МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФНОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
КАФЕДРА ОФ

Проект виртуальной лаборатории  
По дисциплине «Физика»  
Свободные колебания Системы  
Связанных пружинных маятников

|  |  |
| --- | --- |
| Факультет: | ПМИ |
| Группа: | ПМ-34 |
| Студенты: | Буняков И. Д., Рогоза А. А., Романов С. А. |
| Преподаватель: | Баранов А. В. |

2015 г.

# Цели и задачи проекта

Во время выполнения проекта ставились следующие ***цели***:

1. Описание физической модели проводимого эксперимента;
2. Составление математической модели;
3. Разработка программной реализации эксперимента.

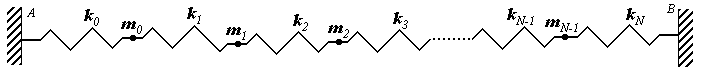
Данные цели предусматривали решение следующих задач:

1. Поиск и изучение информации о моделируемом эксперименте;
2. Поиск оптимальных инструментов для программной реализации;
3. Изучение выбранных инструментов;
4. Разделение ролей по разработке программного продукта;
5. Непосредственно разработка, отладка и тестирование.

# Физическая и математическая модели

## Физическая модель

Модели, представляющие собой линейные цепочки (рис. 1), состоящие из конечного или бесконечного числа связанных осцилляторов, оказались весьма эффективными и в настоящее время используются в различных областях физики: физике твердого тела, физике сплошных сред, химической физике, радиофизике и др. (Например, в [1] для описания системы трех связанных электрических колебательных контуров использована модель, представляющая собой систему шариков с массами *m1*, *m2*, *m3*, cвязанных между собой пружинками одинаковой жесткости *k*.) Используя модели линейных цепочек, оказывается возможным естественным образом осуществить переход к волновым процессам и ввести такие понятия как длина волны, групповая скорость, фазовая скорость, дисперсия и др.



*Рис. 1*

Отмеченные обстоятельства определяют целесообразность рассмотрения данных моделей в соответствующих курсах физики и компьютерного моделирования. Однако необходимо отметить два важных обстоятельства. Во-первых, аналитические решения уравнений движения длинных линейных цепочек () могут быть получены только для относительно небольшого числа случаев [2]: 1) **, **; 2) , ; ; 3) **, , ; 4) , , 5) , . Во-вторых, большинство этих решений оказываются весьма громоздкими и для их последующего анализа приходится использовать ПК.

Рассмотрим ситуацию, когда на тела не действует сила тяжести, отсутствуют внешние силы сопротивления, т.е. система совершает свободные колебания. Ввиду этого, начальная скорость движения тел в системе всегда будет равна нулю. Помимо этого, будем считать, что все тела в системе имеют одинаковую массу, а все пружины имеют одинаковый коэффициент жесткости.

## Математическая модель

Запишем уравнения движения для каждой массы колебательной системы, представленной на рис. 1:

 (1)

Для удобства дальнейшего решения запишем уравнение (1), введя обозначение , в следующем виде:

­ (2)

Следуя общему подходу к решению рассматриваемой задачи, изложенному в [3], ищем решение системы дифференциальных уравнений (2) в виде:

(3)

Подставив (3) в систему (2), сгруппировав члены, пропорциональные *Ai*, и записав систему в матричном виде, получим:

, (4)

где

,

***B*** − трехдиагональная матрица, элементы которой вычисляются по следующим правилам:

,

,, (5)

,

Необходимым и достаточным условием существования решения системы уравнений (4) является равенство нулю определителя матрицы ***B***

(6)

Уравнение (6), называемое характеристическим уравнением, является уравнением степени *N* **−** 1 относительно . Оно имеет в общем случае *N* **−** 1 различных вещественных положительных корней . Каждому собственному числу соответствует собственный вектор, являющийся решением уравнения

, (7)

где **Ω** − трехдиагональная матрица элементы которой вычисляются по следующим правилам:

,

,, (8)

,

Частоту называют частотой нормальных колебаний, а вектор − вектором нормального колебания, отвечающего α-ой частоте. Вектор нормального колебания  меняется во времени по закону

. (9)

Общее решение системы дифференциальных уравнений (2) , есть суперпозиция всех векторов нормальных колебаний :

, (10)

Где − произвольные постоянные, определяемые из начальных условий.

