SpringSys

P. Baillehache

June 16, 2017

Contents

Introduction

SpringSys is a C library to calculate the equilibrium position of a spring-mass system in 2D.

It can calculate the equilibrium for a mix of fixed and non-fixed masses, and unbreakable springs. Masses are massless. SPrings are defined by their length at rest and their stiffness.

1 Mathematical solution

The problem of finding the equilibrium consists of finding the position of each non-fixed mass of the spring-mass system for which all the forces applied by the spring to these masses cancels.

Lets call M the set of masses of the system. k_s and l_s the strength and the resting length of spring s. To generalize the equations below we assume there is a spring between any combination of two masses, and we will set $k_s = 0$ for springs which doesn't actually exist in the system.

The equilibrium can be expressed as:

$$\forall m \in M, \sum_{n \in M} \overrightarrow{F_{mn}} = \overrightarrow{0} \tag{1}$$

where:

$$\begin{cases} \overrightarrow{F_{mn}} = k_{mn}(||\overrightarrow{mn}|| - r_{mn})\frac{\overrightarrow{mn}}{||\overrightarrow{mn}||} & \text{if } m \text{ is non-fixed} \\ \overrightarrow{F_{mn}} = \overrightarrow{0} & \text{if } m \text{ is fixed} \end{cases}$$

(1) is a set of non linear equations of $\overrightarrow{P_m}$, where $\overrightarrow{P_m}$ is the position vector of the mass m. These equations cannot be solved directly, then a solution will be approached using the Newton's method:

1. For each mass m, initialize $\overrightarrow{P_m}$, and lets call it $\overrightarrow{P_m^0}$.
2. Approximate $\overrightarrow{P_m^{i+1}}$ from $\overrightarrow{P_m^i}$
3. Repeat 2. until $||\sum_{m \in M} \sum_{n \in M} \overrightarrow{F_{mn}(\overrightarrow{P_m^{i+1}})}|| < \epsilon$

where the aproximation at step 2. is obtained as follow. We have

$$\overrightarrow{F(\overrightarrow{P_m^{i+1}})} \approx \overrightarrow{F(\overrightarrow{P_m^i})} + F'(\overrightarrow{P_m^i})(\overrightarrow{P_m^{i+1}} - \overrightarrow{P_m^i})$$

where F' is the 2x2 matrix:

$$F'(x,y) = \left[\begin{array}{cc} \frac{\overrightarrow{F(x+\epsilon,y)} - \overrightarrow{F(x-\epsilon,y)}}{2\epsilon}, & \frac{\overrightarrow{F(x,y+\epsilon)} - \overrightarrow{F(x,y-\epsilon)}}{2\epsilon} \end{array} \right]$$

We want

$$\overrightarrow{F(\overrightarrow{P_m^{i+1}})} = \overrightarrow{0}$$

equivalent to

$$\overrightarrow{F(\overrightarrow{P_m^i})} + F'(\overrightarrow{P_m^i})(\overrightarrow{P_m^{i+1}} - \overrightarrow{P_m^i}) \approx \overrightarrow{0}$$

thus

$$\overrightarrow{P_m^{i+1}} \approx \overrightarrow{P_m^i} - (F'(\overrightarrow{P_m^{i+1}}))^{-1} \overrightarrow{F(\overrightarrow{P_m^i})}$$

To avoid divergence, especially in system where all masses are non-fixed, it is beneficial to introduce a dampening coefficient D in the previous equation:

$$\overrightarrow{P_m^{i+1}} \approx \overrightarrow{P_m^i} - D(F'(\overrightarrow{P_m^{i+1}}))^{-1}\overrightarrow{F(P_m^i)}$$

D must be such as $0.0 \leq D \leq 1.0$, and is fixed to 0.5 in SpringSys.
The inverse of F' is obtained as follow:

$$F'^{-1} = \frac{1}{F'_{00}F'_{11} - F'_{01}F'_{10}} \begin{bmatrix} F'_{11} & -F'_{01} \\ -F'_{10} & F'_{00} \end{bmatrix}$$

2 Interface

```
// ============ SpringSys.h ================

#ifndef __SPRINGSYS_H
#define __SPRINGSYS_H

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

typedef struct {
  float _pos[2];
  float _nextPos[2];
  char _fixed;
} SpringSysMass;

typedef struct {
  float _k;
  float _r;
  int _mass[2];
} SpringSysSpring;

typedef struct {
  SpringSysMass *_masses;
  SpringSysSpring *_springs;
  int _nbMass;
  int _nbSpring;
} SpringSys;

// Create a new SpringSys
SpringSys* SpringSysCreate();

// Free the memory used by a SpringSys
void SpringSysFree(SpringSys *sys);

// Initialize the SpringSys
// mass is a vector of mass position (x,y),
// fixed is a vector of status (0: non-fixed mass, 1: fixed mass)
// spring is a vector of indices of mass linked by
// a spring (iMass, jMass)
// coeff is a vector of coefficient for the springs (k, r)
void SpringSysSet(SpringSys *sys, int nbMass, int nbSpring,
```

```
  float *mass, char *fixed, int *spring, float *coeff);

// Find the equilibrium position of masses
void SpringSysFindEquilibrium(SpringSys *sys);

#endif
```

