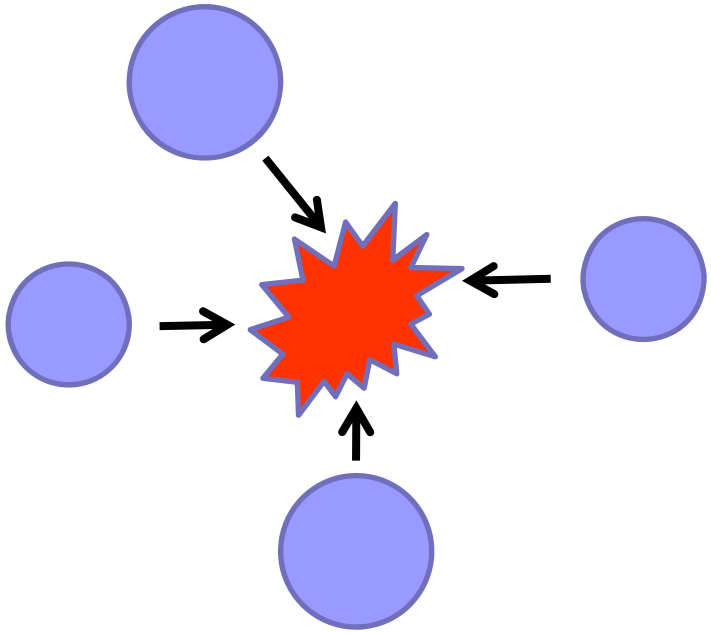
創造設計演習（S&V演習）

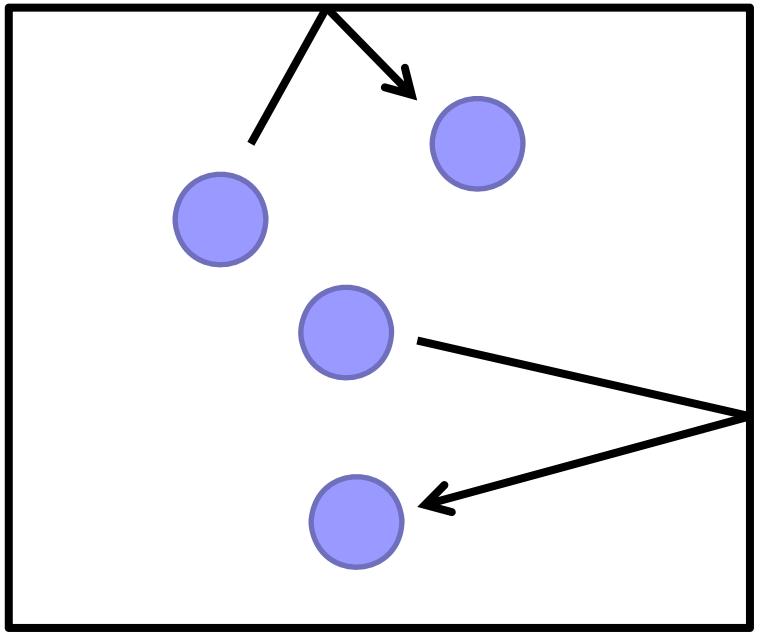
個別要素法の基礎

- Distinct Element Method: DEM -

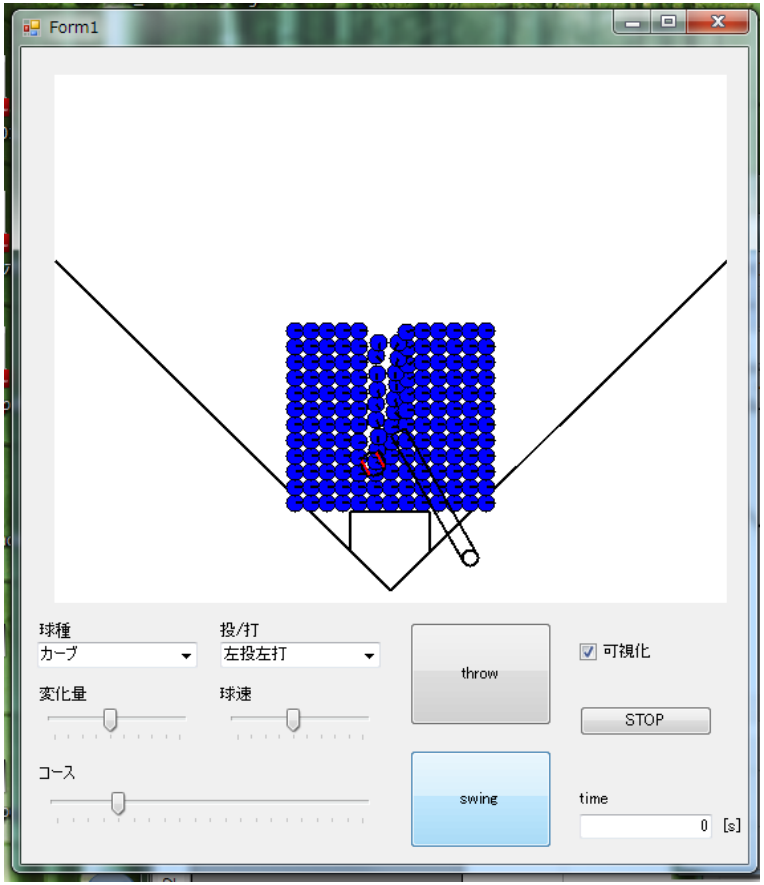
目的

シミュレーション＆ビジュアライジェーション演習では個別要素法（DEM）のソースを基礎に、オリジナルのプログラムを作成する。

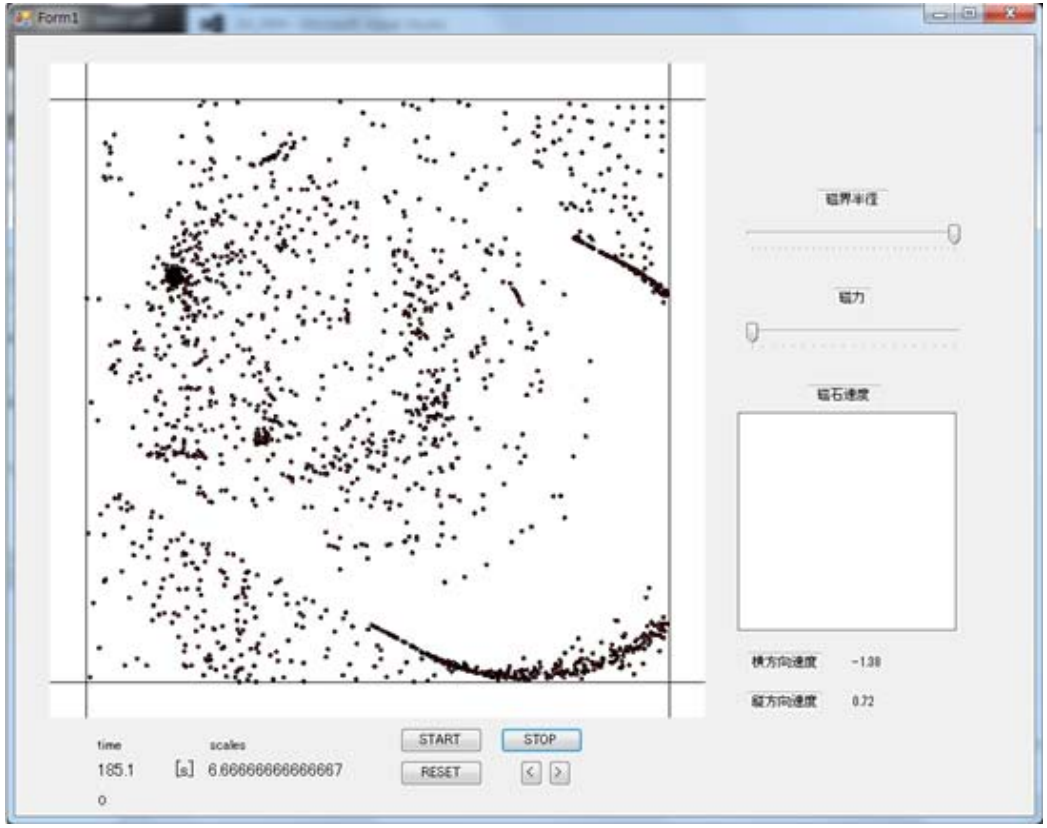




過去の優秀作品

