

SpringSys

P. Baillehache

September 12, 2017

Contents

1	Definition	2
2	Interface	3
3	Code	6
4	Makefile	22
5	Usage	23
6	Output	32
7	springsys.dat	50
8	Plots	52

Introduction

SpringSys is a C library to simulate systems of masses connected by springs.

The number of dimensions of the system can be 1,2 or 3. It has a dissipation coefficient used to simulate dissipation of energy and dampen the system behaviour. A mass is defined by its mass, position and speed. A spring is defined by its rigidity coefficient, length (min, max, current and at rest), and the 2 masses it connects. A spring can be unbreakable or breakable (under stress limit condition).

SpringSys offers functions to create the system by adding/removing masses and springs or by cloning another SpringSys, to step in time the system, to step it until it reach equilibrium, to print it, to get the total stress and momentum of the system, to load and save the system to a text file, to get the nearest mass or spring to a given position.

1 Definition

The physics of the SpringSys is calculated as follow.

Lets call A and B two masses of the system, AB the spring connecting these two masses, and \overrightarrow{AB} the vector from the position of A to the position of B . Note that AB is the same spring as BA . Lets call R_{AB} the rest length of AB , $L_{AB}(t)$ the length of AB at time t , and k_{AB} the rigidity coefficient of AB . Then lets define the stress of AB at time t as:

$$S_{AB}(t) = k_{AB}(L_{AB}(t) - R_{AB})$$

Then lets define the stress on one mass A at time t as:

$$\overrightarrow{S}_A(t) = \sum_{m \in M} \left(E(Am) S_{Am}(t) \frac{\overrightarrow{Am}}{||\overrightarrow{Am}||} \right)$$

where M is the set of masses of the SpringSys and

$$E(Am) = \begin{cases} 1.0 & \text{if the spring } Am \text{ exists in the SpringSys} \\ 0.0 & \text{if the spring } Am \text{ doesn't exist in the SpringSys} \end{cases}$$

Note that $\overrightarrow{S}_A(t)$ corresponds to the acceleration of A due to the springs connected to it. Then the velocity $\overrightarrow{V}_A(t)$ of the mass A at time $t = t_0 + \delta t$ is defined by:

$$\overrightarrow{V}_A(t) = (1 - d)^{\delta t} \overrightarrow{V}_A(t_0) + \overrightarrow{S}_A(t_0) \delta t$$

where d is the dissipation coefficient of the SpringSys. And the position $\overrightarrow{P}_A(t)$ of the mass A at time $t = t_0 + \delta t$ is defined by:

$$\overrightarrow{P}_A(t) = \overrightarrow{P}_A(t_0) + \overrightarrow{V}_A(t_0) \delta t$$

2 Interface

```
// ===== SPRINGSYS.H =====

#ifndef SPRINGSYS_H
#define SPRINGSYS_H

// ===== Include =====

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include "gset.h"

// ===== Define =====

#define SPRINGSYS_EPSILON 0.0000001

// ===== Data structure =====

typedef struct SpringSysMass {
    // ID
    int _id;
    // Position
    float _pos[3];
    // Speed
    float _speed[3];
    // Stress (acceleration due to the springs)
    float _stress[3];
    // Mass
    float _mass;
    // Fixed flag, if true the mass doesn't move
    bool _fixed;
} SpringSysMass;

typedef struct SpringSysSpring {
    // ID
    int _id;
    // Current length
    float _length;
    // K coefficient
    float _k;
    // Length at rest
    float _restLength;
    // Stress (positive = extension, negative = compression)
    float _stress;
    // Limit stress (compression/extension)
    // If the current stress get over the limits the spring breaks (it
    // is removed from the list of springs)
    float _maxStress[2];
    // ID of the masses at the extremities of the spring
    int _mass[2];
    // Breakable flag, if true the spring breaks if its stress goes over
    // the limit _maxStress
    bool _breakable;
} SpringSysSpring;

typedef struct SpringSys {
    // List of masses
```

```

    GSet *_masses;
    // List of springs
    GSet *_springs;
    // Number of dimension of the system (in [1, 3])
    int _nbDim;
    // Dissipation coefficient (applied to speed of masses at each step,
    // 0.0 = no dissipation, 1.0 = total dissipation)
    float _dissip;
} SpringSys;

// ===== Functions declaration =====

// Create a new SpringSys with number of dimensions 'nbDim' (in [1,3])
// Default dissipation coefficient _dissip = 0.1
// Return NULL if we couldn't create the Springsys
SpringSys* SpringSysCreate(int nbDim);

// Clone the SpringSys 'sys'
// Return NULL if we couldn't clone the Springsys
SpringSys* SpringSysClone(SpringSys *sys);

// Load the SpringSys 'sys' from the stream 'stream'
// If 'sys' is already allocated, it is freed before loading
// Return 0 in case of success, or:
// 1: invalid arguments
// 2: can't allocate memory
// 3: invalid data
int SpringSysLoad(SpringSys **sys, FILE *stream);

// Save the SpringSys 'sys' to the stream
// Return 0 upon success, else
int SpringSysSave(SpringSys *sys, FILE *stream);

// Create a default mass, default properties' values are:
// _id = 0;
// _pos[0] = _pos[1] = _pos[2] = 0.0;
// _nextPos[0] = _nextPos[1] = _nextPos[2] = 0.0;
// _speed[0] = _speed[1] = _speed[2] = 0.0;
// _stress[0] = _stress[1] = _stress[2] = 0.0;
// _mass = 1.0;
// _fixed = false;
// Return NULL if memory allocation failed
SpringSysMass* SpringSysCreateMass(void);

// Create a default spring, default properties' values are:
// _id = 0;
// _length = 1.0;
// _k = 1.0;
// _restLength = 1.0;
// _stress = 0.0;
// _maxLength[0] = 0.0;
// _maxLength[1] = 100.0;
// _maxStress[0] = -1000000.0;
// _maxStress[1] = 1000000.0;
// _mass[0] = 0;
// _mass[1] = 0;
// _breakable = false
// Return NULL if memory allocation failed
SpringSysSpring* SpringSysCreateSpring(void);

// Free the memory used by a SpringSys
// Do nothing if arguments are invalid

```

```

void SpringSysFree(SpringSys **sys);

// Free the memory used by a SpringSysMass
// Do nothing if arguments are invalid
void SpringSysMassFree(SpringSysMass **m);

// Free the memory used by a SpringSysSpring
// Do nothing if arguments are invalid
void SpringSysSpringFree(SpringSysSpring **s);

// Print the SpringSys on 'stream'
// Do nothing if arguments are invalid
void SpringSysPrint(SpringSys *sys, FILE *stream);

// Print the SpringSysMass on 'stream'
// Do nothing if arguments are invalid
void SpringSysMassPrint(void *m, FILE *stream);

// Print the SpringSysSpring on 'stream'
// Do nothing if arguments are invalid
void SpringSysSpringPrint(void *s, FILE *stream);

// Set the dissipation coefficient of the SpringSys to 'dissip'
// in [0.0,1.0]
// Do nothing if arguments are invalid
void SpringSysSetDissip(SpringSys *sys, float dissip);

// Get the mass identified by 'id'
// Return NULL if arguments are invalid or if there is no mass
// with this id
SpringSysMass* SpringSysGetMass(SpringSys *sys, int id);

// Get the spring identified by 'id'
// Return NULL if arguments are invalid or if there is no spring
// with this id
SpringSysSpring* SpringSysGetSpring(SpringSys *sys, int id);

// Get the number of mass in the SpringSys
// Return -1 if the argument are invalid
int SpringSysGetNbMass(SpringSys *sys);

// Get the number of spring in the SpringSys
// Return -1 if the argument are invalid
int SpringSysGetNbSpring(SpringSys *sys);

// Add a copy of the mass 'm' to the SpringSys
// Return false if the arguments are invalid or memory allocation failed
// else return true
bool SpringSysAddMass(SpringSys *sys, SpringSysMass *m);

// Add a copy of the spring 's' to the SpringSys
// Return false if the arguments are invalid or memory allocation failed
// else return true
bool SpringSysAddSpring(SpringSys *sys, SpringSysSpring *s);

// Remove the mass identified by 'id'
// Springs connected to this mass are removed as well
// Do nothing if arguments are invalids
void SpringSysRemoveMass(SpringSys *sys, int id);

// Remove spring identified by 'id'
// Do nothing if argument are invalids

```

```

void SpringSysRemoveSpring(SpringSys *sys, int id);

// Step in time by 'dt' the SpringSys
// 'dt' must be carefully choosen, if too big inaccuracy of the
// simulation leads to divergence and then to rupture of springs,
// especially if springs have a high mk coefficient
// Do nothing if arguments are invalid
void SpringSysStep(SpringSys *sys, float dt);

// Step in time by 'dt' the SpringSys until it is in equilibrium
// or 'tMax' has been reached
// 'dt' must be carefully choosen, if too big inaccuracy of the
// simulation leads to divergence and then to rupture of springs,
// especially if springs have a high mk coefficient
// Return a value > tMax if the arguments are invalid or the equilibrium
// couldn't be reached, else return the time it took to
// reach equilibrium
float SpringSysStepToRest(SpringSys *sys, float dt, float tMax);

// Get the momentum (sum of norm(v) of masses) of the SpringSys
// Return 0.0 if the arguments are invalid
float SpringSysGetMomentum(SpringSys *sys);

// Get the stress (sum of abs(stress) of springs) of the SpringSys
// Return 0.0 if the arguments are invalid
float SpringSysGetStress(SpringSys *sys);

// Get the nearest mass to 'pos' in the SpringSys 'sys'
// Return NULL if arguments are invalids
SpringSysMass* SpringSysGetMassByPos(SpringSys *sys, float *pos);

// Get the nearest spring to 'pos' in the SpringSys 'sys'
// Return NULL if arguments are invalids
SpringSysSpring* SpringSysGetSpringByPos(SpringSys *sys, float *pos);

#endif

```

3 Code

```

// ===== SPRINGSYS.C =====

#include "springsys.h"

// ===== Include =====

// Create a new SpringSys with number of dimensions 'nbDim' (in [1,3])
// Default dissipation coefficient _dissip = 0.01
// Return NULL if we couldn't create the Springsys
SpringSys* SpringSysCreate(int nbDim) {
    // Check arguments
    if (nbDim < 1 || nbDim > 3)
        return NULL;
    // Allocate memory for the new SpringSys
    SpringSys *ret = (SpringSys*)malloc(sizeof(SpringSys));
    // If we could allocate memory
    if (ret != NULL) {
        // Set the number of dimensions
        ret->_nbDim = nbDim;
        // Set the dissipation coefficient
        ret->_dissip = 0.1;
    }
}

```

```

    // Create the gset of masses
    ret->_masses = GSetCreate();
    // If we couldn't create the gset
    if (ret->_masses == NULL) {
        // Free memory
        free(ret);
        // Return NULL
        return NULL;
    }
    // Create the gset of springs
    ret->_springs = GSetCreate();
    // If we couldn't create the gset
    if (ret->_springs == NULL) {
        // Free memory
        GSetFree(&(ret->_masses));
        free(ret);
        // Return NULL
        return NULL;
    }
}
return ret;
}

// Clone the SpringSys 'sys'
// Return NULL if we couldn't clone the Springsys
SpringSys* SpringSysClone(SpringSys *sys) {
    // Check arguments
    if (sys == NULL)
        return NULL;
    // Allocate memory for the new SpringSys
    SpringSys *ret = (SpringSys*)malloc(sizeof(SpringSys));
    // If we could allocate memory
    if (ret != NULL) {
        // Set the number of dimensions
        ret->_nbDim = sys->_nbDim;
        // Set the dissipation coefficient
        ret->_dissip = sys->_dissip;
        // Initialize the pointer to gsets of masses and springs
        ret->_masses = NULL;
        ret->_springs = NULL;
        // Create the gset of masses
        ret->_masses = GSetCreate();
        // If we couldn't create the gset
        if (ret->_masses == NULL) {
            // Free memory
            free(ret);
            // Return NULL
            return NULL;
        }
        // Create the gset of springs
        ret->_springs = GSetCreate();
        // If we couldn't create the gset
        if (ret->_springs == NULL) {
            // Free memory
            GSetFree(&(ret->_masses));
            free(ret);
            // Return NULL
            return NULL;
        }
    }
    // If there is a gset of masses
    if (sys->_masses != NULL) {
        // Copy the masses

```

```

GSetElem *m = sys->_masses->_head;
while (m != NULL) {
    // Declare a variable to create the clone of the mass
    SpringSysMass *mass = NULL;
    // If the mass is not null in the SpringSys
    if (m->_data != NULL) {
        // Allocate memory for the clone of the mass
        mass = (SpringSysMass*)malloc(sizeof(SpringSysMass));
        // If we couldn't allocate memory
        if (mass == NULL) {
            // Free the memory
            SpringSysFree(&ret);
            // Return NULL
            return NULL;
        }
        // Clone the mass
        memcpy(mass, m->_data, sizeof(SpringSysMass));
    }
    // Get the current number of masses
    int nbMass = ret->_masses->_nbElem;
    // Append the mass
    GSetAppend(ret->_masses, mass);
    // If we couldn't append the mass
    if (nbMass + 1 != ret->_masses->_nbElem) {
        // Free the memory
        free(mass);
        SpringSysFree(&ret);
        // Return NULL
        return NULL;
    }
    // Move to the next element
    m = m->_next;
}
}
// If there is a gset of springs
if (sys->_springs != NULL) {
    // Copy the springs
    GSetElem *s = sys->_springs->_head;
    while (s != NULL) {
        // Declare a variable to create the clone of the spring
        SpringSysSpring *spring = NULL;
        // If the spring is not null in the SpringSys
        if (s->_data != NULL) {
            // Allocate memory for the clone of the spring
            spring = (SpringSysSpring*)malloc(sizeof(SpringSysSpring));
            // If we couldn't allocate memory
            if (spring == NULL) {
                // Free the memory
                SpringSysFree(&ret);
                // Return NULL
                return NULL;
            }
            // Clone the spring
            memcpy(spring, s->_data, sizeof(SpringSysSpring));
        }
        // Get the current number of springs
        int nbSpring = ret->_springs->_nbElem;
        // Append the spring
        GSetAppend(ret->_springs, spring);
        // If we couldn't append the spring
        if (nbSpring + 1 != ret->_springs->_nbElem) {
            // Free the memory

```



```

        free(spring);
        SpringSysFree(&ret);
        // Return NULL
        return NULL;
    }
    // Move to the next element
    s = s->_next;
}
}
}
return ret;
}

// Load the SpringSys 'sys' from the stream 'stream'
// If 'sys' is already allocated, it is freed before loading
// Return 0 in case of success, or:
// 1: invalid arguments
// 2: can't allocate memory
// 3: invalid data
int SpringSysLoad(SpringSys **sys, FILE *stream) {
    // Check arguments
    if (sys == NULL || stream == NULL)
        return 1;
    // If the SpringSys is already allocated
    if (*sys != NULL)
        // Free memory
        SpringSysFree(sys);
    // Read the number of dimension
    int nbDim;
    int ret = fscanf(stream, "%d\n", &nbDim);
    // Allocate memory for the SpringSys
    *sys = SpringSysCreate(nbDim);
    // If we couldn't allocate memory
    if (*sys == NULL)
        // Stop here
        return 2;
    // Create a mass to read the data
    SpringSysMass *mass = SpringSysCreateMass();
    if (mass == NULL) {
        SpringSysFree(sys);
        return 2;
    }
    // Create a spring to read the data
    SpringSysSpring *spring = SpringSysCreateSpring();
    if (spring == NULL) {
        SpringSysMassFree(&mass);
        SpringSysFree(sys);
        return 2;
    }
    // Read the number of mass
    int nbMass;
    ret = fscanf(stream, "%d\n", &nbMass);
    // For each mass
    for (int iMass = 0; iMass < nbMass; ++iMass) {
        // Read the properties of the mass
        ret = fscanf(stream, "%d\n", &(mass->_id));
        ret = fscanf(stream, "%f %f %f\n", &(mass->_pos[0]),
            &(mass->_pos[1]), &(mass->_pos[2]));
        ret = fscanf(stream, "%f %f %f\n", &(mass->_speed[0]),
            &(mass->_speed[1]), &(mass->_speed[2]));
        ret = fscanf(stream, "%f %f %f\n", &(mass->_stress[0]),
            &(mass->_stress[1]), &(mass->_stress[2]));
    }
}

```

```

    ret = fscanf(stream, "%f\n", &(mass->_mass));
    int b;
    ret = fscanf(stream, "%d\n", &b);
    mass->_fixed = b;
    // Add the mass
    bool ok = SpringSysAddMass(*sys, mass);
    // If we couldn't add the mass
    if (ok == false) {
        SpringSysMassFree(&mass);
        SpringSysSpringFree(&spring);
        SpringSysFree(sys);
        return 3;
    }
}
// Read the number of spring
int nbSpring;
ret = fscanf(stream, "%d\n", &nbSpring);
// For each spring
for (int iSpring = 0; iSpring < nbSpring; ++iSpring) {
    // Read the properties of the spring
    ret = fscanf(stream, "%d\n", &(spring->_id));
    ret = fscanf(stream, "%f\n", &(spring->_length));
    ret = fscanf(stream, "%f\n", &(spring->_k));
    ret = fscanf(stream, "%f\n", &(spring->_restLength));
    ret = fscanf(stream, "%f\n", &(spring->_stress));
    ret = fscanf(stream, "%f %f\n", &(spring->_maxStress[0]),
        &(spring->_maxStress[1]));
    ret = fscanf(stream, "%d %d\n", &(spring->_mass[0]),
        &(spring->_mass[1]));
    int b;
    ret = fscanf(stream, "%d\n", &b);
    spring->_breakable = b;
    // Add the spring
    bool ok = SpringSysAddSpring(*sys, spring);
    // If we couldn't add the spring
    if (ok == false) {
        SpringSysMassFree(&mass);
        SpringSysSpringFree(&spring);
        SpringSysFree(sys);
        return 3;
    }
}
// TODO don't ignore the return value
ret = ret;
// Free memory
SpringSysMassFree(&mass);
SpringSysSpringFree(&spring);
// Return success code
return 0;
}

// Save the SpringSys 'sys' to the stream
// Return 0 upon success, else
// 1: invalid argument
// 2: invalid SpringSys
int SpringSysSave(SpringSys *sys, FILE *stream) {
    // Check arguments
    if (sys == NULL || sys->_masses == NULL ||
        sys->_springs == NULL || stream == NULL)
        return 1;
    // Write the number of dimensions
    fprintf(stream, "%d\n", sys->_nbDim);

