ばね質点モデル

目的

ばね質点モデル (Mass-Spring Model) による変形のシミュレーションを行うプログラムを作成します。これにより数式で示されたモデルをプログラムに置き換える感覚を養います。

1. 点列を折れ線で結んで表示するプログラムを用意しています。このプログラムは点をマウスの左ボタンでドラッグして動かすことができます。
2. 点を重さのある「質点」とし、点と点を結ぶ線分を「ばね」だと考えます。点を動かすと、ばねが伸び縮みします。このとき、ばねの両端の質点に力が加わります。
3. この力と、質点に設定した質量、および重力加速度を使って、質点の位置と更新します。これによって、折れ線がどのような動きをするのかを観察します。
4. 折れ線を縦横に繋いで網を作り、同様の考え方で網または布のシミュレーションをします。

教材

http://marina.sys.wakayama-u.ac.jp/~tokoi/seminar1/spring.zip

- このプログラムは Windows (Visual Studio 2017 以降)、Mac OS X (Xcode 10 以降)、および Linux (ディストリビューションに依存) でコンパイル、実行できます。

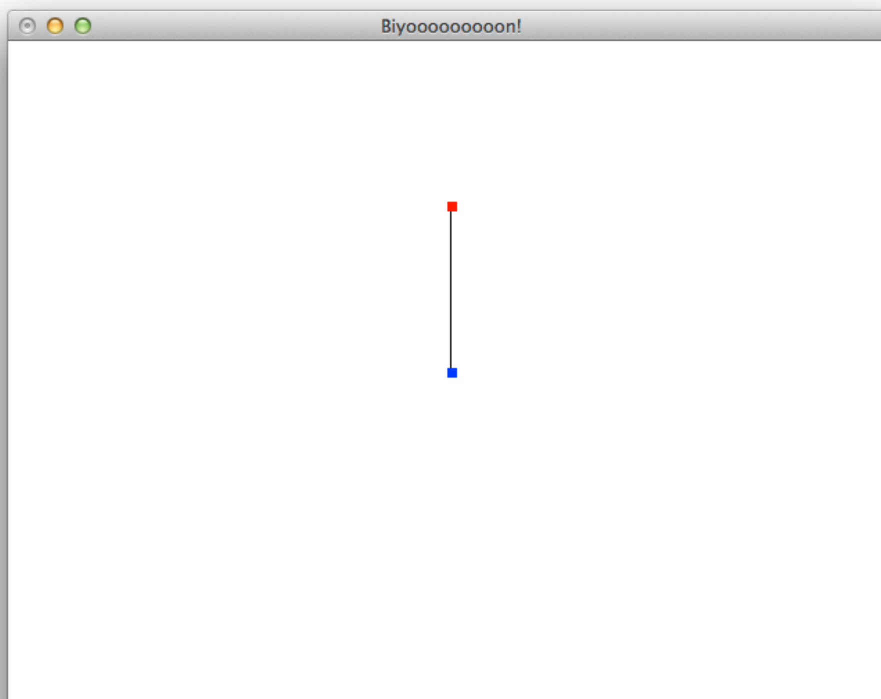
課題 1

1. 教材に含まれる、draw.cpp というソースプログラムを変更します。そのほかのプログラムは補助的なものです。 draw.cpp は以下の内容になっています。

```
//
// 折れ線を描く
//
//      n: 質点の数 (線分の数 + 1)
//      p: 質点の位置 (二次元)
//
extern void polyline(int n, double (*p)[2]);

//
// 時間間隔
//
const double dt = (1.0 / 60.0);

// 質点の初期位置
double position[][2] =
{
  { 0.0, 0.0 },
  { 0.0, 50.0 }
};
```

```
// 質点の数
const int points = sizeof position / sizeof position[0];

// 描画
void draw(void)
{
  polyline(points, position);
}
```