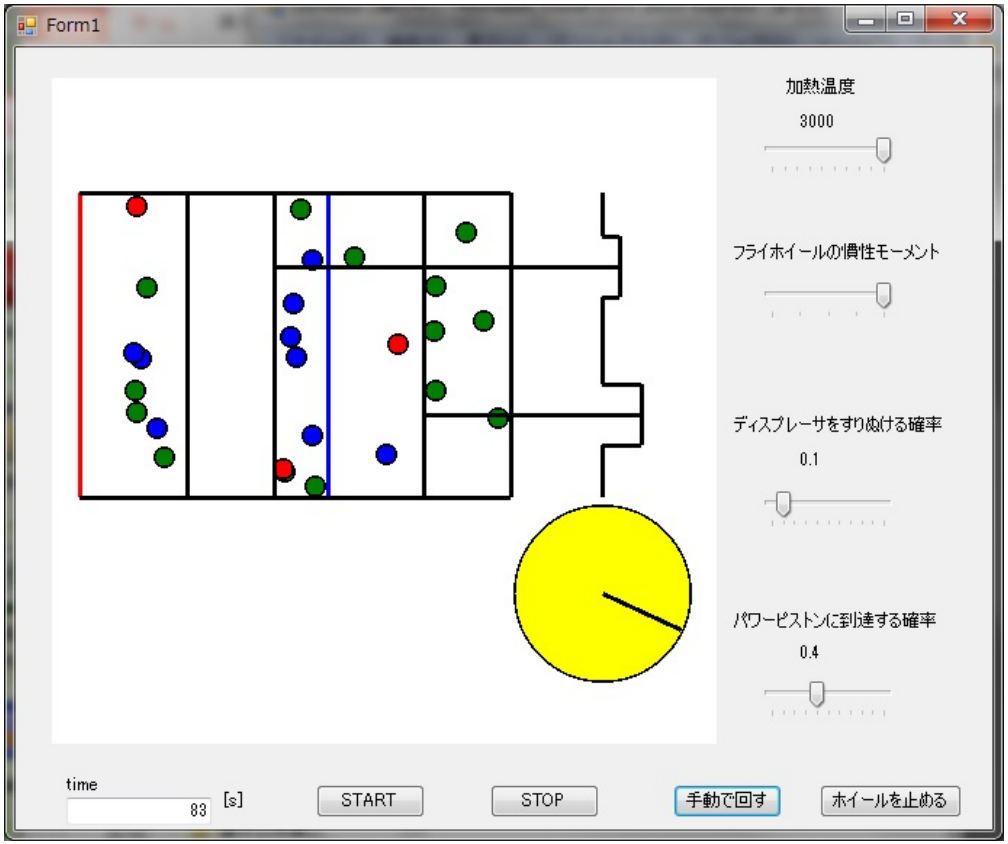


↑ベースボール

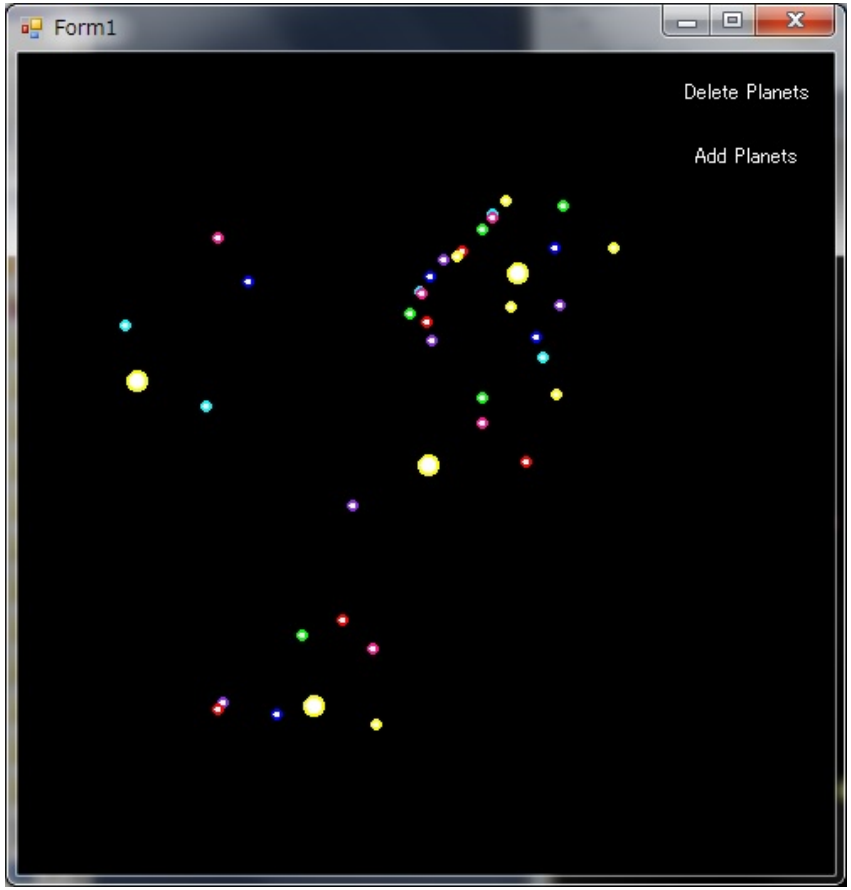


↑磁石まわりの磁界のシミュレーション

過去の優秀作品



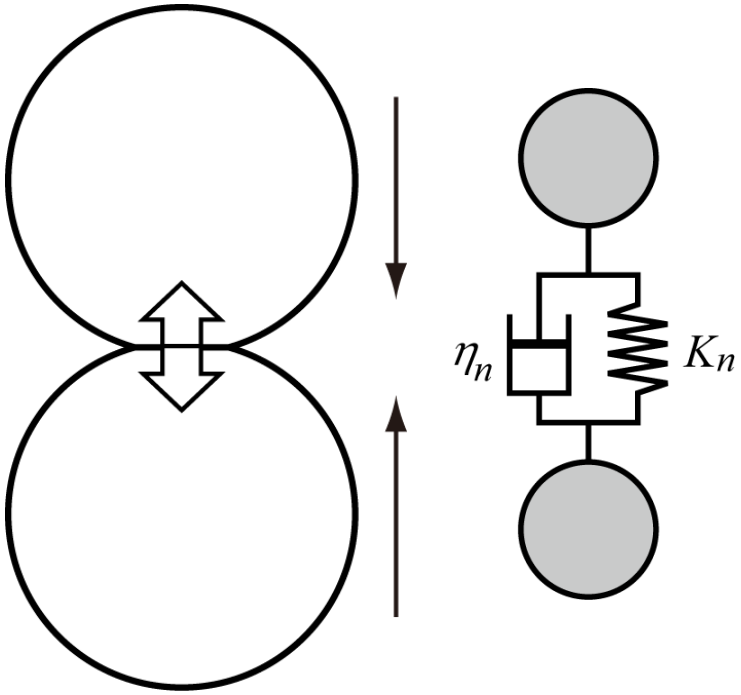
↑スターリングエンジン



↑流星群の軌道

DEMの接触モデル

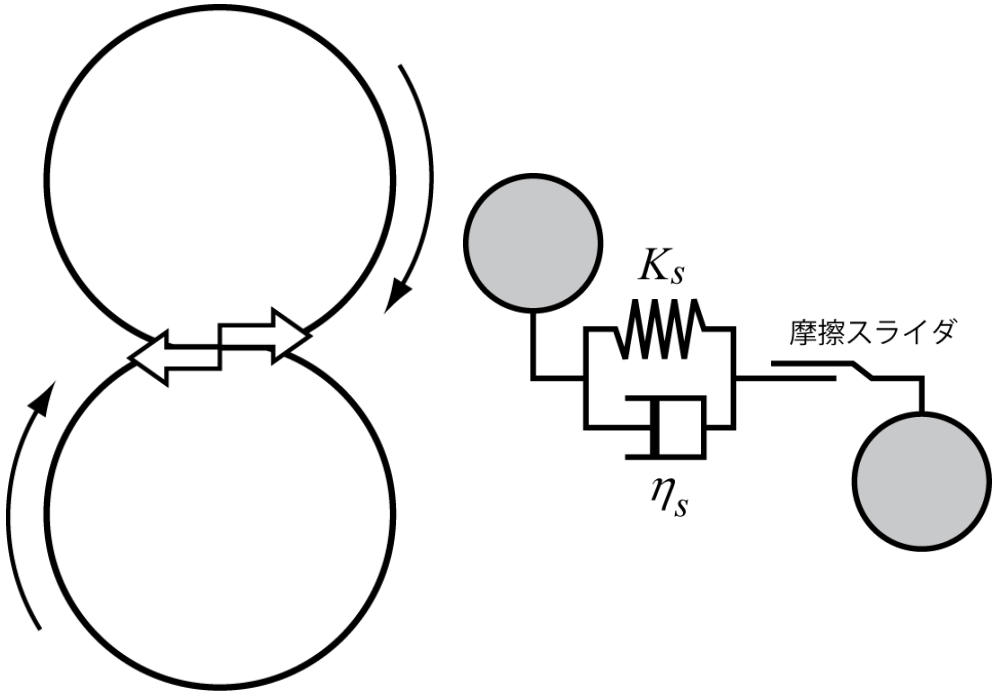
法線方向作用



$$m\frac{du^2}{dt^2}+\eta_n\frac{du}{dt}+K_nu=0$$

