Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа программной инженерии

О Т Ч Е Т

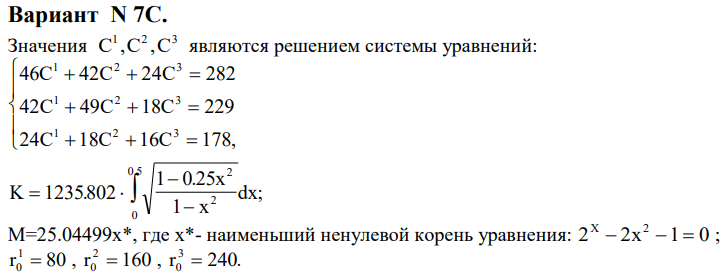
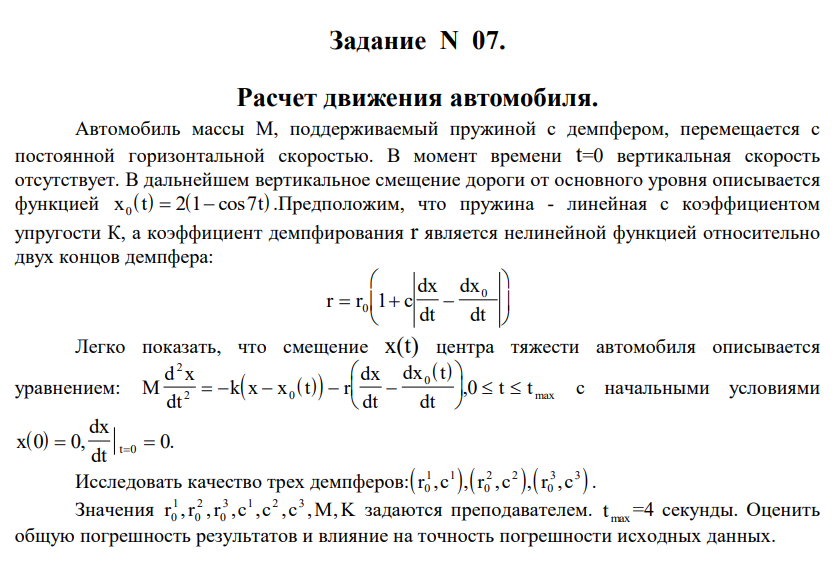
Курсовая работа

по дисциплине «Вычислительная математика»

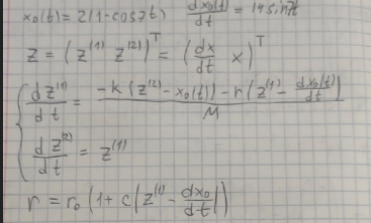
Выполнила студент гр. 3530904/90002 Афанасьев Е. Д.

Преподаватель Воскобойников С. П.