2. これを実行すると、次のような表示が得られます。



3. 関数 `draw()` に、図形を描画する手続きを記述します。この関数は繰り返し実行されます。関数 `polyline()` はあらかじめ用意しています。

4. 関数 `polyline()` は、第一引数 n に指定した数の点を結ぶ折れ線を描きます。各点の座標値は第二引数 p で指定します。 p は 2 要素の配列 n 個からなる二次元配列です。

5. 点は p に指定した配列変数の要素の順に線分で結ばれます。例えば、(0, 0), (10, 20), (30, 40) の三点をこの順に結ぶ折れ線を描くには、p に次のような配列を指定します。これは二次元の表のようなデータで、行は頂点の番号、列は x, y の座標値を表します。

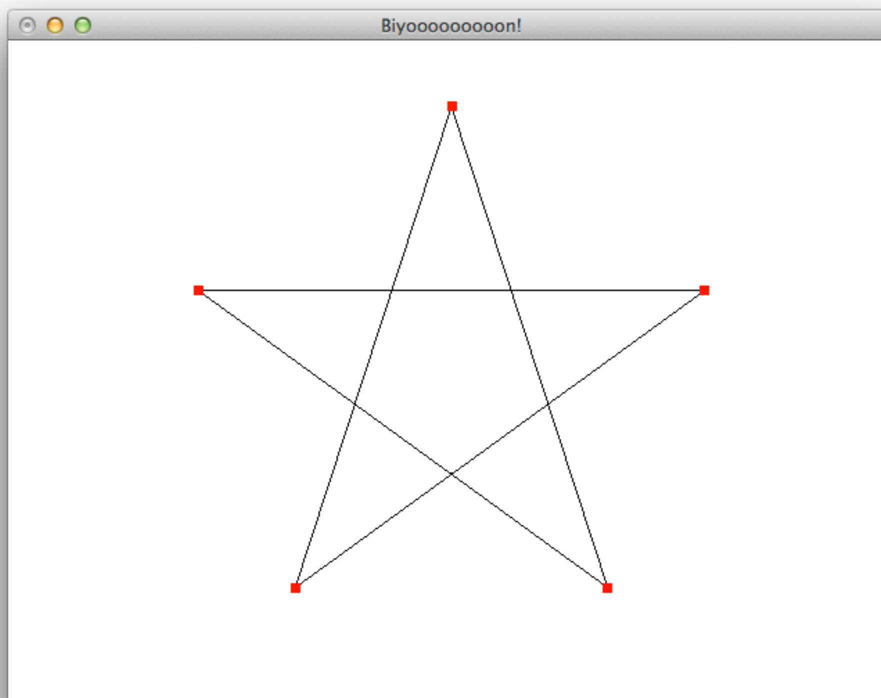
```
// 質点の初期位置
double position[][2] =
{
  {  0.0,  0.0 },
  { 10.0, 20.0 },
  { 30.0, 40.0 }
};
```

	0	1
0	0.0	0.0
1	10.0	20.0
2	30.0	40.0

6. sizeof は変数やデータ型のサイズを求める単項演算子です。sizeof position により配列変数 point の全データのサイズを得ます。また、sizeof position[0] は point の一つの要素のサイズを得ます。したがって、sizeof position / sizeof position[0] により、point の要素数を求めることができます。したがって、次の変数宣言によって points を配列変数 position の要素数で初期化することができます。

```
// 質点の数
const int points = sizeof position / sizeof position[0];
```

7. それでは、このプログラムで次のような「星」を描いてください。



課題2

1. アニメーションを行うには定数 dt で与えられる時間間隔 dt により配列変数 position の値を変化させます。ただし dt は前回表示を行ってからの経過時間なので、前回の状態をもとにして現在の状態を求める必要があります。一秒間に一回転するなら、現在の状態 $\mathbf{p} = (x, y)$ に対する次の状態 $\mathbf{p}' = (x', y')$ を、次式により求めます。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos 2\pi dt & \sin 2\pi dt \\ -\sin 2\pi dt & \cos 2\pi dt \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