u：並進変位

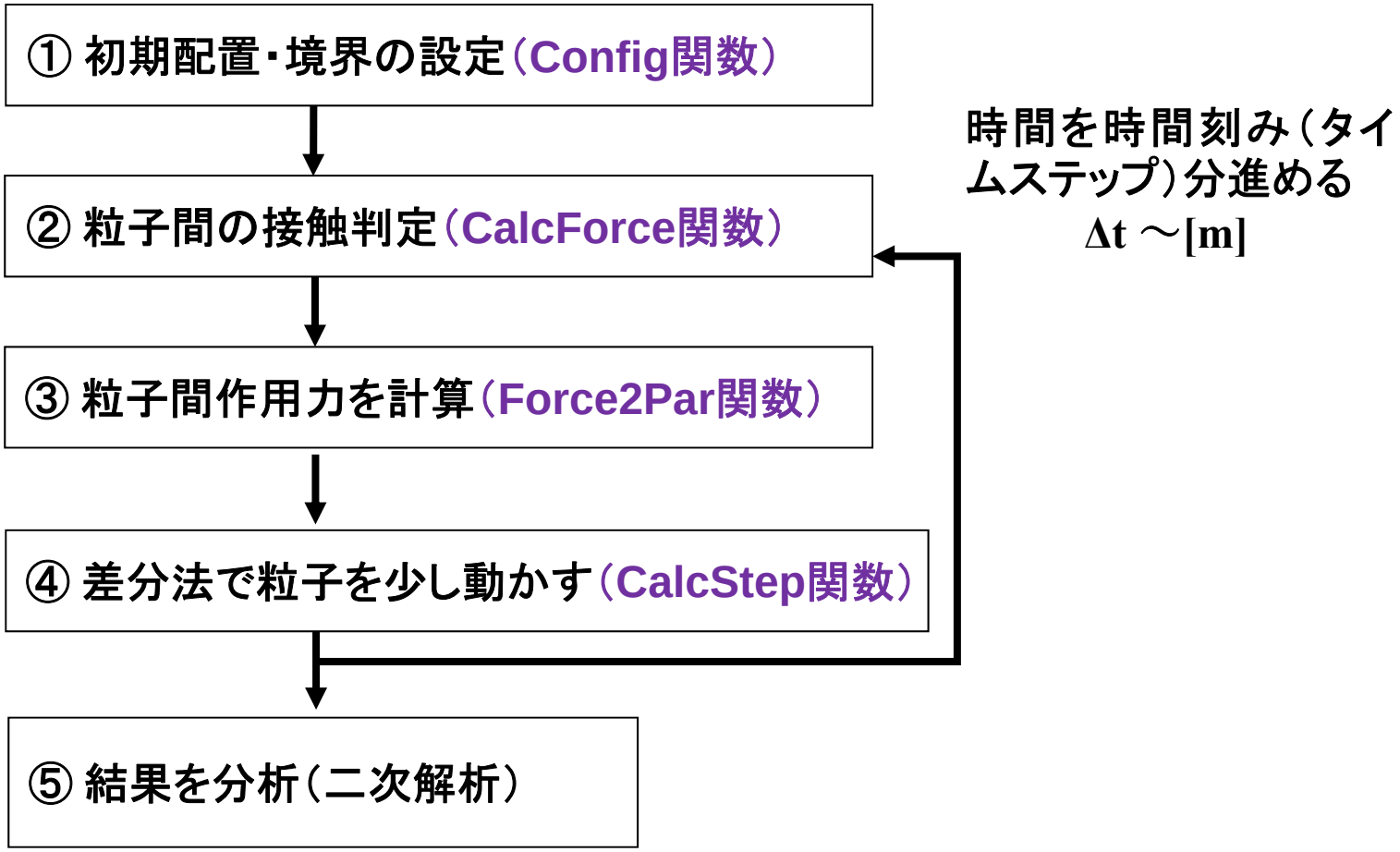
せん断方向作用



$$I\frac{d\phi^2}{dt^2}+\eta_s r^2\frac{d\phi}{dt}+K_s r^2\phi=0$$

ϕ：回転変位

DEMの計算の流れ


① 初期配置・境界の設定（Config関数）
② 粒子間の接触判定（CalcForce関数）
③ 粒子間作用力を計算（Force2Par関数）
④ 差分法で粒子を少し動かす（CalcStep関数）
⑤ 結果を分析（二次解析）
時間を時間刻み（タイムステップ）分進める
Δt ～[m]