```

```

// Write the number of masses
fprintf(stream, "%d\n", sys->_masses->_nbElem);
// Get a pointer to the first element of the list of masses
GSetElem *e = sys->_masses->_head;
// While we are not at the end of the list
while (e != NULL) {
    // Get a pointer to the mass
    SpringSysMass *m = (SpringSysMass*)(e->_data);
    // If the pointer is not null
    if (m != NULL) {
        // Write the mass properties
        fprintf(stream, "%d\n", m->_id);
        fprintf(stream, "%f %f %f\n", m->_pos[0], m->_pos[1], m->_pos[2]);
        fprintf(stream, "%f %f %f\n", m->_speed[0], m->_speed[1],
            m->_speed[2]);
        fprintf(stream, "%f %f %f\n", m->_stress[0], m->_stress[1],
            m->_stress[2]);
        fprintf(stream, "%f\n", m->_mass);
        fprintf(stream, "%d\n", m->_fixed);
    } else {
        // Else, the pointer is null
        // This should never happen
        return 2;
    }
    // Move to the next element
    e = e->_next;
}
// Write the number of springs
fprintf(stream, "%d\n", sys->_springs->_nbElem);
// Get a pointer to the first element of the list of springs
e = sys->_springs->_head;
// While we are not at the end of the list
while (e != NULL) {
    // Get a pointer to the spring
    SpringSysSpring *s = (SpringSysSpring*)(e->_data);
    // If the pointer is not null
    if (s != NULL) {
        // Write the spring properties
        fprintf(stream, "%d\n", s->_id);
        fprintf(stream, "%f\n", s->_length);
        fprintf(stream, "%f\n", s->_k);
        fprintf(stream, "%f\n", s->_restLength);
        fprintf(stream, "%f\n", s->_stress);
        fprintf(stream, "%f %f\n", s->_maxStress[0], s->_maxStress[1]);
        fprintf(stream, "%d %d\n", s->_mass[0], s->_mass[1]);
        fprintf(stream, "%d\n", s->_breakable);
    } else {
        // Else, the pointer is null
        // This should never happen
        return 2;
    }
    // Move to the next element
    e = e->_next;
}
return 0;
}

// Create a default mass
// Return NULL if memory allocation failed
SpringSysMass* SpringSysCreateMass(void) {
    // Allocate memory
    SpringSysMass *ret = (SpringSysMass*)malloc(sizeof(SpringSysMass));

```

```

// If we could allocate memory
if (ret != NULL) {
    // Set default values for the properties
    ret->_id = 0;
    ret->_pos[0] = ret->_pos[1] = ret->_pos[2] = 0.0;
    ret->_speed[0] = ret->_speed[1] = ret->_speed[2] = 0.0;
    ret->_stress[0] = ret->_stress[1] = ret->_stress[2] = 0.0;
    ret->_mass = 1.0;
    ret->_fixed = false;
}
// Return the new mass
return ret;
}

// Create a default spring
// Return NULL if memory allocation failed
SpringSysSpring* SpringSysCreateSpring(void) {
    // Allocate memory
    SpringSysSpring *ret =
        (SpringSysSpring*)malloc(sizeof(SpringSysSpring));
    // If we could allocate memory
    if (ret != NULL) {
        // Set default values for the properties
        ret->_id = 0;
        ret->_length = 1.0;
        ret->_k = 1.0;
        ret->_restLength = 1.0;
        ret->_stress = 0.0;
        ret->_maxStress[0] = -1000000.0;
        ret->_maxStress[1] = 1000000.0;
        ret->_mass[0] = 0;
        ret->_mass[1] = 0;
        ret->_breakable = false;
    }
    // Return the new spring
    return ret;
}

// Free the memory used by a SpringSys
// Do nothing if arguments are invalid
void SpringSysFree(SpringSys **sys) {
    // Check arguments
    if (sys == NULL || *sys == NULL)
        return;
    // If there is a gset of masses
    if ((*sys)->_masses != NULL) {
        // Free the memory used by masses
        GSetElem *m = (*sys)->_masses->_head;
        while (m != NULL) {
            SpringSysMassFree((SpringSysMass**>(&(m->_data)));
            m = m->_next;
        }
    }
    // If there is a gset of springs
    if ((*sys)->_springs != NULL) {
        // Free the memory used by springs
        GSetElem *s = (*sys)->_springs->_head;
        while (s != NULL) {
            SpringSysSpringFree((SpringSysSpring**>(&(s->_data)));
            s = s->_next;
        }
    }
}

```

```

    // Free the gsets
    GSetFree(&((*sys)->_masses));
    GSetFree(&((*sys)->_springs));
    // Free memory
    free(*sys);
    *sys = NULL;
}

// Free the memory used by a SpringSysMass
// Do nothing if arguments are invalid
void SpringSysMassFree(SpringSysMass **m) {
    // Check arguments
    if (m == NULL || *m == NULL)
        return;
    // Free the memory
    free(*m);
    *m = NULL;
}

// Free the memory used by a SpringSysSpring
// Do nothing if arguments are invalid
void SpringSysSpringFree(SpringSysSpring **s) {
    // Check arguments
    if (s == NULL || *s == NULL)
        return;
    // Free the memory
    free(*s);
    *s = NULL;
}

// Print the SpringSys on 'stream'
// Do nothing if arguments are invalid
void SpringSysPrint(SpringSys *sys, FILE *stream) {
    // Check arguments
    if (sys == NULL || stream == NULL)
        return;
    // Print the number of dimension
    fprintf(stream, "Number of dimension: %d\n", sys->_nbDim);
    // Print the dissipation
    fprintf(stream, "Dissipation: %.3f\n", sys->_dissip);
    // Print the masses
    fprintf(stream, "Masses:\n");
    GSetPrint(sys->_masses, stream, &SpringSysMassPrint, (char*)"");
    fprintf(stream, "\n");
    // Print the springs
    fprintf(stream, "Springs:\n");
    GSetPrint(sys->_springs, stream, &SpringSysSpringPrint, (char*)"");
    fprintf(stream, "\n");
}

// Print the SpringSysMass on 'stream'
// Do nothing if arguments are invalid
void SpringSysMassPrint(void *m, FILE *stream) {
    // Check arguments
    if (m == NULL || stream == NULL)
        return;
    // Print the mass properties
    fprintf(stream, "%#d, ", ((SpringSysMass*)m)->_id);
    fprintf(stream, "pos(%.3f,%.3f,%.3f), ",
        ((SpringSysMass*)m)->_pos[0], ((SpringSysMass*)m)->_pos[1],
        ((SpringSysMass*)m)->_pos[2]);
    fprintf(stream, "speed(%.3f,%.3f,%.3f), ",

```

```

        ((SpringSysMass*)m)->_speed[0], ((SpringSysMass*)m)->_speed[1],
        ((SpringSysMass*)m)->_speed[2]);
    fprintf(stream, "stress(%.3f,%.3f,%.3f), ",
        ((SpringSysMass*)m)->_stress[0], ((SpringSysMass*)m)->_stress[1],
        ((SpringSysMass*)m)->_stress[2]);
    fprintf(stream, "mass(%.3f), ", ((SpringSysMass*)m)->_mass);
    fprintf(stream, "fixed(%d)", ((SpringSysMass*)m)->_fixed);
}

// Print the SpringSysSpring on 'stream'
// Do nothing if arguments are invalid
void SpringSysSpringPrint(void *s, FILE *stream) {
    // Check arguments
    if (s == NULL || stream == NULL)
        return;
    // Print the spring properties
    fprintf(stream, "%d, ", ((SpringSysSpring*)s)->_id);
    fprintf(stream, "%d-%d, ", ((SpringSysSpring*)s)->_mass[0],
        ((SpringSysSpring*)s)->_mass[1]);
    fprintf(stream, "length(%.3f), ", ((SpringSysSpring*)s)->_length);
    fprintf(stream, "stress(%.3f), ", ((SpringSysSpring*)s)->_stress);
    fprintf(stream, "k(%.3f), ", ((SpringSysSpring*)s)->_k);
    fprintf(stream, "restLength(%.3f), ",
        ((SpringSysSpring*)s)->_restLength);
    fprintf(stream, "maxStress(%.3f,%.3f), ",
        ((SpringSysSpring*)s)->_maxStress[0],
        ((SpringSysSpring*)s)->_maxStress[1]);
    fprintf(stream, "breakable(%d)", ((SpringSysSpring*)s)->_breakable);
}

// Set the dissipation coefficient of the SpringSys to 'dissip'
// in [0.0,1.0]
// Do nothing if arguments are invalid
void SpringSysSetDissip(SpringSys *sys, float dissip) {
    // Check arguments
    if (sys == NULL || dissip < 0.0 || dissip > 1.0)
        return;
    // Set the dissipation
    sys->_dissip = dissip;
}

// Get the mass identified by 'id'
// Return NULL if arguments are invalid or if there is no mass
// with this id
SpringSysMass* SpringSysGetMass(SpringSys *sys, int id) {
    // Check arguments
    if (sys == NULL)
        return NULL;
    // Declare a pointer to memorize the searched mass
    SpringSysMass *ret = NULL;
    // Get a pointer to the first element in list of masses
    GSetElem *m = sys->_masses->_head;
    // While we are not at the end of the list
    while (m != NULL) {
        // If the current mass is the searched mass
        if (((SpringSysMass*)(m->_data))->_id == id) {
            // Update the result pointer
            ret = (SpringSysMass*)(m->_data);
            // Set the pointer to element to null to end the loop
            m = NULL;
        }
        // Else, the current ass is not the searched mass
    }
    return ret;
}

```

```

        // Move to the next element
        m = m->_next;
    }
    // Return the result pointer
    return ret;
}

// Get the spring identified by 'id'
// Return NULL if arguments are invalid or if there is no spring
// with this id
SpringSysSpring* SpringSysGetSpring(SpringSys *sys, int id) {
    // Check arguments
    if (sys == NULL || sys->_springs == NULL)
        return NULL;
    // Declare a pointer to memorize the searched spring
    SpringSysSpring *ret = NULL;
    // Get a pointer to the first element of the list of springs
    GSetElem *m = sys->_springs->_head;
    // While we are not at the end of the list
    while (m != NULL) {
        // If the current spring is the searched one
        if (((SpringSysSpring*)(m->_data))->_id == id) {
            // Update the result pointer
            ret = (SpringSysSpring*)(m->_data);
            // Set the pointer to element to null to end the loop
            m = NULL;
        } else {
            // Move to the next element
            m = m->_next;
        }
    }
    // Return the result pointer
    return ret;
}

// Get the number of mass in the SpringSys
// Return -1 if the argument are invalid
int SpringSysGetNbMass(SpringSys *sys) {
    // Check arguments
    if (sys == NULL || sys->_masses == NULL)
        return -1;
    // Return the number of masses
    return sys->_masses->_nbElem;
}

// Get the number of spring in the SpringSys
// Return -1 if the argument are invalid
int SpringSysGetNbSpring(SpringSys *sys) {
    // Check arguments
    if (sys == NULL || sys->_springs == NULL)
        return -1;
    // Return the number of springs
    return sys->_springs->_nbElem;
}

// Add a copy of the mass 'm' to the SpringSys
// Return false if the arguments are invalid or memory allocation failed
// else return true
bool SpringSysAddMass(SpringSys *sys, SpringSysMass *m) {
    // Check arguments
    if (sys == NULL || m == NULL)

```

```

        return false;
// If the mass properties are incorrect
if (m->_mass < 0.0)
    // Return false
    return false;
// Allocate memory for the new mass
SpringSysMass *mass = (SpringSysMass*)malloc(sizeof(SpringSysMass));
// If we could allocate memory
if (mass != NULL) {
    // Copy the properties of the mass
    memcpy(mass, m, sizeof(SpringSysMass));
    // Add the mass
    GSetAppend(sys->_masses, mass);
// Else, we couldn't allocate the memory
} else
    // Return false
    return false;
// Return true
return true;
}

// Add a copy of the spring 's' to the SpringSys
// Return false if the arguments are invalid or memory allocation failed
// else return true
bool SpringSysAddSpring(SpringSys *sys, SpringSysSpring *s) {
    // Check arguments
    if (sys == NULL || s == NULL)
        return false;
// If the spring properties are incorrect
if (s->_mass[0] == s->_mass[1] || s->_length < 0.0 ||
    s->_k < 0.0 || s->_restLength < 0.0 ||
    s->_maxStress[0] >= 0.0 || s->_maxStress[1] <= 0.0)
    // Return false
    return false;
SpringSysMass *m[2];
m[0] = SpringSysGetMass(sys, s->_mass[0]);
m[1] = SpringSysGetMass(sys, s->_mass[1]);
if (m[0] == NULL || m[1] == NULL)
    // Return false
    return false;
// Allocate memory for the new spring
SpringSysSpring *spring =
    (SpringSysSpring*)malloc(sizeof(SpringSysSpring));
// If we could allocate memory
if (spring != NULL) {
    // Copy the properties of the spring
    memcpy(spring, s, sizeof(SpringSysSpring));
    // Add the spring to the list of springs
    GSetAppend(sys->_springs, spring);
// Else, we couldn't allocate the memory
} else
    // Return false
    return false;
// Return true
return true;
}

// Remove the mass identified by 'id'
// Springs connected to this mass are removed as well
// Do nothing if arguments are invalids
void SpringSysRemoveMass(SpringSys *sys, int id) {
    // Check arguments

```



```

if (sys == NULL)
    return;
// Get a pointer to the first element in the list of mass
GSetElem *e = sys->_masses->_head;
// While we are not at the end of the list
while (e != NULL) {
    // Get a pointer to the mass
    SpringSysMass *m = (SpringSysMass*)(e->_data);
    // Move to next mass
    e = e->_next;
    // If the pointer is not null and this is the searched mass
    if (m != NULL && m->_id == id) {
        // Remove the element
        GSetRemoveFirst(sys->_masses, m);
        // Free memory used by the mass
        SpringSysMassFree(&m);
    }
}
// Get a pointer to the first element in the list of spring
e = sys->_springs->_head;
// While we are not at the end of the list
while (e != NULL) {
    // Get a pointer to the spring
    SpringSysSpring *s = (SpringSysSpring*)(e->_data);
    // Move to next spring
    e = e->_next;
    // If the pointer is not null and this is a spring connected to
    // the searched mass
    if (s != NULL && (s->_mass[0] == id || s->_mass[1] == id)) {
        // Remove the element
        GSetRemoveFirst(sys->_springs, s);
        // Free memory used by the spring
        SpringSysSpringFree(&s);
    }
}
}
}

// Remove spring idenitfied by 'id'
// Do nothing if argument are invalids
void SpringSysRemoveSpring(SpringSys *sys, int id) {
    // Check arguments
    if (sys == NULL)
        return;
    // Get a pointer to the first element in the list of spring
    GSetElem *e = sys->_springs->_head;
    // While we are not at the end of the list
    while (e != NULL) {
        // Get a pointer to the spring
        SpringSysSpring *s = (SpringSysSpring*)(e->_data);
        // Move to next spring
        e = e->_next;
        // If the pointer is not null and this is the searched spring
        if (s != NULL && s->_id == id) {
            // Remove the element
            GSetRemoveFirst(sys->_springs, s);
            // Free memory used by the spring
            SpringSysSpringFree(&s);
        }
    }
}

// Step in time by 'dt' the SpringSys

```

```

// Do nothing if arguments are invalid
void SpringSysStep(SpringSys *sys, float dt) {
    // Check arguments
    if (sys == NULL || dt <= 0.0 || sys->_masses == NULL ||
        sys->_springs == NULL)
        return;
    // Reset the stress for each unfixed mass
    // Get a pointer to the first element in the list of mass
    GSetElem *e = sys->_masses->_head;
    // While we are not at the end of the list
    while (e != NULL) {
        // Get a pointer to the mass
        SpringSysMass *m = (SpringSysMass*)(e->_data);
        // If the pointer is not null and the mass is not fixed
        if (m != NULL && m->_fixed == false)
            // For each dimension
            for (int iDim = 0; iDim < sys->_nbDim; ++iDim)
                // Reset the stress
                m->_stress[iDim] = 0.0;
        // Move to next mass
        e = e->_next;
    }
    // Update length and stress of each springs
    e = sys->_springs->_head;
    while (e != NULL) {
        // Get a pointer to the spring
        SpringSysSpring *s = (SpringSysSpring*)(e->_data);
        // If the pointer is not null
        if (s != NULL) {
            // Declare a variable to memorize if there has been a rupture
            bool flagRupture = false;
            // Get the two masses at extremities of the spring
            SpringSysMass* m[2];
            m[0] = SpringSysGetMass(sys, s->_mass[0]);
            m[1] = SpringSysGetMass(sys, s->_mass[1]);
            // If both masses are not null
            if (m[0] != NULL && m[1] != NULL) {
                // Get the distance between the masses
                float l = 0.0;
                for (int iDim = 0; iDim < sys->_nbDim; ++iDim)
                    l += pow(m[0]->_pos[iDim] - m[1]->_pos[iDim], 2.0);
                s->_length = sqrt(l);
                // Get the stress
                s->_stress = (s->_length - s->_restLength) * s->_k;
                // If the spring is breakable, check for rupture
                if (s->_breakable == true &&
                    ((s->_stress > 0.0 && s->_stress >= s->_maxStress[1]) ||
                     (s->_stress < 0.0 && s->_stress <= s->_maxStress[0]))) {
                    // Memorize there has been a rupture
                    flagRupture = true;
                    // Remove this spring from the sets of spring
                    GSetRemoveFirst(sys->_springs, s);
                    // Free memory for the spring
                    SpringSysSpringFree(&s);
                } else {
                    // Update the stress to the masses which are not fixed
                    for (int iDim = 0; iDim < sys->_nbDim; ++iDim) {
                        for (int iMass = 0; iMass < 2; ++iMass) {
                            float d = s->_length * (1.0 + m[iMass]->_mass);
                            if (m[iMass]->_fixed == false && d > SPRINGSYS_EPSILON)
                                m[iMass]->_stress[iDim] += s->_stress *
                                    (m[1 - iMass]->_pos[iDim] - m[iMass]->_pos[iDim]) / d;
                        }
                    }
                }
            }
        }
        e = e->_next;
    }
}

```

```

        }
    }
}
// If there has been no rupture
if (flagRupture == false) {
    // Move to the next spring
    e = e->_next;
}
// Else, the pointer is yet on the following element
// Else, the pointer to the spring is null
} else {
    // Move to the next element in list
    e = e->_next;
}
}
// Apply speed to masses which are not fixed
// Get a pointer to the first element of the list of mass
e = sys->_masses->_head;
// While we are not at the end of the list
while (e != NULL) {
    // Get a pointer to the mass
    SpringSysMass *m = (SpringSysMass*)(e->_data);
    // If the pointer is not null and the mass is not fixed
    if (m != NULL && m->_fixed == false)
        // For each dimension
        for (int iDim = 0; iDim < sys->_nbDim; ++iDim) {
            // Apply the dissipation to the speed
            m->_speed[iDim] *= pow(1.0 - sys->_dissip, dt);
            // Apply the stress to the speed
            m->_speed[iDim] += m->_stress[iDim] * dt;
            // Apply the speed to the position
            m->_pos[iDim] += m->_speed[iDim] * dt;
        }
    // Move to next mass
    e = e->_next;
}
}

// Step in time by 'dt' the SpringSys until it is in equilibrium
// or 'tMax' has been reached
// Return a value > tMax if the arguments are invalid or the equilibrium
// couldn't be reached, else return the time it took to
// reach equilibrium
float SpringSysStepToRest(SpringSys *sys, float dt, float tMax) {
    // Declare a variable to memorize time
    float t = tMax + dt;
    // If arguments are valid
    if (sys != NULL && dt > 0.0 && tMax > dt) {
        // Declare a variable to memorize the momentum of the system
        float m = 0.0;
        // Declare variables to memorize the stress of the system at current
        // step and previous step
        float s = 0.0;
        float sp = 0.0;
        // Loop until the momentum is null and the stress stops varying or
        // tMax is reached
        t = 0.0;
        do {
            // Update current stress
            s = sp;
            // Step the SpringSys