Скорость движения масс, можно определить, продифференцировав (10) по времени:

. (11)

Для решения задачи Коши системы дифференциальных уравнений (2) необходимо задать значения координат и скоростей каждого тела системы в начальный момент времени и решить систему уравнений

(12)

относительно неизвестных .

###### Запишем (12) в матричном виде

, (13)

где

, (14)

, (15)

, (16)

, (17)

**Z** − нулевая матрица, размерности *N−*1×*N* **−**9.

Система уравнений (13) оказывается нелинейной, однако, блочная структура матрицы, позволяет найти решение данной системы не прибегая к численным методам. Для этого, сначала, решив две линейные системы уравнений

(18)

, (19)

найдем векторы ,, затем координаты вектора ***С***

, (20)

и далее значения начальных фаз каждого нормального колебания:

. (21)

|  |
| --- |
|  |
| Рис. 2. К выбору правильного значения угла |

Отметим, что функция arctan на интервале [0;2π] является двузначной (рис. 2), поэтому для выбора правильных значений данной функции необходимо контролировать знаки числителя и знаменателя дроби в выражении (20). Как очевидно из рис. 2, правильное значение угла выбирается по следующим правилам:

(22)

# Алгоритм решения задачи

1. Задать число тел, образующих цепочку *N*.
2. Задать массы тел , *i*=0,1…, *N*−9.
3. Задать значения коэффициентов жесткости пружин , *i*=0,1…, *N*.
4. Вычислить элементы матрицы **Ω** в соответствие с (8).
5. Найти собственные числа матрицы **Ω**.
6. Найти собственные векторы , соответствующие набору собственных частот .
7. Задать начальные условия.
8. Решить систему линейных уравнений (18), (19), относительно векторов ***С***1 и ***С***2, соответственно.
9. Вычислить координаты вектора ***С*** в соответствии с (20).
10. Вычислить значения начальных фаз нормальных колебаний *ϕi* в соответствии с (21).

# Текст программы

Предоставим лишь наиболее интересные участки текста программы:

## Функция, отвечающая за отрисовку 3D модели в главном окне программы:

void Cscene3D::*paintGL*()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glScalef(nSca, nSca, nSca);

glRotatef(xRot, 0.0f, -1.0f, 0.0f);

glRotatef(90.0f, 1.0f, 0.0f,0.0f);

int n = theStorage.getNumOfSprings() - 1;

float balance = - 0.2 - 0.8 - (0.01 \* 9.81 / 50.0);

float spring\_step = -0.3f;

glTranslatef(0.0f, shift, 0.0f);

switch(n)

{

case 2:

wall\_left\_2.draw(); break;

case 3:

wall\_left\_3.draw(); break;

case 4:

wall\_left\_4.draw(); break;

}

glTranslatef(0.0f, -2\*shift, 0.0f);

spring\_end.draw();

wall\_right.draw();

glTranslatef(0.0f, 2\*shift, 0.0f);

double springs\_length = 0;

shift =( -(n+1)\*balance - n\*spring\_step)/2.0;

for(int i = 0; i < n; i++) {

if( i != 0) {

springs[i].resize( m\_action[i-1].x - balance - m\_action[i].x);

glTranslatef(0.0f, i\*(balance+spring\_step) + m\_action[i-1].x ,0.0f);

spring\_start.draw();

springs[i].draw();

glTranslatef(0.0f, -i\*(balance+spring\_step) - m\_action[i-1].x ,0.0f);

}

else {

spring\_start.draw();

springs[i].resize(- m\_action[i].x - balance);

springs[i].draw();

}

glTranslatef(0.0f, m\_action[i].x + (i+1)\*balance + i\*spring\_step,0.0f);

sphere.draw();

spring\_end.draw();

glTranslatef(0.0f, spring\_step ,0.0f);

springs\_length += spring\_step - m\_action[i].x + balance;

glTranslatef(0.0f, -(i+1)\*(balance + spring\_step) - m\_action[i].x ,0.0f);

}

glTranslatef(0.0f, n\*(balance+spring\_step) + m\_action[n-1].x ,0.0f);

spring\_start.draw();

springs[n].resize(2\*shift + n\*balance + n\*spring\_step + m\_action[n-1].x);

springs[n].draw();

}

## Функции, отвечающие за вычисление дифференциального уравнения

void Calculations::initializeCalculations( int n, int koefficient, float mass, vd & beginingShifts )

{

\_cleared = false;

alglib::real\_2d\_array OMEGA, Sigma1, VR, SigmaV, SigmaV1;

alglib::real\_1d\_array Teta, WI;

