Правообладатель:

**Русяк Иван Григорьевича, Королев Станислав Анатольевичб**

**Мансуров Рустам Ренатовичв,**

**а 426069, Российская Федерация, Ижевск, ул. 30 лет Победы, д. 43 , кв. 458**

**б 426069, Российская Федерация, Ижевск, ул. 30 лет Победы, д. 43, кв. 155**

**в 426072, Российская Федерация, Ижевск, ул. Молодежная, д. 45, кв. 3**

Программа для ЭВМ

**Программа для решения задачи оптимизации параметров внутренней и внешней баллистики активно-реактивного снаряда с целью повышения дальности стрельбы**

Фрагменты исходного текста программы, листов – **\_32\_**

Материалы аудиовизуальных отображений,

порождаемых программой для ЭВМ, листов – **\_2\_**

Авторы: И.Г. Русяк

С.А. Королев

Р.Р. Мансуров

## Ижевск – 2023

# Фрагменты исходного текста программы для ЭВМ

# Файл «Form1.cs»

using Newtonsoft.Json;

using System.Text.Json;

using Newtonsoft.Json.Converters;

using JsonSerializer = Newtonsoft.Json.JsonSerializer;

namespace Externum\_ballistics

{

public partial class Form1 : Form

{

static uint N = 8;

static int n = 9;

double R = 346.9;

string path;

double[] Y0 = new double[n];

Projectile OFM29 = new Projectile();// Создадим экземпляр класса для снаряда ОФМ29

BallisticSolver solver = new BallisticSolver();

ExternumParametrs parametrs = new ExternumParametrs();

InletParametrs inletParametrs = new InletParametrs();

Jetparametrs jetparametrs = new Jetparametrs();

Externum\_ballistics CalcExternumBall = new Externum\_ballistics(8);

Inlet\_ballistics CalcInletBall = new Inlet\_ballistics(4);

Optimization optimizer = new Optimization();

public Form1()

{

InitializeComponent();

}

#region Сохранение и загрузка данных

private void jsonToolStripMenuItem\_Click(object sender, EventArgs e)//Сохранить данные снаряда в JSON

{

JsonSerializer serializer = new JsonSerializer();

SaveFileDialog openDialog = new SaveFileDialog();

openDialog.Filter = "Файл данных|\*.json";

if (openDialog.ShowDialog() == DialogResult.Cancel) return;

using (StreamWriter sw = new StreamWriter(openDialog.FileName))

using (JsonWriter writer = new JsonTextWriter(sw))

serializer.Serialize(writer, OFM29);

}

private void jsonToolStripMenuItem1\_Click(object sender, EventArgs e)//Загрузить данные снаряда в JSON

{

OpenFileDialog openDialog = new OpenFileDialog();

openDialog.Filter = "Файл данных|\*.json";

if (openDialog.ShowDialog() == DialogResult.Cancel) return;

path = openDialog.FileName;

string text = File.ReadAllText(path);

OFM29 = JsonConvert.DeserializeObject<Projectile>(text);

propertyGrid1.SelectedObject = OFM29;

}

private void xMLToolStripMenuItem\_Click(object sender, EventArgs e)//Загрузить данные снаряда в XML

{

OpenFileDialog openDialog = new OpenFileDialog();

openDialog.Filter = "Файл данных|\*.snl";

openxml snaryad = new openxml();

if (openDialog.ShowDialog() == DialogResult.Cancel) return;

// OFM29 = snaryad.load(openDialog.FileName, "");

propertyGrid1.SelectedObject = OFM29;

}

#endregion

private void начатьToolStripMenuItem\_Click(object sender, EventArgs e)//Начать вычисления

{

N = 8;

n = 9;

List<double[]> result = new List<double[]>();

result = CalcExternumBall.CalcExternum(N, parametrs, n);

double [] datax = new double[result.Count];

double[] datat = new double[result.Count];

double [] datay = new double[result.Count];

double[] dataV = new double[result.Count];

double[] dataOmega = new double[result.Count];

double[] dataSigma = new double[result.Count];

double[] koef1 = new double[result.Count];

double[] koef2 = new double[result.Count];

List<double> dataxJet = new List<double>();

List<double> datayJet = new List<double>();

List<double> dataJetV = new List<double>();

List<double> dataJett = new List<double>();

List<double> dataOmegaJet = new List<double>();

for (int i = 0; i < result.Count; i++)

{

dataGridView1.Rows.Add();

datat[i] = result[i][0];

datax[i] = result[i][1];

datay[i] = result[i][2];

dataV[i] = result[i][4];

dataOmega[i] = result[i][7];

dataSigma[i] = result[i][8];

if (result[i][0] >= parametrs.t\_start && result[i][0] <= parametrs.t\_start + parametrs.t\_delta )

{

dataxJet.Add(result[i][1]);

datayJet.Add(result[i][2]);

dataJetV.Add(result[i][4]);

dataJett.Add(result[i][0]);

dataOmegaJet.Add(result[i][7]);

}

for (int j = 0; j < N-1; j++)

{

dataGridView1.Rows[i].Cells[j].Value = result[i][j];

}

}

double[] dataY2 = new double[datayJet.Count];

double[] dataX2 = new double[dataxJet.Count];

double[] dataV2 = new double[dataxJet.Count];

double[] datat2 = new double[dataxJet.Count];

double[] dataOmegaJet2 = new double[dataxJet.Count];

for (int k = 0; k < dataxJet.Count; k++)

{

dataX2[k] = dataxJet[k];

dataY2[k] = datayJet[k];

dataV2[k] = dataJetV[k];

datat2[k] = dataJett[k];

datat2[k] = dataJett[k];

dataOmegaJet2[k] = dataOmegaJet[k];