**Постанова задачи**



**Результат работы**

Преобразованная система уравнений и производная х0

M = 9.97886, K = 640

r1 = 80, c1 = 0

r2 = 160, c2 = 1

r3 = 240, c3 = 10

r1, c1

0 0 0

0.04 0.141668 0.00172691

0.08 0.72762 0.0173205

0.12 1.89206 0.0677118

0.16 3.63108 0.1764

0.2 5.82092 0.36425

0.24 8.24711 0.645249

0.28 10.6401 1.02358

0.32 12.7134 1.49219

0.36 14.1997 2.03281

0.4 14.8823 2.61742

0.44 14.6194 3.21075

0.48 13.3593 3.77363

0.52 11.1458 4.26674

0.56 8.11486 4.65436

0.6 4.48153 4.90787

0.64 0.520905 5.00851

0.68 -3.45657 4.94932

0.72 -7.13247 4.73601

0.76 -10.208 4.38675

0.8 -12.4296 3.93079

0.84 -13.611 3.40628

0.88 -13.6487 2.85719

0.92 -12.5307 2.32983

0.96 -10.3386 1.86909

1 -7.24035 1.51486

1.04 -3.47748 1.29875

1.08 0.654242 1.24159

1.12 4.82846 1.35166

1.16 8.71405 1.62401

1.2 12.0015 2.04078

1.24 14.4272 2.57261

1.28 15.7949 3.18082

1.32 15.9905 3.82053

1.36 14.9917 4.4441

1.4 12.8696 5.00486

1.44 9.78297 5.46077

1.48 5.96646 5.77774

1.52 1.71167 5.93223

1.56 -2.6553 5.91317

1.6 -6.79946 5.72279

1.64 -10.403 5.37644

1.68 -13.1902 4.90146

1.72 -14.9489 4.33497

1.76 -15.547 3.72106

1.8 -14.943 3.10728

1.84 -13.189 2.54099

1.88 -10.4267 2.06563

1.92 -6.87642 1.71738

1.96 -2.81988 1.52228

2 1.42176 1.49426

2.04 5.51291 1.63401

2.08 9.1298 1.92896

2.12 11.9856 2.35422

2.16 13.8528 2.87459

2.2 14.5812 3.44723

2.24 14.109 4.02504

2.28 12.4684 4.56033

2.32 9.7826 5.00854

2.36 6.2563 5.33171

2.4 2.15983 5.50143

2.44 -2.19188 5.50108

2.48 -6.46396 5.32714

2.52 -10.3275 4.98943

2.56 -13.4854 4.51037

2.6 -15.6951 3.92327

2.64 -16.788 3.26969

2.68 -16.6822 2.59625

2.72 -15.389 1.95098

2.76 -13.012 1.37961

2.8 -9.73909 0.921993

2.84 -5.82766 0.609017

2.88 -1.58471 0.46021

2.92 2.65714 0.482224

2.96 6.56555 0.668323

3 9.83437 0.998918

3.04 12.2075 1.4431

3.08 13.4987 1.96106

3.12 13.6063 2.50718

3.16 12.5211 3.03362

3.2 10.3269 3.49404

3.24 7.19406 3.84722

3.28 3.36642 4.06028

3.32 -0.857948 4.11124

3.36 -5.14986 3.99074

3.4 -9.17463 3.70282

3.44 -12.6182 3.26454

3.48 -15.2114 2.70473

3.52 -16.7514 2.0617

3.56 -17.117 1.38031

3.6 -16.2781 0.708444

3.64 -14.2986 0.0933091

3.68 -11.3306 -0.422231

3.72 -7.60344 -0.802994

3.76 -3.40506 -1.02421