① 初期配置・境界の設定

■初期条件の設定

時刻t=0における粒子の位置、角度、運動量、角運動量を与える

■境界条件の設定

□自由境界条件

□反射境界条件（壁）

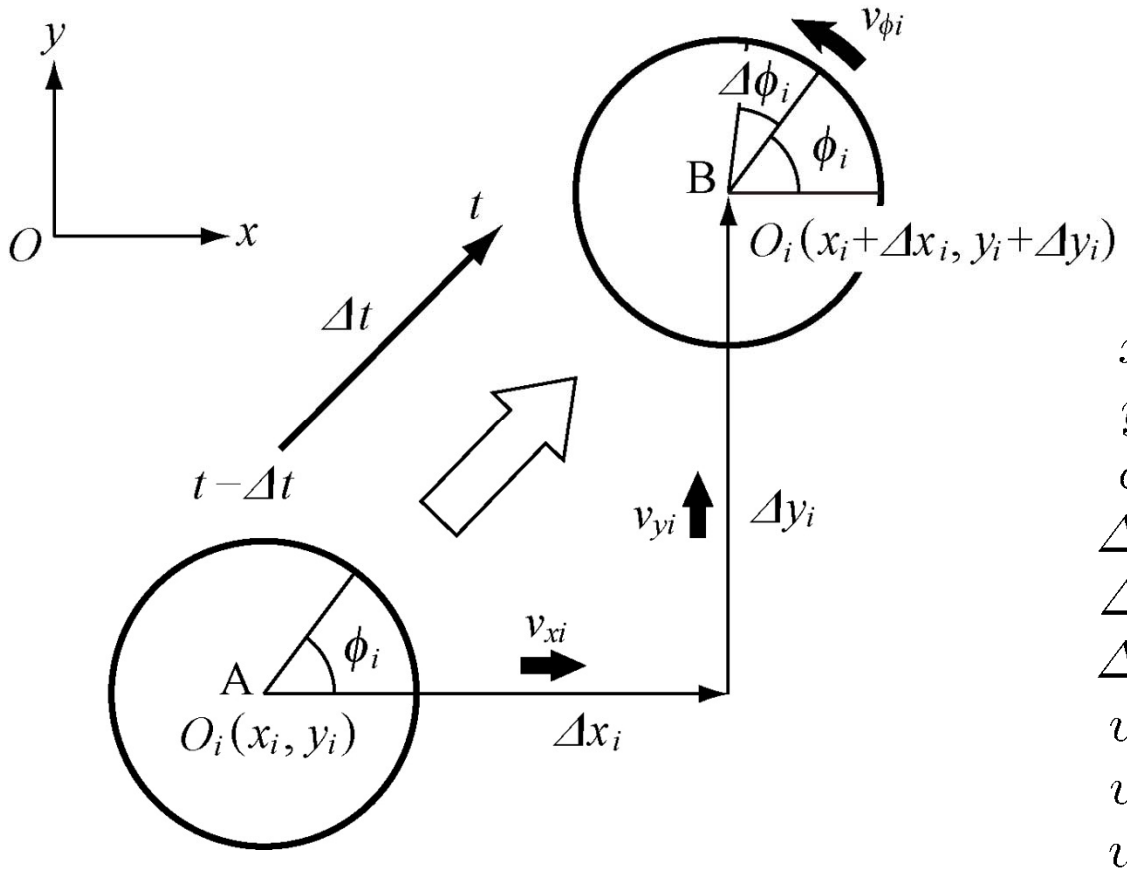
1) ミラーリング（入射角=反射角）

2) バネーダンパ相互作用

□周期境界条件（周期構造）

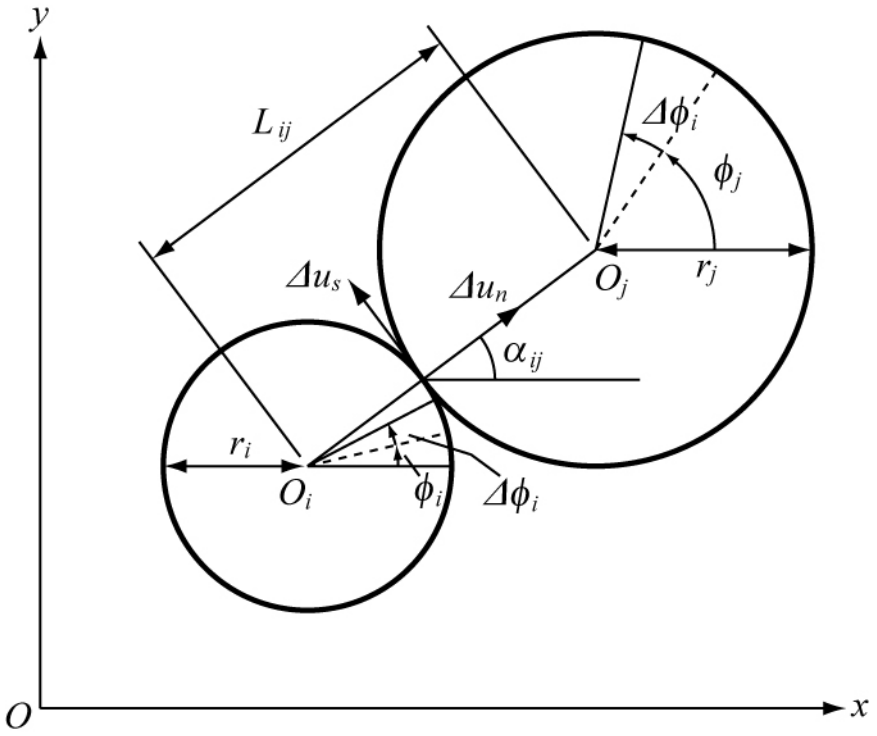
① 粒子配置の変数

粒子*i* の移動



x_i ：粒子i のx座標
y_i ：粒子i のy座標
ϕ_i ：粒子i の角度
Δx_i ：粒子i のx方向変位増分
Δy_i ：粒子i のy方向変位増分
$\Delta \phi_i$ ：粒子i の回転変位増分
v_{xi} ：粒子i のx方向速度
v_{yi} ：粒子i のy方向速度
$v_{\phi i}$ ：粒子i の角速度

② 粒子間の接触判定



r_i :粒子i の半径
r_j :粒子j の半径
L_{ij} :粒子ij 間の距離
α_{ij} :粒子ij 間の角度
Δu_n :相対的な法線方向変位増分
Δu_s :相対的な接線方向変位増分

接触判定

$$r_i + r_j \geq L_{ij}$$

$$r_i + r_j \geq L_{ij}$$

ここで、

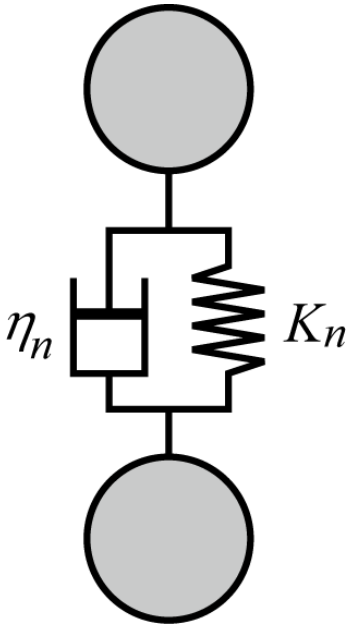
$$L_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

粒子間角度

$$\sin\alpha_{ij} = -(y_i - y_j)/L_{ij}$$
$$\cos\alpha_{ij} = -(x_i - x_j)/L_{ij}$$

③ 粒子間作用力の計算

法線方向成分



K_n :法線方向の弾性係数
η_n :法線方向の粘性係数

時刻tにおける法線方向の作用力を求める。

時刻tの相対変位増分Δu_nによる力:

$$\Delta e_n = K_n \Delta u_n, \quad \Delta d_n = \eta_n \frac{\Delta u_n}{\Delta t}$$

ただし、圧縮力を正とする。
時刻tにおける弾性力$e_n(t)$と粘性抵抗力$d_n(t)$は、

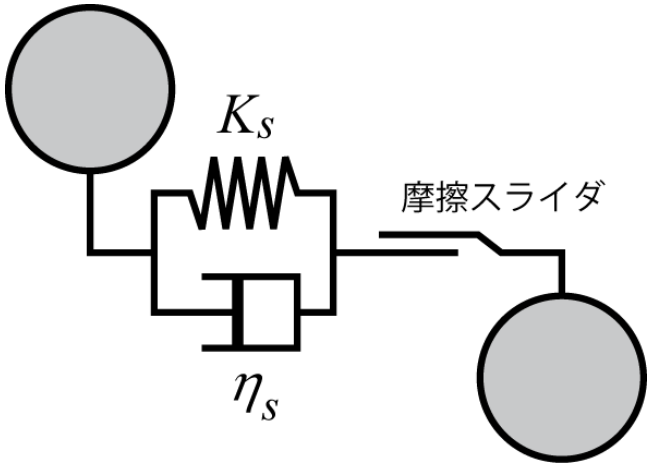
$$e_n(t) = e_n(t - \Delta t) + \Delta e_n(t), \quad d_n(t) = \Delta d_n$$