}

for (int i = 0; i < koef1.Length; i++)

{

koef1[i] = 0.9;

koef2[i] = 0.6;

}

formsPlot1.Plot.AddScatter(datax, datay,markerShape:ScottPlot.MarkerShape.none, lineWidth:3);

formsPlot1.Plot.XLabel("X, метров");

formsPlot1.Plot.YLabel("Y, метров");

formsPlot1.Refresh();

formsPlot1.Plot.AddScatter(dataX2, dataY2, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3);

formsPlot1.Refresh();

formsPlot2.Plot.AddScatter(datat, dataV, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3);

formsPlot2.Plot.XLabel("t, секунд");

formsPlot2.Plot.YLabel("V, м/с");

formsPlot2.Refresh();

formsPlot2.Plot.AddScatter(datat2, dataV2, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3);

formsPlot2.Refresh();

formsPlot3.Plot.AddScatter(datat, dataOmega, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3);

formsPlot3.Plot.XLabel("t, секунд");

formsPlot3.Plot.YLabel("Omega, рад/с");

formsPlot3.Refresh();

formsPlot3.Plot.AddScatter(datat2, dataOmegaJet2, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3);

formsPlot3.Refresh();

formsPlot4.Plot.AddScatter(datat, dataSigma, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3);

formsPlot4.Plot.AddHorizontalLine(0.6);

formsPlot4.Plot.AddHorizontalLine(0.9);

formsPlot4.Plot.XLabel("t, секунд");

formsPlot4.Plot.YLabel("Критерий устойчивости");

formsPlot4.Refresh();

}

private void начальныеУсловияToolStripMenuItem\_Click(object sender, EventArgs e)// Задание начальных условий

{

parametrs = parametrs.Get\_Initial\_Conditions(parametrs);

propertyGrid1.SelectedObject = parametrs;

}

private void Form1\_Load(object sender, EventArgs e)// Чтение файла снаряда при запуске

{

path = @"net6.0-windowsOFM29.json";

string text = File.ReadAllText(path);

OFM29 = JsonConvert.DeserializeObject<Projectile>(text);

propertyGrid1.SelectedObject = OFM29;

}

private void button1\_Click(object sender, EventArgs e)

{

formsPlot1.Plot.Clear();

formsPlot2.Plot.Clear();

formsPlot3.Plot.Clear();

formsPlot4.Plot.Clear();

}

private void изменитьПараметрыToolStripMenuItem\_Click(object sender, EventArgs e)

{

propertyGrid1.SelectedObject = OFM29;

}

private void внутренняяБаллистикаToolStripMenuItem\_Click(object sender, EventArgs e)

{

N = 4;

List<double[]> result = new List<double[]>();

result = CalcInletBall.CalcInlet(4, inletParametrs, 5);

for (int i = 0; i < result.Count; i++)

{

dataGridView2.Rows.Add();

for (int j = 0; j < 15; j++)

{

dataGridView2.Rows[i].Cells[j].Value = result[i][j];

}

}

double[] datat = new double[result.Count];

double[] datap = new double[result.Count];

double[] datap\_sn = new double[result.Count];

double[] datap\_kn = new double[result.Count];

double[] dataV = new double[result.Count];

double[] datax = new double[result.Count];

double[] dataz = new double[result.Count];

double[] dataPsi = new double[result.Count];

double[] dataW = new double[result.Count];

for (int i = 0; i < result.Count; i++)

{

dataGridView1.Rows.Add();

datat[i] = result[i][0];

datap[i] = result[i][6];

datap\_sn[i] = result[i][7];

datap\_kn[i] = result[i][8];

datax[i] = result[i][4];

dataV[i] = result[i][3];

dataz[i] = result[i][1];

dataPsi[i] = result[i][2];

dataW[i] = result[i][13];

}

formsPlot1.Plot.AddScatter(datat, datap, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3, label: "p");

formsPlot1.Plot.XLabel("t, секунд");

formsPlot1.Plot.YLabel("Среднее давление, МПа");

formsPlot1.Refresh();

formsPlot1.Plot.AddScatter(datat, datap\_sn, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3, label: "p\_сн");

formsPlot1.Refresh();

formsPlot1.Plot.Legend();

formsPlot1.Plot.AddScatter(datat, datap\_kn, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3, label: "p\_кн");

formsPlot1.Refresh();

formsPlot2.Plot.AddScatter(datat, datax, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3);

formsPlot2.Plot.XLabel("t, секунд");

formsPlot2.Plot.YLabel("x, м");

formsPlot2.Refresh();

formsPlot3.Plot.AddScatter(datat, dataV, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3);

formsPlot3.Plot.XLabel("t, секунд");

formsPlot3.Plot.YLabel("V, м/с");

formsPlot3.Refresh();

formsPlot4.Plot.AddScatter(datat, dataW, markerShape: ScottPlot.MarkerShape.none, lineWidth: 3);

formsPlot4.Plot.XLabel("t, секунд");

formsPlot4.Plot.YLabel("psi, доля сгоревшего пороха");

formsPlot4.Refresh();

}

private void оптимизацияВнешнебаллистическихПараметровToolStripMenuItem\_Click(object sender, EventArgs e)

{

double[] x = new double[3];

x = optimizer.Optimize();

MessageBox.Show(x[0].ToString());

MessageBox.Show(x[1].ToString());

MessageBox.Show(x[2].ToString());

}

private void начальныеПараметрыВнутреннейБаллистикиToolStripMenuItem\_Click(object sender, EventArgs e)

{

inletParametrs = inletParametrs.Get\_Initial\_Conditions(inletParametrs);

propertyGrid1.SelectedObject = inletParametrs;