3 Code

```
// ============ SpringSys.c ================

#include "SpringSys.h"

#define EPSILON 0.00001
#define DAMP 0.5

SpringSys* SpringSysCreate() {
  SpringSys *sys = (SpringSys*)malloc(sizeof(SpringSys));
  if (sys == NULL) return NULL;
  sys->_nbMass = 0;
  sys->_nbSpring = 0;
  sys->_masses = NULL;
  sys->_springs = NULL;
  return sys;
}

void SpringSysFree(SpringSys *sys) {
  if (sys == NULL) return;
  if (sys->_masses != NULL)
    free(sys->_masses);
  if (sys->_springs != NULL)
    free(sys->_springs);
  sys->_nbMass = 0;
  sys->_nbSpring = 0;
}

float SpringSysNormVec(float *v) {
  return sqrt(pow(v[0], 2.0) + pow(v[1], 2.0));
}

void SpringSysGetStressMass(SpringSys *sys, SpringSysMass *mass,
  int iMass, float *stress) {
  stress[0] = stress[1] = 0.0;
  if (mass->_fixed == 0)
    for (int iSpring = sys->_nbSpring; iSpring--;) {
      SpringSysSpring *spring = &(sys->_springs[iSpring]);
      SpringSysMass *massp = NULL;
      if (spring->_mass[0] == iMass)
        massp = &(sys->_masses[spring->_mass[1]]);
      else if (spring->_mass[1] == iMass)
        massp = &(sys->_masses[spring->_mass[0]]);
      if (massp != NULL) {
        float mn[2];
        mn[0] = massp->_pos[0] - mass->_pos[0];
        mn[1] = massp->_pos[1] - mass->_pos[1];
        float l = SpringSysNormVec(mn);
        if (fabs(l) > EPSILON) {
          float c = spring->_k * (l - spring->_r) / l;
          stress[0] += c * mn[0]; stress[1] += c * mn[1];
        }
```

```
      }
    }
}

float SpringSysGetStress(SpringSys *sys) {
  float stress = 0.0;
  for (int iMass = sys->_nbMass; iMass--;) {
    SpringSysMass *mass = &(sys->_masses[iMass]);
    float stressMass[2];
    SpringSysGetStressMass(sys, mass, iMass, stressMass);
    stress += SpringSysNormVec(stressMass);
  }
  return stress;
}

void SpringSysGetNextPosMass(SpringSys *sys, SpringSysMass *mass,
  int iMass) {
  float jacob[4];
  float stressp[2];
  float stressq[2];
  SpringSysMass m;
  m._fixed = 0;
  m._pos[0] = mass->_pos[0] + EPSILON;
  m._pos[1] = mass->_pos[1];
  SpringSysGetStressMass(sys, &m, iMass, stressp);
  m._pos[0] = mass->_pos[0] - EPSILON;
  m._pos[1] = mass->_pos[1];
  SpringSysGetStressMass(sys, &m, iMass, stressq);
  jacob[0] = (stressp[0] - stressq[0]) / (2.0 * EPSILON);
  jacob[2] = (stressp[1] - stressq[1]) / (2.0 * EPSILON);
  m._pos[0] = mass->_pos[0];
  m._pos[1] = mass->_pos[1] + EPSILON;
  SpringSysGetStressMass(sys, &m, iMass, stressp);
  m._pos[0] = mass->_pos[0];
  m._pos[1] = mass->_pos[1] - EPSILON;
  SpringSysGetStressMass(sys, &m, iMass, stressq);
  jacob[1] = (stressp[0] - stressq[0]) / (2.0 * EPSILON);
  jacob[3] = (stressp[1] - stressq[1]) / (2.0 * EPSILON);
  float det = jacob[0] * jacob[3] - jacob[1] * jacob[2];
  float stress[2];
  SpringSysGetStressMass(sys, mass, iMass, stress);
  if (fabs(det) > EPSILON) {
    float invJacob[4];
    invJacob[0] = jacob[3] / det;
    invJacob[1] = -1.0 * jacob[1] / det;
    invJacob[2] = -1.0 * jacob[2] / det;
    invJacob[3] = jacob[0] / det;
    // dampen to avoid divergence (especially when all masses are
    // non-fixed)
    mass->_nextPos[0] = mass->_pos[0] -
      (invJacob[0] * stress[0] + invJacob[1] * stress[1]) * DAMP;
    mass->_nextPos[1] = mass->_pos[1] -
      (invJacob[2] * stress[0] + invJacob[3] * stress[1]) * DAMP;
  }
}

void SpringSysFindEquilibrium(SpringSys *sys) {
  if (sys == NULL) return;
  float stress = 0.0;
  int nbIter = 0;
  int nbMaxIter = 1000;
  do {
```

```
    for (int iMass = sys->_nbMass; iMass--;) {
      SpringSysMass *mass = &(sys->_masses[iMass]);
      if (mass->_fixed == 0)
        SpringSysGetNextPosMass(sys, mass, iMass);
    }
    for (int iMass = sys->_nbMass; iMass--;) {
      SpringSysMass *mass = &(sys->_masses[iMass]);
      if (mass->_fixed == 0) {
        mass->_pos[0] = mass->_nextPos[0];
        mass->_pos[1] = mass->_nextPos[1];
      }
    }
    stress = SpringSysGetStress(sys);
    ++nbIter;
  } while (fabs(stress) > EPSILON && nbIter < nbMaxIter);
}

void SpringSysSet(SpringSys *sys, int nbMass, int nbSpring,
  float *mass, char *fixed, int *spring, float *coeff) {
  if (sys == NULL) return;
  sys->_masses =
    (SpringSysMass*)malloc(nbMass * sizeof(SpringSysMass));
  if (sys->_masses == NULL)
    return;
  sys->_nbMass = nbMass;
  sys->_springs =
    (SpringSysSpring*)malloc(nbSpring * sizeof(SpringSysSpring));
  if (sys->_springs == NULL)
    return;
  sys->_nbSpring = nbSpring;
  for (int iMass = 0; iMass < nbMass; ++iMass) {
    SpringSysMass *m = &(sys->_masses[iMass]);
    m->_pos[0] = mass[2 * iMass]; m->_pos[1] = mass[2 * iMass + 1];
    m->_fixed = fixed[iMass];
  }
  for (int iSpring = 0; iSpring < nbSpring; ++iSpring) {
    SpringSysSpring *s = &(sys->_springs[iSpring]);
    s->_mass[0] = spring[2 * iSpring];
    s->_mass[1] = spring[2 * iSpring + 1];
    s->_k = coeff[2 * iSpring];
    s->_r = coeff[2 * iSpring + 1];
  }
}
```