Teta.setlength( n );

SigmaV.setlength( n, n );

SigmaV1.setlength( n, n );

Sigma1.setlength( n, n );

WI.setlength( n );

OMEGA.setlength( n, n );

\_sigma.setlength( n, n );

VR.setlength( n, n );

vd m( n, mass );

vd k( n+1, koefficient );

vvd omega( n+1, vd(n, 0) );

int i, j;

for ( i = 0; i < n + 1; ++i )

for ( j = 0; j < n; ++j )

omega[i][j] = k[i] / m[j];

for ( i = 0; i < n; ++i )

for ( j = 0; j < n; ++j )

OMEGA[i][j] = 0;

for ( i = 0; i < n; ++i) {

if ( i == 0 ) {

OMEGA[i][i] = omega[0][0] + omega[1][0];

OMEGA[0][1] = -omega[1][0];

}

else if ( i > 0 ) {

if ( i < n-1 ) {

OMEGA[i][i - 1] = -omega[i][i];

OMEGA[i][i] = omega[i][i] + omega[i + 1][i];

OMEGA[i][i + 1] = -omega[i + 1][i];

}

else {

OMEGA[i][i - 1] = -omega[i][i];

OMEGA[i][i] = omega[i][i] + omega[i + 1][i];

}

}

}

alglib::rmatrixevd( OMEGA, n, 3, Teta, WI, VR, \_sigma );

double sum;

// normalize

for ( j = 0; j < n; ++j ) {

sum = 0;

for ( i = 0; i < n; ++i )

sum += \_sigma[i][j] \* \_sigma[i][j];

for ( i = 0; i < n; ++i )

\_sigma[i][j] /= sqrt( sum );

}

\_omega.resize( n, 0 );

for ( i = 0; i < n; ++i ) {

Teta[i] = sqrt( Teta[i] );

\_omega[i] = Teta[i];

}

for ( i = 0; i < n; ++i )

for( j = 0; j < n; ++j )

SigmaV[j][i] = -Teta[i] \* \_sigma(j, i);

for ( i = 0; i < n; ++i )

for ( j = 0; j < n; ++j ) {

Sigma1[i][j] = \_sigma[i][j];

SigmaV1[i][j] = SigmaV[i][j];

}

alglib::matinvreport rep;

alglib::ae\_int\_t g;

alglib::rmatrixinverse( Sigma1, g, rep );

alglib::rmatrixinverse( SigmaV1, g, rep );

vd C1( n, 0 );

vd C2( n, 0 );

vd V0( n, 0 );

multip( Sigma1, beginingShifts, C1 );

multip( SigmaV1, V0, C2 );

\_preAmplitudes.resize( n, 0 );

for ( i = 0; i < n; ++i )

\_preAmplitudes[i] = sqrt( C1[i] \* C1[i] + C2[i] \* C2[i] );

\_phita.resize( n, 0 );

for ( i = 0; i < n; ++i ) {

if ( \_preAmplitudes[i] == 0 )

\_phita[i] = 0;

else {

\_phita[i] = atan( C2[i] / C1[i] );

int ex = C1[i] \* 1000;

if ( ex < 0 )

\_phita[i] += M\_PI ;

else if ( ex > 0 ) {

int ex2 = C2[i] \* 1000;

if ( ex2 < 0 )

\_phita[i] += 2 \* M\_PI;

}

}

}

}

void Calculations::calculateShifts( double t, vd & x )

{

int i, j;

int n = \_preAmplitudes.size();

vd amplitude( n, 0 );

for ( int i = 0; i < n; ++i )

x[i] = 0;

for ( i = 0; i < n; ++i ) {

calculateAmplitudes( \_preAmplitudes, \_sigma, amplitude, i );

for ( j = 0; j < n; ++j )

x[j] += amplitude[j] \* cos( \_omega[i]\*t + \_phita[i] );