}

}

}

# Файл «ExternumParametrs.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

namespace Externum\_ballistics

{

public class ExternumParametrs

{

BallisticSolver solver = new BallisticSolver();

double R = 346.9;

#region Положение в пространстве

[Category("Положение в пространстве"), DescriptionAttribute("Описание"), DisplayName("X, м")]

public double X { get; set; }

[Category("Положение в пространстве"), DescriptionAttribute("Описание"), DisplayName("Y, м")]

public double Y { get; set; }

[Category("Положение в пространстве"), DescriptionAttribute("Описание"), DisplayName("Z, м")]

public double Z { get; set; }

#endregion

#region Начальные условия

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("V, м")]

public double V { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("omega, рад/с")]

public double Omega { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("teta, градусов")]

public double teta { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("g, м/с")]

public double g { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("Скоростной напор в воздухе, кг/м^2")]

public double q { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("Угол направления, град")]

public double psi { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("Площадь Миделева сечения, м^2")]

public double Sm { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("m, кг")]

public double Mass { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("Плотность воздуха, кг/м^3")]

public double ro { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("Давление воздуха, кПа")]

public double p { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("Температура, К")]

public double T { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("Число маха")]

public double Mah { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("Скорость звука, м/с")]

public double a { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("Длина, м")]

public double Length{ get; set; }

[Category("Начальные условия"), DescriptionAttribute("Описание"), DisplayName("Диаметр, м")]

public double d { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Масса")]

public double m { get; set; }

#endregion

#region Моменты и коэффициенты

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Аксиальный момент инерции")]

public double I\_x { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Экваториальный момент инерции")]

public double I\_z { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Длина хода нарезов")]

public double Rifling\_stroke { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Закон сопротивления")]

public double cx\_law { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Коэффициент бокового отклонения")]

public double iz { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Коэффициент аксиального демпф. момента")]

public double mx\_wx { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Диапозон чисел Маха")]

public double[] Ma { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Сха")]

public double[,] Cxa { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Сх")]

public double Cx { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Сx")]

public double Cy { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("Сz")]

public double Cz { get; set; }

[Category("Различные моменты и коэффициенты"), DescriptionAttribute("Описание"), DisplayName("mz")]

public double mz { get; set; }

#endregion

#region Реактивный двигаетель

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Площадь выходного сечения сопла, м^2")]

public double Sv { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Общая сила тяги реактивного двигателя, кг/с")]

public double Psigma { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Сила тяги реактивного двигателя с учетом вращения, кг/с")]

public double P { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Момент вращения двигателя")]

public double Mpx { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Суммарный импульс тяги двигателя, м/с")]

public double It { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Скорость горения, м/с")]

public double u { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Давление в камере сгорания, Па")]

public double pk { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Расход продуктов горения через сопло, кг/с")]

public double G { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Скорость горения в выходном сечении, м/с")]

public double uv { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Скорость звука в выходном сечении, м/с")]

public double akr { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Радиус расположения ребер сопла")]

public double re { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Внешнее давление, Па")]

public double pv { get; set; }

[Category("Сопло с ребрами"), DescriptionAttribute("Описание"), DisplayName("Ускорение угловой скорости, Рад/с")]

public double delta\_omega { get; set; }

#endregion

#region Константы

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Высота ребер на внутренней поверхности сопла, м")]

public double h { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Диаметр выходного сечения сопла, м")]

public double dv { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Угол наклона ребер, град")]

public double beta { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Внутреннее давление, Па")]

public double pn { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Доля тяги на устойчивость")]

public double nu { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Время старта работы двигателя, сек")]

public double t\_start { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Время работы двигателя, сек")]

public double t\_delta { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Единичная скорость горения, м/с")]

public double u1 { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Плотность топлива, кг/м^3")]

public double pT { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Площадь горящего свода, м^2")]

public double Sg { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Температура камеры, К")]

public double Tk { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Площадь критического сечения, м^2")]

public double Skr { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Коэффициент расхода сопла")]

public double A { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Лямбда")]

public double lambda { get; set; }

[Category("Константы"), DescriptionAttribute("Описание"), DisplayName("Показатель адиабаты")]

public double k { get; set; }

#endregion

#region Устойчивость

[Category("Устойчивость"), DescriptionAttribute("Описание"), DisplayName("Критерий устойчивости")]

public double sigma { get; set; }

[Category("Устойчивость"), DescriptionAttribute("Описание"), DisplayName("Коэффициент гироскопического момента")]

public double alfa { get; set; }

[Category("Устойчивость"), DescriptionAttribute("Описание"), DisplayName("Коэффициент аэродинамического момента")]

public double beta1 { get; set; }

#endregion

[Category("Устойчивость"), DescriptionAttribute("Описание"), DisplayName("Коэффициент аэродинамического момента")]

public double IsARS { get; set; }

public ExternumParametrs Get\_Initial\_Conditions(ExternumParametrs parametrs)// Получить начальные параметры

{

parametrs.g = solver.g(0, 0);

parametrs.T = solver.T(0);

parametrs.a = solver.a(parametrs.T);

parametrs.d = 0.152;

parametrs.mx\_wx = 0.0004;

parametrs.Sm = solver.Sm(parametrs.d);

parametrs.p = solver.p(0);

parametrs.ro = solver.ro(parametrs.p, parametrs.T);

parametrs.Mah = solver.Mah(parametrs.V, parametrs.a);

parametrs.q = solver.q(parametrs.ro, parametrs.V);

parametrs.V = 960;

parametrs.X = 0;

parametrs.Y = 1;

parametrs.Z = 0;

parametrs.psi = solver.psi(parametrs.Cz, parametrs.q, parametrs.Sm, parametrs.Mass, parametrs.V, parametrs.teta);