```

```

        SpringSysStep(sys, dt);
        // Get the momentum
        m = SpringSysGetMomentum(sys);
        // Get the stress
        sp = SpringSysGetStress(sys);
        // Increment time
        t += dt;
    } while ((m > SPRINGSYS_EPSILON ||
        fabs(sp - s) > SPRINGSYS_EPSILON) && t <= tMax);
}
// Return the time
return t;
}

// Get the momentum (sum of norm(v) of masses) of the SpringSys
// Return 0.0 if the arguments are invalid
float SpringSysGetMomentum(SpringSys *sys) {
    // Check arguments
    if (sys == NULL || sys->_masses == NULL)
        return 0.0;
    // Declare a variable to memorize the sum
    float sum = 0.0;
    // Declare a pointer to the first element of the list of masses
    GSetElem *e = sys->_masses->_head;
    // While we are not at the end of the list
    while (e != NULL) {
        // Declare a pointer to the mass
        SpringSysMass *m = (SpringSysMass*)(e->_data);
        // If the pointer is not null
        if (m != NULL) {
            // Declare a variable to calculate the norm of the speed
            // of the mass
            float v = 0.0;
            // Calculate the norm of the speed of the mass and sum it
            for (int iDim = 0; iDim < sys->_nbDim; ++iDim)
                v += pow(m->_speed[iDim], 2.0);
            sum += sqrt(v);
        }
        // Move to the next mass
        e = e->_next;
    }
    // Return the sum
    return sum;
}

// Get the stress (sum of abs(stress) of springs) of the SpringSys
// Return 0.0 if the arguments are invalid
float SpringSysGetStress(SpringSys *sys) {
    // Check arguments
    if (sys == NULL || sys->_springs == NULL)
        return 0.0;
    // Declare a variable to memorize the sum
    float sum = 0.0;
    // Declare a pointer to the first element of the list of springs
    GSetElem *e = sys->_springs->_head;
    // While we are not at the end of the list
    while (e != NULL) {
        // Get a pointer toward the spring
        SpringSysSpring *s = (SpringSysSpring*)(e->_data);
        // If the pointer is not null
        if (s != NULL)
            // Add the absolute value of the stress of the spring

```

```

        sum += fabs(s->_stress);
        // Move to the next spring
        e = e->_next;
    }
    // Return the sum
    return sum;
}

// Get the nearest mass to 'pos' in the SpringSys 'sys'
// Return NULL if arguments are invalids
SpringSysMass* SpringSysGetMassByPos(SpringSys *sys, float *pos) {
    // Check arguments
    if (sys == NULL || pos == NULL || sys->_masses == NULL)
        return NULL;
    // Declare a pointer to memorize the nearest mass
    SpringSysMass *ret = NULL;
    // Declare a variable to memorize the distance to nearest mass
    float d = 0.0;
    // Declare a pointer to the first element of the list of masses
    GSetElem *e = sys->_masses->_head;
    // While we are not at the end of the list
    while (e != NULL) {
        // Declare a pointer to the mass
        SpringSysMass *m = (SpringSysMass*)(e->_data);
        // If the pointer is not null
        if (m != NULL) {
            // Declare a variable to calculate the distance
            float v = 0.0;
            // Calculate the distance
            for (int iDim = 0; iDim < sys->_nbDim; ++iDim)
                v += pow(m->_pos[iDim] - pos[iDim], 2.0);
            v = sqrt(v);
            // If the distance is shorter than the current one
            if (ret == NULL || d > v) {
                // Update the distance
                d = v;
                // Update the nearest mass
                ret = m;
            }
        }
        // Move to the next mass
        e = e->_next;
    }
    // Return the nearest mass
    return ret;
}

// Get the nearest spring to 'pos' in the SpringSys 'sys'
// Return NULL if arguments are invalids
SpringSysSpring* SpringSysGetSpringByPos(SpringSys *sys, float *pos) {
    // Check arguments
    if (sys == NULL || pos == NULL || sys->_springs == NULL ||
        sys->_masses == NULL)
        return NULL;
    // Declare a pointer to memorize the nearest spring
    SpringSysSpring *ret = NULL;
    // Declare a variable to memorize the distance to nearest mass
    float d = 0.0;
    // Declare a pointer to the first element of the list of springs
    GSetElem *e = sys->_springs->_head;
    // While we are not at the end of the list
    while (e != NULL) {

```

```

// Get a pointer toward the spring
SpringSysSpring *s = (SpringSysSpring*)(e->_data);
// If the pointer is not null
if (s != NULL) {
    // Get the two masses at the extremity of the spring
    SpringSysMass *mA = SpringSysGetMass(sys, s->_mass[0]);
    SpringSysMass *mB = SpringSysGetMass(sys, s->_mass[1]);
    if (mA != NULL && mB != NULL) {
        // Declare a variable to memorize the center of the spring
        float center[2];
        // Calculate the center of the spring
        center[0] = 0.5 * (mA->_pos[0] + mB->_pos[0]);
        center[1] = 0.5 * (mA->_pos[1] + mB->_pos[1]);
        // Declare a variable to calculate the distance
        float v = 0.0;
        // Calculate the distance
        for (int iDim = 0; iDim < sys->_nbDim; ++iDim)
            v += pow(center[iDim] - pos[iDim], 2.0);
        v = sqrt(v);
        // If the distance is shorter than the current one
        if (ret == NULL || d > v) {
            // Update the distance
            d = v;
            // Update the nearest spring
            ret = s;
        }
    }
}
// Move to the next spring
e = e->_next;
}
// Return the nearest spring
return ret;
}

```

4 Makefile

```

OPTIONS_DEBUG=-ggdb -g3 -Wall
OPTIONS_RELEASE=-O3
OPTIONS=$(OPTIONS_DEBUG)
INCPATH=./Include
LIBPATH=./Include

all : main

main: main.o springsys.o $(LIBPATH)/tgapaint.o $(LIBPATH)/gset.o Makefile
    gcc $(OPTIONS) main.o springsys.o $(LIBPATH)/tgapaint.o $(LIBPATH)/gset.o -o main -lm

main.o : main.c springsys.h Makefile
    gcc $(OPTIONS) -I$(INCPATH) -c main.c

springsys.o : springsys.c springsys.h $(INCPATH)/gset.h Makefile
    gcc $(OPTIONS) -I$(INCPATH) -c springsys.c

clean :
    rm -rf *.o main

video:
    avconv -r 25 -i ./Frames/%02d.tga -b:v 2048k video.mp4

```

5 Usage

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

// Function to draw the SpringSys of the first example in one dimension
void DrawLegendTGA_1D(SpringSys *sys, TGA *tga, float t, float dt,
    float tMax, float lMax, float lPixel, int margin) {
    // Create a pencil with black color
    TGAPencil *pen = TGAGetPencil();
    TGAPixel *black = TGAGetBlackPixel();
    TGAPencilSetColor(pen, black);
    TGAPencilSetAntialias(pen, true);
    TGAFreePixel(&black);
    // Create a font
    TGAFont *font = TGAFontCreate(tgaFontDefault);
    TGAFontSetSize(font, 0.5 * (float)margin);
    // Declare variables for tracing
    float p[2];
    char s[100] = {0};
    float q[2];
    // Draw the absciss
    q[1] = p[1] = 0.25 * (float)margin;
    for (p[0] = 0.0; p[0] < tMax; p[0] += 5.0) {
        // Update the label
        sprintf(s, "%.1f", p[0]);
        // Print the label
        q[0] = p[0] / dt + 0.5 * (float)margin;
        TGAPrintString(tga, pen, font, (unsigned char*)s, q);
    }
    // Draw the ordinate
    q[0] = p[0] = 0.25 * (float)margin;
    for (p[1] = 0.0; p[1] < lMax; p[1] += 1.0) {
        // Update the label
        sprintf(s, "%d", (int)round(p[1]));
        // Print the label
        q[1] = p[1] / lPixel + (float)margin;
        TGAPrintString(tga, pen, font, (unsigned char*)s, q);
    }
    p[0] = q[0] = (float)margin;
    p[1] = (float)margin;
    q[1] = lMax / lPixel + (float)margin;
    TGAPencilSetShapePixel(pen);
    TGAPencilSetAntialias(pen, false);
    TGADrawLine(tga, p, q, pen);
    // Free the pencil
    TGAFreePencil(&pen);
    // Free the font
    TGAFreeFont(&font);
}

// Function to draw the SpringSys of the first example in one dimension
void DrawTGA_1D(SpringSys *sys, TGA *tga, float t, float dt,
    float lPixel, int margin) {
    // Create a pencil with black color
    TGAPencil *pen = TGAGetPencil();
    TGAPixel *black = TGAGetBlackPixel();
```

```

    TGAPencilSetColor(pen, black);
    TGAPencilSetShapePixel(pen);
    TGAFreePixel(&black);
    // Declare variables for tracing
    float p[2];
    // Position in absciss is the center of the pixel corresponding to
    // the time (scale by dt and shift by margin)
    p[0] = t / dt + 0.5 * dt + (float)margin;
    // Draw the masses of the SpringSys
    int nbMass = SpringSysGetNbMass(sys);
    for (int iMass = 0; iMass < nbMass; ++iMass) {
        // Get the mass
        SpringSysMass *m = SpringSysGetMass(sys, iMass);
        // If the mass is not null
        if (m != NULL) {
            p[1] = m->_pos[0] / lPixel + (float)margin;
            TGAStrokePix(tga, p, pen);
        }
    }
    // Free the pencil
    TGAFreePencil(&pen);
}

// Function to draw the SpringSys of the second example in two dimensions
void DrawLegendTGA_2D(SpringSys *sys, TGA *tga, float lMax,
    float lPixel, int margin, float slope, float k) {
    // Create a pencil with black color
    TGAPencil *pen = TGAGetPencil();
    TGAPixel *black = TGAGetBlackPixel();
    TGAPencilSetColor(pen, black);
    TGAPencilSetAntialias(pen, true);
    TGAFreePixel(&black);
    // Create a font
    TGAFont *font = TGAFontCreate(tgaFontDefault);
    TGAFontSetSize(font, 0.25 * (float)margin);
    // Declare variables for tracing
    float p[2];
    char s[100] = {0};
    float q[2];
    // Draw the absciss
    q[1] = p[1] = 0.25 * (float)margin;
    for (p[0] = 0.0; p[0] < lMax; p[0] += 0.5) {
        // Update the label
        sprintf(s, "%.1f", p[0]);
        // Print the label
        q[0] = p[0] / lPixel + 0.75 * (float)margin;
        TGAPrintString(tga, pen, font, (unsigned char*)s, q);
    }
    // Draw the ordinate
    q[0] = p[0] = 0.25 * (float)margin;
    for (p[1] = 0.0; p[1] < lMax; p[1] += 1.0) {
        // Update the label
        sprintf(s, "%.1f", p[1]);
        // Print the label
        q[1] = p[1] / lPixel + (float)margin;
        TGAPrintString(tga, pen, font, (unsigned char*)s, q);
    }
    // Draw the k coefficient
    TGAFontSetSize(font, 0.5 * (float)margin);
    sprintf(s, "k=%.1f", k);
    q[0] = 0.25 / lPixel + (float)margin;
    q[1] = 3.0 / lPixel + (float)margin;
}

```



```

    TGAPrintString(tga, pen, font, (unsigned char*)s, q);
    // Draw the axis
    TGAPencilSetShapePixel(pen);
    TGAPencilSetAntialias(pen, false);
    p[0] = p[1] = (float)margin;
    q[0] = (float)margin;
    q[1] = lMax / lPixel + (float)margin;
    TGADrawLine(tga, p, q, pen);
    q[0] = lMax / lPixel + (float)margin;
    q[1] = (float)margin;
    TGADrawLine(tga, p, q, pen);
    // Draw the ground
    q[1] = slope * lMax / lPixel + (float)margin;
    TGADrawLine(tga, p, q, pen);
    // Free the pencil
    TGAFreePencil(&pen);
    // Free the font
    TGAFreeFont(&font);
}

// Function to draw the SpringSys of the second example in two dimensions
void DrawTGA_2D(SpringSys *sys, TGA *tga, float lPixel, int margin) {
    // Create a pencil with black color
    TGAPencil *pen = TGAPencil();
    TGAPixel *black = TGAGetBlackPixel();
    TGAPencilSetColor(pen, black);
    TGAPencilSetShapePixel(pen);
    TGAFreePixel(&black);
    // Declare variables for tracing
    float p[2];
    float q[2];
    // Set the color of each mass
    unsigned char rgba[16] =
        {255,0,0,255, 0,255,0,255, 0,0,255,255, 255,255,0,255};
    for (int iColor = 0; iColor < 4; ++iColor) {
        TGAPencilSelectColor(pen, iColor);
        TGAPencilSetColRGBA(pen, rgba + iColor * 4);
    }
    // Draw the border of the square
    int nbSpring = 4;
    for (int iSpring = 0; iSpring < nbSpring; ++iSpring) {
        // Get the spring
        SpringSysSpring *s = SpringSysGetSpring(sys, iSpring);
        // If the spring is not null
        if (s != NULL) {
            // Get the masses
            SpringSysMass *mA = SpringSysGetMass(sys, s->_mass[0]);
            SpringSysMass *mB = SpringSysGetMass(sys, s->_mass[1]);
            // Draw the line between the mass
            p[0] = mA->_pos[0] / lPixel + (float)margin;
            p[1] = mA->_pos[1] / lPixel + (float)margin;
            q[0] = mB->_pos[0] / lPixel + (float)margin;
            q[1] = mB->_pos[1] / lPixel + (float)margin;
            TGAPencilSetModeColorBlend(pen, s->_mass[0], s->_mass[1]);
            TGADrawLine(tga, p, q, pen);
        }
    }
    // Free the pencil
    TGAFreePencil(&pen);
}

int main(int argc, char **argv) {

```

```

// Create a first example in one dimension,
// a chain of spring aligned and fixed at one extremity,
// initially compressed and with no velocity,
// default values for masses and springs
fprintf(stdout, " ----- 1D example -----\n");
int nbDim = 1;
// Allocate memory for the system
SpringSys *theSpringSys = SpringSysCreate(nbDim);
// Declare a variable to create the masses
SpringSysMass *mass = SpringSysCreateMass();
if (mass == NULL) {
    fprintf(stderr, "Couldn't allocate memory for mass\n");
}
// Declare a variable to create the springs
SpringSysSpring *spring = SpringSysCreateSpring();
if (spring == NULL) {
    fprintf(stderr, "Couldn't allocate memory for spring\n");
}
// Create the masses
int nbMass = 5;
for (int iMass = 0; iMass < nbMass; ++iMass) {
    // Set the mass properties
    mass->_id = iMass;
    mass->_pos[0] = 0.5 * (float)iMass;
    mass->_mass = 0.0;
    if (iMass == 0)
        mass->_fixed = true;
    else
        mass->_fixed = false;
    // Add the mass to the system
    bool ret = SpringSysAddMass(theSpringSys, mass);
    if (ret == false) {
        // Something went wrong when adding the mass
        fprintf(stderr, "Couldn't add the mass\n");
        // Print the mass for debugging
        SpringSysMassPrint(mass, stderr);
        fprintf(stderr, "\n");
        // Free memory
        SpringSysFree(&theSpringSys);
        SpringSysMassFree(&mass);
        SpringSysSpringFree(&spring);
        // Stop
        return 1;
    }
}
// Create the springs
int nbSpring = nbMass - 1;
for (int iSpring = 0; iSpring < nbSpring; ++iSpring) {
    // Set the spring properties
    spring->_id = iSpring;
    spring->_mass[0] = iSpring;
    spring->_mass[1] = iSpring + 1;
    // Add the spring to the system
    bool ret = SpringSysAddSpring(theSpringSys, spring);
    if (ret == false) {
        // Something went wrong when adding the spring
        fprintf(stderr, "Couldn't add the spring\n");
        // Print the spring for debugging
        SpringSysSpringPrint(spring, stderr);
        fprintf(stderr, "\n");
        // Free memory
        SpringSysFree(&theSpringSys);
    }
}

```

```

        SpringSysMassFree(&mass);
        SpringSysSpringFree(&spring);
        // Stop
        return 1;
    }
}

// Print the SpringSys
fprintf(stdout, "SpringSys:\n");
SpringSysPrint(theSpringSys, stdout);
// Declare some variable to memorize the parameters of the simulation
float t = 0.0;
float tMax = 100.0;
float dt = 0.1;
float lMax = (float)(1 + nbMass);
float lPixel = 0.01;
int margin = 20;
// Create a TGA to draw the SpringSys
TGAPixel *white = TGAGetWhitePixel();
short dim[2];
dim[0] = (int)round(tMax / dt) + margin * 2;
dim[1] = (int)round(lMax / lPixel) + margin * 2;
TGA* tga = TGACreate(dim, white);
TGAFreePixel(&white);
// If the TGA couldn't be created
if (tga == NULL) {
    // Free memory
    SpringSysFree(&theSpringSys);
    SpringSysMassFree(&mass);
    SpringSysSpringFree(&spring);
    // Stop
    return 1;
}

// Draw the legend of the TGA
DrawLegendTGA_1D(theSpringSys, tga, t, dt, tMax, lMax, lPixel, margin);
// Draw the initial state to the TGA
DrawTGA_1D(theSpringSys, tga, t, dt, lPixel, margin);
// Simulate the SpringSys behaviour
while (t < tMax) {
    // Step the SpringSys
    SpringSysStep(theSpringSys, dt);
    // Draw the new state to the TGA
    DrawTGA_1D(theSpringSys, tga, t, dt, lPixel, margin);
    // Display masses position
    fprintf(stdout, "%.3f: ", t);
    for (int iMass = 0; iMass < nbMass; ++iMass) {
        SpringSysMass *m = SpringSysGetMass(theSpringSys, iMass);
        float p = m->_pos[0];
        fprintf(stdout, "%.3f, ", p);
    }
    fprintf(stdout, "\n");
    // Increment time
    t += dt;
}

// Save the TGA
TGASave(tga, "./springSys1D.tga");
// Reset the initial position of the mass
for (int iMass = 0; iMass < nbMass; ++iMass) {
    SpringSysMass *m = SpringSysGetMass(theSpringSys, iMass);
    m->_pos[0] = 0.5 * (float)iMass;
}

// Get the equilibrium
tMax = 1000.0;

```

```

t = SpringSysStepToRest(theSpringSys, dt, tMax);
// If we could reach the equilibrium
if (t <= tMax) {
    fprintf(stderr, "Equilibrium reach in %.3f second\n", t);
    // Display masses position
    fprintf(stdout, "Masses position: ");
    for (int iMass = 0; iMass < nbMass; ++iMass) {
        SpringSysMass *m = SpringSysGetMass(theSpringSys, iMass);
        float p = m->_pos[0];
        fprintf(stdout, "%.3f, ", p);
    }
    fprintf(stdout, "\n");
} else {
    fprintf(stderr, "Couldn't reach the equilibrium\n");
}
// Free the TGA
TGAFree(&tga);
// Free memory
SpringSysFree(&theSpringSys);