2. それでは、この星を一秒間に一回転の速度で回転させてください。$t = 2\pi dt$ をあらかじめ計算しておけばプログラムが単純になります。

課題3

3. それでは、前の課題で追加した回転のプログラムを一端削除してください（星の頂点のデータは後で使いますので、残しておいてください）。ここでは六つの頂点のうち、最初の二点 $\mathbf{p}_0$ と $\mathbf{p}_1$、すなわち `position[0]` と `position[1]` のみについて考えます。

4. ばねに力をかけない状態の長さを、ばねの自然長と呼ぶことにします。この長さを l_0 とします。l_0 はとりあえず 30 にします。

```
// ばねの自然長
const double l0 = 30.0;
```

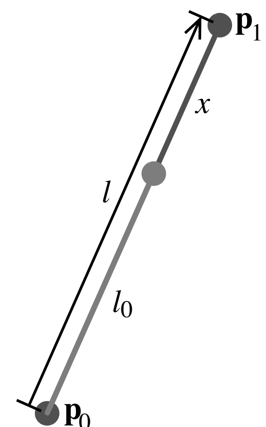
5. ばねの強さは、ばね定数で決まります。これを k とします。k はとりあえず 300 にします。

```
// ばね定数
const double k = 300.0;
```

6. すべての質点の質量を m とします。m はとりあえず 1 にします。

```
// 質点の質量
const double m = 1.0;
```

7. ばねの両端の質点の位置をそれぞれ $\mathbf{p}_0$, $\mathbf{p}_1$ とします。$\mathbf{p}_0$ は、プログラム上では配列変数の要素 `position[0]` に、$\mathbf{p}_1$ は `position[1]` に相当する二次元のベクトルです。このとき、ばねの長さ l は次式で求めることができます。



$$l = |\mathbf{p}_1 - \mathbf{p}_0|$$

8. $\mathbf{p}_0 = (x_0, y_0)$、$\mathbf{p}_1 = (x_1, y_1)$ とすれば、$\mathbf{p}_1 - \mathbf{p}_0 = (x_1 - x_0, y_1 - y_0)$ ですから、$(x_1 - x_0, y_1 - y_0) = (dx, dy)$ を次のようにして求めます。

```
// 二点の変位
const double dx = position[1][0] - position[0][0];
const double dy = position[1][1] - position[0][1];
```

なお、$\sqrt{dx^2 + dy^2}$ は `sqrt(dx * dx + dy * dy)` または `hypot(dx, dy)` で計算できます。

9. ばねの力 f_s は、ばねの伸びの量を $x = l - l_0$ とすると、フックの法則により、次式で求めることができます。

$$f_s = -k\,x$$

10. この力が、ばねが伸び縮みする方向 $\mathbf{e}$ に加わります。ばねの一方の端 $\mathbf{p}_0$ を固定し、もう一方の端 $\mathbf{p}_1$ を動かしたとすれば、ばねが伸び縮みする方向 $\mathbf{e}$ は次式で求めることができます。$\mathbf{e}$ も二次元のベクトルだということに気をつけてください。

$$\mathbf{e} = \frac{\mathbf{p}_1 - \mathbf{p}_0}{l}$$

11. したがって、$\mathbf{p}_1$ の質点に加わる力のベクトル $\mathbf{f}$ は次式で得られます。

$$\mathbf{f} = f_s\mathbf{e}$$

12. このとき、この力を受ける $\mathbf{p}_1$ の質点の加速度 $\mathbf{a}$ は、その質量を m として、運動方程式 $\mathbf{f} = m\mathbf{a}$ より次式で求めることができます。

$$\mathbf{a} = \frac{\mathbf{f}}{m}$$

13. この加速度で動く質点の時間間隔 dt 秒後の速度 $\mathbf{v}$ は、初速度を $\mathbf{v}_1$ とすると、次式で得られます。

$$\mathbf{v} = \mathbf{v}_1 + \mathbf{a}dt$$

14. この $\mathbf{p}_1$ の初速度 $\mathbf{v}_1$ は、質点の数 `points` を要素数とする次の配列変数 `velocity` の要素 `velocity[1]` に格納されているものとし、その初期値(初速度)は 0 とします。

```
// 質点の速度
static double velocity[points][2];
```