parametrs.Cx = solver.Cx(parametrs.Mah, parametrs.t\_start, parametrs.t\_delta, 0);

parametrs.Cy = 0;

parametrs.Cz = 0;

parametrs.mz = 0.8918; //solver.mz(parametrs.Mah, parametrs.Initial\_angular\_velocity);

parametrs.I\_x = 0.1455;

parametrs.I\_z = 1.4417;

parametrs.Omega = 1560;

parametrs.m = 46;

parametrs.G = solver.G(parametrs.Skr, parametrs.pk, parametrs.A, R, parametrs.Tk);

parametrs.teta = 52;

parametrs.Length = 0.709878;

parametrs.t\_start = 22;

parametrs.t\_delta = 3;// Время работы РД

parametrs.h = 0.026;// Высота ребер

parametrs.dv = 0.04; // Выходной диаметр

parametrs.beta = 15;// Угол наклона ребер

parametrs.pn = 0.101325;// Нормальное атмосферное давление

parametrs.nu = 0.5;// Доля тяги на вращение

parametrs.u1 = 6.53 \* 1e-6f;// Единичная скорость горения

parametrs.pT = 1600;// Плотность топлива

parametrs.Sg = 0.015394;// Площадь горящего свода

parametrs.Tk = 2478;// Температура???

parametrs.Skr = 0.000115;// Площадь критического сопла

parametrs.A = 0.652;// Коэффициент расхода сопла

parametrs.lambda = 2.376;// Лямбда

parametrs.k = 1.22;// Показатель адиабаты

parametrs.Sv = Math.Round(solver.Sv(parametrs.dv), 2);// Площадь внешняя

parametrs.pk = solver.pk(parametrs.u1, parametrs.Sg, 0.98, R, parametrs.Tk, 0.98, parametrs.Skr, parametrs.nu);// Давление в камере

parametrs.u = solver.u(parametrs.u1, parametrs.pk, parametrs.nu);// Скорость горения топлива

parametrs.G = Math.Round(solver.G(parametrs.Skr, parametrs.pk, parametrs.A, R, parametrs.Tk), 2);// Массовый расход топлива в секунду

parametrs.akr = solver.akr(parametrs.k, R, parametrs.Tk);// Скорость звука в критическом срезе

parametrs.uv = solver.uv(parametrs.akr, parametrs.lambda);// Внешнее u

parametrs.re = solver.re(parametrs.dv);// радиус сопла

parametrs.pv = solver.pv(parametrs.pk, parametrs.k, parametrs.lambda);// Внешнее давление

parametrs.Psigma = Math.Round(solver.Psigma(parametrs.G, parametrs.uv, parametrs.Sv, parametrs.pv, parametrs.pn), 2);// Суммарная тяга с учетом вращения

parametrs.P = Math.Round(solver.P(parametrs.Psigma, parametrs.nu, parametrs.beta), 2);// Тяга без учета вращения

parametrs.It = Math.Round(solver.It(parametrs.Psigma, parametrs.t\_delta), 2);// Импульс двигателя

parametrs.Mpx = Math.Round(solver.Mpx(parametrs.Psigma, parametrs.nu, parametrs.re, parametrs.beta), 2);// Коэффициент тяги на вращение

parametrs.alfa = Math.Round(solver.alfa(parametrs.I\_x, parametrs.I\_z, parametrs.Omega), 2);

parametrs.beta1 = Math.Round(solver.beta1(parametrs.mz, parametrs.ro, parametrs.Sm, parametrs.Length, parametrs.I\_z, parametrs.V), 2);

parametrs.sigma = Math.Round(solver.sigma(parametrs.alfa, parametrs.beta1), 2);

parametrs.psi = 0;

parametrs.IsARS = 0;

return parametrs;

}

}

}

# Файл «InletParametrs.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ComponentModel;

namespace Externum\_ballistics

{

public class InletParametrs

{

InletBallisticSolver solver = new InletBallisticSolver();

#region Начальные условия

[Category("Начальные условия"), DescriptionAttribute("Масса пороха"), DisplayName("omega, кг")]

public double omega { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Число каналов в пороховом элементе"), DisplayName("n")]

public double n { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Внешний диаметр порохового элемента"), DisplayName("D0, м")]

public double D0 { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Диаметр канала порохового элемента"), DisplayName("d0, м")]

public double d0 { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Толщина горящего свода"), DisplayName("e1, м")]

public double e1 { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Теплоемкость пороха"), DisplayName("c\_poroh, Дж/кг")]

public double c\_poroh { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Масса воспламенителя"), DisplayName("omegaV, кг")]

public double omegaV { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Плотность пороха"), DisplayName("delta, кг/м^3")]

public double delta { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Масса снаряда"), DisplayName("m, кг")]

public double m { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("S, м^2")]

public double S { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("Lambda0")]

public double Lambda0 { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("P")]

public double P { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("Q")]

public double Q { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("beta")]

public double beta { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("lambda")]

public double lambda { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("uk")]

public double uk { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("uk")]

public double u1 { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("mu")]

public double mu { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("J1")]

public double J1 { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("J2")]

public double J2 { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("J3")]

public double J3 { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("l\_n")]

public double[] l\_n { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Единичная скорость горения пороха"), DisplayName("L\_k")]

public double L\_k { get; set; }

#endregion

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("k")]

public double k { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("z")]

public double z { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("psiP")]

public double psiP { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("psi")]

public double psi { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("teat")]

public double teta { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("p")]

public double p { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("T")]

public double T { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("p\_kn")]

public double p\_kn { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("p\_sn")]

public double p\_sn { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("W\_sn")]

public double W\_sn { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("W\_km")]

public double W\_km { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("eta")]

public double eta { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("V")]

public double V { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("sigma")]

public double sigma { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("S\_sn")]

public double S\_sn { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("S0")]

public double S0 { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("p\_f")]

public double p\_f { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("x")]

public double x { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("L0")]

public double L0 { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("S\_km")]

public double S\_km { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("d\_km")]

public double[] d\_km { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("S\_kn")]

public double S\_kn { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("d\_kn")]

public double d\_kn { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("alfa")]

public double alfa { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("f")]

public double f { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("cv")]

public double cv { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("cp")]

public double cp { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("cp")]

public double kappa\_ { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("cp")]

public double lambda\_ { get; set; }

[Category("Начальные условия"), DescriptionAttribute("Показатель адиабаты"), DisplayName("cp")]

public double L { get; set; }

public InletParametrs Get\_Initial\_Conditions(InletParametrs parametrs)// Получить начальные параметры