// Create a second example in two dimensions,
// a 2d square of masses (link side by side and diagonals)
// falling under gravity onto an inclined ground
fprintf(stdout, " ----- 2D example ----- \n");
nbDim = 2;
// Allocate memory for the system
theSpringSys = SpringSysCreate(nbDim);
// Create the masses and add them to the system
mass->_mass = 0.1;
mass->_fixed = false;
nbMass = 4;
// Shift the upper right corner to create the SpringSys in
// an instable state
// Then we'll let it reach equilibrium before starting the simulation
// This is a trick to position the square slightly rotated relatively
// to the horizontal and avoid corner to be aligned during fall
// in the TGA
float posMass[8] = {1.1, 3.0, 2.1, 3.0, 0.9, 2.0, 1.9, 2.0};
for (int iMass = 0; iMass < nbMass; ++iMass) {
    // Set the mass position
    mass->_id = iMass;
    mass->_pos[0] = posMass[2 * iMass];
    mass->_pos[1] = posMass[2 * iMass + 1];
    // Add the mass to the system
    bool ret = SpringSysAddMass(theSpringSys, mass);
    if (ret == false) {
        // Something went wrong when adding the mass
        fprintf(stderr, "Couldn't add the mass\n");
        // Print the mass for debugging
        SpringSysMassPrint(mass, stderr);
        fprintf(stderr, "\n");
        // Free memory
        SpringSysFree(&theSpringSys);
        SpringSysMassFree(&mass);
        SpringSysSpringFree(&spring);
        // Stop
        return 1;
    }
}
}
// Create the springs

```

```

nbSpring = 6;
int springDef[12] = {0, 1, 1, 3, 2, 3, 0, 2, 1, 2, 0, 3};
float k = 2.0;
spring->_k = k;
for (int iSpring = 0; iSpring < nbSpring; ++iSpring) {
    // Set the spring properties
    spring->_id = iSpring;
    spring->_mass[0] = springDef[iSpring * 2];
    spring->_mass[1] = springDef[iSpring * 2 + 1];
    if (iSpring < 4)
        spring->_length = spring->_restLength = 1.0;
    else
        spring->_length = spring->_restLength = sqrt(2.0);
    // Add the spring to the system
    bool ret = SpringSysAddSpring(theSpringSys, spring);
    if (ret == false) {
        // Something went wrong when adding the spring
        fprintf(stderr, "Couldn't add the spring\n");
        // Print the spring for debugging
        SpringSysSpringPrint(spring, stderr);
        fprintf(stderr, "\n");
        // Free memory
        SpringSysFree(&theSpringSys);
        SpringSysMassFree(&mass);
        SpringSysSpringFree(&spring);
        // Stop
        return 1;
    }
}
// Reach equilibrium
dt = 1.0 / 25.0;
t = SpringSysStepToRest(theSpringSys, dt, tMax);
if (t <= tMax) {
    fprintf(stderr, "Equilibrium reach in %.3f second\n", t);
} else {
    fprintf(stderr, "Couldn't reach the equilibrium, start anyway the \
simulation with current state\n");
}
// Print the SpringSys
fprintf(stdout, "SpringSys:\n");
SpringSysPrint(theSpringSys, stdout);
// Declare a variable to memorize the slope of the ground
float slope = 0.1;
// Create a TGA to draw the SpringSys
white = TGAGetWhitePixel();
lMax = 3.0;
margin = 30;
dim[0] = (int)round(lMax / lPixel) + margin * 2;
dim[1] = (int)round(lMax / lPixel) + margin * 2;
tga = TGACreate(dim, white);
// If the TGA couldn't be created
if (tga == NULL) {
    // Free memory
    SpringSysFree(&theSpringSys);
    SpringSysMassFree(&mass);
    SpringSysSpringFree(&spring);
    TGAFreePixel(&white);
    // Stop
    return 1;
}
// Draw the legend of the TGA

```

```

DrawLegendTGA_2D(theSpringSys, tga, lMax, lPixel, margin, slope, k);
// Draw the intial state to the TGA
DrawTGA_2D(theSpringSys, tga, lPixel, margin);
// Declare a variable to memorize the vector used during bouncing
float v[2];
v[0] = -1.0 * slope / sqrt(1.0 + pow(slope, 2.0));
v[1] = 1.0 / sqrt(1.0 + pow(slope, 2.0));
// Run the simulation
t = 0.0;
tMax = 30.0;
int iFrame = 0;
while (t < tMax) {
    // For each mass
    for (int iMass = 0; iMass < nbMass; ++iMass) {
        // Get the mass
        SpringSysMass *m = SpringSysGetMass(theSpringSys, iMass);
        // If the mass is not null
        if (m != NULL) {
            // Apply attraction toward bottom
            m->_speed[1] -= 1.0 * dt * m->_mass;
        }
    }
    // Step the SpringSys
    SpringSysStep(theSpringSys, dt);
    // For each mass
    for (int iMass = 0; iMass < nbMass; ++iMass) {
        // Get the mass
        SpringSysMass *m = SpringSysGetMass(theSpringSys, iMass);
        // If the mass is not null and has collided with the ground
        if (m != NULL && m->_pos[1] < m->_pos[0] * slope) {
            // Correct the mass position
            m->_pos[1] = m->_pos[0] * slope;
            // Bounce the mass off the ground
            float ls = sqrt(pow(m->_speed[0], 2.0) + pow(m->_speed[1], 2.0));
            if (ls > SPRINGSYS_EPSILON) {
                float s[2];
                s[0] = m->_speed[0] / ls;
                s[1] = m->_speed[1] / ls;
                float w[2];
                w[0] = s[0] + v[0] * 2.0;
                w[1] = s[1] + v[1] * 2.0;
                float lw = sqrt(pow(w[0], 2.0) + pow(w[1], 2.0));
                m->_speed[0] = 0.9 * w[0] / lw * ls;
                m->_speed[1] = 0.9 * w[1] / lw * ls;
            } else {
                m->_speed[0] = 0.0;
                m->_speed[1] = 0.0;
            }
        }
    }
}
// Draw the SpringSys
DrawTGA_2D(theSpringSys, tga, lPixel, margin);
// Save the frame for animation
char fileName[100];
sprintf(fileName, "./Frames/%02d.tga", iFrame);
TGA* tgaFrame = TGACreate(dim, white);
if (tgaFrame != NULL) {
    DrawLegendTGA_2D(theSpringSys, tgaFrame,
        lMax, lPixel, margin, slope, k);
    DrawTGA_2D(theSpringSys, tgaFrame, lPixel, margin);
    TGASave(tgaFrame, fileName);
    TGAFree(&tgaFrame);
}

```

```

    }
    // Increment time and frame index
    t += dt;
    iFrame++;
}
// Save the TGA
TGASave(tga, "./springSys2D.tga");
// Free the TGA
TGAFree(&tga);
// Search the nearest mass to (1.0,1.0)
printf("Nearest mass to (1.0,1.0):\n");
float pos[2] = {1.0, 1.0};
SpringSysMass *nearestMass = SpringSysGetMassByPos(theSpringSys, pos);
printf("#%d %.3f,%.3f\n", nearestMass->_id, nearestMass->_pos[0],
    nearestMass->_pos[1]);
// Search the nearest spring to (1.0,1.0)
printf("Nearest spring to (1.0,1.0):\n");
SpringSysSpring *nearestSpring =
    SpringSysGetSpringByPos(theSpringSys, pos);
printf("#%d %d,%d\n", nearestSpring->_id, nearestSpring->_mass[0],
    nearestSpring->_mass[1]);
// Save the SpringSys
printf("Saved SpringSys:\n");
SpringSysPrint(theSpringSys, stdout);
FILE *stream = fopen("./springsys.dat", "w");
int ret = SpringSysSave(theSpringSys, stream);
fclose(stream);
if (ret != 0) {
    fprintf(stderr, "Couldn't save the SpringSys (%d)\n", ret);
} else {
    stream = fopen("./springsys.dat", "r");
    SpringSys *loadSys = NULL;
    ret = SpringSysLoad(&loadSys, stream);
    if (ret != 0) {
        fprintf(stderr, "Couldn't load the SpringSys (%d)\n", ret);
    } else {
        printf("Loaded SpringSys:\n");
        SpringSysPrint(loadSys, stdout);
        SpringSysFree(&loadSys);
    }
    fclose(stream);
}
// Remove one spring
printf("Remove spring #4:\n");
SpringSysRemoveSpring(theSpringSys, 4);
SpringSysPrint(theSpringSys, stdout);
// Remove one node
printf("Remove mass #3:\n");
SpringSysRemoveMass(theSpringSys, 3);
SpringSysPrint(theSpringSys, stdout);
// Free memory
TGAFreePixel(&white);
SpringSysFree(&theSpringSys);
SpringSysMassFree(&mass);
SpringSysSpringFree(&spring);
// Free memory
SpringSysFree(&theSpringSys);
return 0;
}

```

6 Output