よって、2粒子間の法線方向の合力 $f_n(t)$ は、

$$f_n(t) = e_n(t) + d_n(t)$$

③ 粒子間作用力の計算

せん断方向成分



K_s :せん断方向の弾性係数
η_s :せん断方向の粘性係数

時刻tにおけるせん断方向の作用力を求める。

時刻tの相対変位増分Δu_nによる力:

$$\Delta e_s = K_s \Delta u_s, \quad \Delta d_s = \eta_s \frac{\Delta u_s}{\Delta t}$$

ただし、粒子iの時計回りの方向を正とする。
時刻tにおける弾性力$e_s(t)$と粘性抵抗力$d_s(t)$は、

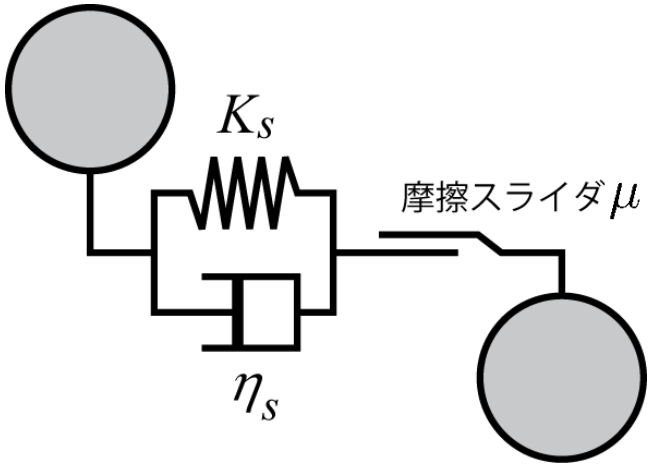
$$e_s(t) = e_s(t - \Delta t) + \Delta e_s(t), \quad d_s(t) = \Delta d_s$$

2粒子間のせん断方向の合力 $f_s(t)$:

$$f_s(t) = e_s(t) + d_s(t)$$

③ 粒子間作用力の計算

せん断方向成分



K_s :せん断方向の弾性係数
η_s :せん断方向の粘性係数
μ :摩擦係数

クーロンの摩擦の法則から得られるせん断力より
大きな力は作用しない

せん断力の条件:

$$\text{if } e_n(t) \le 0, \quad e_s(t) = d_s(t) = 0$$

$$\text{elseif } |e_s(t) \ge \mu e_n(t)|,$$

$$|e_s(t) = \mu e_n(t) \times \text{SIGN}(e_s(t))|, \quad d_s(t) = 0$$

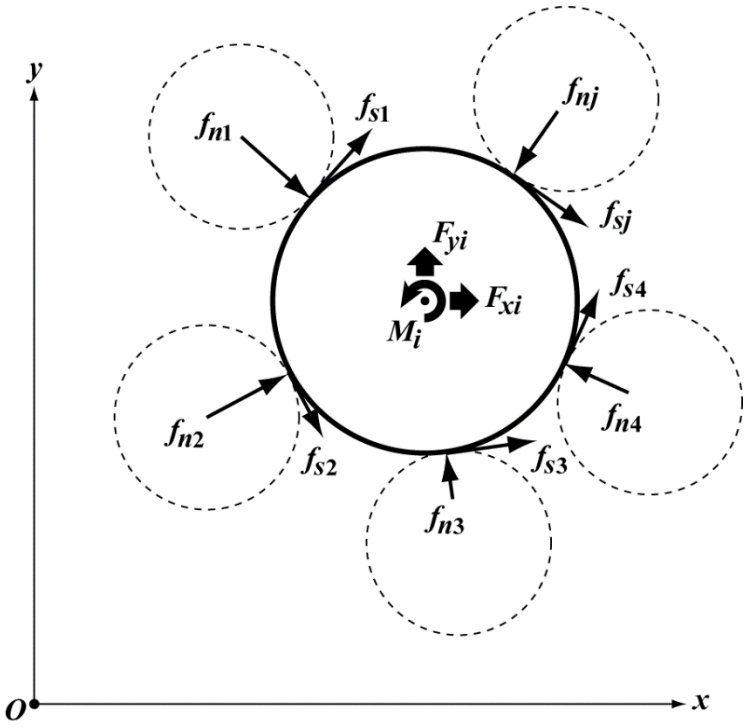
ここで、μ は粒子間の摩擦係数、
SIGNは$e_s(t)$の正負を表す関数とする。

2粒子間のせん断方向の合力 $f_s(t)$:

$$f_s(t) = e_s(t) + d_s(t)$$

③ 粒子間作用力の計算

粒子i の合力



着目粒子i に接触する全ての粒子jからの接触力[f_{nj}, f_{sj}]を
粒子i に作用する力[F_{xi}, F_{yi}, M_i]に変換して足し合わせる

$$F_{xi} = \sum_j \left(-f_{nj}\cos\alpha_{ij} + f_{sj}\sin\alpha_{ij} + m_i g\right)$$

重力有の場合

$$F_{yi} = \sum_j \left(-f_{nj}\sin\alpha_{ij} + f_{sj}\cos\alpha_{ij}\right)$$

$$M_i = -r_i \sum_j f_{sj}$$

M_i : 粒子i の中心回りのモーメント（反時計回りが正）

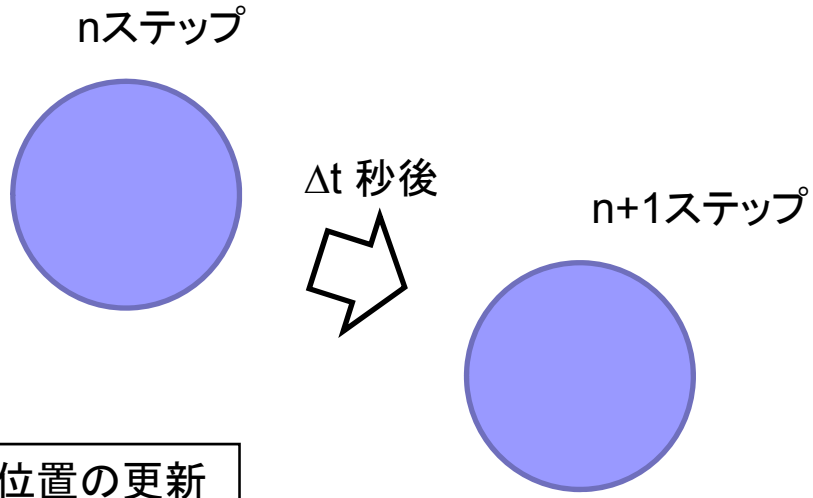
④ 差分法で粒子を少し動かす

差分法（Euler法）

粒子i のnステップにおける加速度を計算

$$\begin{cases} a_x^n = f_x^n / m \\ a_y^n = f_y^n / m \\ a_\phi^n = M^n / I \end{cases}$$

※添字i を省略



粒子の速度の更新

$$\begin{cases} v_x^{n+1} = v_x^n + a_x^n \Delta t \\ v_y^{n+1} = v_y^n + a_y^n \Delta t \\ v_\phi^{n+1} = v_\phi^n + a_\phi^n \Delta t \end{cases}$$

粒子の位置の更新

$$\begin{cases} x^{n+1} = x^n + \Delta x \ (\Delta x = v_x^n \Delta t) \\ y^{n+1} = y^n + \Delta y \ (\Delta y = v_y^n \Delta t) \\ \phi^{n+1} = \phi^n + \Delta \phi \ (\Delta \phi = v_\phi^n \Delta t) \end{cases}$$