{

double[] d\_m = { 0.214, 0.196, 0.164, 0.155, 0.155, 0.1524 };

double[] x\_m = { 0, 0.85, 0.960, 1.015, 1.045, 1.1225 };

double[] l\_m = x\_m.Skip(1).Select((x, i) => x - x\_m[i]).ToArray();

parametrs.psi = 0;

parametrs.z = 0;

parametrs.V = 0;

parametrs.x = x\_m.Last();

parametrs.f = 900000;

parametrs.d0 = 0.0009;

parametrs.D0 = 0.0115;

parametrs.L0 = 0.019;

parametrs.L = 1.093;

parametrs.d\_km = d\_m;

parametrs.d\_kn = 0.1524;

parametrs.c\_poroh = 1298;

parametrs.l\_n = l\_m;

parametrs.S\_kn = solver.S(d\_kn);

parametrs.L\_k = x\_m.Last();

parametrs.S\_sn = solver.S(d\_km.Last());

parametrs.Lambda0 = solver.Lambda0(parametrs.D0, parametrs.d0, parametrs.L0);

parametrs.Q = solver.Q(parametrs.D0, parametrs.d0, parametrs.L0);

parametrs.e1 = 0.0009;

parametrs.alfa = 0.00095;

parametrs.cv = 1497.4;

parametrs.cp = 1838.8;

parametrs.omega = 19;

parametrs.omegaV = 0.810;

parametrs.teta = solver.teta(parametrs.cv, parametrs.cp);

parametrs.m = 46;

parametrs.delta = 1520;

parametrs.J1 = 1 / 3f;

parametrs.J2 = 1 / 2f;

parametrs.u1 = 0.775 \* 1e-9;

parametrs.p\_f = 25000000;

parametrs.eta = 0;

parametrs.J3 = 1 / 6f;

parametrs.beta = solver.beta(parametrs.e1, parametrs.L0);

parametrs.P = solver.P(parametrs.D0, parametrs.d0, parametrs.L0);

parametrs.k = solver.kappa(parametrs.P, parametrs.Q, parametrs.beta);

parametrs.lambda = solver.lambda(parametrs.P, parametrs.Q, parametrs.beta);

parametrs.mu = solver.mu(parametrs.P, parametrs.Q, parametrs.beta);

parametrs.psiP = solver.psiP(parametrs.k, parametrs.lambda, parametrs.mu);

parametrs.W\_km = solver.W\_km(parametrs.l\_n, parametrs.S\_kn, parametrs.L\_k, parametrs.d\_km);

parametrs.W\_sn = solver.W\_sn(parametrs.W\_km, parametrs.S\_sn, parametrs.x, parametrs.L\_k);

parametrs.kappa\_ = solver.kappa\_(parametrs.k, parametrs.mu);

parametrs.lambda\_ = solver.lambda\_(parametrs.k, parametrs.lambda, parametrs.mu, parametrs.kappa\_);

parametrs.sigma = solver.sigma(parametrs.lambda\_, parametrs.kappa\_, parametrs.psi, parametrs.psiP);

parametrs.p = solver.p(parametrs.W\_sn, parametrs.alfa, parametrs.psi, parametrs.omega, parametrs.omegaV, parametrs.f, parametrs.m, parametrs.J1, parametrs.teta, parametrs.V, parametrs.delta);

parametrs.T = solver.T(parametrs.W\_sn, parametrs.alfa, parametrs.psi, parametrs.omega, parametrs.omegaV, parametrs.delta, parametrs.cp, parametrs.cv, parametrs.p);

parametrs.p\_sn = solver.p\_sn(parametrs.p, parametrs.omega, parametrs.omegaV, parametrs.m, parametrs.J1, parametrs.J2, parametrs.J3, parametrs.V, parametrs.W\_sn);

parametrs.p\_kn = solver.p\_kn(parametrs.p\_sn, parametrs.omega, parametrs.omegaV, parametrs.m, parametrs.J2, parametrs.V, parametrs.W\_sn);

parametrs.S0 = solver.S0(parametrs.d0, parametrs.D0);

parametrs.uk = solver.uk(parametrs.u1, parametrs.p, parametrs.p\_f);

return parametrs;

}

}

}

# Файл «BallisticSolver.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Externum\_ballistics

{

public class BallisticSolver

{

double a0 = 340.7;// Начальная скорость звука

double T0 = 288.9;// Начальная температура

double A1 = 0.6523864;// Коэффициент для формулы Бори

#region Дифференциальные уравнения

public double X(double V, double teta, double psi)// Дальность в плоскости стрельбы

{

return V \* Math.Cos(teta \* Math.PI / 180) \* Math.Cos(psi \* Math.PI / 180);

}

public double Y(double V, double teta)// Высота полёта снаряда

{

return V \* Math.Sin(teta \* Math.PI / 180);

}

public double Z(double V, double teta, double psi)// Боковое отклонение

{

return -V \* Math.Cos(teta \* Math.PI / 180) \* Math.Sin(psi \* Math.PI / 180);

