{r} v_\phi - \frac{eB_0}{m}\frac{v_r}{r} \tag{8.24}$$

となり一階微分しかでてこなくなった。4 つの変数 (r, v_r, ϕ, v_ϕ) に対する一階の微分方程式系なので、ルンゲ=クッタ法などで簡単に解くことができる。

8.2.2 実例



4 次のルンゲ=クッタ法で上の式 (8.21) から (8.24) を解くプログラムを作った。質点（荷電粒子）の構造体は以下のように定義した：

Listing 8.1: 荷電質点用構造体

```
1
2 struct particle_ {
3     double mass;   // mass of the particle
4     double charge;   // electric charge
```

```
5       double pos[4]; // {r,v_r,phi,v_phi}
6       GLfloat color_diffuse[4];
7       GLdouble sphere_radius;
8       GLint    sphere_slices, sphere_stacks;
9   };
```

4 次ルンゲ＝クッタ法を使っているのは次の部分である：

Listing 8.2: 4 次ルンゲ＝クッタ法

```
1   void runge_kutta4(struct particle_ *particle, double dt)
2   {
3           const double ONE_SIXTH = 1.0/6.0;
4           const double ONE_THIRD = 1.0/3.0;
5
6           double pos_before[4], work_pos[4];
7           double dpos1[4], dpos2[4], dpos3[4], dpos4[4];
8
9           pos_before[0] = particle->pos[0];
10          pos_before[1] = particle->pos[1];
11          pos_before[2] = particle->pos[2];
12          pos_before[3] = particle->pos[3];
13
14          double   mass = particle->mass;
15          double charge = particle->charge;
16
17          //step 1
18          equation_of_motion(mass, charge, pos_before, dpos1, dt);
19          runge_kutta4_advance(work_pos, pos_before, dpos1, 0.5);
20          //step 2
21          equation_of_motion(mass, charge, work_pos, dpos2, dt);
22          runge_kutta4_advance(work_pos, pos_before, dpos2, 0.5);
23          //step 3
24          equation_of_motion(mass, charge, work_pos, dpos3, dt);
25          runge_kutta4_advance(work_pos, pos_before, dpos3, 1.0);
26          //step 4
27          equation_of_motion(mass, charge, work_pos, dpos4, dt);
28
29          //the result
30          particle->pos[0] = pos_before[0] + (  ONE_SIXTH*dpos1[0]
31                                              + ONE_THIRD*dpos2[0]
32                                              + ONE_THIRD*dpos3[0]
33                                              + ONE_SIXTH*dpos4[0] );
34          particle->pos[1] = pos_before[1] + (  ONE_SIXTH*dpos1[1]
35                                              + ONE_THIRD*dpos2[1]
36                                              + ONE_THIRD*dpos3[1]
37                                              + ONE_SIXTH*dpos4[1] );
```

```
38          particle->pos[2] = pos_before[2] + (   ONE_SIXTH*dpos1[2]
39                                               + ONE_THIRD*dpos2[2]
40                                               + ONE_THIRD*dpos3[2]
41                                               + ONE_SIXTH*dpos4[2]  );
42          particle->pos[3] = pos_before[3] + (   ONE_SIXTH*dpos1[3]
43                                               + ONE_THIRD*dpos2[3]
44                                               + ONE_THIRD*dpos3[3]
45                                               + ONE_SIXTH*dpos4[3]  );
46          std::cout << "total energy: "
47                          << std::scientific
48                          << total_energy(mass,charge,particle->pos)
49                          << std::endl;
50  }
```

運動方程式 (8.21) から (8.24) は以下のようにコーディングされている：なお、以下のプログラムで CENTER_CHARGE_Q の値は、符号付き（実際には負の値）で定義しているため、式 (8.22) とは中心電荷 Q が現れる項の符号が逆になっているので注意。