数値計算の精度

差分法の時間増分Δtの取り方には注意が必要

- Δtが小さい → 計算時間大、精度良
- Δtが大きい → 計算時間小、精度悪

差分法（Euler法）では以下の条件を満足しないと数値計算が発散する

$$\Delta t \leq 2\sqrt{\frac{m}{K_n}}$$

Euler法よりも精度の良い数値積分法が多く開発されている。

例）Adams-Baxhforth法

$$v^{n+1} = v^n + (3f^n - f^{n-1})\frac{\Delta t}{2m}$$

$$x^{n+1} = x^n + (v^n + v^{n-1})\frac{\Delta t}{2}$$

プログラムの構成

■Form1.h

描画やアクションの設定を行う。
（フォームデザインの内容は右図を参照）

■main_DEM.cpp

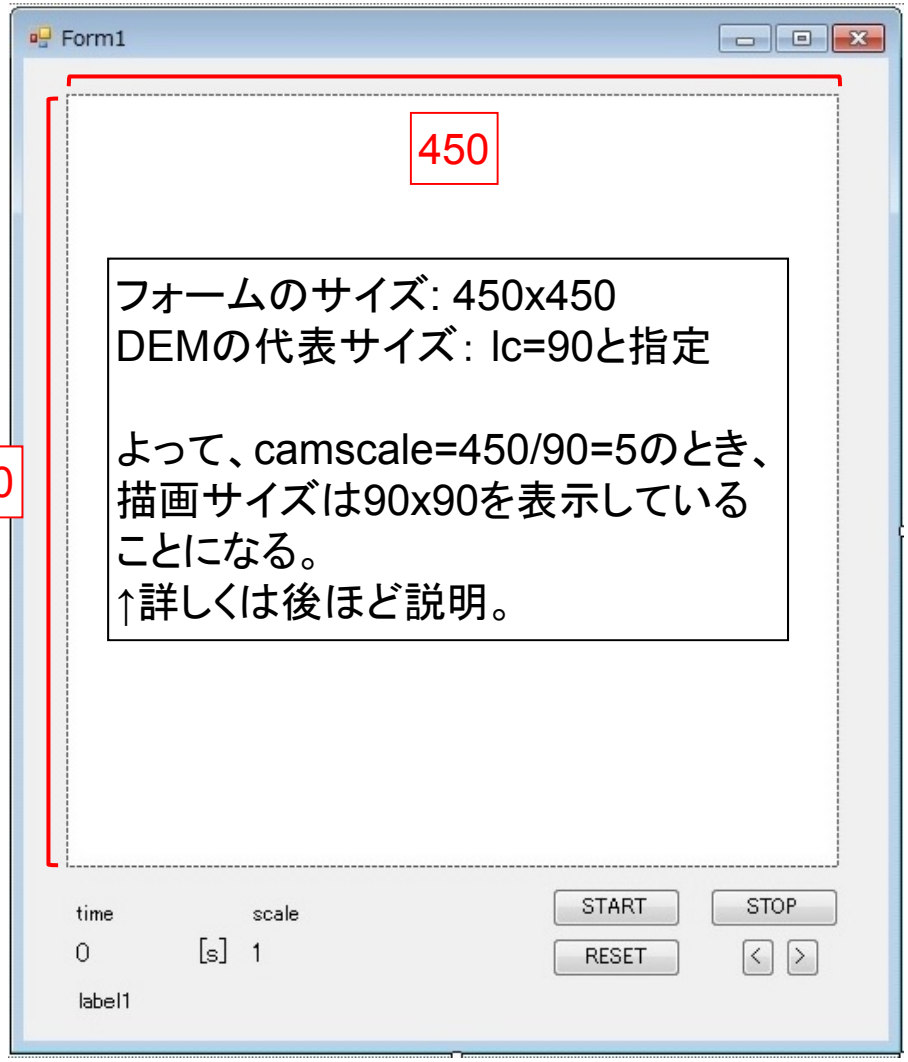
個別要素法の計算コード。
以下の4つの関数で構成されている。

□setParam()
　各変数の初期化／設定。

□Config()
　各粒子の初期条件(位置／速度)を与える。

□CalcStep()
　粒子間の接触判定を行い、CalcForce関数で計算した接触力より、時間刻みステップにおける各粒子の位置座標を計算する。

□CalcForce()
　粒子間の接触力（相互作用力）をサブ関数Force2Para()をコールして計算。



Main_DEM.cpp

```
//おまじない
//(意味: ここから下は普通のC++の構文で書いてありますよ)
#pragma unmanaged

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "PEMstruct.h"

const double PI = 3.1415926535;        //円周率
const double G = 0;                    //重力加速度
//const double G = 9.80665;            //重力加速度
const double rho = 10;                 //粒子の密度

//力の計算をする関数
void CalcForce(int pe_n, int wa_n, PEM *pe, double dt, double lc, double *param);
//DEM用の，2要素の力を計算して代入する関数
void Force2Par(PEM *pe, int pe_n, int i, int j, double ld, double cos_a, double sin_a, double dt);
```

粒子の属性を表す構造体をインクルード

新しい変数を作成した場合、必ずここにも追加

setParam関数・・・各変数の初期化／設定

```
void setParam(PEM *pe, int i, int n, bool move, double r, double x, ... , double vy, double vphi){
        pe[i].exist = true; //計算で使うよう設定

        //引数で与えられた値の設定
        pe[i].move = move;
        pe[i].r = r;
        pe[i].x = x;   pe[i].y = y;   pe[i].phi = phi;
        pe[i].vx = vx; pe[i].vy = vy; pe[i].vphi = vphi;

        //加速度,変位増分,力を0で初期化
        pe[i].ax = pe[i].ay = pe[i].aphi = 0.0;
        pe[i].dx = pe[i].dy = pe[i].dphi = 0.0;
        pe[i].fx = pe[i].fy = pe[i].fm   = 0.0;

        //質量,慣性モーメントの設定
        pe[i].m = 4.0/3.0*PI*rho*r*r*r;
        pe[i].Ir = PI*rho*r*r*r*r/2.0;

        //弾性力の初期化
        for(int j=0; j<n ;j++){
                pe[i].en[j] = 0.0;
                pe[i].es[j] = 0.0;
        }
}
```

半径、位置、回転、速度、回転速度の設定

Config関数･･･各粒子の初期条件の設定

```
void Config(int pe_n, int wa_n, PEM *pe){

        int n=pe_n+wa_n;

        //存在初期化
        for(int i=0 ;i<n ;i++){
                pe[i].exist=false;
        }

        //粒子の初期条件
        //setParam(pe,  i, n, move,    r,   x,   y,phi,vx,vy,vphi);
        setParam(pe,  0, n, true,  5.0, -10,   0,  0,  4, 0, 0);
        setParam(pe,  1, n, true,  5.0,  10,   0,  0, -4, 0, 4);
        /*
        //壁の初期条件 [四方が壁]
        setParam(pe, pe_n  , n,false, 0.01, -45,   0, 0.5*PI, 0, 0, 0); //動かない& 壁
        setParam(pe, pe_n+1, n,false, 0.01,  45,   0, 0.5*PI, 0, 0, 0);
        setParam(pe, pe_n+2, n,false, 0.01,   0,  45, 0,       0, 0, 0);
        setParam(pe, pe_n+3, n,false, 0.01,   0, -45, 0,       0, 0, 0);
        */
}
```

各粒子の初期条件(半径、位置、回転、速度、回転速度)を与える。

CalcStep関数･･･ 差分法による粒子の更新

```
void CalcStep(int pe_n, int wa_n, PEM *pe, double dt, double lc, double *param){
        int i;

        //各粒子の力の初期化
        for(i = 0; i < pe_n+wa_n; i++){
                pe[i].fx=0.0; pe[i].fy=0.0; pe[i].fm=0.0;
        }
        //力の計算
        CalcForce(pe_n, wa_n, pe, dt, lc, param);

        //------------------ここから位置などの計算------------
        //力から加速度，速度，変位の更新
        for(i = 0; i < pe_n+wa_n; i++){
                if(pe[i].exist == false || pe[i].move == false) continue; //要素を使わないか、動かない場合は飛ばす

                pe[i].ax   = pe[i].fx / pe[i].m;
                pe[i].ay   = pe[i].fy / pe[i].m;
                pe[i].aphi = pe[i].fm / pe[i].Ir;

        //------------------ 省略 ------------
        //エネルギーなどの計算結果
        param[1]=0.0;
        }
}
```