```
----- 1D example -----
SpringSys:
Number of dimension: 1
Dissipation: 0.100
Masses:
#0, pos(0.000,0.000,0.000), speed(0.000,0.000,0.000),
  stress(0.000,0.000,0.000), mass(0.000), fixed(1)
#1, pos(0.500,0.000,0.000), speed(0.000,0.000,0.000),
  stress(0.000,0.000,0.000), mass(0.000), fixed(0)
#2, pos(1.000,0.000,0.000), speed(0.000,0.000,0.000),
  stress(0.000,0.000,0.000), mass(0.000), fixed(0)
#3, pos(1.500,0.000,0.000), speed(0.000,0.000,0.000),
  stress(0.000,0.000,0.000), mass(0.000), fixed(0)
#4, pos(2.000,0.000,0.000), speed(0.000,0.000,0.000),
  stress(0.000,0.000,0.000), mass(0.000), fixed(0)
Springs:
#0, 0-1, length(1.000), stress(0.000), k(1.000), restLength(1.000),
  maxStress(-1000000.000,1000000.000), breakable(0)
#1, 1-2, length(1.000), stress(0.000), k(1.000), restLength(1.000),
  maxStress(-1000000.000,1000000.000), breakable(0)
#2, 2-3, length(1.000), stress(0.000), k(1.000), restLength(1.000),
  maxStress(-1000000.000,1000000.000), breakable(0)
#3, 3-4, length(1.000), stress(0.000), k(1.000), restLength(1.000),
  maxStress(-1000000.000,1000000.000), breakable(0)
0.000: 0.000, 0.500, 1.000, 1.500, 2.005,
0.100: 0.000, 0.500, 1.000, 1.500, 2.015,
0.200: 0.000, 0.500, 1.000, 1.500, 2.030,
0.300: 0.000, 0.500, 1.000, 1.501, 2.049,
0.400: 0.000, 0.500, 1.000, 1.502, 2.072,
0.500: 0.000, 0.500, 1.000, 1.503, 2.100,
0.600: 0.000, 0.500, 1.000, 1.506, 2.131,
0.700: 0.000, 0.500, 1.000, 1.510, 2.166,
0.800: 0.000, 0.500, 1.000, 1.515, 2.204,
0.900: 0.000, 0.500, 1.001, 1.522, 2.244,
1.000: 0.000, 0.500, 1.001, 1.530, 2.287,
1.100: 0.000, 0.500, 1.002, 1.541, 2.332,
1.200: 0.000, 0.500, 1.003, 1.555, 2.378,
1.300: 0.000, 0.500, 1.005, 1.571, 2.426,
1.400: 0.000, 0.500, 1.007, 1.589, 2.475,
1.500: 0.000, 0.500, 1.010, 1.611, 2.524,
1.600: 0.000, 0.501, 1.014, 1.635, 2.574,
1.700: 0.000, 0.501, 1.019, 1.663, 2.624,
1.800: 0.000, 0.502, 1.025, 1.693, 2.673,
1.900: 0.000, 0.502, 1.032, 1.726, 2.723,
2.000: 0.000, 0.504, 1.041, 1.762, 2.771,
2.100: 0.000, 0.505, 1.052, 1.800, 2.820,
2.200: 0.000, 0.507, 1.065, 1.840, 2.867,
2.300: 0.000, 0.509, 1.080, 1.883, 2.914,
2.400: 0.000, 0.512, 1.096, 1.928, 2.960,
2.500: 0.000, 0.515, 1.116, 1.974, 3.005,
2.600: 0.000, 0.520, 1.137, 2.021, 3.049,
2.700: 0.000, 0.525, 1.161, 2.069, 3.093,
2.800: 0.000, 0.531, 1.188, 2.118, 3.136,
2.900: 0.000, 0.539, 1.216, 2.167, 3.178,
3.000: 0.000, 0.548, 1.248, 2.217, 3.220,
3.100: 0.000, 0.558, 1.281, 2.266, 3.261,
3.200: 0.000, 0.570, 1.317, 2.314, 3.302,
3.300: 0.000, 0.583, 1.355, 2.363, 3.343,
3.400: 0.000, 0.599, 1.395, 2.410, 3.383,
```


3.500: 0.000, 0.616, 1.437, 2.456, 3.423,
 3.600: 0.000, 0.635, 1.481, 2.502, 3.463,
 3.700: 0.000, 0.655, 1.525, 2.546, 3.503,
 3.800: 0.000, 0.678, 1.571, 2.590, 3.543,
 3.900: 0.000, 0.703, 1.617, 2.632, 3.584,
 4.000: 0.000, 0.729, 1.664, 2.673, 3.624,
 4.100: 0.000, 0.758, 1.711, 2.713, 3.664,
 4.200: 0.000, 0.788, 1.759, 2.752, 3.704,
 4.300: 0.000, 0.819, 1.806, 2.790, 3.745,
 4.400: 0.000, 0.852, 1.852, 2.828, 3.785,
 4.500: 0.000, 0.886, 1.898, 2.865, 3.825,
 4.600: 0.000, 0.921, 1.942, 2.902, 3.866,
 4.700: 0.000, 0.956, 1.986, 2.938, 3.906,
 4.800: 0.000, 0.992, 2.028, 2.974, 3.946,
 4.900: 0.000, 1.028, 2.070, 3.010, 3.986,
 5.000: 0.000, 1.064, 2.109, 3.046, 4.026,
 5.100: 0.000, 1.099, 2.147, 3.082, 4.066,
 5.200: 0.000, 1.133, 2.184, 3.118, 4.105,
 5.300: 0.000, 1.166, 2.219, 3.154, 4.145,
 5.400: 0.000, 1.198, 2.252, 3.191, 4.183,
 5.500: 0.000, 1.228, 2.284, 3.227, 4.222,
 5.600: 0.000, 1.256, 2.315, 3.264, 4.260,
 5.700: 0.000, 1.281, 2.344, 3.301, 4.298,
 5.800: 0.000, 1.304, 2.372, 3.338, 4.335,
 5.900: 0.000, 1.325, 2.398, 3.374, 4.372,
 6.000: 0.000, 1.342, 2.423, 3.411, 4.409,
 6.100: 0.000, 1.357, 2.447, 3.447, 4.445,
 6.200: 0.000, 1.370, 2.470, 3.483, 4.481,
 6.300: 0.000, 1.379, 2.492, 3.519, 4.517,
 6.400: 0.000, 1.385, 2.512, 3.553, 4.552,
 6.500: 0.000, 1.389, 2.532, 3.587, 4.587,
 6.600: 0.000, 1.391, 2.550, 3.620, 4.621,
 6.700: 0.000, 1.390, 2.568, 3.652, 4.655,
 6.800: 0.000, 1.387, 2.584, 3.683, 4.689,
 6.900: 0.000, 1.382, 2.599, 3.713, 4.722,
 7.000: 0.000, 1.375, 2.613, 3.741, 4.755,
 7.100: 0.000, 1.368, 2.626, 3.768, 4.788,
 7.200: 0.000, 1.359, 2.637, 3.793, 4.820,
 7.300: 0.000, 1.349, 2.647, 3.817, 4.851,
 7.400: 0.000, 1.339, 2.656, 3.839, 4.881,
 7.500: 0.000, 1.329, 2.663, 3.859, 4.911,
 7.600: 0.000, 1.319, 2.669, 3.878, 4.940,
 7.700: 0.000, 1.310, 2.673, 3.895, 4.968,
 7.800: 0.000, 1.301, 2.676, 3.911, 4.996,
 7.900: 0.000, 1.293, 2.677, 3.925, 5.021,
 8.000: 0.000, 1.286, 2.677, 3.937, 5.046,
 8.100: 0.000, 1.280, 2.676, 3.948, 5.069,
 8.200: 0.000, 1.275, 2.674, 3.956, 5.091,
 8.300: 0.000, 1.272, 2.670, 3.964, 5.112,
 8.400: 0.000, 1.270, 2.665, 3.970, 5.130,
 8.500: 0.000, 1.269, 2.660, 3.974, 5.147,
 8.600: 0.000, 1.269, 2.654, 3.977, 5.162,
 8.700: 0.000, 1.271, 2.647, 3.978, 5.175,
 8.800: 0.000, 1.273, 2.640, 3.978, 5.186,
 8.900: 0.000, 1.277, 2.632, 3.977, 5.194,
 9.000: 0.000, 1.281, 2.625, 3.975, 5.201,
 9.100: 0.000, 1.286, 2.618, 3.971, 5.205,
 9.200: 0.000, 1.291, 2.611, 3.966, 5.206,
 9.300: 0.000, 1.297, 2.604, 3.960, 5.206,
 9.400: 0.000, 1.302, 2.598, 3.953, 5.202,
 9.500: 0.000, 1.307, 2.593, 3.945, 5.197,
 9.600: 0.000, 1.313, 2.588, 3.936, 5.189,

9.700: 0.000, 1.317, 2.585, 3.927, 5.178,
 9.800: 0.000, 1.321, 2.582, 3.916, 5.165,
 9.900: 0.000, 1.325, 2.579, 3.905, 5.150,
 10.000: 0.000, 1.328, 2.578, 3.893, 5.132,
 10.100: 0.000, 1.330, 2.577, 3.880, 5.112,
 10.200: 0.000, 1.331, 2.577, 3.867, 5.090,
 10.300: 0.000, 1.331, 2.577, 3.853, 5.066,
 10.400: 0.000, 1.330, 2.577, 3.839, 5.041,
 10.500: 0.000, 1.329, 2.578, 3.824, 5.013,
 10.600: 0.000, 1.327, 2.578, 3.809, 4.984,
 10.700: 0.000, 1.324, 2.579, 3.794, 4.953,
 10.800: 0.000, 1.320, 2.579, 3.778, 4.921,
 10.900: 0.000, 1.316, 2.578, 3.761, 4.888,
 11.000: 0.000, 1.311, 2.577, 3.745, 4.854,
 11.100: 0.000, 1.306, 2.574, 3.728, 4.820,
 11.200: 0.000, 1.301, 2.571, 3.710, 4.785,
 11.300: 0.000, 1.295, 2.566, 3.692, 4.749,
 11.400: 0.000, 1.289, 2.560, 3.674, 4.713,
 11.500: 0.000, 1.283, 2.552, 3.655, 4.677,
 11.600: 0.000, 1.277, 2.543, 3.635, 4.642,
 11.700: 0.000, 1.271, 2.532, 3.615, 4.606,
 11.800: 0.000, 1.265, 2.519, 3.594, 4.572,
 11.900: 0.000, 1.259, 2.505, 3.572, 4.537,
 12.000: 0.000, 1.253, 2.489, 3.549, 4.504,
 12.100: 0.000, 1.246, 2.472, 3.526, 4.471,
 12.200: 0.000, 1.240, 2.453, 3.502, 4.439,
 12.300: 0.000, 1.233, 2.432, 3.477, 4.408,
 12.400: 0.000, 1.226, 2.411, 3.451, 4.378,
 12.500: 0.000, 1.219, 2.388, 3.424, 4.349,
 12.600: 0.000, 1.211, 2.364, 3.397, 4.321,
 12.700: 0.000, 1.203, 2.339, 3.368, 4.295,
 12.800: 0.000, 1.194, 2.313, 3.339, 4.269,
 12.900: 0.000, 1.185, 2.286, 3.309, 4.244,
 13.000: 0.000, 1.174, 2.259, 3.279, 4.220,
 13.100: 0.000, 1.163, 2.232, 3.248, 4.197,
 13.200: 0.000, 1.152, 2.204, 3.217, 4.175,
 13.300: 0.000, 1.139, 2.177, 3.185, 4.153,
 13.400: 0.000, 1.125, 2.149, 3.154, 4.132,
 13.500: 0.000, 1.111, 2.121, 3.122, 4.112,
 13.600: 0.000, 1.096, 2.094, 3.091, 4.091,
 13.700: 0.000, 1.080, 2.067, 3.060, 4.071,
 13.800: 0.000, 1.063, 2.040, 3.030, 4.051,
 13.900: 0.000, 1.045, 2.014, 3.000, 4.031,
 14.000: 0.000, 1.027, 1.988, 2.971, 4.011,
 14.100: 0.000, 1.008, 1.962, 2.943, 3.991,
 14.200: 0.000, 0.989, 1.937, 2.916, 3.970,
 14.300: 0.000, 0.970, 1.913, 2.890, 3.949,
 14.400: 0.000, 0.951, 1.889, 2.865, 3.928,
 14.500: 0.000, 0.932, 1.866, 2.841, 3.906,
 14.600: 0.000, 0.913, 1.844, 2.818, 3.884,
 14.700: 0.000, 0.894, 1.822, 2.797, 3.862,
 14.800: 0.000, 0.876, 1.801, 2.776, 3.839,
 14.900: 0.000, 0.859, 1.781, 2.757, 3.815,
 15.000: 0.000, 0.843, 1.761, 2.739, 3.792,
 15.100: 0.000, 0.827, 1.742, 2.721, 3.768,
 15.200: 0.000, 0.812, 1.724, 2.705, 3.744,
 15.300: 0.000, 0.799, 1.707, 2.689, 3.719,
 15.400: 0.000, 0.787, 1.691, 2.674, 3.695,
 15.500: 0.000, 0.776, 1.676, 2.659, 3.671,
 15.600: 0.000, 0.766, 1.662, 2.645, 3.647,
 15.700: 0.000, 0.758, 1.648, 2.631, 3.623,
 15.800: 0.000, 0.751, 1.636, 2.618, 3.599,

15.900: 0.000, 0.746, 1.625, 2.604, 3.576,
 16.000: 0.000, 0.742, 1.615, 2.591, 3.553,
 16.100: 0.000, 0.740, 1.607, 2.578, 3.531,
 16.200: 0.000, 0.738, 1.599, 2.564, 3.510,
 16.300: 0.000, 0.738, 1.592, 2.551, 3.489,
 16.400: 0.000, 0.739, 1.587, 2.537, 3.470,
 16.500: 0.000, 0.742, 1.583, 2.524, 3.451,
 16.600: 0.000, 0.745, 1.579, 2.510, 3.433,
 16.700: 0.000, 0.749, 1.577, 2.497, 3.416,
 16.800: 0.000, 0.753, 1.576, 2.483, 3.400,
 16.900: 0.000, 0.759, 1.575, 2.470, 3.385,
 17.000: 0.000, 0.765, 1.575, 2.457, 3.371,
 17.100: 0.000, 0.771, 1.576, 2.445, 3.358,
 17.200: 0.000, 0.778, 1.578, 2.433, 3.346,
 17.300: 0.000, 0.784, 1.580, 2.422, 3.335,
 17.400: 0.000, 0.791, 1.583, 2.412, 3.325,
 17.500: 0.000, 0.798, 1.586, 2.403, 3.316,
 17.600: 0.000, 0.805, 1.589, 2.395, 3.308,
 17.700: 0.000, 0.811, 1.592, 2.388, 3.301,
 17.800: 0.000, 0.817, 1.596, 2.382, 3.295,
 17.900: 0.000, 0.822, 1.599, 2.377, 3.290,
 18.000: 0.000, 0.827, 1.603, 2.374, 3.286,
 18.100: 0.000, 0.832, 1.606, 2.373, 3.282,
 18.200: 0.000, 0.835, 1.610, 2.372, 3.280,
 18.300: 0.000, 0.839, 1.613, 2.374, 3.278,
 18.400: 0.000, 0.841, 1.616, 2.376, 3.278,
 18.500: 0.000, 0.843, 1.619, 2.380, 3.278,
 18.600: 0.000, 0.844, 1.621, 2.386, 3.280,
 18.700: 0.000, 0.844, 1.624, 2.392, 3.282,
 18.800: 0.000, 0.844, 1.626, 2.400, 3.286,
 18.900: 0.000, 0.843, 1.628, 2.409, 3.291,
 19.000: 0.000, 0.842, 1.630, 2.419, 3.297,
 19.100: 0.000, 0.840, 1.633, 2.429, 3.304,
 19.200: 0.000, 0.837, 1.635, 2.440, 3.312,
 19.300: 0.000, 0.835, 1.637, 2.452, 3.321,
 19.400: 0.000, 0.832, 1.639, 2.464, 3.332,
 19.500: 0.000, 0.828, 1.642, 2.477, 3.344,
 19.600: 0.000, 0.825, 1.645, 2.489, 3.357,
 19.700: 0.000, 0.822, 1.647, 2.502, 3.371,
 19.800: 0.000, 0.818, 1.651, 2.515, 3.386,
 19.900: 0.000, 0.815, 1.654, 2.527, 3.403,
 20.000: 0.000, 0.812, 1.658, 2.540, 3.420,
 20.100: 0.000, 0.810, 1.662, 2.552, 3.439,
 20.200: 0.000, 0.808, 1.666, 2.565, 3.459,
 20.300: 0.000, 0.806, 1.671, 2.577, 3.479,
 20.400: 0.000, 0.805, 1.676, 2.589, 3.500,
 20.500: 0.000, 0.805, 1.682, 2.600, 3.522,
 20.600: 0.000, 0.805, 1.687, 2.612, 3.544,
 20.700: 0.000, 0.807, 1.694, 2.624, 3.567,
 20.800: 0.000, 0.809, 1.700, 2.635, 3.590,
 20.900: 0.000, 0.811, 1.707, 2.647, 3.614,
 21.000: 0.000, 0.815, 1.715, 2.659, 3.637,
 21.100: 0.000, 0.820, 1.722, 2.671, 3.661,
 21.200: 0.000, 0.825, 1.730, 2.683, 3.684,
 21.300: 0.000, 0.831, 1.739, 2.696, 3.707,
 21.400: 0.000, 0.837, 1.748, 2.709, 3.730,
 21.500: 0.000, 0.845, 1.757, 2.723, 3.752,
 21.600: 0.000, 0.853, 1.767, 2.737, 3.773,
 21.700: 0.000, 0.861, 1.777, 2.751, 3.795,
 21.800: 0.000, 0.870, 1.787, 2.766, 3.815,
 21.900: 0.000, 0.879, 1.799, 2.782, 3.835,
 22.000: 0.000, 0.889, 1.810, 2.798, 3.854,

22.100: 0.000, 0.899, 1.823, 2.815, 3.872,
 22.200: 0.000, 0.909, 1.835, 2.832, 3.890,
 22.300: 0.000, 0.919, 1.849, 2.850, 3.906,
 22.400: 0.000, 0.929, 1.863, 2.868, 3.922,
 22.500: 0.000, 0.939, 1.877, 2.886, 3.938,
 22.600: 0.000, 0.949, 1.892, 2.904, 3.952,
 22.700: 0.000, 0.959, 1.908, 2.923, 3.966,
 22.800: 0.000, 0.968, 1.924, 2.942, 3.980,
 22.900: 0.000, 0.977, 1.940, 2.960, 3.993,
 23.000: 0.000, 0.986, 1.957, 2.979, 4.005,
 23.100: 0.000, 0.995, 1.975, 2.998, 4.017,
 23.200: 0.000, 1.004, 1.992, 3.016, 4.029,
 23.300: 0.000, 1.012, 2.010, 3.034, 4.040,
 23.400: 0.000, 1.020, 2.028, 3.052, 4.051,
 23.500: 0.000, 1.028, 2.045, 3.069, 4.063,
 23.600: 0.000, 1.036, 2.063, 3.085, 4.074,
 23.700: 0.000, 1.043, 2.080, 3.102, 4.085,
 23.800: 0.000, 1.051, 2.097, 3.117, 4.096,
 23.900: 0.000, 1.058, 2.114, 3.132, 4.107,
 24.000: 0.000, 1.065, 2.130, 3.147, 4.119,
 24.100: 0.000, 1.072, 2.145, 3.161, 4.130,
 24.200: 0.000, 1.079, 2.160, 3.174, 4.142,
 24.300: 0.000, 1.086, 2.174, 3.187, 4.154,
 24.400: 0.000, 1.093, 2.187, 3.199, 4.166,
 24.500: 0.000, 1.100, 2.199, 3.210, 4.178,
 24.600: 0.000, 1.107, 2.210, 3.221, 4.191,
 24.700: 0.000, 1.113, 2.220, 3.232, 4.204,
 24.800: 0.000, 1.120, 2.229, 3.242, 4.216,
 24.900: 0.000, 1.126, 2.237, 3.251, 4.229,
 25.000: 0.000, 1.132, 2.243, 3.260, 4.242,
 25.100: 0.000, 1.138, 2.249, 3.269, 4.256,
 25.200: 0.000, 1.144, 2.254, 3.277, 4.269,
 25.300: 0.000, 1.149, 2.258, 3.284, 4.282,
 25.400: 0.000, 1.153, 2.261, 3.292, 4.295,
 25.500: 0.000, 1.158, 2.263, 3.299, 4.307,
 25.600: 0.000, 1.161, 2.265, 3.305, 4.320,
 25.700: 0.000, 1.164, 2.265, 3.312, 4.332,
 25.800: 0.000, 1.167, 2.266, 3.318, 4.344,
 25.900: 0.000, 1.168, 2.266, 3.323, 4.356,
 26.000: 0.000, 1.169, 2.265, 3.329, 4.367,
 26.100: 0.000, 1.169, 2.264, 3.334, 4.377,
 26.200: 0.000, 1.169, 2.263, 3.339, 4.387,
 26.300: 0.000, 1.167, 2.262, 3.343, 4.397,
 26.400: 0.000, 1.165, 2.260, 3.347, 4.406,
 26.500: 0.000, 1.163, 2.259, 3.351, 4.414,
 26.600: 0.000, 1.159, 2.257, 3.354, 4.421,
 26.700: 0.000, 1.155, 2.256, 3.357, 4.428,
 26.800: 0.000, 1.151, 2.254, 3.360, 4.434,
 26.900: 0.000, 1.146, 2.253, 3.363, 4.439,
 27.000: 0.000, 1.141, 2.251, 3.365, 4.443,
 27.100: 0.000, 1.135, 2.250, 3.366, 4.447,
 27.200: 0.000, 1.130, 2.249, 3.368, 4.449,
 27.300: 0.000, 1.124, 2.247, 3.368, 4.451,
 27.400: 0.000, 1.118, 2.246, 3.369, 4.452,
 27.500: 0.000, 1.113, 2.245, 3.369, 4.452,
 27.600: 0.000, 1.108, 2.243, 3.369, 4.451,
 27.700: 0.000, 1.103, 2.242, 3.368, 4.450,
 27.800: 0.000, 1.099, 2.240, 3.367, 4.447,
 27.900: 0.000, 1.095, 2.238, 3.365, 4.444,
 28.000: 0.000, 1.091, 2.237, 3.363, 4.440,
 28.100: 0.000, 1.088, 2.234, 3.361, 4.436,
 28.200: 0.000, 1.086, 2.232, 3.358, 4.430,

28.300: 0.000, 1.084, 2.230, 3.354, 4.424,
 28.400: 0.000, 1.083, 2.227, 3.350, 4.418,
 28.500: 0.000, 1.083, 2.224, 3.346, 4.411,
 28.600: 0.000, 1.083, 2.221, 3.341, 4.403,
 28.700: 0.000, 1.084, 2.218, 3.335, 4.394,
 28.800: 0.000, 1.085, 2.215, 3.329, 4.386,
 28.900: 0.000, 1.087, 2.212, 3.322, 4.376,
 29.000: 0.000, 1.089, 2.208, 3.315, 4.366,
 29.100: 0.000, 1.091, 2.205, 3.308, 4.356,
 29.200: 0.000, 1.094, 2.201, 3.300, 4.346,
 29.300: 0.000, 1.096, 2.197, 3.291, 4.335,
 29.400: 0.000, 1.099, 2.193, 3.282, 4.323,
 29.500: 0.000, 1.102, 2.190, 3.273, 4.312,
 29.600: 0.000, 1.104, 2.186, 3.263, 4.300,
 29.700: 0.000, 1.106, 2.182, 3.253, 4.288,
 29.800: 0.000, 1.108, 2.178, 3.243, 4.276,
 29.900: 0.000, 1.109, 2.175, 3.233, 4.263,
 30.000: 0.000, 1.110, 2.171, 3.222, 4.251,
 30.100: 0.000, 1.111, 2.167, 3.212, 4.238,
 30.200: 0.000, 1.111, 2.163, 3.201, 4.225,
 30.300: 0.000, 1.110, 2.159, 3.190, 4.212,
 30.400: 0.000, 1.109, 2.155, 3.179, 4.199,
 30.500: 0.000, 1.107, 2.150, 3.169, 4.186,