15. さらに、この質点の経過時間 `dt` 秒後の位置 $\mathbf{p}$ は、現在の位置 $\mathbf{p}_1$ から次の位置に移動します。

$$\mathbf{p} = \mathbf{p}_1 + \mathbf{v}dt$$

16. したがって、得られたこれらの速度や位置で初速度や現在の位置を更新すれば、質点が動くアニメーションになります。

$$\mathbf{p}_1 \leftarrow \mathbf{p}$$
$$\mathbf{v}_1 \leftarrow \mathbf{v}$$

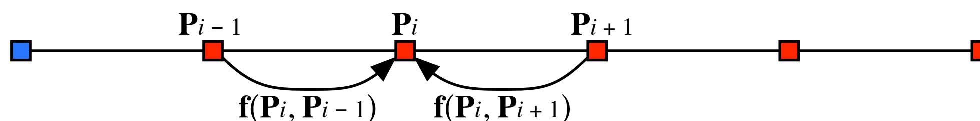
17. $\mathbf{p}_0$、$\mathbf{p}_1$ を結ぶ線分についてこの力を計算し、それにより $\mathbf{p}_1$、すなわち `position[1]` の位置を更新して、一個の質点が動くアニメーションを作ってください。更新した速度は質点の数 `points` を要素数とする配列変数 `velocity` の対応する要素に格納します。

課題4

1. このプログラムは、点を右ボタンでクリックすれば、点の色が[赤][青]に交互に切り替わります。最初は一つ目の点だけが[青]、残りは全部[赤]になっています。
2. 最初に `polyline()` を呼び出したときは、引数 `p` に与えたすべての点の位置が、表示する点の位置に反映されます。その後も[赤]の点の位置は `polyline()` を呼び出すごとに `p` の値が反映されます。これに対して、[青]の点は現在位置あるいはマウスで設定した位置から移動しません。したがって、[青]の点の位置は固定されます。

3. この折れ線の点は、最初の点 `position[0]` と最後の点 `position[points - 1]` 以外は二つの線分で共有されています。したがって、線分をばねだと考えれば、これらの点に加わる力は二つのばねの力を合成したものになります。

4. このことをもとに、折れ線の各点に加わる力を求め、すべての点が動くようにしてください。下の例では、i 番目の点に加わる力は、i 番目と $i-1$ 番目の点を結ぶばねの力と、i 番目と $i+1$ 番目の点を結ぶばねの力を合計したものになります。



5. このとき、任意の 2 点間を結ぶばねの力を求める手続きを関数にまとめておくと、プログラムが簡単になります。

```
// p0, p1 間を結ぶばねの力を f に求める
void force(double *f, const double *p0, const double *p1)
{
  // ここに p0, p1 間を結ぶばねの力を求めて f に格納するプログラムを書く
}
```

6. なお、`position[0]` の点は `position[1]` の点との間にあるばねの力だけを受けるものとし、`position[points - 1]` の点は `position[points - 2]` の点との間にあるばねの力だけを受けるものとします。