オイラー差分(pp.14参照)

出力パラメータの計算(今回は使わない)

CalcForce関数（その1）･･･ 接触粒子間の力を計算

```
void CalcForce(int pe_n, int wa_n, PEM *pe, double dt, double lc, double *param){
	int i,j;
	double cos_a, sin_a, ld; //衝突時の角度、中心間距離

	//2要素間の全ペアの力の計算
	for(i = 0; i < pe_n; i++){
		if(pe[i].exist == false) continue; //要素を使わない場合はforループを1周飛ばす

		//粒子-粒子間
		for(j = i+1; j < pe_n; j++){
			if(pe[j].exist == false) continue;
			if(pe[i].move == false && pe[j].move == false) continue;
			//両方動かない場合も力を計算する意味が無いので飛ばす。

			double lx = pe[j].x - pe[i].x;
			double ly = pe[j].y - pe[i].y;
			ld = sqrt(lx*lx+ly*ly);

			//ぶつかっている場合
			if(pe[i].r+pe[j].r > ld ){
				cos_a = lx/ld;
```

粒子iと粒子jの中心間距離ldの計算

CalcForce関数（その2）･･･ 接触粒子間の力を計算

```
                    double lx = pe[j].x - pe[i].x;
                    double ly = pe[j].y - pe[i].y;
                    ld = sqrt(lx*lx+ly*ly);

                    //ぶつかっている場合
                    if(pe[i].r+pe[j].r > ld ){
                        cos_a = lx/ld;
                        sin_a = ly/ld;
                        Force2Par(pe, pe_n, i, j, ld, cos_a, sin_a, dt);
                    }else{
                        pe[i].en[j] = 0.0;
                        pe[i].es[j] = 0.0;
                    }
                }
            }

            //外力の計算
            for(i = 0; i < pe_n+wa_n ; i++){
                if(pe[i].exist == false || pe[i].move == false) continue; //要素を使わないか、動かない場合は飛ばす
                pe[i].fy += -G * pe[i].m;   //重力
            }
        }
```

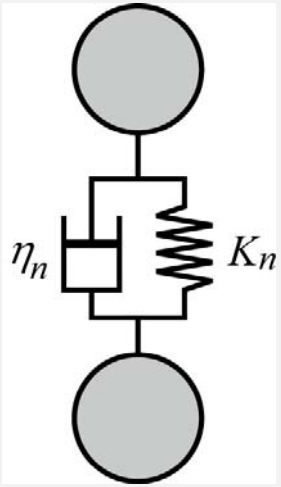
$r_i+r_j>ld$のとき、サブ関数Force2Par()より、2粒子間に働く力を求める。

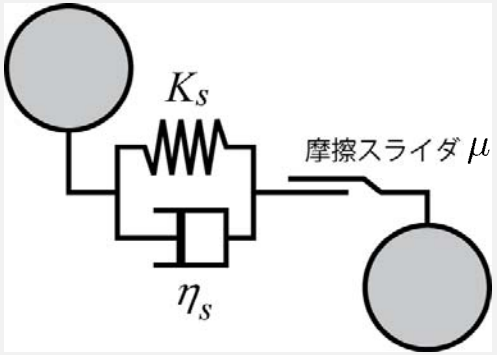
Force2Par関数(その1)

```
void Force2Par(PEM* pe, int pe_n, int i, int j, double ld, double cos_a, double sin_a, double dt){
        //PEM用の係数変数
        //nが法線・sがせん断方向の値
        double un;    double us;     //相対変位増分
        double veln;  double vels;   //相対速度増分
        double hn;    double hs;     //合力(法線,せん断)
        double kn;    double ks;
        double etan; double etas;
        double frc;

        //弾性・粘性係数の設定
        if(j<pe_n){
                kn   = 100000;    ks   = 5000;
                etan = 5000;      etas = 1000;
                frc  = 10;
        }else{
                kn   = 500000;   ks   = 1000;
                etan = 1000;      etas = 500;
                frc  = 1.0;
        }
```

粒子と壁で場合分けして材料定数を設定





Force2Par関数(その2)

2粒子間の法線力とせん断力を計算するために、変位ベクトルを局所座標系に変換する。

```
//法線方向成分とせん断方向成分の相対的変位増分の計算
un   = +(pe[i].dx-pe[j].dx)*cos_a + (pe[i].dy-pe[j].dy)*sin_a;
us   = -(pe[i].dx-pe[j].dx)*sin_a + (pe[i].dy-pe[j].dy)*cos_a
       +(pe[i].r*pe[i].dphi + pe[j].r*pe[j].dphi);

veln = +(pe[i].vx-pe[j].vx)*cos_a + (pe[i].vy-pe[j].vy)*sin_a;
vels = -(pe[i].vx-pe[j].vx)*sin_a + (pe[i].vy-pe[j].vy)*cos_a
       +(pe[i].r*pe[i].vphi + pe[j].r*pe[j].vphi);

//法線方向成分とせん断方向成分の合力の計算
 //バネの力を追加
pe[i].en[j] = pe[i].en[j];
pe[i].es[j] = pe[i].es[j];
 //ダンパの力を追加
hn = pe[i].en[j];
hs = pe[i].es[j];

if(hn <= 0.0){
       hs=0.0;
}else if((abs(hs)-frc*hn) >= 0.0){
       hs=frc*fabs(hn)*hs/fabs(hs);
}
```

ここに相応しいプログラムを記述(課題1,2)
pp.10, pp.11を参照

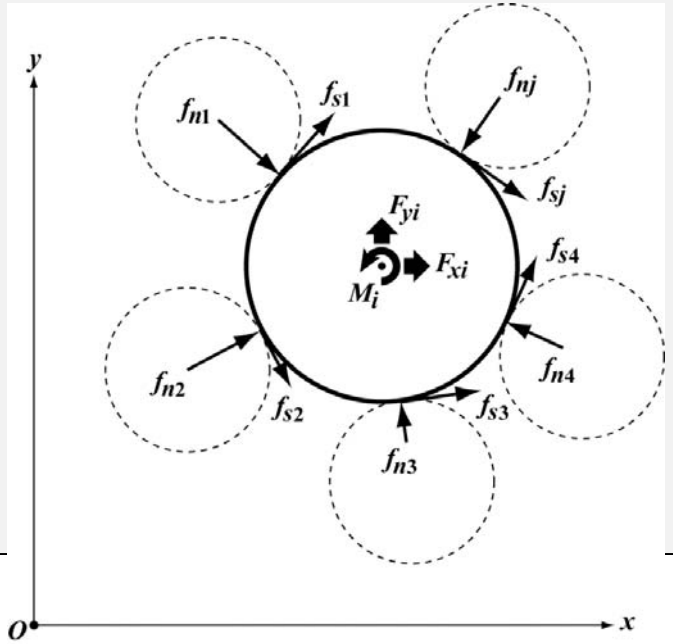
条件付きせん断力の計算
(pp.12を参照)

Force2Par関数(その3)

```
        //粒子i,粒子jのx方向成分とy方向成分の合力の計算
        pe[i].fx = -hn*cos_a + hs*sin_a + pe[i].fx;
        pe[i].fy = -hn*sin_a - hs*cos_a + pe[i].fy;
        pe[i].fm = pe[i].fm - pe[i].r*hs;

        pe[j].fx = hn*cos_a - hs*sin_a + pe[j].fx;
        pe[j].fy = hn*sin_a + hs*cos_a + pe[j].fy;
        pe[j].fm = pe[j].fm - pe[j].r*hs;
}
```