Listing 8.3: 運動方程式

```
1
2  void equation_of_motion(double mass, double charge,
3                          double *pos, double *dpos, double dt)
4  {
5      const double B = 0.01; // Magnetic field
6
7      double r  = pos[0];
8      double vr = pos[1];
9      double p  = pos[2];
10     double vp = pos[3];
11
12     double  rvp = r*vp;
13     double vr_r = vr/r;
14
15     double  eB_m = B*charge/mass;
16     double  eQ_m = CENTER_CHARGE_Q*charge/mass;
17
18     dpos[0] = ( vr ) * dt;
19     dpos[1] = (  rvp*vp + eQ_m / (r*r)
20                + eB_m*rvp ) * dt;
21     dpos[2] = ( vp ) * dt;
22     dpos[3] = ( -2*vr_r*vp - eB_m*vr_r) * dt;
23 }
```

8.2.3 練習問題

これは、この講義の冒頭でとりあげた問題である。二つの放物線

$$y = x^2 + 1 \quad \text{（上側）}$$

と

$$y = -x^2 - 1 \quad \text{（下側）}$$

に二つの質点（質量 m）がそれぞれ放物線上に拘束されて動くとする。質点同士はバネ（バネ定数 k、自然長 ℓ_0）で結ばれている。摩擦は無視する。この質点の運動を解くためのプログラムを作ろう。

解答例

上と下の放物線を動く質点の座標をそれぞれ (x_1, y_1) と (x_2, y_2) とする。上の質点の速度を v_1 とすると、

$$v_1^2 = \left(\frac{dx_1}{dt}\right)^2 + \left(\frac{dy_1}{dt}\right)^2 = \dot{x}_1^2 + \left(\frac{dy_1}{dx_1}\right)^2 \dot{x}_1^2 = (4x_1^2 + 1)\dot{x}_1^2$$

2 つの質点間の距離を s とすると

$$s = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} = \sqrt{(x_1 - x_2)^2 + (x_1^2 + x_2^2 + 2)^2}$$

従ってラグランジアンは

$$L(x_1, x_2, \dot{x}_1, \dot{x}_2) = \frac{m}{2}\left\{(4x_1^2 + 1)\dot{x}_1^2 + (4x_2^2 + 1)\dot{x}_2^2\right\} - \frac{k}{2}(s - s_0)^2$$

である。ここではバネの自然長を s_0 とした。

ラグランジュの運動方程式を導出しよう。

$$\frac{\partial L}{\partial \dot{x}_1} = m(4x_1^2 + 1)\dot{x}_1$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_1}\right) = m(4x_1^2 + 1)\ddot{x}_1 + 8mx_1\dot{x}_1^2$$

$$\frac{\partial L}{\partial x_1} = 4mx_1\dot{x}_1^2 - k\frac{s - s_0}{s}\left\{(x_1 - x_2) + 2x_1(x_1^2 + x_2^2 + 2)\right\}$$

より上の放物線を動く質点の運動方程式は

$$m(4x_1^2 + 1)\ddot{x}_1 + 8mx_1\dot{x}_1^2 = 4mx_1\dot{x}_1^2 - k\frac{s - s_0}{s}(\Delta x + 2x_1\Delta y)$$

ここで、プログラミング上の便宜を考え、

$$\Delta x = x_1 - x_2$$

$$\Delta y = y_1 - y_2$$

を導入した。この式を書き直すと、

$$\ddot{x}_1 = -\frac{1}{4x_1^2+1}\left\{4x_1\dot{x}_1^2 + \frac{k}{m}\frac{s-s_0}{s}\left(\Delta x + 2x_1\Delta y\right)\right\}$$

が上の放物線を動く質点の運動方程式である。同様に下の質点の運動方程式は、

$$\ddot{x}_2 = -\frac{1}{4x_2^2+1}\left\{4x_2\dot{x}_2^2 + \frac{k}{m}\frac{s-s_0}{s}\left(-\Delta x + 2x_2\Delta y\right)\right\}$$