3.8 0.939967 -1.07344

3.84 5.09584 -0.951542

3.88 8.74168 -0.672591

3.92 11.5965 -0.26278

3.96 13.4412 0.241629

4 14.1354 0.797138

r2, c2

0 0 0

0.04 0.229894 0.00297778

0.08 0.913914 0.0245135

0.12 1.98781 0.0812876

0.16 3.21686 0.18566

0.2 4.28426 0.336284

0.24 5.20711 0.52649

0.28 6.02841 0.751515

0.32 6.75467 1.0075

0.36 7.38092 1.29056

0.4 7.89876 1.59653

0.44 8.29706 1.92086

0.48 8.55985 2.25849

0.52 8.66004 2.60349

0.56 8.53988 2.94839

0.6 8.02715 3.28164

0.64 6.99377 3.58166

0.68 6.53364 3.85033

0.72 6.39334 4.10836

0.76 6.36475 4.36323

0.8 6.41295 4.61855

0.84 6.52697 4.87714

0.88 6.70027 5.1415

0.92 6.92686 5.41388

0.96 7.19986 5.69627

1 7.51072 5.99037

1.04 7.84869 6.29749

1.08 8.20052 6.61845

1.12 8.55049 6.95351

1.16 8.88052 7.30223

1.2 9.17115 7.66342

1.24 9.40326 8.03514

1.28 9.56049 8.41468

1.32 9.63214 8.79883

1.36 9.61529 9.18407

1.4 9.51521 9.56694

1.44 9.34369 9.94433

1.48 9.11634 10.3137

1.52 8.8499 10.6731

1.56 8.56036 11.0214

1.6 8.26191 11.3578

1.64 7.96635 11.6824

1.68 7.6831 11.9953

1.72 7.419 12.2973

1.76 7.17841 12.5891

1.8 6.96325 12.8719

1.84 6.77303 13.1465

1.88 6.60487 13.414

1.92 6.45359 13.6751

1.96 6.31188 13.9304

2 6.17054 14.1801

2.04 6.01877 14.4239

2.08 5.84504 14.6613

2.12 5.63765 14.8911

2.16 5.38596 15.1117

2.2 5.08156 15.3213

2.24 4.71936 15.5175

2.28 4.29838 15.698

2.32 3.82194 15.8606

2.36 3.29723 16.0031

2.4 2.73439 16.1239

2.44 2.14529 16.2215

2.48 1.5424 16.2953

2.52 0.937781 16.3449

2.56 0.342337 16.3705

2.6 -0.234689 16.3725

2.64 -0.786046 16.352

2.68 -1.30666 16.3101

2.72 -1.79374 16.2479

2.76 -2.24684 16.167

2.8 -2.6678 16.0686

2.84 -3.06073 15.954

2.88 -3.43182 15.8241

2.92 -3.78917 15.6796

2.96 -4.14239 15.521

3 -4.50207 15.3481

3.04 -4.87903 15.1606

3.08 -5.28323 14.9575

3.12 -5.7227 14.7375

3.16 -6.20227 14.4991

3.2 -6.72272 14.2407

3.24 -7.28035 13.9608

3.28 -7.86727 13.6579

3.32 -8.47234 13.3312

3.36 -9.08228 12.9801

3.4 -9.68311 12.6047

3.44 -10.2611 12.2057

3.48 -10.8038 11.7843

3.52 -11.3004 11.342

3.56 -11.7421 10.881

3.6 -12.1221 10.4035

3.64 -12.4362 9.9121

3.68 -12.6818 9.40951

3.72 -12.8588 8.89847

3.76 -12.9689 8.3817

3.8 -13.0156 7.86181

3.84 -13.0042 7.34123

3.88 -12.9417 6.82216

3.92 -12.8365 6.30647