7. プログラムを動かして、もし、点をマウスでドラックできそうだったら、ドラッグして点の位置を変えてみてください。

課題5

1. 重力がある場合、この点の加速度 $\mathbf{a}$ は、ばねの力による加速度と重力加速度 $\mathbf{g}$ とを合成したものになります。

$$\mathbf{a} = \frac{\mathbf{f}}{m} + \mathbf{g}$$

2. 重力加速度を $\mathbf{g} = (0, -980)$ として、このプログラムに重力の影響を追加してください。

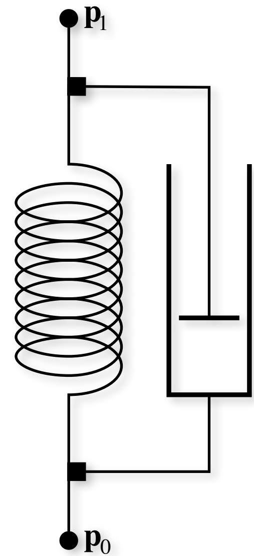
```
// 重力加速度
const double g[] = { 0.0, -980.0 };
```

課題6

1. 現実のばねは、無限に動き続けることはありません。そこで、このばねを、ばねとばねの動きを抑えるダンパ(ショックアブゾーバ)で構成されていると考えます。

2. このダンパによる抵抗力は、質点間の相対速度に比例するものとします。$\mathbf{p}_0$ の質点の速度を $\mathbf{v}_0$、$\mathbf{p}_1$ の質点の速度を $\mathbf{v}_1$、ダンパの減衰係数を $c > 0$ とすると、ばねの力 $\mathbf{f}$ は次のようになります。

$$\mathbf{f} = m\mathbf{a} + c\left(\mathbf{v}_1 - \mathbf{v}_0\right)$$



3. したがって $\mathbf{p}_1$ の質点の加速度 $\mathbf{a}$ は、次式で求められます。

$$\mathbf{a} = \frac{\mathbf{f} - c\left(\mathbf{v}_1 - \mathbf{v}_0\right)}{m}$$

4. これにより 2 点間にかかる力は、その 2 点を結ぶばねによる力 $\mathbf{f}$ に対して、$\mathbf{f} - c(\mathbf{v}_1 - \mathbf{v}_0)$ となります。そこで減衰係数を $c = 5$ として、このダンパによる効果と、重力の影響を追加してください。

```
// 減衰係数
const double c = 5.0;
```

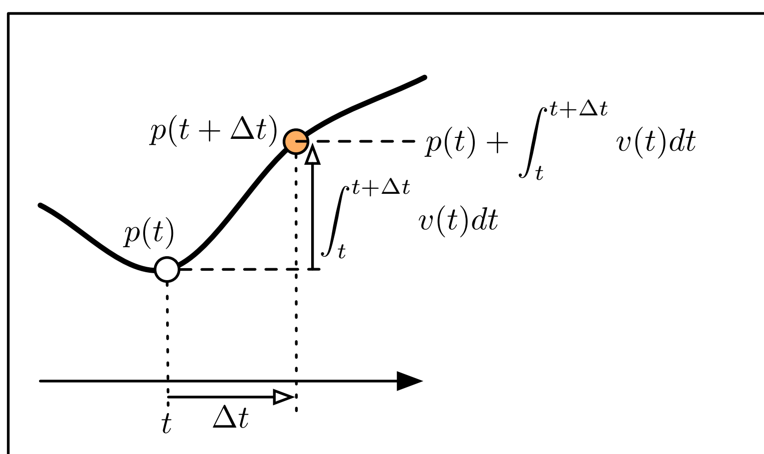
微分方程式の数値解法

微分方程式は自然現象など、様々な現象をモデル化するために使われます。現象は、例えば投げたボールの軌跡が放物線になるように、関数として記述できることもあります。しかし、そのような関数が未知の場合もよくあります。そこで、未知の関数とその微分、すなわち導関数の関係によって現象を記述しようとするのが微分方程式です。

関数の形が簡単には決められない複雑な現象であっても、「今」の状態、すなわち現象の過去からの積み重ねの結果と、これからどうなるのか、どっちに向かうのかという「性質」は決めることができます。