}

public double V(double g, double teta, double Cx, double q, double Sm, double m, double P)// Скорость центра масс снаряда

{

return -g \* Math.Sin(teta \* Math.PI / 180) + (P - Cx \* q \* Sm) / m;

}

public double teta(double g, double teta, double V, double Cy, double q, double Sm, double m)// Угол наклона траектории

{

return (180 / Math.PI) \* -(g \* Math.Cos(teta \* Math.PI / 180) / V) - (Cy \* q \* Sm) / (m \* V);

}

public double psi(double Cz, double q, double Sm, double m, double V, double teta)// Угол направления

{

return -(0 \* q \* Sm) / (m \* V \* Math.Cos(teta \* Math.PI / 180));

}

public double omega(double mx, double q, double Sm, double l, double Ix, double Mpx)// Аксиальная угловая скорость

{

return -mx \* q \* Sm \* l / Ix + Mpx/Ix;

}

#endregion

#region Реактивный двигатель

public double Sv(double dv)// Площадь выходного сечения сопла

{

return Math.PI\*dv\*dv/4;

}

public double Psigma(double G, double uv, double Sv, double pv, double pn)// Общая реактивная тяга двигателя

{

double P = 0;

P = G \* uv + Sv \* (pv - pn);

return P;

}

public double P(double Psigma, double nu, double beta)// Тяга с учетом вращения

{

return Psigma\*((1-nu)+ nu\*Math.Cos(beta\*Math.PI/180));

}

public double Mpx(double Psigma, double nu, double re, double beta)// Момент вращения

{

return Psigma\*nu\*re\*Math.Sin(beta\*Math.PI/180);

}

public double re(double dv)// Радиус расположения ребер сопла

{

return dv/2;

}

public double It(double P, double t)// Суммарный импульс

{

return P\*t;

}

public double u(double u1, double pk, double nu)// Скорость горения

{

double u = 0;

u = u1 \* pk \* nu;

return Math.Round(u,2);

}

public double pk(double u1, double Sg, double Hi, double R, double Tk, double fc, double Skr, double v)// Давление в камере сгорания (Формула Бори)

{

double pk = 0;

pk = Math.Pow((1600 \* u1 \* Sg \* Math.Sqrt(0.98 \* R \* Tk)) / (0.98 \* Skr \* A1), 1 / (1 - v));

return Math.Round(pk,2);

}

public double G(double Skr, double pk, double A, double R, double Tk)// Расход продуктов горения через сопло

{

return (Skr \* pk \* A) / (Math.Sqrt(R \* Tk));

}

public double A(double k)// Коэффициент для формулы Бори

{

return Math.Sqrt(k\*Math.Pow(2/(k+1),(k+1)/(k-1)));

}

public double beta1(double mz, double ro, double Sm, double l, double Iy, double V)// коэффициент аэродинамического момента

{

return (mz \* ro \* V \* V / 2 \* Sm \* l) / Iy;

}

public double sigma(double alfa, double beta1)// Критерий устойчивости

{

return (1 - beta1 / (alfa\*alfa));

}

public double alfa(double Ix, double Iy, double omega)// Коэффициент гироскопического момента

{

return Ix / (2 \* Iy) \* omega;

}

public double uv (double akr, double lambda)// Скорость газов в выходном сечении

{

return Math.Round(akr \* lambda,2);

}

public double akr (double k, double R, double T)// Скорость звука в критическом сечении

{

return Math.Round(Math.Sqrt(2 \* k / (k + 1) \* R \* T),2);

}

public double pv (double pk, double k, double lambda)

{

double pv = 0;

pv = pk \* Math.Pow((1 - (k - 1) / (k + 1) \* lambda \* lambda), (1 / (k - 1)));

return Math.Round(pv,2);

}

public double m(double G, double t\_start, double t)

{

return -G ;

}

#endregion

#region Линейные уравнения характеристик снаряда

public double Mah (double V, double a)// Число Маха

{

return Math.Round(V / a,2);

}

public double Cx (double M, double t\_delta, double t\_start, double t)

{

double [] a = new double[4];

double Res = 0;

if(M > 0 && M <= 0.8)

{

a[0] = 0.1860;

a[1] = 0;

a[2] = 0;

a[3] = 0;

}

else if(M > 0.8 && M <= 1)

{

a[0] = -0.7794;

a[1] = 4.7477;

a[2] = -7.6523;

a[3] = 4.038;

}

else if (M > 1 && M <= 1.2)

{

a[0] = -17.441;

a[1] = 44.811;

a[2] = -37.284;

a[3] = 10.298;

}

else if (M > 1.2)

{

a[0] = 0.7088;

a[1] = -0.2797;

a[2] = 0.0512;

a[3] = -0.0035;

}

Res = a[0] + a[1] \* M + a[2] \* Math.Pow(M, 2) + a[3] \* Math.Pow(M, 3);

return Math.Round(Res,2);

}

public double q(double ro, double V)// Скоростной напор в воздухе

{

return Math.Round(ro\*V\*V/2,2);

}

public double ro(double p, double T)// Плотность воздуха

{

double M = 29;

double R = 8.31;

return Math.Round((p\*M)/(R\*T),2);

}

public double a(double T)// Скорость звука

{

return Math.Round(a0 \* Math.Sqrt(T / T0),2);

}

public double T(double height)// Температура на высоте h

{

return Math.Round(5e-8 \* height \* height - 0.00682858 \* height + 288.72637363,2);

}

public double p(double height)// Давление на высоте h

{

return Math.Round((-1e-8 \* height \* height \* height + 0.00055417 \* height \* height - 11.96119603 \* height + 101310.54945055) / 10e+2, 2);