3.96 -12.6984 5.79568

4 -12.5389 5.29088

r3, c3

0 0 0

0.04 0.393801 0.00575976

0.08 1.07093 0.0345767

0.12 1.90139 0.0941025

0.16 2.60879 0.18454

0.2 3.27595 0.302323

0.24 3.91779 0.44628

0.28 4.53412 0.615406

0.32 5.12294 0.808643

0.36 5.68166 1.02484

0.4 6.20736 1.26273

0.44 6.69659 1.52094

0.48 7.14485 1.79792

0.52 7.5449 2.09189

0.56 7.88139 2.40067

0.6 8.10167 2.72093

0.64 8.00199 3.04337

0.68 8.20632 3.3671

0.72 8.47121 3.70056

0.76 8.75547 4.04505

0.8 9.05017 4.40114

0.84 9.35116 4.76915

0.88 9.65571 5.14928

0.92 9.96144 5.54162

0.96 10.2659 5.94618

1 10.5661 6.36284

1.04 10.8589 6.79137

1.08 11.1402 7.23139

1.12 11.4057 7.68237

1.16 11.6504 8.14357

1.2 11.8695 8.61406

1.24 12.0582 9.09273

1.28 12.2129 9.57827

1.32 12.3312 10.0693

1.36 12.4127 10.5643

1.4 12.4586 11.0618

1.44 12.4714 11.5605

1.48 12.4544 12.0591

1.52 12.4114 12.5565

1.56 12.3458 13.0517

1.6 12.261 13.5439

1.64 12.1595 14.0324

1.68 12.0439 14.5165

1.72 11.9158 14.9957

1.76 11.7762 15.4696

1.8 11.6257 15.9377

1.84 11.4643 16.3995

1.88 11.2915 16.8547

1.92 11.1061 17.3027

1.96 10.9065 17.743

2 10.6908 18.175

2.04 10.4568 18.598

2.08 10.2021 19.0112

2.12 9.92445 19.4139

2.16 9.62204 19.8049

2.2 9.29348 20.1833

2.24 8.93826 20.548

2.28 8.55668 20.898

2.32 8.15001 21.2322

2.36 7.72018 21.5497

2.4 7.26992 21.8495

2.44 6.80222 22.131

2.48 6.32029 22.3935

2.52 5.82727 22.6365

2.56 5.32609 22.8596

2.6 4.81937 23.0625

2.64 4.30926 23.2451

2.68 3.79747 23.4072

2.72 3.28515 23.5489

2.76 2.77294 23.6701

2.8 2.26095 23.7707

2.84 1.7488 23.8509

2.88 1.23568 23.9106

2.92 0.720404 23.9498

2.96 0.201569 23.9682

3 -0.322329 23.9658

3.04 -0.852693 23.9423

3.08 -1.39063 23.8975

3.12 -1.93675 23.831

3.16 -2.49101 23.7424

3.2 -3.05259 23.6316

3.24 -3.61991 23.4982

3.28 -4.1907 23.342

3.32 -4.76211 23.1629

3.36 -5.33097 22.961

3.4 -5.89398 22.7365

3.44 -6.44789 22.4896

3.48 -6.98965 22.2208

3.52 -7.51661 21.9306

3.56 -8.02652 21.6197

3.6 -8.51768 21.2888

3.64 -8.98894 20.9386

3.68 -9.43971 20.5699

3.72 -9.87 20.1837

3.76 -10.2804 19.7806

3.8 -10.6718 19.3615

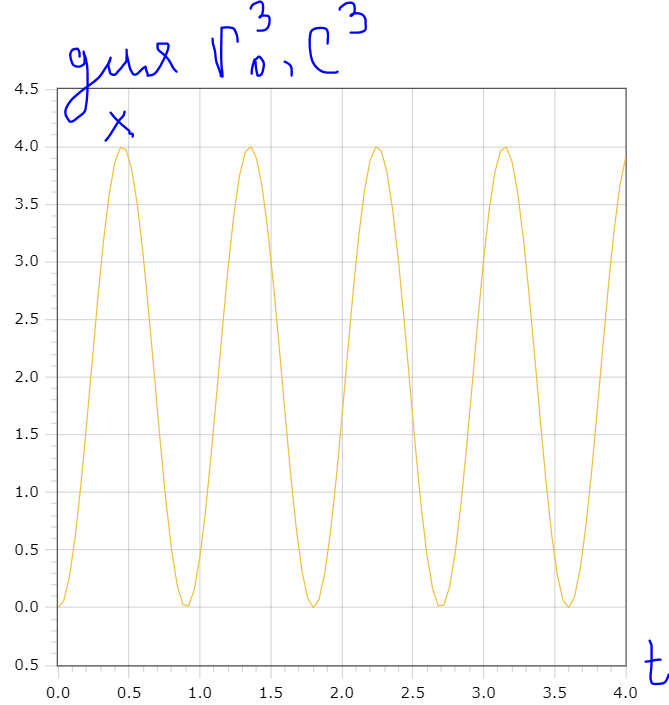
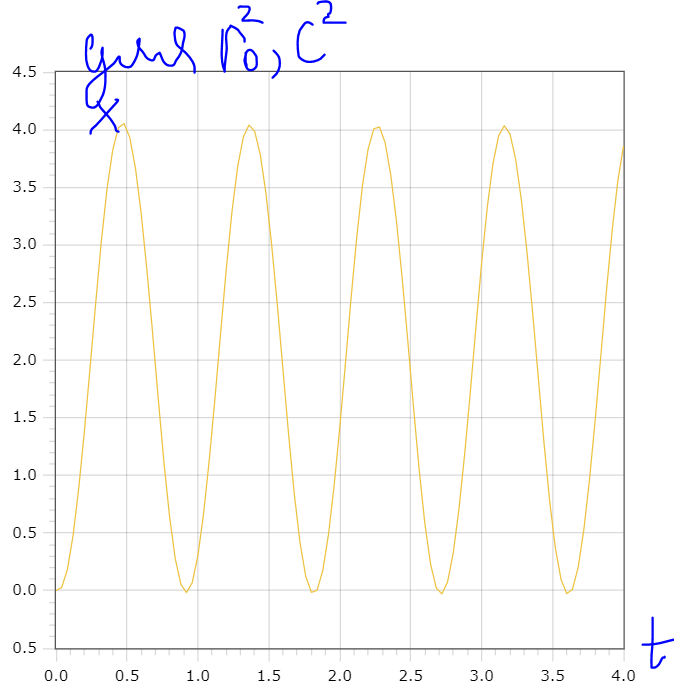
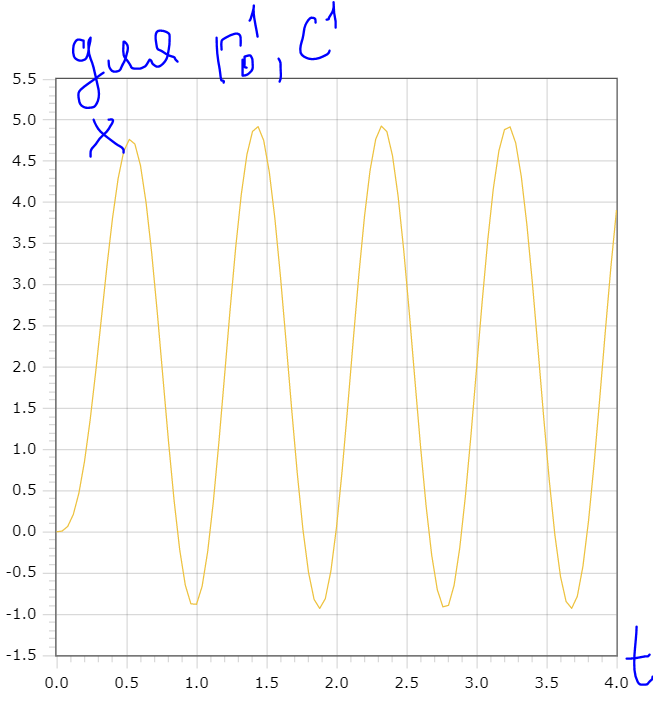
3.84 -11.046 18.9271