 30.600: 0.000, 1.105, 2.146, 3.158, 4.172,
 30.700: 0.000, 1.101, 2.141, 3.147, 4.159,
 30.800: 0.000, 1.098, 2.136, 3.137, 4.146,
 30.900: 0.000, 1.093, 2.131, 3.127, 4.133,
 31.000: 0.000, 1.089, 2.125, 3.117, 4.120,
 31.100: 0.000, 1.083, 2.119, 3.107, 4.107,
 31.200: 0.000, 1.078, 2.112, 3.098, 4.094,
 31.300: 0.000, 1.072, 2.105, 3.089, 4.081,
 31.400: 0.000, 1.065, 2.098, 3.080, 4.069,
 31.500: 0.000, 1.059, 2.090, 3.071, 4.057,
 31.600: 0.000, 1.052, 2.082, 3.062, 4.045,
 31.700: 0.000, 1.045, 2.073, 3.053, 4.033,
 31.800: 0.000, 1.038, 2.064, 3.045, 4.022,
 31.900: 0.000, 1.031, 2.055, 3.036, 4.011,
 32.000: 0.000, 1.024, 2.045, 3.028, 4.000,
 32.100: 0.000, 1.017, 2.035, 3.019, 3.990,
 32.200: 0.000, 1.010, 2.025, 3.011, 3.980,
 32.300: 0.000, 1.003, 2.014, 3.002, 3.971,
 32.400: 0.000, 0.996, 2.004, 2.993, 3.962,
 32.500: 0.000, 0.990, 1.993, 2.984, 3.953,
 32.600: 0.000, 0.983, 1.982, 2.975, 3.945,
 32.700: 0.000, 0.977, 1.972, 2.966, 3.937,
 32.800: 0.000, 0.971, 1.961, 2.957, 3.930,
 32.900: 0.000, 0.966, 1.951, 2.948, 3.923,
 33.000: 0.000, 0.961, 1.941, 2.938, 3.916,
 33.100: 0.000, 0.955, 1.931, 2.929, 3.910,
 33.200: 0.000, 0.951, 1.922, 2.919, 3.903,
 33.300: 0.000, 0.946, 1.913, 2.909, 3.897,
 33.400: 0.000, 0.942, 1.904, 2.900, 3.892,
 33.500: 0.000, 0.938, 1.895, 2.890, 3.886,
 33.600: 0.000, 0.934, 1.888, 2.881, 3.880,
 33.700: 0.000, 0.930, 1.880, 2.871, 3.875,
 33.800: 0.000, 0.927, 1.873, 2.862, 3.869,
 33.900: 0.000, 0.924, 1.867, 2.853, 3.864,
 34.000: 0.000, 0.921, 1.861, 2.845, 3.858,
 34.100: 0.000, 0.918, 1.856, 2.837, 3.853,
 34.200: 0.000, 0.916, 1.851, 2.829, 3.847,
 34.300: 0.000, 0.914, 1.846, 2.822, 3.841,
 34.400: 0.000, 0.912, 1.842, 2.815, 3.835,

34.500: 0.000, 0.910, 1.839, 2.809, 3.829,
 34.600: 0.000, 0.908, 1.836, 2.803, 3.823,
 34.700: 0.000, 0.907, 1.833, 2.798, 3.816,
 34.800: 0.000, 0.906, 1.831, 2.794, 3.810,
 34.900: 0.000, 0.905, 1.829, 2.790, 3.803,
 35.000: 0.000, 0.904, 1.828, 2.787, 3.797,
 35.100: 0.000, 0.903, 1.827, 2.784, 3.790,
 35.200: 0.000, 0.903, 1.826, 2.782, 3.783,
 35.300: 0.000, 0.903, 1.826, 2.780, 3.777,
 35.400: 0.000, 0.903, 1.826, 2.779, 3.770,
 35.500: 0.000, 0.903, 1.826, 2.778, 3.764,
 35.600: 0.000, 0.904, 1.827, 2.777, 3.758,
 35.700: 0.000, 0.904, 1.827, 2.777, 3.752,
 35.800: 0.000, 0.905, 1.828, 2.777, 3.746,
 35.900: 0.000, 0.906, 1.830, 2.777, 3.741,
 36.000: 0.000, 0.907, 1.831, 2.778, 3.736,
 36.100: 0.000, 0.909, 1.833, 2.778, 3.732,
 36.200: 0.000, 0.910, 1.835, 2.779, 3.728,
 36.300: 0.000, 0.912, 1.837, 2.779, 3.725,
 36.400: 0.000, 0.913, 1.839, 2.780, 3.722,
 36.500: 0.000, 0.915, 1.842, 2.781, 3.720,
 36.600: 0.000, 0.917, 1.844, 2.781, 3.719,
 36.700: 0.000, 0.919, 1.847, 2.782, 3.718,
 36.800: 0.000, 0.921, 1.849, 2.783, 3.718,
 36.900: 0.000, 0.923, 1.852, 2.783, 3.718,
 37.000: 0.000, 0.926, 1.855, 2.784, 3.719,
 37.100: 0.000, 0.928, 1.857, 2.785, 3.721,
 37.200: 0.000, 0.930, 1.860, 2.786, 3.724,
 37.300: 0.000, 0.932, 1.862, 2.787, 3.727,
 37.400: 0.000, 0.934, 1.865, 2.788, 3.730,
 37.500: 0.000, 0.936, 1.867, 2.789, 3.734,
 37.600: 0.000, 0.938, 1.869, 2.790, 3.739,
 37.700: 0.000, 0.940, 1.871, 2.792, 3.744,
 37.800: 0.000, 0.942, 1.873, 2.794, 3.749,
 37.900: 0.000, 0.943, 1.875, 2.797, 3.755,
 38.000: 0.000, 0.945, 1.877, 2.799, 3.761,
 38.100: 0.000, 0.946, 1.878, 2.802, 3.768,
 38.200: 0.000, 0.947, 1.880, 2.806, 3.775,
 38.300: 0.000, 0.949, 1.881, 2.810, 3.782,
 38.400: 0.000, 0.949, 1.882, 2.814, 3.789,
 38.500: 0.000, 0.950, 1.884, 2.819, 3.797,
 38.600: 0.000, 0.951, 1.885, 2.824, 3.804,
 38.700: 0.000, 0.951, 1.886, 2.829, 3.812,
 38.800: 0.000, 0.951, 1.888, 2.835, 3.820,
 38.900: 0.000, 0.951, 1.889, 2.841, 3.827,
 39.000: 0.000, 0.951, 1.891, 2.847, 3.835,
 39.100: 0.000, 0.951, 1.893, 2.854, 3.843,
 39.200: 0.000, 0.951, 1.895, 2.861, 3.851,
 39.300: 0.000, 0.950, 1.897, 2.868, 3.859,
 39.400: 0.000, 0.950, 1.899, 2.875, 3.867,
 39.500: 0.000, 0.950, 1.902, 2.882, 3.875,
 39.600: 0.000, 0.949, 1.905, 2.889, 3.883,
 39.700: 0.000, 0.949, 1.908, 2.896, 3.891,
 39.800: 0.000, 0.949, 1.911, 2.904, 3.899,
 39.900: 0.000, 0.949, 1.915, 2.911, 3.907,
 40.000: 0.000, 0.949, 1.919, 2.918, 3.915,
 40.100: 0.000, 0.949, 1.923, 2.925, 3.922,
 40.200: 0.000, 0.950, 1.927, 2.932, 3.930,
 40.300: 0.000, 0.951, 1.932, 2.939, 3.938,
 40.400: 0.000, 0.952, 1.937, 2.945, 3.945,
 40.500: 0.000, 0.953, 1.942, 2.952, 3.953,
 40.600: 0.000, 0.955, 1.947, 2.958, 3.960,

40.700: 0.000, 0.957, 1.952, 2.964, 3.968,
 40.800: 0.000, 0.960, 1.958, 2.970, 3.975,
 40.900: 0.000, 0.962, 1.963, 2.976, 3.982,
 41.000: 0.000, 0.966, 1.969, 2.982, 3.989,
 41.100: 0.000, 0.969, 1.974, 2.987, 3.996,
 41.200: 0.000, 0.973, 1.980, 2.993, 4.002,
 41.300: 0.000, 0.977, 1.986, 2.998, 4.009,
 41.400: 0.000, 0.982, 1.991, 3.004, 4.015,
 41.500: 0.000, 0.986, 1.997, 3.009, 4.021,
 41.600: 0.000, 0.991, 2.003, 3.014, 4.027,
 41.700: 0.000, 0.996, 2.008, 3.019, 4.033,
 41.800: 0.000, 1.001, 2.014, 3.024, 4.039,
 41.900: 0.000, 1.007, 2.019, 3.029, 4.044,
 42.000: 0.000, 1.012, 2.024, 3.035, 4.049,
 42.100: 0.000, 1.017, 2.029, 3.040, 4.054,
 42.200: 0.000, 1.022, 2.035, 3.045, 4.059,
 42.300: 0.000, 1.027, 2.040, 3.050, 4.063,
 42.400: 0.000, 1.032, 2.045, 3.055, 4.068,
 42.500: 0.000, 1.036, 2.049, 3.060, 4.072,
 42.600: 0.000, 1.040, 2.054, 3.065, 4.076,
 42.700: 0.000, 1.044, 2.059, 3.070, 4.080,
 42.800: 0.000, 1.048, 2.064, 3.075, 4.084,
 42.900: 0.000, 1.051, 2.068, 3.079, 4.087,
 43.000: 0.000, 1.054, 2.073, 3.084, 4.091,
 43.100: 0.000, 1.056, 2.077, 3.089, 4.094,
 43.200: 0.000, 1.058, 2.081, 3.093, 4.098,
 43.300: 0.000, 1.060, 2.085, 3.098, 4.101,
 43.400: 0.000, 1.061, 2.089, 3.102, 4.104,
 43.500: 0.000, 1.062, 2.093, 3.106, 4.107,
 43.600: 0.000, 1.063, 2.096, 3.110, 4.110,
 43.700: 0.000, 1.063, 2.099, 3.114, 4.113,
 43.800: 0.000, 1.063, 2.102, 3.117, 4.116,
 43.900: 0.000, 1.063, 2.105, 3.121, 4.119,
 44.000: 0.000, 1.063, 2.107, 3.124, 4.122,
 44.100: 0.000, 1.062, 2.110, 3.127, 4.125,
 44.200: 0.000, 1.062, 2.111, 3.130, 4.128,
 44.300: 0.000, 1.061, 2.113, 3.132, 4.131,
 44.400: 0.000, 1.060, 2.114, 3.134, 4.134,
 44.500: 0.000, 1.059, 2.115, 3.136, 4.137,
 44.600: 0.000, 1.058, 2.115, 3.138, 4.140,
 44.700: 0.000, 1.057, 2.115, 3.140, 4.143,
 44.800: 0.000, 1.056, 2.115, 3.141, 4.145,
 44.900: 0.000, 1.055, 2.114, 3.142, 4.148,
 45.000: 0.000, 1.054, 2.113, 3.143, 4.151,
 45.100: 0.000, 1.054, 2.112, 3.144, 4.153,
 45.200: 0.000, 1.053, 2.111, 3.144, 4.155,
 45.300: 0.000, 1.052, 2.109, 3.144, 4.158,
 45.400: 0.000, 1.052, 2.107, 3.144, 4.160,
 45.500: 0.000, 1.051, 2.105, 3.144, 4.162,
 45.600: 0.000, 1.050, 2.103, 3.144, 4.163,
 45.700: 0.000, 1.050, 2.101, 3.143, 4.165,
 45.800: 0.000, 1.049, 2.098, 3.142, 4.166,
 45.900: 0.000, 1.049, 2.096, 3.141, 4.167,
 46.000: 0.000, 1.048, 2.094, 3.140, 4.168,
 46.100: 0.000, 1.048, 2.091, 3.138, 4.168,
 46.200: 0.000, 1.047, 2.089, 3.137, 4.168,
 46.300: 0.000, 1.047, 2.087, 3.135, 4.168,
 46.400: 0.000, 1.046, 2.085, 3.133, 4.168,
 46.500: 0.000, 1.045, 2.083, 3.131, 4.167,
 46.600: 0.000, 1.044, 2.081, 3.129, 4.166,
 46.700: 0.000, 1.044, 2.079, 3.127, 4.164,
 46.800: 0.000, 1.043, 2.078, 3.125, 4.162,

46.900: 0.000, 1.042, 2.076, 3.123, 4.160,
 47.000: 0.000, 1.041, 2.075, 3.120, 4.157,
 47.100: 0.000, 1.039, 2.074, 3.118, 4.154,
 47.200: 0.000, 1.038, 2.073, 3.115, 4.151,
 47.300: 0.000, 1.037, 2.072, 3.113, 4.148,
 47.400: 0.000, 1.036, 2.071, 3.110, 4.144,
 47.500: 0.000, 1.035, 2.070, 3.108, 4.139,
 47.600: 0.000, 1.034, 2.069, 3.105, 4.135,
 47.700: 0.000, 1.032, 2.068, 3.102, 4.130,
 47.800: 0.000, 1.031, 2.068, 3.100, 4.125,
 47.900: 0.000, 1.030, 2.067, 3.097, 4.120,
 48.000: 0.000, 1.029, 2.066, 3.094, 4.115,
 48.100: 0.000, 1.028, 2.065, 3.091, 4.109,
 48.200: 0.000, 1.028, 2.064, 3.089, 4.104,
 48.300: 0.000, 1.027, 2.063, 3.086, 4.098,
 48.400: 0.000, 1.026, 2.061, 3.083, 4.092,
 48.500: 0.000, 1.026, 2.060, 3.079, 4.086,
 48.600: 0.000, 1.025, 2.058, 3.076, 4.081,
 48.700: 0.000, 1.025, 2.057, 3.073, 4.075,
 48.800: 0.000, 1.025, 2.055, 3.070, 4.069,
 48.900: 0.000, 1.025, 2.053, 3.066, 4.063,
 49.000: 0.000, 1.024, 2.051, 3.062, 4.058,
 49.100: 0.000, 1.024, 2.049, 3.059, 4.052,
 49.200: 0.000, 1.024, 2.046, 3.055, 4.047,
 49.300: 0.000, 1.024, 2.044, 3.051, 4.042,
 49.400: 0.000, 1.024, 2.041, 3.047, 4.037,
 49.500: 0.000, 1.023, 2.039, 3.042, 4.032,
 49.600: 0.000, 1.023, 2.036, 3.038, 4.027,
 49.700: 0.000, 1.023, 2.033, 3.034, 4.022,
 49.800: 0.000, 1.022, 2.030, 3.029, 4.018,
 49.900: 0.000, 1.021, 2.027, 3.025, 4.013,
 50.000: 0.000, 1.021, 2.024, 3.020, 4.009,
 50.100: 0.000, 1.020, 2.021, 3.015, 4.005,
 50.200: 0.000, 1.018, 2.018, 3.011, 4.002,
 50.300: 0.000, 1.017, 2.015, 3.006, 3.998,
 50.400: 0.000, 1.015, 2.012, 3.002, 3.994,
 50.500: 0.000, 1.014, 2.009, 2.997, 3.991,
 50.600: 0.000, 1.012, 2.006, 2.993, 3.987,
 50.700: 0.000, 1.010, 2.002, 2.989, 3.984,
 50.800: 0.000, 1.008, 1.999, 2.985, 3.981,
 50.900: 0.000, 1.005, 1.996, 2.981, 3.978,
 51.000: 0.000, 1.003, 1.992, 2.977, 3.975,
 51.100: 0.000, 1.000, 1.989, 2.973, 3.972,
 51.200: 0.000, 0.998, 1.986, 2.970, 3.969,
 51.300: 0.000, 0.995, 1.982, 2.967, 3.966,
 51.400: 0.000, 0.992, 1.979, 2.963, 3.963,
 51.500: 0.000, 0.989, 1.976, 2.961, 3.960,
 51.600: 0.000, 0.987, 1.972, 2.958, 3.957,
 51.700: 0.000, 0.984, 1.969, 2.955, 3.954,
 51.800: 0.000, 0.981, 1.966, 2.953, 3.952,
 51.900: 0.000, 0.978, 1.963, 2.951, 3.949,
 52.000: 0.000, 0.976, 1.960, 2.949, 3.946,
 52.100: 0.000, 0.973, 1.956, 2.947, 3.944,
 52.200: 0.000, 0.971, 1.954, 2.945, 3.941,
 52.300: 0.000, 0.969, 1.951, 2.943, 3.939,
 52.400: 0.000, 0.967, 1.948, 2.941, 3.936,
 52.500: 0.000, 0.965, 1.946, 2.939, 3.934,
 52.600: 0.000, 0.963, 1.943, 2.938, 3.932,
 52.700: 0.000, 0.962, 1.941, 2.936, 3.929,
 52.800: 0.000, 0.960, 1.939, 2.934, 3.927,
 52.900: 0.000, 0.959, 1.937, 2.932, 3.925,
 53.000: 0.000, 0.958, 1.936, 2.931, 3.923,

53.100: 0.000, 0.957, 1.934, 2.929, 3.922,
 53.200: 0.000, 0.957, 1.933, 2.927, 3.920,
 53.300: 0.000, 0.957, 1.932, 2.926, 3.918,
 53.400: 0.000, 0.956, 1.931, 2.924, 3.917,
 53.500: 0.000, 0.956, 1.930, 2.922, 3.915,
 53.600: 0.000, 0.957, 1.930, 2.920, 3.914,
 53.700: 0.000, 0.957, 1.929, 2.919, 3.913,
 53.800: 0.000, 0.958, 1.929, 2.917, 3.911,
 53.900: 0.000, 0.958, 1.929, 2.916, 3.910,
 54.000: 0.000, 0.959, 1.930, 2.914, 3.909,
 54.100: 0.000, 0.960, 1.930, 2.913, 3.908,
 54.200: 0.000, 0.961, 1.930, 2.912, 3.907,
 54.300: 0.000, 0.962, 1.931, 2.911, 3.906,
 54.400: 0.000, 0.963, 1.932, 2.910, 3.905,
 54.500: 0.000, 0.964, 1.932, 2.909, 3.905,
 54.600: 0.000, 0.965, 1.933, 2.909, 3.904,
 54.700: 0.000, 0.967, 1.934, 2.909, 3.903,
 54.800: 0.000, 0.968, 1.935, 2.909, 3.903,
 54.900: 0.000, 0.969, 1.936, 2.909, 3.902,
 55.000: 0.000, 0.970, 1.937, 2.909, 3.902,
 55.100: 0.000, 0.971, 1.938, 2.910, 3.901,
 55.200: 0.000, 0.972, 1.940, 2.910, 3.901,
 55.300: 0.000, 0.973, 1.941, 2.911, 3.901,
 55.400: 0.000, 0.974, 1.942, 2.912, 3.901,
 55.500: 0.000, 0.975, 1.943, 2.914, 3.901,
 55.600: 0.000, 0.976, 1.944, 2.915, 3.901,
 55.700: 0.000, 0.977, 1.946, 2.917, 3.901,
 55.800: 0.000, 0.977, 1.947, 2.918, 3.902,
 55.900: 0.000, 0.978, 1.948, 2.920, 3.902,
 56.000: 0.000, 0.978, 1.949, 2.922, 3.903,
 56.100: 0.000, 0.978, 1.951, 2.924, 3.904,
 56.200: 0.000, 0.979, 1.952, 2.926, 3.905,
 56.300: 0.000, 0.979, 1.953, 2.928, 3.906,
 56.400: 0.000, 0.979, 1.954, 2.930, 3.908,
 56.500: 0.000, 0.979, 1.955, 2.932, 3.910,
 56.600: 0.000, 0.980, 1.957, 2.934, 3.912,
 56.700: 0.000, 0.980, 1.958, 2.936, 3.914,
 56.800: 0.000, 0.980, 1.959, 2.938, 3.916,
 56.900: 0.000, 0.980, 1.960, 2.940, 3.919,
 57.000: 0.000, 0.980, 1.961, 2.942, 3.922,
 57.100: 0.000, 0.980, 1.963, 2.944, 3.924,
 57.200: 0.000, 0.980, 1.964, 2.946, 3.928,
 57.300: 0.000, 0.980, 1.965, 2.947, 3.931,
 57.400: 0.000, 0.980, 1.966, 2.949, 3.934,
 57.500: 0.000, 0.981, 1.967, 2.951, 3.938,
 57.600: 0.000, 0.981, 1.968, 2.953, 3.941,
 57.700: 0.000, 0.981, 1.969, 2.955, 3.945,
 57.800: 0.000, 0.982, 1.970, 2.957, 3.948,
 57.900: 0.000, 0.982, 1.971, 2.959, 3.952,
 58.000: 0.000, 0.983, 1.971, 2.960, 3.956,
 58.100: 0.000, 0.983, 1.972, 2.962, 3.960,
 58.200: 0.000, 0.984, 1.973, 2.964, 3.964,
 58.300: 0.000, 0.985, 1.974, 2.967, 3.967,
 58.400: 0.000, 0.985, 1.975, 2.969, 3.971,
 58.500: 0.000, 0.986, 1.976, 2.971, 3.975,
 58.600: 0.000, 0.987, 1.977, 2.973, 3.978,
 58.700: 0.000, 0.988, 1.978, 2.976, 3.982,
 58.800: 0.000, 0.989, 1.979, 2.978, 3.985,
 58.900: 0.000, 0.989, 1.980, 2.980, 3.988,
 59.000: 0.000, 0.990, 1.982, 2.983, 3.991,
 59.100: 0.000, 0.991, 1.983, 2.986, 3.994,
 59.200: 0.000, 0.992, 1.984, 2.988, 3.997,

59.300: 0.000, 0.993, 1.986, 2.991, 4.000,
 59.400: 0.000, 0.994, 1.988, 2.993, 4.003,
 59.500: 0.000, 0.994, 1.989, 2.996, 4.005,
 59.600: 0.000, 0.995, 1.991, 2.999, 4.008,
 59.700: 0.000, 0.996, 1.993, 3.001, 4.010,
 59.800: 0.000, 0.997, 1.995, 3.004, 4.012,
 59.900: 0.000, 0.998, 1.997, 3.006, 4.014,
 60.000: 0.000, 0.999, 1.999, 3.009, 4.016,
 60.100: 0.000, 0.999, 2.002, 3.011, 4.018,
 60.200: 0.000, 1.000, 2.004, 3.014, 4.020,
 60.300: 0.000, 1.001, 2.006, 3.016, 4.022,
 60.400: 0.000, 1.002, 2.009, 3.019, 4.023,
 60.500: 0.000, 1.003, 2.011, 3.021, 4.025,
 60.600: 0.000, 1.004, 2.013, 3.023, 4.027,
 60.700: 0.000, 1.005, 2.016, 3.025, 4.028,
 60.800: 0.000, 1.006, 2.018, 3.027, 4.029,
 60.900: 0.000, 1.007, 2.020, 3.029, 4.031,
 61.000: 0.000, 1.008, 2.023, 3.031, 4.032,
 61.100: 0.000, 1.010, 2.025, 3.032, 4.034,
 61.200: 0.000, 1.011, 2.027, 3.034, 4.035,
 61.300: 0.000, 1.012, 2.029, 3.036, 4.036,
 61.400: 0.000, 1.014, 2.030, 3.037, 4.037,
 61.500: 0.000, 1.015, 2.032, 3.038, 4.039,
 61.600: 0.000, 1.016, 2.034, 3.040, 4.040,
 61.700: 0.000, 1.018, 2.035, 3.041, 4.041,
 61.800: 0.000, 1.019, 2.036, 3.042, 4.042,
 61.900: 0.000, 1.020, 2.037, 3.043, 4.043,
 62.000: 0.000, 1.021, 2.039, 3.044, 4.045,
 62.100: 0.000, 1.023, 2.039, 3.045, 4.046,
 62.200: 0.000, 1.024, 2.040, 3.046, 4.047,
 62.300: 0.000, 1.025, 2.041, 3.047, 4.048,
 62.400: 0.000, 1.026, 2.041, 3.048, 4.049,
 62.500: 0.000, 1.027, 2.042, 3.049, 4.050,
 62.600: 0.000, 1.027, 2.042, 3.049, 4.051,
 62.700: 0.000, 1.028, 2.043, 3.050, 4.052,
 62.800: 0.000, 1.028, 2.043, 3.050, 4.053,
 62.900: 0.000, 1.029, 2.043, 3.051, 4.054,
 63.000: 0.000, 1.029, 2.043, 3.051, 4.055,
 63.100: 0.000, 1.029, 2.043, 3.052, 4.056,
 63.200: 0.000, 1.029, 2.043, 3.052, 4.057,
 63.300: 0.000, 1.028, 2.043, 3.052, 4.058,
 63.400: 0.000, 1.028, 2.043, 3.053, 4.058,