}

public double g(double phi, double h)

{

return Math.Round(9.780318 \* (1 + 0.005302 \* Math.Sin(phi \* Math.PI / 180) - 0.000006 \* Math.Sin(2 \* phi \* Math.PI / 180) \* Math.Sin(2 \* phi \* Math.PI / 180)) - 0.000003086 \* h,2);

}

public double Sm(double d)// Площадь миделева сечения снаряда

{

return Math.Round(Math.PI \* d \* d / 4,2);

}

public double mz(double M, double omega)

{

double[] b = new double[3];

b[0] = 0.000617;

b[1] = -0.00022;

b[2] = 2.92e-5;

return (b[0] + b[1]\*M + b[2] \* M \* M) \* omega;

}

#endregion

}

}

# Файл «InletBallisticSolver.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Externum\_ballistics

{

public class InletBallisticSolver

{

#region Дифференциальные уравнения

public double psi(double z, double psiP, double psi, double uk, double e1, double sigma, double k, double S0, double Lambda0, double p) // Относительная доля сгоревшего пороха

{

double res = 0;

if (z <= 1 || psi <= psiP)// До фазы распада пороховых элементов

{

res = (k / e1) \* sigma \* uk;

return res;

}

else // После фазы распада пороховых элементов

{

res = S0 / Lambda0 \* sigma \* uk;

return res;

}

}

public double z(double uk, double e1, double p)// Относительная толщина горящего свода

{

double res = (uk / e1);

return res;

}

public double V(double m, double p\_sn, double S, double eta, double p\_f)// Уравнение дульной скорости снаряда

{

if (eta == 0)

{

return 0;

}

return (p\_sn \* S \* eta)/m;

}

public double x (double V, double eta, double p\_sn, double p\_f)// Уравнение движения снаряда

{

return V;

}

#endregion

#region Константы

public double S(double d)// Площадь сечений

{

return Math.PI \* (d \* d) / 4;

}

public double S0(double d, double D)// Площадь сечений

{

return S(D) - 7 \* S(d);

}

public double Lambda0(double D0, double d0, double L0)// Площадь сечений

{

return Math.PI / 4 \* (D0 \* D0 - 7 \* d0 \* d0) \* L0;

}

public double P(double D0, double d0, double L0)// Площадь сечений

{

return (D0 + 7 \* d0) / L0;

}

public double Q(double D0, double d0, double L0)// Площадь сечений

{

return (D0 \* D0 - 7 \* d0 \* d0) / (L0 \* L0);

}

public double beta(double e1, double L0)// Площадь сечений

{

return (2 \* e1) / L0;

}

public double kappa(double P, double Q, double beta)// Площадь сечений

{

return (Q + 2 \* P) / Q \* beta;

}

public double lambda(double P, double Q, double beta)// Площадь сечений

{

return (2 \* (3 - P)) / (Q + 2 \* P) \* beta;

}

public double kappa\_(double kappa, double mu)// Площадь сечений

{

return kappa - 0.5 \* kappa\*mu;

}

public double lambda\_(double kappa, double lambda, double mu, double kappa\_)// Площадь сечений

{

return (kappa\*lambda+1.5\*kappa\*mu)/kappa\_;

}

public double mu(double P, double Q, double beta)// Площадь сечений

{

return (-6 \* beta \* beta) / (Q + 2 \* P);

}

public double uk(double u1, double p, double p\_f)// Площадь сечений

{

double u1\_3 = 0;

double u2\_3 = 0;

u2\_3 = 2 \* u1 \* p\_f / (Math.Pow(2 \* p\_f, 2 / 3));

u1\_3 = u2\_3 \* Math.Pow(p\_f, 2 / 3) / (Math.Pow(p\_f, 1 / 3));

if (p <= p\_f)

{

return u1\_3 \* Math.Pow(p, 1 / 3);

}

else

if (p < p\_f && p < 2 \* p\_f)

{

return u2\_3 \* Math.Pow(p, 2 / 3);

}

else

{

return u1\* p;

}

}

public double J1()

{

return 1 / 3;

}

public double J2()

{

return 1 / 2;

}

public double J3()

{

return 1 / 6;

}

#endregion

#region Линейные уравнения

public double psiP(double k, double lambda, double mu)// Относительная доля сгоревшего пороха P

{

return k \* (1 + lambda + mu);

}

public double sigma(double lambda\_, double kappa\_, double psi, double psiP) // Уравнение горения

{

if (psi <= psiP)// До фазы распада пороховых элементов

{

return Math.Sqrt(1+4\*lambda\_/kappa\_\*psi);

}

else // После фазы распада пороховых элементов

{

return Math.Sqrt(1 + 4 \* lambda\_ / kappa\_ \* psi)\*Math.Sqrt((1-psi)/(1-psiP));

}

}

public double p(double W, double alfa, double psi, double omega, double omegaV, double f, double m, double J1, double teta, double V, double delta)// Уравнение энергии (Среднее давление в стволе)

{

return ((omega \* psi + omegaV) \* f - (1 + (omega + omegaV) / m \* J1) \* teta \* m \* V\*V / 2) / (W - omega / delta \* (1 - psi) - alfa \* (omega \* psi + omegaV));

}

public double T(double W, double alfa, double psi, double omega, double omegaV, double delta, double cp, double cv, double p)// Уравнение состояния (Определение температуры)

{

return p\*(W - omega / delta \* (1 - psi) - alfa \* (omega \* psi + omegaV))/ ((omega \* psi + omegaV)\*(341.4));

}

public double p\_kn(double p\_sn, double omega, double omega\_v, double m, double J2, double V, double W)// Давление на дно канала