全体座標系に変換して合力の総和を計算



$$F_{xi} = \sum_j (-f_{nj}\cos\alpha_{ij} + f_{sj}\sin\alpha_{ij} + m_i g)$$

重力有の場合

$$F_{yi} = \sum_j (-f_{nj}\sin\alpha_{ij} + f_{sj}\cos\alpha_{ij})$$

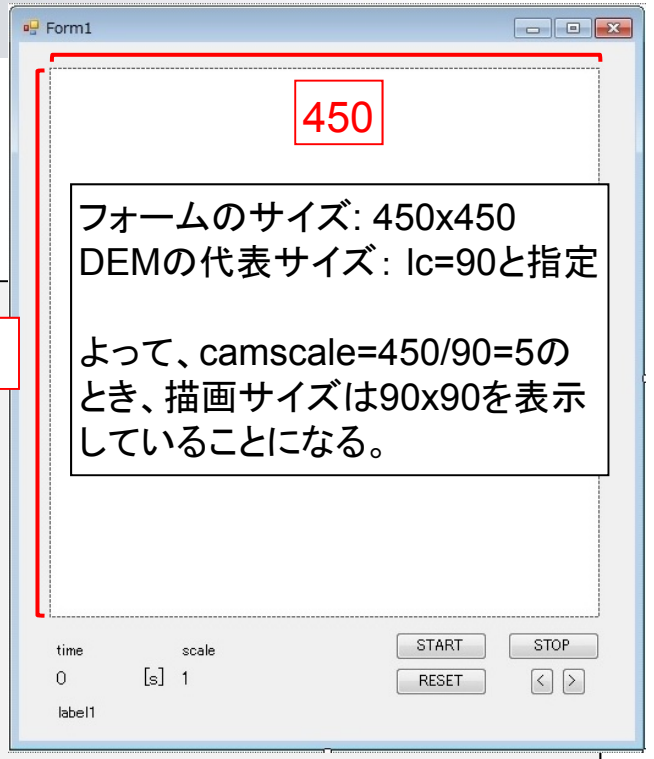
$$M_i = -r_i \sum_j f_{sj}$$

※重力は関数CalcForce()のところで計算するため、
ここでは考慮しない(pp.22を参照)。

Form1.hの説明

※59行目

```
Form1()
{
    InitializeComponent();
    //dt=1e-15;                 //計算の時間刻み(s)(MD)
    dt=0.005;                   //計算の時間刻み(s)(DEM)
    //lc=1e-9;                  //MDの代表サイズ
    lc=90;                      //DEMの代表サイズ
    pe_n=2;                     //粒子数の設定
    wa_n=0;                         //壁の数の設定
    param = new double[10];     //計算結果などの格納用のメモリ確保
    pe = new PEM[pe_n+wa_n];    //粒子構造体のメモリ確保
    for(int i=0; i<pe_n+wa_n ;i++){
        pe[i].en = new double[pe_n+wa_n];
        pe[i].es = new double[pe_n+wa_n];
    }
    Config(pe_n, wa_n, pe); //粒子の初期値設定(main_MD.cpp参照)
```



フォームのサイズ: 450x450
DEMの代表サイズ: lc=90と指定

よって、camscale=450/90=5のとき、描画サイズは90x90を表示していることになる。

変数の定義（主要なモノのみ）

n	:粒子数
x[i], y[i], phi[i]	:粒子*i* の位置、角度
vx[i], vy[i], vphi[i]	:粒子*i* の速度、角速度
ax[i], ay[i], aphi[i]	:粒子*i* の加速度、角加速度
dx[i], dy[i], dphi[i]	:粒子*i* の変位増分、回転変位増分
r[i], m[i], Ir[i]	:粒子*i* の半径、重量、慣性モーメント
fx[i], fy[i], fm[i]	:粒子*i* の*x*, *y* 方向合力とモーメント
en[i][j], es[i][j]	:粒子*i* と粒子*j* の法線方向接触力とせん断方向接触力
cos_a, sin_a	:粒子間角度αによる余弦成分と正弦成分
kn, ks	:法線方向／せん断方向の弾性係数
etan, etas	:法線方向／せん断方向の粘性係数
frc	:摩擦係数

今日の課題

配布したプログラムを基に以下の課題に取り組み、粒子の相互作用を理解する。

課題①
粒子間の法線方向作用のみを考慮したモデルを作り、数値計算より確認する。

課題②
粒子間のせん断方向作用を追加し、数値計算より回転力の伝達を確認する。

課題③
粒子数の変更、初期条件の変更、重力項の追加などを行い、解析する。

応用課題①
壁モデルを導入し、反射や衝突による粒子の運動の減衰を確認する。

応用課題②（できなくてもよいが数値解が正しいかどうかを判断する上で非常に重要）
時間ステップや差分スキームを変更し、数値積分の誤差を評価する。

課題1～3のヒント

■課題１＆２

pp.24の該当部分にプログラムを追加。

（法線方向）

pe[i].en[j]：粒子i の粒子jから受ける力を更新して法線合力hnを求める。

↑ 変位unおよび速度velnと弾性係数knおよび粘性係数etanを用いる。

（接線方向）

pe[i].es[j]：粒子i の粒子jから受ける力を更新して接線合力hsを求める。

↑ 変位usおよび速度velsと弾性係数ksおよび粘性係数etasを用いる。

■課題3

pp.17のGに重力加速度の値を入力。

そうすると、pp.22下段の外力の計算がプログラムに反映される。

※ひとつひとつデバッグしながら慎重にプログラムを書き換える。

※エラーメッセージをよく読む。

課題4のヒント（その1）

■四方に壁を挿入することを想定した準備

1. Form1.hにおいてwa_n=4に変更する（pp.26参照）
2. Config関数の壁情報をコメントアウトする（pp.19参照）

```
/*←消去
//壁の初期条件 [四方が壁]
setParam(pe, pe_n  , n,false, 0.01, -45,   0, 0.5*PI, 0, 0, 0); //動かない& 壁
setParam(pe, pe_n+1, n,false, 0.01,  45,   0, 0.5*PI, 0, 0, 0);
setParam(pe, pe_n+2, n,false, 0.01,   0,  45, 0,       0, 0, 0);
setParam(pe, pe_n+3, n,false, 0.01,   0, -45, 0,       0, 0, 0);
*/←消去
```

※基本的に粒子と扱いはいっしょ。

```
//setParam(pe,  i, n, move,    r,   x,   y,phi,vx,vy,vphi);
```

※壁番号は粒子番号のつづきとなる。すなわち、n～n+3。

課題4のヒント（その2）

```
//粒子-壁間だった場合
for(j = pe_n; j < pe_n+wa_n ; j++){
    if(pe[j].exist == false) continue;

    //直線の式
    //la*x + lb*y + lc = 0
    double la =  sin(pe[j].phi);
    double lb = -cos(pe[j].phi);
    double lc = -(la*pe[j].x + lb*pe[j].y);
    double length_raw = la*pe[i].x + lb*pe[i].y + lc; //la^2 + lb^2  = 1です
    ld = abs(length_raw);

    //ぶつかっている場合
    if(pe[i].r+pe[j].r - ld > 0){
        if(length_raw < 0){
            cos_a = la;
            sin_a = lb;
        }else{
            cos_a = -la;
            sin_a = -lb;
        }
        Force2Par(pe, pe_n, i, j, ld, cos_a, sin_a, dt);
    }else{
        pe[i].en[j]=0.0;
        pe[i].es[j]=0.0;
    }
}
```

ここに書かれたコード全体をCalcForce関数の適切な場所に挿入することで、粒子と壁の相互作用が計算できるようになる。

以上、プログラムが上手く動いて、内容が理解できたらスタッフを呼んで出欠チェックをしてもらってください。