 63.500: 0.000, 1.027, 2.043, 3.053, 4.059,
 63.600: 0.000, 1.027, 2.042, 3.053, 4.059,
 63.700: 0.000, 1.026, 2.042, 3.053, 4.060,
 63.800: 0.000, 1.025, 2.042, 3.053, 4.060,
 63.900: 0.000, 1.024, 2.041, 3.053, 4.061,
 64.000: 0.000, 1.023, 2.041, 3.053, 4.061,
 64.100: 0.000, 1.022, 2.041, 3.053, 4.061,
 64.200: 0.000, 1.021, 2.040, 3.053, 4.061,
 64.300: 0.000, 1.020, 2.039, 3.053, 4.061,
 64.400: 0.000, 1.019, 2.039, 3.053, 4.061,
 64.500: 0.000, 1.018, 2.038, 3.053, 4.061,
 64.600: 0.000, 1.017, 2.037, 3.052, 4.061,
 64.700: 0.000, 1.016, 2.037, 3.052, 4.060,
 64.800: 0.000, 1.015, 2.036, 3.051, 4.060,
 64.900: 0.000, 1.014, 2.035, 3.051, 4.060,
 65.000: 0.000, 1.013, 2.034, 3.050, 4.059,
 65.100: 0.000, 1.013, 2.033, 3.050, 4.058,
 65.200: 0.000, 1.012, 2.032, 3.049, 4.058,
 65.300: 0.000, 1.012, 2.031, 3.048, 4.057,
 65.400: 0.000, 1.011, 2.030, 3.047, 4.056,

65.500: 0.000, 1.011, 2.029, 3.046, 4.055,
 65.600: 0.000, 1.010, 2.028, 3.045, 4.054,
 65.700: 0.000, 1.010, 2.027, 3.044, 4.053,
 65.800: 0.000, 1.010, 2.026, 3.043, 4.052,
 65.900: 0.000, 1.010, 2.025, 3.042, 4.050,
 66.000: 0.000, 1.010, 2.024, 3.041, 4.049,
 66.100: 0.000, 1.010, 2.023, 3.039, 4.048,
 66.200: 0.000, 1.010, 2.022, 3.038, 4.046,
 66.300: 0.000, 1.010, 2.022, 3.036, 4.045,
 66.400: 0.000, 1.010, 2.021, 3.035, 4.043,
 66.500: 0.000, 1.010, 2.020, 3.033, 4.041,
 66.600: 0.000, 1.010, 2.019, 3.032, 4.040,
 66.700: 0.000, 1.010, 2.019, 3.030, 4.038,
 66.800: 0.000, 1.010, 2.018, 3.028, 4.036,
 66.900: 0.000, 1.010, 2.017, 3.027, 4.034,
 67.000: 0.000, 1.010, 2.017, 3.025, 4.032,
 67.100: 0.000, 1.010, 2.016, 3.023, 4.030,
 67.200: 0.000, 1.010, 2.016, 3.022, 4.028,
 67.300: 0.000, 1.010, 2.015, 3.020, 4.026,
 67.400: 0.000, 1.010, 2.015, 3.019, 4.024,
 67.500: 0.000, 1.010, 2.014, 3.017, 4.022,
 67.600: 0.000, 1.009, 2.014, 3.016, 4.019,
 67.700: 0.000, 1.009, 2.013, 3.014, 4.017,
 67.800: 0.000, 1.009, 2.013, 3.013, 4.015,
 67.900: 0.000, 1.008, 2.012, 3.011, 4.013,
 68.000: 0.000, 1.008, 2.011, 3.010, 4.010,
 68.100: 0.000, 1.007, 2.011, 3.008, 4.008,
 68.200: 0.000, 1.007, 2.010, 3.007, 4.006,
 68.300: 0.000, 1.006, 2.009, 3.006, 4.004,
 68.400: 0.000, 1.006, 2.008, 3.005, 4.002,
 68.500: 0.000, 1.005, 2.007, 3.003, 4.000,
 68.600: 0.000, 1.004, 2.006, 3.002, 3.998,
 68.700: 0.000, 1.004, 2.005, 3.001, 3.996,
 68.800: 0.000, 1.003, 2.004, 3.000, 3.994,
 68.900: 0.000, 1.002, 2.003, 2.999, 3.992,
 69.000: 0.000, 1.002, 2.002, 2.997, 3.991,
 69.100: 0.000, 1.001, 2.001, 2.996, 3.989,
 69.200: 0.000, 1.000, 2.000, 2.995, 3.987,
 69.300: 0.000, 1.000, 1.998, 2.994, 3.986,
 69.400: 0.000, 0.999, 1.997, 2.993, 3.985,
 69.500: 0.000, 0.998, 1.996, 2.991, 3.983,
 69.600: 0.000, 0.997, 1.995, 2.990, 3.982,
 69.700: 0.000, 0.997, 1.993, 2.989, 3.981,
 69.800: 0.000, 0.996, 1.992, 2.987, 3.980,
 69.900: 0.000, 0.995, 1.991, 2.986, 3.979,
 70.000: 0.000, 0.995, 1.989, 2.985, 3.979,
 70.100: 0.000, 0.994, 1.988, 2.983, 3.978,
 70.200: 0.000, 0.993, 1.987, 2.982, 3.977,
 70.299: 0.000, 0.993, 1.986, 2.981, 3.976,
 70.399: 0.000, 0.992, 1.985, 2.979, 3.976,
 70.499: 0.000, 0.992, 1.983, 2.978, 3.975,
 70.599: 0.000, 0.991, 1.982, 2.977, 3.975,
 70.699: 0.000, 0.990, 1.981, 2.976, 3.974,
 70.799: 0.000, 0.990, 1.980, 2.975, 3.974,
 70.899: 0.000, 0.989, 1.979, 2.973, 3.973,
 70.999: 0.000, 0.989, 1.978, 2.972, 3.973,
 71.099: 0.000, 0.988, 1.978, 2.972, 3.973,
 71.199: 0.000, 0.987, 1.977, 2.971, 3.972,
 71.299: 0.000, 0.987, 1.976, 2.970, 3.972,
 71.399: 0.000, 0.986, 1.975, 2.969, 3.971,
 71.499: 0.000, 0.986, 1.975, 2.969, 3.971,
 71.599: 0.000, 0.986, 1.974, 2.968, 3.970,

71.699: 0.000, 0.985, 1.974, 2.968, 3.970,
71.799: 0.000, 0.985, 1.973, 2.967, 3.969,
71.899: 0.000, 0.985, 1.973, 2.967, 3.969,
71.999: 0.000, 0.984, 1.972, 2.967, 3.968,
72.099: 0.000, 0.984, 1.972, 2.967, 3.967,
72.199: 0.000, 0.984, 1.972, 2.967, 3.967,
72.299: 0.000, 0.984, 1.972, 2.967, 3.966,
72.399: 0.000, 0.984, 1.972, 2.967, 3.966,
72.499: 0.000, 0.984, 1.972, 2.967, 3.965,
72.599: 0.000, 0.984, 1.972, 2.967, 3.965,
72.699: 0.000, 0.984, 1.972, 2.967, 3.964,
72.799: 0.000, 0.984, 1.972, 2.967, 3.964,
72.899: 0.000, 0.984, 1.973, 2.968, 3.963,
72.999: 0.000, 0.984, 1.973, 2.968, 3.963,
73.099: 0.000, 0.984, 1.973, 2.968, 3.963,
73.199: 0.000, 0.985, 1.974, 2.968, 3.963,
73.299: 0.000, 0.985, 1.974, 2.968, 3.962,
73.399: 0.000, 0.985, 1.975, 2.968, 3.962,
73.499: 0.000, 0.986, 1.975, 2.969, 3.962,
73.599: 0.000, 0.986, 1.976, 2.969, 3.962,
73.699: 0.000, 0.987, 1.977, 2.969, 3.962,
73.799: 0.000, 0.987, 1.977, 2.969, 3.962,
73.899: 0.000, 0.988, 1.978, 2.969, 3.963,
73.999: 0.000, 0.988, 1.979, 2.970, 3.963,
74.099: 0.000, 0.989, 1.979, 2.970, 3.963,
74.199: 0.000, 0.989, 1.980, 2.970, 3.964,
74.299: 0.000, 0.990, 1.980, 2.970, 3.964,
74.399: 0.000, 0.990, 1.981, 2.971, 3.965,
74.499: 0.000, 0.991, 1.982, 2.971, 3.965,
74.599: 0.000, 0.992, 1.982, 2.972, 3.966,
74.699: 0.000, 0.992, 1.983, 2.972, 3.966,
74.799: 0.000, 0.993, 1.983, 2.972, 3.967,
74.899: 0.000, 0.993, 1.984, 2.973, 3.968,
74.999: 0.000, 0.994, 1.984, 2.974, 3.969,
75.099: 0.000, 0.994, 1.985, 2.974, 3.970,
75.199: 0.000, 0.995, 1.985, 2.975, 3.970,
75.299: 0.000, 0.995, 1.986, 2.976, 3.971,
75.399: 0.000, 0.995, 1.986, 2.977, 3.972,
75.499: 0.000, 0.995, 1.987, 2.977, 3.973,
75.599: 0.000, 0.996, 1.987, 2.978, 3.974,
75.699: 0.000, 0.996, 1.988, 2.979, 3.975,
75.799: 0.000, 0.996, 1.988, 2.980, 3.977,
75.899: 0.000, 0.996, 1.988, 2.981, 3.978,
75.999: 0.000, 0.996, 1.989, 2.982, 3.979,
76.099: 0.000, 0.996, 1.989, 2.983, 3.980,
76.199: 0.000, 0.996, 1.990, 2.984, 3.981,
76.299: 0.000, 0.996, 1.990, 2.985, 3.982,
76.399: 0.000, 0.996, 1.990, 2.986, 3.984,
76.499: 0.000, 0.996, 1.991, 2.988, 3.985,
76.599: 0.000, 0.996, 1.991, 2.989, 3.986,
76.699: 0.000, 0.996, 1.992, 2.990, 3.987,
76.799: 0.000, 0.996, 1.992, 2.991, 3.989,
76.899: 0.000, 0.996, 1.993, 2.992, 3.990,
76.999: 0.000, 0.996, 1.993, 2.993, 3.991,
77.099: 0.000, 0.996, 1.994, 2.994, 3.992,
77.199: 0.000, 0.995, 1.994, 2.995, 3.994,
77.299: 0.000, 0.996, 1.995, 2.996, 3.995,
77.399: 0.000, 0.996, 1.996, 2.997, 3.996,
77.499: 0.000, 0.996, 1.996, 2.997, 3.997,
77.599: 0.000, 0.996, 1.997, 2.998, 3.999,
77.699: 0.000, 0.996, 1.997, 2.999, 4.000,
77.799: 0.000, 0.996, 1.998, 3.000, 4.001,

77.899: 0.000, 0.997, 1.998, 3.001, 4.002,
 77.999: 0.000, 0.997, 1.999, 3.002, 4.004,
 78.099: 0.000, 0.997, 1.999, 3.002, 4.005,
 78.199: 0.000, 0.998, 2.000, 3.003, 4.006,
 78.299: 0.000, 0.998, 2.001, 3.004, 4.007,
 78.399: 0.000, 0.999, 2.001, 3.005, 4.008,
 78.499: 0.000, 0.999, 2.002, 3.005, 4.009,
 78.599: 0.000, 1.000, 2.002, 3.006, 4.010,
 78.699: 0.000, 1.000, 2.003, 3.007, 4.011,
 78.799: 0.000, 1.001, 2.004, 3.007, 4.011,
 78.899: 0.000, 1.002, 2.004, 3.008, 4.012,
 78.999: 0.000, 1.002, 2.005, 3.009, 4.013,
 79.099: 0.000, 1.003, 2.005, 3.010, 4.014,
 79.199: 0.000, 1.003, 2.006, 3.010, 4.014,
 79.299: 0.000, 1.004, 2.007, 3.011, 4.015,
 79.399: 0.000, 1.005, 2.007, 3.012, 4.015,
 79.499: 0.000, 1.005, 2.008, 3.012, 4.016,
 79.599: 0.000, 1.006, 2.009, 3.013, 4.016,
 79.699: 0.000, 1.006, 2.009, 3.013, 4.017,
 79.799: 0.000, 1.007, 2.010, 3.014, 4.017,
 79.899: 0.000, 1.007, 2.011, 3.015, 4.018,
 79.999: 0.000, 1.007, 2.011, 3.015, 4.018,
 80.099: 0.000, 1.008, 2.012, 3.016, 4.018,
 80.199: 0.000, 1.008, 2.013, 3.016, 4.018,
 80.299: 0.000, 1.008, 2.013, 3.017, 4.019,
 80.399: 0.000, 1.008, 2.014, 3.017, 4.019,
 80.499: 0.000, 1.009, 2.015, 3.018, 4.019,
 80.599: 0.000, 1.009, 2.015, 3.018, 4.019,
 80.699: 0.000, 1.009, 2.016, 3.019, 4.019,
 80.799: 0.000, 1.009, 2.016, 3.019, 4.020,
 80.899: 0.000, 1.009, 2.017, 3.019, 4.020,
 80.999: 0.000, 1.009, 2.017, 3.020, 4.020,
 81.099: 0.000, 1.009, 2.017, 3.020, 4.020,
 81.199: 0.000, 1.009, 2.018, 3.020, 4.020,
 81.299: 0.000, 1.009, 2.018, 3.021, 4.020,
 81.399: 0.000, 1.009, 2.018, 3.021, 4.020,
 81.499: 0.000, 1.009, 2.018, 3.021, 4.021,
 81.599: 0.000, 1.009, 2.018, 3.021, 4.021,
 81.699: 0.000, 1.009, 2.018, 3.021, 4.021,
 81.799: 0.000, 1.009, 2.018, 3.021, 4.021,
 81.899: 0.000, 1.009, 2.018, 3.021, 4.021,
 81.999: 0.000, 1.009, 2.018, 3.021, 4.021,
 82.099: 0.000, 1.009, 2.017, 3.021, 4.021,
 82.199: 0.000, 1.009, 2.017, 3.021, 4.022,
 82.299: 0.000, 1.009, 2.017, 3.021, 4.022,
 82.399: 0.000, 1.009, 2.017, 3.021, 4.022,
 82.499: 0.000, 1.009, 2.016, 3.021, 4.022,
 82.599: 0.000, 1.009, 2.016, 3.020, 4.022,
 82.699: 0.000, 1.009, 2.015, 3.020, 4.022,
 82.799: 0.000, 1.009, 2.015, 3.020, 4.022,
 82.899: 0.000, 1.008, 2.015, 3.020, 4.022,
 82.999: 0.000, 1.008, 2.014, 3.019, 4.022,
 83.099: 0.000, 1.008, 2.014, 3.019, 4.022,
 83.199: 0.000, 1.008, 2.013, 3.019, 4.022,
 83.299: 0.000, 1.008, 2.013, 3.018, 4.022,
 83.399: 0.000, 1.007, 2.012, 3.018, 4.022,
 83.499: 0.000, 1.007, 2.012, 3.018, 4.022,
 83.599: 0.000, 1.007, 2.012, 3.017, 4.022,
 83.699: 0.000, 1.006, 2.011, 3.017, 4.021,
 83.799: 0.000, 1.006, 2.011, 3.016, 4.021,
 83.899: 0.000, 1.006, 2.011, 3.016, 4.021,
 83.999: 0.000, 1.005, 2.010, 3.016, 4.020,

84.099: 0.000, 1.005, 2.010, 3.015, 4.020,
 84.199: 0.000, 1.005, 2.009, 3.015, 4.020,
 84.299: 0.000, 1.004, 2.009, 3.014, 4.019,
 84.399: 0.000, 1.004, 2.009, 3.014, 4.019,
 84.499: 0.000, 1.004, 2.009, 3.014, 4.018,
 84.599: 0.000, 1.003, 2.008, 3.013, 4.017,
 84.699: 0.000, 1.003, 2.008, 3.013, 4.017,
 84.799: 0.000, 1.003, 2.008, 3.012, 4.016,
 84.899: 0.000, 1.003, 2.007, 3.012, 4.015,
 84.999: 0.000, 1.002, 2.007, 3.011, 4.015,
 85.099: 0.000, 1.002, 2.007, 3.011, 4.014,
 85.199: 0.000, 1.002, 2.006, 3.011, 4.013,
 85.299: 0.000, 1.002, 2.006, 3.010, 4.012,
 85.399: 0.000, 1.002, 2.006, 3.010, 4.011,
 85.499: 0.000, 1.002, 2.006, 3.009, 4.010,
 85.599: 0.000, 1.002, 2.005, 3.009, 4.010,
 85.699: 0.000, 1.002, 2.005, 3.008, 4.009,
 85.799: 0.000, 1.001, 2.005, 3.007, 4.008,
 85.899: 0.000, 1.001, 2.004, 3.007, 4.007,
 85.999: 0.000, 1.001, 2.004, 3.006, 4.006,
 86.099: 0.000, 1.002, 2.004, 3.006, 4.006,
 86.199: 0.000, 1.002, 2.003, 3.005, 4.005,
 86.299: 0.000, 1.002, 2.003, 3.005, 4.004,
 86.399: 0.000, 1.002, 2.003, 3.004, 4.003,
 86.499: 0.000, 1.002, 2.003, 3.003, 4.002,
 86.599: 0.000, 1.002, 2.002, 3.003, 4.002,
 86.699: 0.000, 1.002, 2.002, 3.002, 4.001,
 86.799: 0.000, 1.002, 2.002, 3.001, 4.000,
 86.899: 0.000, 1.002, 2.001, 3.001, 3.999,
 86.999: 0.000, 1.002, 2.001, 3.000, 3.999,
 87.099: 0.000, 1.001, 2.001, 2.999, 3.998,
 87.199: 0.000, 1.001, 2.000, 2.999, 3.997,
 87.299: 0.000, 1.001, 2.000, 2.998, 3.997,
 87.399: 0.000, 1.001, 2.000, 2.997, 3.996,
 87.499: 0.000, 1.001, 1.999, 2.997, 3.996,
 87.599: 0.000, 1.001, 1.999, 2.996, 3.995,
 87.699: 0.000, 1.001, 1.999, 2.996, 3.995,
 87.799: 0.000, 1.000, 1.998, 2.995, 3.994,
 87.899: 0.000, 1.000, 1.998, 2.995, 3.993,
 87.999: 0.000, 1.000, 1.998, 2.994, 3.993,
 88.099: 0.000, 1.000, 1.997, 2.994, 3.992,
 88.199: 0.000, 0.999, 1.997, 2.993, 3.992,
 88.299: 0.000, 0.999, 1.997, 2.993, 3.992,
 88.399: 0.000, 0.999, 1.996, 2.993, 3.991,
 88.499: 0.000, 0.998, 1.996, 2.992, 3.991,
 88.599: 0.000, 0.998, 1.995, 2.992, 3.990,
 88.699: 0.000, 0.998, 1.995, 2.992, 3.990,
 88.799: 0.000, 0.997, 1.994, 2.991, 3.990,
 88.899: 0.000, 0.997, 1.994, 2.991, 3.989,
 88.999: 0.000, 0.997, 1.994, 2.991, 3.989,
 89.099: 0.000, 0.996, 1.993, 2.991, 3.989,
 89.199: 0.000, 0.996, 1.993, 2.990, 3.988,
 89.299: 0.000, 0.996, 1.992, 2.990, 3.988,
 89.399: 0.000, 0.995, 1.992, 2.990, 3.988,
 89.499: 0.000, 0.995, 1.992, 2.990, 3.988,
 89.599: 0.000, 0.995, 1.991, 2.990, 3.988,
 89.699: 0.000, 0.995, 1.991, 2.989, 3.987,
 89.799: 0.000, 0.994, 1.991, 2.989, 3.987,
 89.899: 0.000, 0.994, 1.991, 2.989, 3.987,
 89.999: 0.000, 0.994, 1.990, 2.989, 3.987,
 90.099: 0.000, 0.994, 1.990, 2.989, 3.987,
 90.199: 0.000, 0.994, 1.990, 2.988, 3.987,

90.299: 0.000, 0.994, 1.990, 2.988, 3.987,
 90.399: 0.000, 0.994, 1.990, 2.988, 3.987,
 90.499: 0.000, 0.994, 1.989, 2.988, 3.987,
 90.599: 0.000, 0.993, 1.989, 2.988, 3.987,
 90.699: 0.000, 0.993, 1.989, 2.987, 3.987,
 90.799: 0.000, 0.994, 1.989, 2.987, 3.987,
 90.899: 0.000, 0.994, 1.989, 2.987, 3.987,
 90.999: 0.000, 0.994, 1.989, 2.987, 3.987,
 91.099: 0.000, 0.994, 1.989, 2.987, 3.987,
 91.199: 0.000, 0.994, 1.989, 2.987, 3.987,
 91.299: 0.000, 0.994, 1.989, 2.987, 3.987,
 91.399: 0.000, 0.994, 1.989, 2.987, 3.987,
 91.499: 0.000, 0.994, 1.990, 2.987, 3.987,
 91.599: 0.000, 0.994, 1.990, 2.987, 3.987,
 91.699: 0.000, 0.995, 1.990, 2.987, 3.987,
 91.799: 0.000, 0.995, 1.990, 2.987, 3.987,
 91.899: 0.000, 0.995, 1.990, 2.987, 3.987,
 91.999: 0.000, 0.995, 1.990, 2.987, 3.987,
 92.099: 0.000, 0.995, 1.991, 2.987, 3.987,
 92.199: 0.000, 0.995, 1.991, 2.988, 3.987,
 92.299: 0.000, 0.996, 1.991, 2.988, 3.987,
 92.399: 0.000, 0.996, 1.991, 2.988, 3.987,
 92.499: 0.000, 0.996, 1.991, 2.988, 3.987,
 92.599: 0.000, 0.996, 1.992, 2.989, 3.987,
 92.699: 0.000, 0.996, 1.992, 2.989, 3.987,
 92.799: 0.000, 0.996, 1.992, 2.989, 3.987,
 92.899: 0.000, 0.996, 1.993, 2.990, 3.987,
 92.999: 0.000, 0.997, 1.993, 2.990, 3.987,
 93.099: 0.000, 0.997, 1.993, 2.990, 3.988,
 93.199: 0.000, 0.997, 1.993, 2.990, 3.988,
 93.299: 0.000, 0.997, 1.994, 2.991, 3.988,
 93.399: 0.000, 0.997, 1.994, 2.991, 3.988,
 93.499: 0.000, 0.997, 1.994, 2.991, 3.988,
 93.599: 0.000, 0.997, 1.995, 2.992, 3.989,
 93.699: 0.000, 0.997, 1.995, 2.992, 3.989,
 93.799: 0.000, 0.998, 1.995, 2.992, 3.989,
 93.899: 0.000, 0.998, 1.996, 2.993, 3.990,
 93.999: 0.000, 0.998, 1.996, 2.993, 3.990,
 94.099: 0.000, 0.998, 1.996, 2.993, 3.991,
 94.199: 0.000, 0.998, 1.996, 2.993, 3.991,
 94.299: 0.000, 0.998, 1.997, 2.994, 3.991,
 94.399: 0.000, 0.998, 1.997, 2.994, 3.992,
 94.499: 0.000, 0.998, 1.997, 2.994, 3.992,
 94.599: 0.000, 0.999, 1.997, 2.995, 3.993,
 94.699: 0.000, 0.999, 1.997, 2.995, 3.993,
 94.799: 0.000, 0.999, 1.997, 2.995, 3.994,
 94.899: 0.000, 0.999, 1.998, 2.996, 3.994,
 94.999: 0.000, 0.999, 1.998, 2.996, 3.995,
 95.099: 0.000, 0.999, 1.998, 2.996, 3.995,
 95.199: 0.000, 0.999, 1.998, 2.997, 3.996,
 95.299: 0.000, 0.999, 1.998, 2.997, 3.997,
 95.399: 0.000, 0.999, 1.998, 2.997, 3.997,
 95.499: 0.000, 0.999, 1.998, 2.998, 3.998,
 95.599: 0.000, 1.000, 1.998, 2.998, 3.998,
 95.699: 0.000, 1.000, 1.999, 2.998, 3.999,
 95.799: 0.000, 1.000, 1.999, 2.999, 3.999,
 95.899: 0.000, 1.000, 1.999, 2.999, 4.000,
 95.999: 0.000, 1.000, 1.999, 2.999, 4.000,
 96.099: 0.000, 1.000, 1.999, 3.000, 4.001,
 96.199: 0.000, 1.000, 1.999, 3.000, 4.001,
 96.299: 0.000, 1.000, 2.000, 3.000, 4.002,
 96.399: 0.000, 1.000, 2.000, 3.001, 4.002,