例えば、現在時刻 t における位置 $p(t)$ と、その時刻における速度 $v(t)$、すなわち位置の微分が分かっているとします。すると現在時刻から Δt 後の位置 $p(t+\Delta t)$ は、速度 $v(t)$ と時間 Δt を掛けたものを現在位置 $p(t)$ に足して得られます。ただし時刻 t から $t+\Delta t$ の間にも速度 $v(t)$ は変化しますから、これは Δt を細かく刻んで、速度 $v(t)$ に微小時間 dt を掛けたものを時刻 t から $t+\Delta t$ の間で累積、すなわち積分する必要があります。

$$p(t+\Delta t) = p(t) + \int_t^{t+\Delta t} v(t)dt$$



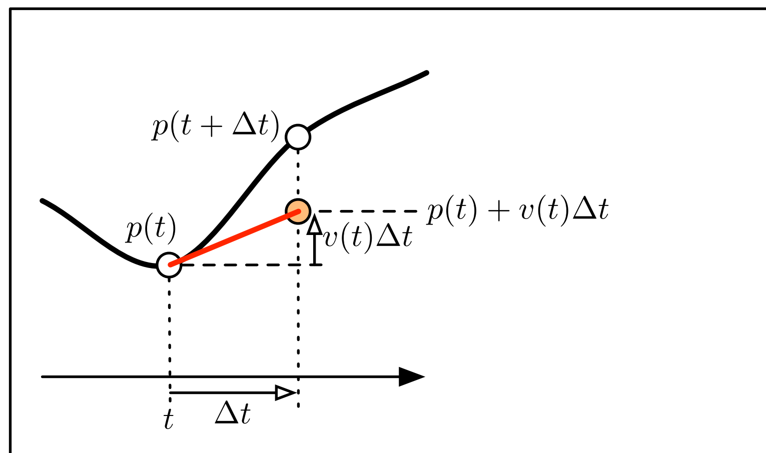
ですが、この $v(t)$ の積分が簡単に求まるなら苦労はしません。速度はアクセルを吹かすとか、狸が飛び出てブレーキをかけるとか、風が吹くとか、様々な理由で変化します。

そこで、時刻 t における速度 $v(t)$ を使って Δt 後の位置 $p(t+\Delta t)$ を予測してみます。時刻 t から $t+\Delta t$ の間、速度 $v(t)$ が変化しないと仮定すれば、次のような予測ができます。

$$p(t+\Delta t) = p(t) + v(t)\Delta t$$

このような予測の方法を、**オイラー法**と呼びます。**課題 3** 〜 **課題 6** では、この方法を使って質点の動きを求めていました。

しかし、この予測は結構よく外れます。Δt を小さくすれば予測の精度は向上しますが、それでもこの方法による予測の誤差は大きく、この予測を繰り返すと誤差が累積されて、結構具合の悪いことになります。



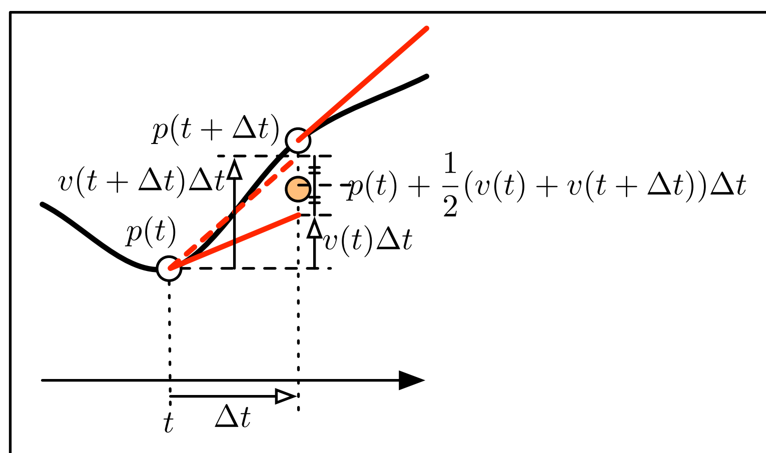
そこで、何とかして $t+\Delta t$ における速度 $v(t+\Delta t)$ を何とかして求めます。速度であれば、加速度を積分して求めることができます。時刻 t における加速度 $a(t)$ は、その時にかかっている力、例えばエンジンの出力なんかから知ることができます。これを使えば、次式により $t+\Delta t$ における速度を予測します。