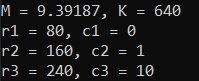
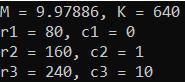
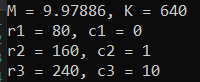
3.88 -11.4046 18.478

3.92 -11.7497 18.0149

3.96 -12.0835 17.5382

4 -12.4077 17.0483



Эти результаты получены с погрешностью 10е-5. Если ее увеличивать, то это будет сильно влиять на параметр М. Вот начальные значения при погрешности 10е-2, 10е-3 и 10е-4

**Вывод**

Результат зависит от начальных значений, особенно параметр М.

Первый демпфер имеет наибольшее отклонение среди всех, при чем оно еще бывает меньше нуля.

Второй демпфер имеет меньшее отклонение, чем первый.

Третий демпфер почти такой же, как второй, но самую малость лучше.

**Код программы**

Course.cpp

#include <iostream>

#include <vector>

#include "get\_c.hpp"

#include "get\_m.hpp"

#include "get\_k.hpp"

#include "rkf45.c"

Matrix R = {80, 160, 240};

Matrix C;

double M, K;

struct pair

{

double r0, c;

};

std::vector<pair> get\_pairs\_of\_r\_c(Matrix C, Matrix R);

void do\_rkf(int (\*f)(int n, double t, double x[], double xp[]));

double x0(double t);

double dx0dt(double t);

double r1(double t, double x);

int func1(int n, double t, double x[], double dxdt[]);

double r2(double t, double x);

int func2(int n, double t, double x[], double dxdt[]);

double r3(double t, double x);

int func3(int n, double t, double x[], double dxdt[]);

int main()

{

M = get\_m();

K = get\_k();

C = get\_c();

std::vector<pair> pairs\_r\_c = get\_pairs\_of\_r\_c(get\_c(), R);

std::cout << "M = " << M << ", " << "K = " << K << '\n';

for (int i = 0; i < 3; i++)

{

std::cout << "r" << i + 1 << " = " << pairs\_r\_c.at(i).r0 << ", c" << i + 1 << " = " << pairs\_r\_c.at(i).c << '\n';

}

std::cout << "r1, c1\n";

do\_rkf(func1);

std::cout << "r2, c2\n";

do\_rkf(func2);

std::cout << "r3, c3\n";

do\_rkf(func3);

}

void do\_rkf(int (\*f)(int n, double t, double x[], double xp[]))

{

Matrix x\_start = {0, 0};

Matrix dxdt(2);

double t\_init = 0, out = 0, re = 1e-5, ae = 1e-5, h = 0;

int flag = 1, fail = 1, nfe = 0, maxnfe = 10e7;

rkfinit(2, &fail);

for (double t = 0; t <= 4.001; t += 0.04)

{

rkf45(f, 2, x\_start.data(), dxdt.data(), &t\_init, t, &re, ae, &h, &nfe, maxnfe, &flag);

std::cout << t << '\t' << x\_start[0] << ' ' << x\_start[1] << '\n';

}

//rkfend;

}

std::vector<pair> get\_pairs\_of\_r\_c(Matrix C, Matrix R)

{

std::vector<pair> pairs;

for (int i = 0; i < 3; i++)

{

pairs.push\_back({R.at(i), C.at(i)});

}

return pairs;