4 Makefile

```
OPTIONS_DEBUG=-ggdb -g3 -Wall
OPTIONS_RELEASE=-O3 -s
OPTIONS=$(OPTIONS_RELEASE) -lm

all : main

main: main.o SpringSys.o Makefile
        gcc -o main main.o SpringSys.o $(OPTIONS)

main.o : main.c SpringSys.h Makefile
        gcc -c main.c $(OPTIONS)

SpringSys.o : SpringSys.c SpringSys.h Makefile
        gcc -c SpringSys.c $(OPTIONS)
```

```
clean :
        rm -rf *.o main
```

5 Usage

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include "SpringSys.h"

int main(int argc, char **argv) {
  // Allocate memory for the system
  SpringSys *sys = SpringSysCreate();
  // Set the system
  int nbMass = 6;
  int nbSpring = 5;
  float mass[12] =
    {0.0, 0.0,
     0.0, 1.0,
     1.0, 1.0,
     1.0, 0.0,
     -0.5, -1.0,
     2.0, 2.0};
  char fixed[6] = {1, 1, 1, 1, 0, 0};
  int spring[10] =
    {0, 4,
     1, 4,
     4, 5,
     2, 5,
     3, 5};
  float coeff[10] =
    {1.0, 0.2,
     1.0, 0.2,
     1.0, 0.2,
     1.0, 0.2,
     1.0, 0.2};
  SpringSysSet(sys, nbMass, nbSpring, mass, fixed, spring, coeff);
  // Find the equilibrium
  SpringSysFindEquilibrium(sys);
  // Display the masses' position
  for (int iMass = 0; iMass < sys->_nbMass; ++iMass) {
    SpringSysMass *mass = &(sys->_masses[iMass]);
    printf("%f,%f\n", mass->_pos[0], mass->_pos[1]);
  }
  // Free memory
  SpringSysFree(sys);
  return 0;
}
```

Output:

```
0.000000,0.000000
0.000000,1.000000
1.000000,1.000000
1.000000,0.000000
0.243833,0.500000
0.756170,0.500000
```