$$v(t+\Delta t)=v(t)+a(t)\Delta t$$

って、これはさっき具合の悪いって言ったオイラー法そのまんまです。でも、ここではもうそんなことは気にしないことにします。

というわけで、これで $v(t)$ に加えて、とりあえず $v(t+\Delta t)$ が手に入りました。そこで、この二つの平均を使って予測してみます。

$$p(t+\Delta t)=p(t)+\frac{1}{2}(v(t)+v(t+\Delta t))\Delta t$$



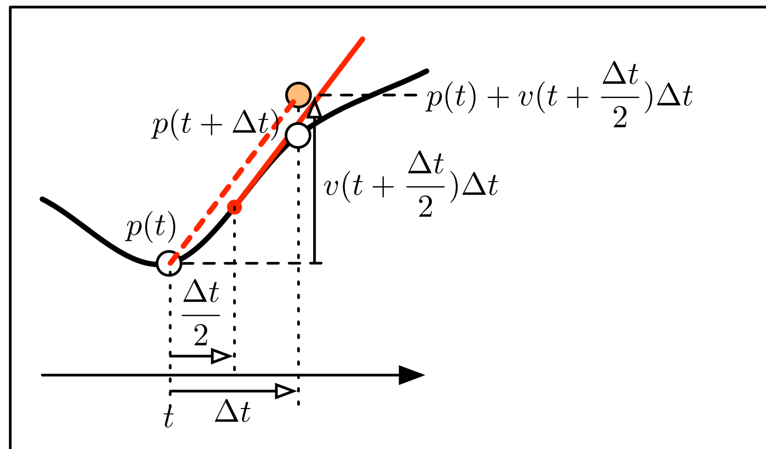
実はこの予測はそれなりに正確で、特に位置の二次微分である加速度を使って予測する場合には安定して答えが得られます。もちろん、これは $a(t)$ が時刻 t から $t+\Delta t$ の間一定だと見なしていますが、この加速度こそ、アクセルや狸や風に影響されて予測が尽きません。なので、今の加速度がしばらく続くと仮定するしかないんですね実際。

これは**修正オイラー法** (Heun の台形公式、Heun は「ホイン」と読みます) といいます。

なお、時刻 t から $t + \Delta t$ の区間の中央における速度 $v(t + \Delta t / 2)$ を使って予測する方法もあります。これもまずオイラー法を使って予測します。

$$v(t + \frac{1}{2}\Delta t) = v(t) + a(t)\frac{1}{2}\Delta t$$

$$p(t + \Delta t) = p(t) + v(t + \frac{1}{2}\Delta t)\Delta t$$



この方法は予測に使う区間 $\Delta t / 2$ がオイラー法の半分であることから、やはりオイラー法よりも正確なものになります。これは**改良オイラー法** (Heun の矩形公式) といいます。

実は、加速度 $a(t)$ が時刻 t から $t + \Delta t$ の間一定だという仮定をすると、改良オイラー法は修正オイラー法と同じです。

$$\frac{1}{2}(v(t) + v(t + \Delta t)) = \frac{1}{2}(v(t) + v(t) + a(t)\Delta t)) = v(t) + a(t)\frac{1}{2}\Delta t$$

修正オイラー法や改良オイラー法は、二次の**ルンゲクッタ法**とも呼ばれます。より高い精度の予測を行うには、四次のルンゲクッタ法や、それをさらに改良した**ルンゲクッタギル法**が用いられます。このほか、マリオのジャンプで有名な**ベレ法**という手法もあります。