}

}

# Снимки экрана заставки, основного и дополнительных окон разработанной программы

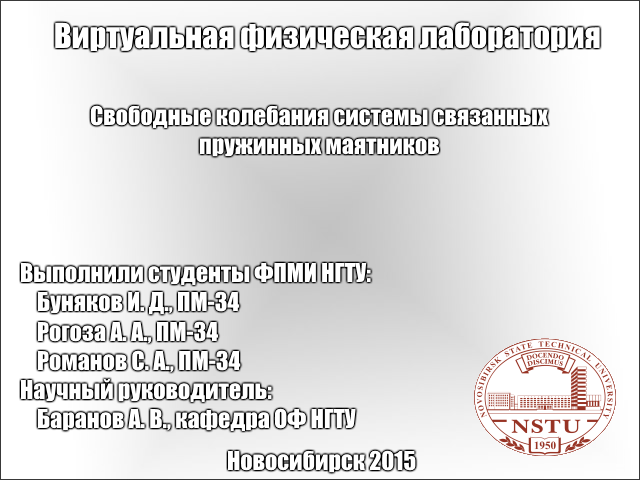


Рисунок 1. Заставка программы

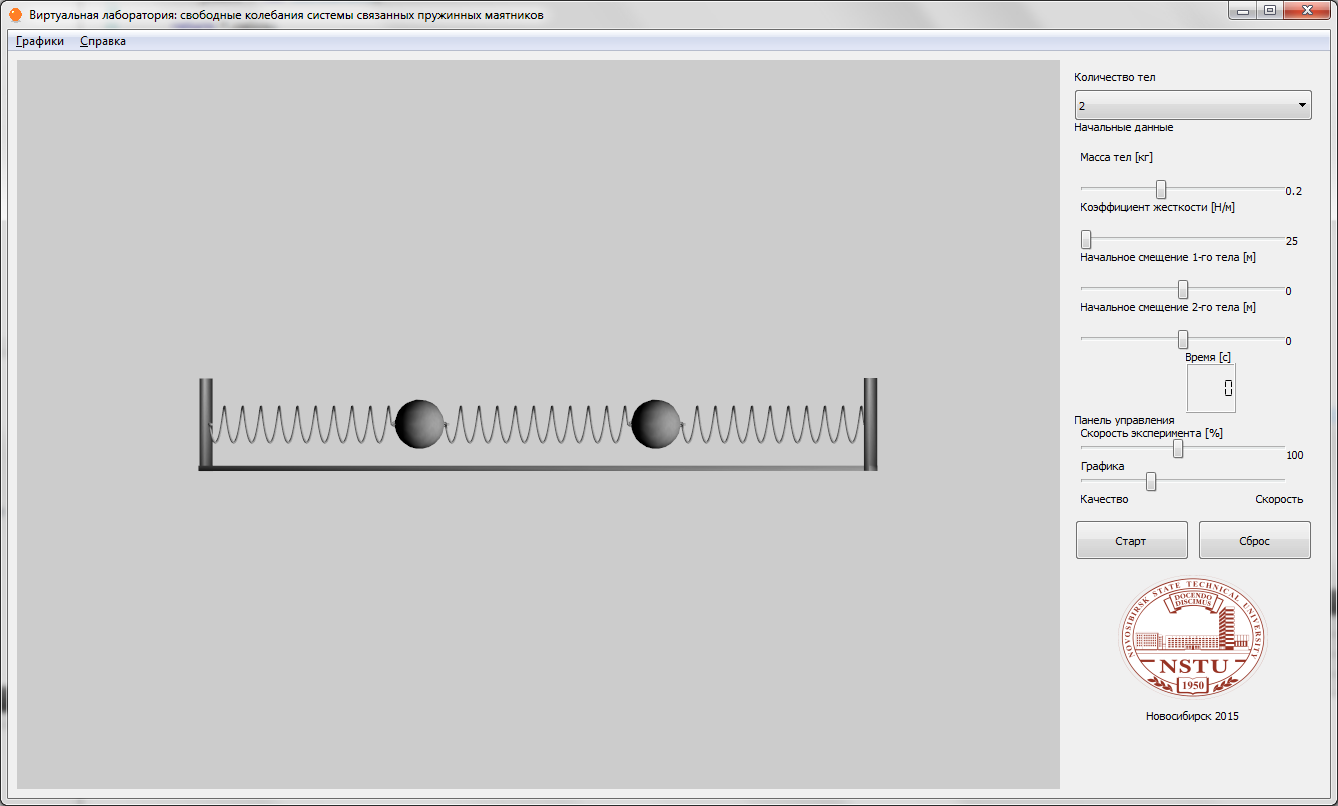


Рисунок 2. Главное окно

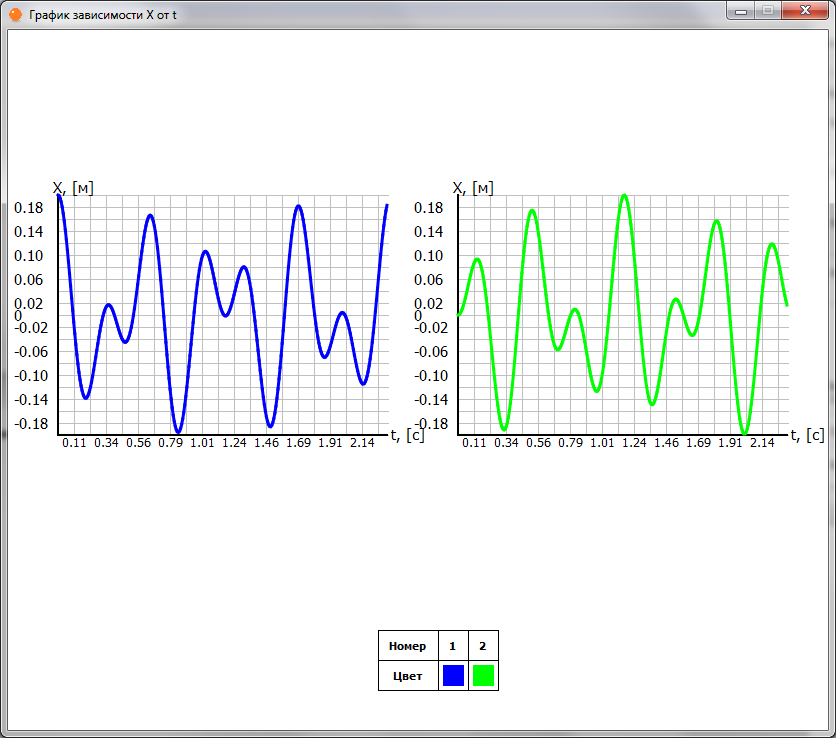


Рисунок 3. Один из графиков

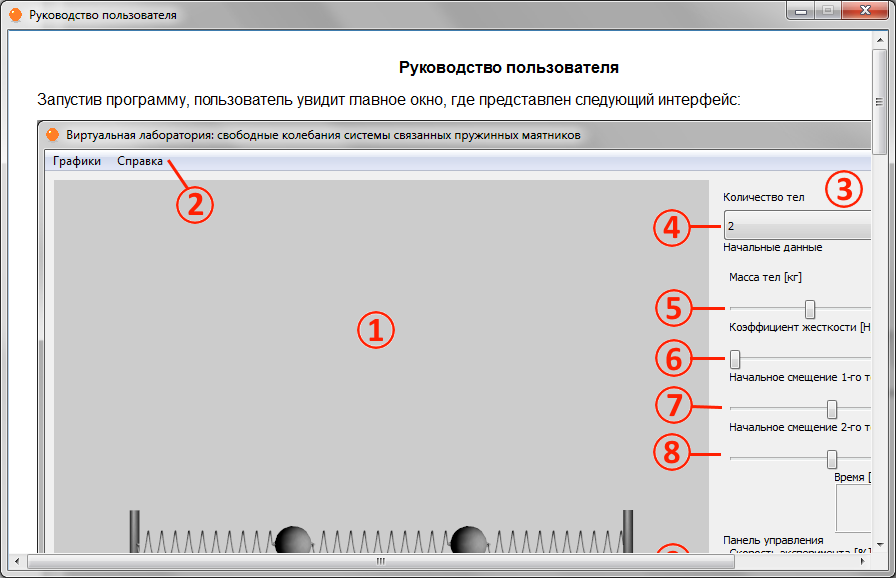


Рисунок 4. Руководство пользователя

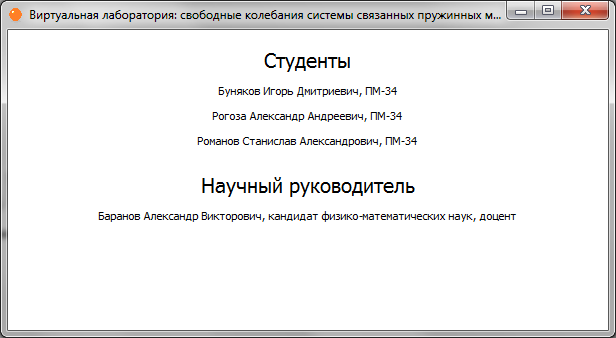


Рисунок 5. Информация о разработчиках

# Результаты проведения экспериментов

# Результаты проекта

## Выводы

## Пожелания

Хотелось бы выразить благодарность нашему научному руководителю за своевременные и содержательные консультации по природе физического явления, а так же общее руководство к действиям.