運動方程式の実装例は以下の通り。

Listing 8.4: two_balls_parabola_rev_equation

```
1  void equation_of_motion(double *pos, double *dpos, double dt)
2  {
3    //     Lagrangian
4    //        L(x1,x2,x1',x2') = (m/2)*(x1'^2+4*x1^2*x1'^2)
5    //                         + (m/2)*(x2'^2+4*x2^2*x2'^2)
6    //                         - (k/2)*(s-S0)^2
7    //     where
8    //          s = sqrt(dx^2+dy^2), dx=x1-x2, dy=x1^2+x2^2+2
9    //
10     double x1     = pos[0];
11     double x1_dot = pos[1];
12     double x2     = pos[2];
13     double x2_dot = pos[3];
14
15     double dx   = x1 - x2;
16     double x1sq = x1*x1, x1_dot_sq = x1_dot*x1_dot;
17     double x2sq = x2*x2, x2_dot_sq = x2_dot*x2_dot;
18     double dy   = x1sq + x2sq + 2;
19     double s    = sqrt(dx*dx + dy*dy);
20
21     double f1 = (K/M)*(s-S0)/s*( dx+2*x1*dy);
22     double f2 = (K/M)*(s-S0)/s*(-dx+2*x2*dy);
23
24     dpos[0] = ( x1_dot ) * dt;
25     dpos[1] = ( -1.0/(1+4*x1sq)*(4*x1*x1_dot_sq + f1) ) * dt;
26     dpos[2] = ( x2_dot ) * dt;
27     dpos[3] = ( -1.0/(1+4*x2sq)*(4*x2*x2_dot_sq + f2) ) * dt;
28 }
```

ルンゲ=クッタ法による積分ルーチンの実装例。質量と電荷を渡す部分以外、本質的には何も変えてないことに注目せよ。

Listing 8.5: two_balls_parabola_rev_rk4

```
1  void runge_kutta4(struct particle_ *particle, double dt)
2  {
3          const double ONE_SIXTH = 1.0/6.0;
4          const double ONE_THIRD = 1.0/3.0;
5
6          double pos_before[4], work_pos[4];
7          double dpos1[4], dpos2[4], dpos3[4], dpos4[4];
8
9          pos_before[0] = particle->pos[0];
10         pos_before[1] = particle->pos[1];
11         pos_before[2] = particle->pos[2];
12         pos_before[3] = particle->pos[3];
13
14         //step 1
15         equation_of_motion(pos_before, dpos1, dt);
16         runge_kutta4_advance(work_pos, pos_before, dpos1, 0.5);
17         //step 2
18         equation_of_motion(work_pos, dpos2, dt);
19         runge_kutta4_advance(work_pos, pos_before, dpos2, 0.5);
20         //step 3
21         equation_of_motion(work_pos, dpos3, dt);
22         runge_kutta4_advance(work_pos, pos_before, dpos3, 1.0);
23         //step 4
24         equation_of_motion(work_pos, dpos4, dt);
25
26         //the result
27         particle->pos[0] = pos_before[0] + (  ONE_SIXTH*dpos1[0]
28                                             + ONE_THIRD*dpos2[0]
29                                             + ONE_THIRD*dpos3[0]
30                                             + ONE_SIXTH*dpos4[0] );
31         particle->pos[1] = pos_before[1] + (  ONE_SIXTH*dpos1[1]
32                                             + ONE_THIRD*dpos2[1]
33                                             + ONE_THIRD*dpos3[1]
34                                             + ONE_SIXTH*dpos4[1] );
35         particle->pos[2] = pos_before[2] + (  ONE_SIXTH*dpos1[2]
36                                             + ONE_THIRD*dpos2[2]
37                                             + ONE_THIRD*dpos3[2]
38                                             + ONE_SIXTH*dpos4[2] );
39         particle->pos[3] = pos_before[3] + (  ONE_SIXTH*dpos1[3]
40                                             + ONE_THIRD*dpos2[3]
41                                             + ONE_THIRD*dpos3[3]
42                                             + ONE_SIXTH*dpos4[3] );
```

```
43
44          std::cout << "total energy: "
45                            << std::scientific
46                            << total_energy(particle->pos)
47                            << std::endl;
48  }
```