課題7

1. **課題 6** のプログラムを修正オイラー法もしくは改良オイラー法を使うように書き換えてください。

教材

http://marina.sys.wakayama-u.ac.jp/~tokoi/seminar1/cloth.zip

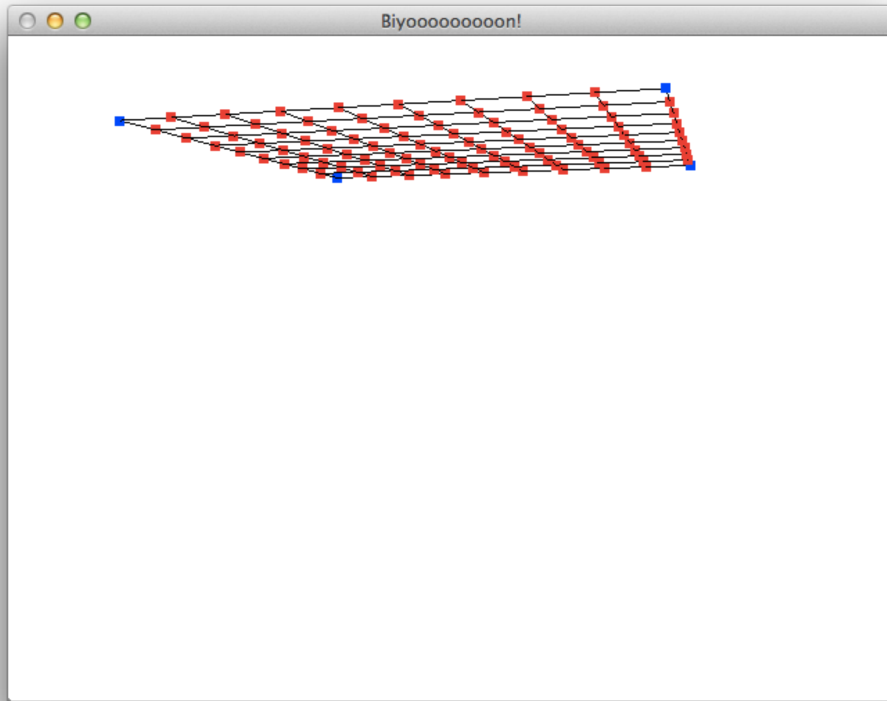
- このプログラムは Windows (Visual Studio 2017 以降)、Mac OS X (Xcode 10 以降)、および Linux (ディストリビューションに依存) でコンパイル、実行できます。

課題8

1. 教材に含まれる、draw.cpp というソースプログラムを変更します。そのほかのプログラムは補助的なものです。 draw.cpp は以下の内容になっています。

```
//
// 網を描く
//
//      m: 横方向の質点の数 (線分の数 + 1)
//      n: 縦方向の質点の数 (線分の数 + 1)
//      p: 質点の位置 (三次元)
//

extern void net(int m, int n, void *p);

// 時間間隔
const double dt = (1.0 / 60.0);

// 結び目の数
const int pointsi = 10;
const int pointsj = 10;

// 結び目の位置
double position[pointsi][pointsj][3];

// 描画
void draw(void)
{
  // 網
  net(m, n, position);

  //
  // position の更新は net() の実行後に行う
  //
}
```

2. これを実行すると、次のような表示が得られます。マウスの右ボタンでドラッグすれば、図形を回転できます。
3. 関数 `draw()` に、図形を描画する手続きを記述します。この関数は繰り返し実行されます。関数 `net()` はあらかじめ用意しています。
4. 関数 `net()` は、第一引数 m、第二引数 n に指定した数の点を結ぶ網を描きます。各点の座標値は第三引数 p で指定します。 p は 3 要素の配列 m×n 個からなる三次元配列です。
5. これを布らしくしてください。布らしくするためには、縦・横・斜め方向のばねを考える必要があります。頂点の間隔は 1 です。


Biyoooooooooon!