}

double x0(double t)

{

return 2 \* (1 - cos(7 \* t));

}

double dx0dt(double t)

{

return 14 \* sin(t);

}

double r1(double t, double x)

{

return R[0] \* (1 + C[0] \* abs(x - dx0dt(t)));

}

int func1(int n, double t, double x[], double dxdt[])

{

dxdt[0] = (K \* (x[1] - x0(t)) + r1(t, x[0]) \* (x[0] - dx0dt(t))) / (-M);

dxdt[1] = x[0];

return 0;

}

double r2(double t, double x)

{

return R[1] \* (1 + C[1] \* abs(x - dx0dt(t)));

}

int func2(int n, double t, double x[], double dxdt[])

{

dxdt[0] = (K \* (x[1] - x0(t)) + r2(t, x[0]) \* (x[0] - dx0dt(t))) / (-M);

dxdt[1] = x[0];

return 0;

}

double r3(double t, double x)

{

return R[2] \* (1 + C[2] \* abs(x - dx0dt(t)));

}

int func3(int n, double t, double x[], double dxdt[])

{

dxdt[0] = (K \* (x[1] - x0(t)) + r3(t, x[0]) \* (x[0] - dx0dt(t))) / (-M);

dxdt[1] = x[0];

return 0;

}

Get\_m.hpp

#pragma once

double get\_m();

double func(double& x);

double bisection(double (\*func)(double& x), double a, double b, double eps);

get\_m.cpp

#include "get\_m.hpp"

#include <cmath>

double get\_m()

{

double x = bisection(func, 0, 2, 10e-4);

return x \* 25.04499;

}

double func(double& x)

{

return std::pow(2, x) - 2 \* x \* x - 1;

}

double bisection(double (\*func)(double& x), double begin, double end, double eps)

{

double dis = 0;

while (end - begin > eps)

{

double center = (begin + end) / 2;

int last = 0;

double f\_end = func(end);

double f\_center = func(center);

if (f\_end \* f\_center < 0)

{

begin = center;

}

else

{

end = center;

}

dis = f\_center;

}

return begin;

}

Get\_k.hpp

#pragma once

double get\_k();

double func(double x);

double get\_integral();

get\_k.cpp

#include "get\_k.hpp"

#include <cmath>

#include "cmath.h"

#include "quanc8.c"

double get\_k()

{

return 1235.802 \* get\_integral();

}

double func(double x)

{

return std::sqrt((1 - 0.25 \* x \* x) / (1 - x \* x));

}

double get\_integral()

{

double a = 0, b = 0.5;

double abserr = 10e-3, relerr = 10e-3;

double result = 0, errest = 0;

int nofunr = 0;

double posnr = 0;

int flag = 0;

quanc8(func, a, b, abserr, relerr, &result, &errest, &nofunr, &posnr, &flag);

return result;

}

Get\_c.hpp

#pragma once

#include <vector>

using Matrix = std::vector<double>;

Matrix get\_c();

Matrix create\_inverse\_matrix(Matrix m);

Matrix create\_identity\_matrix();

Get\_c.cpp

#include "get\_c.hpp"

#include "cmath.h"

#include "decomp.c"

Matrix a = {46, 42, 24, 42, 49, 18, 24, 18, 16};

Matrix b = {282, 229, 178};

const int size = 3;

double cond = 0;

Matrix get\_c()

{

Matrix x(size);

Matrix inv\_a = create\_inverse\_matrix(a);

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

{

for (int j = 0; j < size; j++)

{

x[i] += inv\_a[i \* size + j] \* b[j];

}

}

return x;

}

Matrix create\_inverse\_matrix(Matrix m)

{

Matrix matrix(size \* size);

int pivot[size];

int flag = 0;

decomp(size, size, m.data(), &cond, pivot, &flag);

Matrix e = create\_identity\_matrix();

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

{

Matrix b(size);

for (int j = 0; j < size; j++)

{

b[j] = e[j \* size + i];

}

solve(size, size, m.data(), b.data(), pivot);

for (int k = 0; k < size; k++)

{

matrix[k \* size + i] = b[k];

}

}

return matrix;

}

Matrix create\_identity\_matrix()

{

Matrix m(size \* size);

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

{

for (int j = 0; j < size; j++)

{

m[i \* size + j] = (i == j) ? 1 : 0;

}

}

return m;

}