```

96.499: 0.000, 1.000, 2.000, 3.001, 4.002,
96.599: 0.000, 1.000, 2.000, 3.002, 4.003,
96.699: 0.000, 1.000, 2.000, 3.002, 4.003,
96.799: 0.000, 1.000, 2.001, 3.002, 4.004,
96.899: 0.000, 1.000, 2.001, 3.003, 4.004,
96.999: 0.000, 1.000, 2.001, 3.003, 4.004,
97.099: 0.000, 1.000, 2.001, 3.003, 4.004,
97.199: 0.000, 1.000, 2.002, 3.004, 4.005,
97.299: 0.000, 1.000, 2.002, 3.004, 4.005,
97.399: 0.000, 1.000, 2.002, 3.004, 4.005,
97.499: 0.000, 1.000, 2.002, 3.004, 4.006,
97.599: 0.000, 1.001, 2.003, 3.005, 4.006,
97.699: 0.000, 1.001, 2.003, 3.005, 4.006,
97.799: 0.000, 1.001, 2.003, 3.005, 4.006,
97.899: 0.000, 1.001, 2.003, 3.005, 4.006,
97.999: 0.000, 1.001, 2.004, 3.006, 4.007,
98.099: 0.000, 1.001, 2.004, 3.006, 4.007,
98.199: 0.000, 1.001, 2.004, 3.006, 4.007,
98.299: 0.000, 1.002, 2.004, 3.006, 4.007,
98.399: 0.000, 1.002, 2.005, 3.006, 4.007,
98.499: 0.000, 1.002, 2.005, 3.007, 4.007,
98.599: 0.000, 1.002, 2.005, 3.007, 4.007,
98.699: 0.000, 1.002, 2.005, 3.007, 4.008,
98.799: 0.000, 1.003, 2.005, 3.007, 4.008,
98.899: 0.000, 1.003, 2.005, 3.007, 4.008,
98.999: 0.000, 1.003, 2.006, 3.007, 4.008,
99.099: 0.000, 1.003, 2.006, 3.007, 4.008,
99.199: 0.000, 1.003, 2.006, 3.007, 4.008,
99.299: 0.000, 1.004, 2.006, 3.007, 4.008,
99.399: 0.000, 1.004, 2.006, 3.008, 4.008,
99.499: 0.000, 1.004, 2.006, 3.008, 4.008,
99.599: 0.000, 1.004, 2.006, 3.008, 4.008,
99.699: 0.000, 1.004, 2.006, 3.008, 4.008,
99.799: 0.000, 1.004, 2.006, 3.008, 4.008,
99.899: 0.000, 1.004, 2.006, 3.008, 4.008,
99.999: 0.000, 1.004, 2.006, 3.008, 4.008,
Equilibrium reach in 302.309 second
Masses position: 0.000, 1.000, 2.000, 3.000, 4.000,
----- 2D example -----
Couldn't reach the equilibrium, start anyway the simulation with current state
SpringSys:
Number of dimension: 2
Dissipation: 0.100
Masses:
#0, pos(1.053,3.048,0.000), speed(-0.000,-0.000,0.000),
    stress(-0.000,-0.000,0.000), mass(0.100), fixed(0)
#1, pos(2.048,2.947,0.000), speed(0.000,-0.000,0.000),
    stress(0.000,-0.000,0.000), mass(0.100), fixed(0)
#2, pos(0.952,2.053,0.000), speed(-0.000,0.000,0.000),
    stress(-0.000,0.000,0.000), mass(0.100), fixed(0)
#3, pos(1.947,1.952,0.000), speed(0.000,0.000,0.000),
    stress(0.000,0.000,0.000), mass(0.100), fixed(0)
Springs:
#0, 0-1, length(1.000), stress(-0.000), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#1, 1-3, length(1.000), stress(0.000), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#2, 2-3, length(1.000), stress(-0.000), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#3, 0-2, length(1.000), stress(0.000), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#4, 1-2, length(1.414), stress(0.000), k(2.000), restLength(1.414),

```



```

    maxStress(-1000000.000,1000000.000), breakable(0)
#5, 0-3, length(1.414), stress(0.000), k(2.000), restLength(1.414),
    maxStress(-1000000.000,1000000.000), breakable(0)
Nearest mass to (1.0,1.0):
#1 0.709,1.026
Nearest spring to (1.0,1.0):
#1 1,3
Saved SpringSys:
Number of dimension: 2
Dissipation: 0.100
Masses:
#0, pos(-0.305,0.937,0.000), speed(-0.037,0.024,0.000),
    stress(0.009,0.107,0.000), mass(0.100), fixed(0)
#1, pos(0.709,1.026,0.000), speed(-0.045,0.027,0.000),
    stress(-0.012,0.128,0.000), mass(0.100), fixed(0)
#2, pos(-0.218,-0.007,0.000), speed(-0.044,0.033,0.000),
    stress(0.038,-0.122,0.000), mass(0.100), fixed(0)
#3, pos(0.812,0.081,0.000), speed(-0.040,0.118,0.000),
    stress(-0.036,-0.112,0.000), mass(0.100), fixed(0)
Springs:
#0, 0-1, length(1.018), stress(0.036), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#1, 1-3, length(0.947), stress(-0.106), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#2, 2-3, length(1.034), stress(0.067), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#3, 0-2, length(0.949), stress(-0.103), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#4, 1-2, length(1.389), stress(-0.051), k(2.000), restLength(1.414),
    maxStress(-1000000.000,1000000.000), breakable(0)
#5, 0-3, length(1.404), stress(-0.020), k(2.000), restLength(1.414),
    maxStress(-1000000.000,1000000.000), breakable(0)
Loaded SpringSys:
Number of dimension: 2
Dissipation: 0.100
Masses:
#0, pos(-0.305,0.937,0.000), speed(-0.037,0.024,0.000),
    stress(0.009,0.107,0.000), mass(0.100), fixed(0)
#1, pos(0.709,1.026,0.000), speed(-0.045,0.027,0.000),
    stress(-0.012,0.128,0.000), mass(0.100), fixed(0)
#2, pos(-0.218,-0.007,0.000), speed(-0.044,0.033,0.000),
    stress(0.038,-0.122,0.000), mass(0.100), fixed(0)
#3, pos(0.812,0.081,0.000), speed(-0.040,0.118,0.000),
    stress(-0.036,-0.112,0.000), mass(0.100), fixed(0)
Springs:
#0, 0-1, length(1.018), stress(0.036), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#1, 1-3, length(0.947), stress(-0.106), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#2, 2-3, length(1.034), stress(0.067), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#3, 0-2, length(0.949), stress(-0.103), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#4, 1-2, length(1.389), stress(-0.051), k(2.000), restLength(1.414),
    maxStress(-1000000.000,1000000.000), breakable(0)
#5, 0-3, length(1.404), stress(-0.020), k(2.000), restLength(1.414),
    maxStress(-1000000.000,1000000.000), breakable(0)
Remove spring #4:
Number of dimension: 2
Dissipation: 0.100
Masses:
#0, pos(-0.305,0.937,0.000), speed(-0.037,0.024,0.000),

```

```

    stress(0.009,0.107,0.000), mass(0.100), fixed(0)
#1, pos(0.709,1.026,0.000), speed(-0.045,0.027,0.000),
    stress(-0.012,0.128,0.000), mass(0.100), fixed(0)
#2, pos(-0.218,-0.007,0.000), speed(-0.044,0.033,0.000),
    stress(0.038,-0.122,0.000), mass(0.100), fixed(0)
#3, pos(0.812,0.081,0.000), speed(-0.040,0.118,0.000),
    stress(-0.036,-0.112,0.000), mass(0.100), fixed(0)
Springs:
#0, 0-1, length(1.018), stress(0.036), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#1, 1-3, length(0.947), stress(-0.106), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#2, 2-3, length(1.034), stress(0.067), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#3, 0-2, length(0.949), stress(-0.103), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#5, 0-3, length(1.404), stress(-0.020), k(2.000), restLength(1.414),
    maxStress(-1000000.000,1000000.000), breakable(0)
Remove mass #3:
Number of dimension: 2
Dissipation: 0.100
Masses:
#0, pos(-0.305,0.937,0.000), speed(-0.037,0.024,0.000),
    stress(0.009,0.107,0.000), mass(0.100), fixed(0)
#1, pos(0.709,1.026,0.000), speed(-0.045,0.027,0.000),
    stress(-0.012,0.128,0.000), mass(0.100), fixed(0)
#2, pos(-0.218,-0.007,0.000), speed(-0.044,0.033,0.000),
    stress(0.038,-0.122,0.000), mass(0.100), fixed(0)
Springs:
#0, 0-1, length(1.018), stress(0.036), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)
#3, 0-2, length(0.949), stress(-0.103), k(2.000), restLength(1.000),
    maxStress(-1000000.000,1000000.000), breakable(0)

```

7 springsys.dat

```

2
4
0
-0.304744 0.936908 0.000000
-0.036794 0.023556 0.000000
0.009352 0.106919 0.000000
0.100000
0
1
0.708834 1.026311 0.000000
-0.044818 0.026868 0.000000
-0.011725 0.127543 0.000000
0.100000
0
2
-0.218265 -0.007302 0.000000
-0.043558 0.033407 0.000000
0.038321 -0.122175 0.000000
0.100000
0
3
0.812110 0.081211 0.000000
-0.039871 0.118388 0.000000
-0.035948 -0.112287 0.000000

```

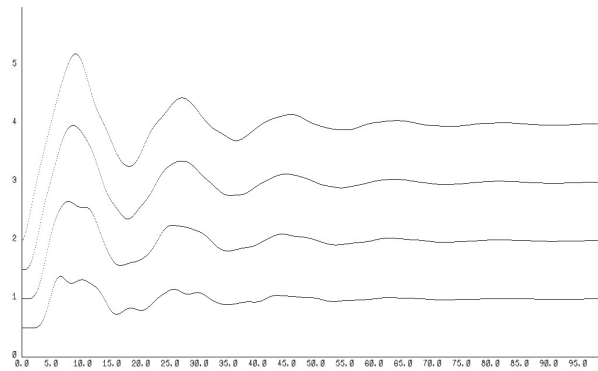
```

0.100000
0
6
0
1.017822
2.000000
1.000000
0.035643
-1000000.000000 1000000.000000
0 1
0
1
0.946968
2.000000
1.000000
-0.106063
-1000000.000000 1000000.000000
1 3
0
2
1.033539
2.000000
1.000000
0.067077
-1000000.000000 1000000.000000
2 3
0
3
0.948579
2.000000
1.000000
-0.102842
-1000000.000000 1000000.000000
0 2
0
4
1.388705
2.000000
1.414214
-0.051016
-1000000.000000 1000000.000000
1 2
0
5
1.404266
2.000000
1.414214
-0.019896
-1000000.000000 1000000.000000
0 3
0

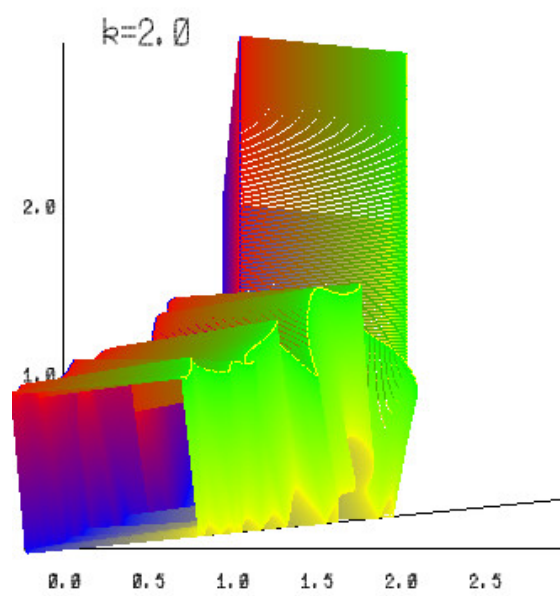
```

8 Plots

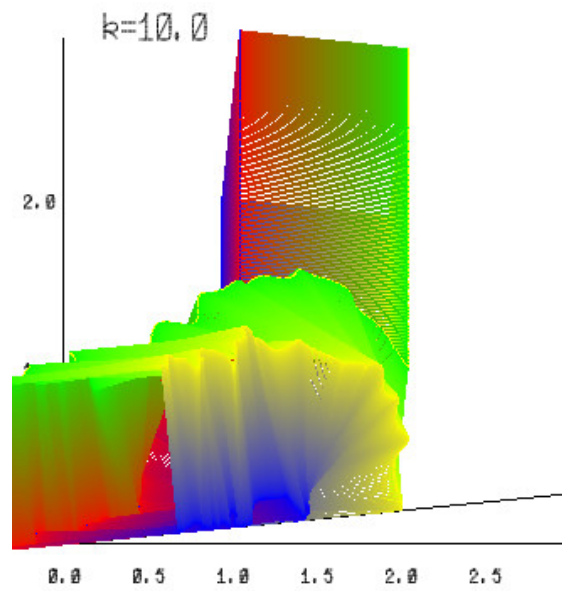
./springSys1D.jpg



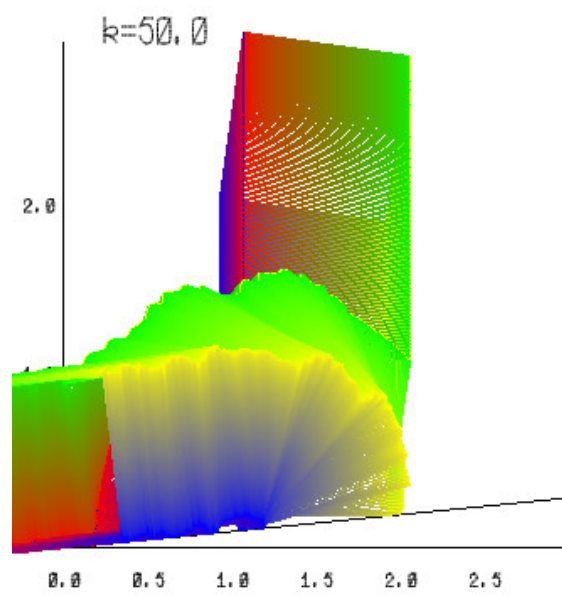
./springSys2Dk20.jpg



./springSys2Dk50.jpg



./springSys2Dk100.jpg



./springSys2Dk500.jpg