{

return p\_sn\*(1+(omega+omega\_v)/m\*J2)+(omega + omega\_v)\*V\*V/W\*(1/2f-J2);

}

public double p\_sn(double p, double omega, double omega\_v, double m, double J1, double J2, double J3, double V, double W)// Давление на дно снаряда

{

return (p+(omega+omega\_v)\*V\*V/W\*(1/2f\*J1+J2-J3-1/2f))/(1+(omega+omega\_v)/m\*(J2-J3));

}

public double W\_sn (double W\_km, double S\_sn, double x, double L)// Объём заснарядного пространства

{

return W\_km + S\_sn\*(x - L);

}

public double W\_km (double[] l\_n, double S\_kn, double L\_k, double [] d\_km)// Объём каморы

{

double sum = 0;

for (int i = 1; i < l\_n.Length; i++)

{

sum+= 1/3f \* Math.PI\*l\_n[i-1] \* (d\_km[i-1] \* d\_km[i - 1] + d\_km[i - 1] \* d\_km[i] + d\_km[i] \* d\_km[i])/4;

// sum += Math.PI\*S(d\_km[i-1]) \* l\_n[i-1] + 1 / 3f \* (l\_n[i-1] - l\_n[i]) \* (S(d\_km[i-1]) + Math.Sqrt(S(d\_km[i-1]) + S\_kn) + S\_kn) + S\_kn \* (L\_k - l\_n[i]);

}

return sum;

//return 0.018;

}

public double eta (double p\_sn, double p\_f)// Функция Хевисайда

{

if ((p\_sn - p\_f) < 0)

{

return 0;

}

else

{

return 1;

}

}

public double teta(double cv, double cp)

{

return cp/cv - 1;

}

#endregion

}

}

# Файл «RungeKutta.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Externum\_ballistics

{

public abstract class RungeKutta

{

public BallisticSolver solver = new BallisticSolver();

public InletBallisticSolver Inlet\_solver = new InletBallisticSolver();

public ExternumParametrs parametrs = new ExternumParametrs();

public InletParametrs inletparametrs = new InletParametrs();

/// <summary>

/// Текущее время

/// </summary>

public double t;

public bool IsThisExternumBallistic = true;

/// <summary>

/// Искомое решение Y[0] — само решение, Y[i] — i-я производная решения

/// </summary>

public double[] Y;

/// <summary>

/// Внутренние переменные

/// </summary>

double[] YY, Y1, Y2, Y3, Y4;

protected double[] FY;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="N">размерность системы</param>

public RungeKutta(uint N)

{

Init(N);

}

/// <summary>

/// Выделение памяти под рабочие массивы

/// </summary>

/// <param name="N">Размерность массивов</param>

public void Init(uint N)

{

Y = new double[N];

YY = new double[N];

Y1 = new double[N];

Y2 = new double[N];

Y3 = new double[N];

Y4 = new double[N];

FY = new double[N];

}

/// <summary>

/// Установка начальных условий

/// </summary>

/// <param name="t0">Начальное время</param>

/// <param name="Y0">Начальное условие</param>

public void SetInit(double t0, ExternumParametrs start\_parametrs)

{

parametrs = start\_parametrs;

t = t0;

Y[0] = parametrs.X;

Y[1] = parametrs.Y;

Y[2] = parametrs.Z;

Y[3] = parametrs.V;

Y[4] = parametrs.teta;

Y[5] = parametrs.psi;

Y[6] = parametrs.Omega;

}

public void SetInit(double t0, InletParametrs start\_parametrs)

{

inletparametrs = start\_parametrs;

t = t0;

Y[0] = inletparametrs.z;

Y[1] = inletparametrs.psi;

Y[2] = inletparametrs.V;

Y[3] = inletparametrs.x;

}

/// <summary>

/// Расчет правых частей системы

/// </summary>

/// <param name="t">текущее время</param>

/// <param name="Y">вектор решения</param>

/// <returns>правая часть</returns>

abstract public double[] F(double t);

public void update(bool IsThisExternum)

{

parametrs.T = solver.T(parametrs.Y);

parametrs.p = solver.p(parametrs.Y);

parametrs.ro = solver.ro(parametrs.p, parametrs.T);

parametrs.q = solver.q(parametrs.ro, parametrs.V);

parametrs.a = solver.a(parametrs.T);

parametrs.g = solver.g(0, parametrs.Y);

parametrs.Mah = solver.Mah(parametrs.V, parametrs.a);

parametrs.Cx = solver.Cx(parametrs.Mah, parametrs.t\_delta, parametrs.t\_start, t);

parametrs.alfa = solver.alfa(parametrs.I\_x, parametrs.I\_z, parametrs.Omega);

parametrs.beta1 = solver.beta1(parametrs.mz, parametrs.ro, parametrs.Sm, parametrs.Length, parametrs.I\_z, parametrs.V);

parametrs.sigma = solver.sigma(parametrs.alfa, parametrs.beta1);

}

public void update()

{

inletparametrs.sigma = Inlet\_solver.sigma(inletparametrs.lambda\_, inletparametrs.kappa\_, inletparametrs.psi, inletparametrs.psiP);

inletparametrs.p = Inlet\_solver.p(inletparametrs.W\_sn, inletparametrs.alfa, inletparametrs.psi, inletparametrs.omega, inletparametrs.omegaV, inletparametrs.f, inletparametrs.m, inletparametrs.J1, inletparametrs.teta, inletparametrs.V, inletparametrs.delta);

inletparametrs.T = Inlet\_solver.T(inletparametrs.W\_sn, inletparametrs.alfa, inletparametrs.psi, inletparametrs.omega, inletparametrs.omegaV, inletparametrs.delta, inletparametrs.cp, inletparametrs.cv, inletparametrs.p);

inletparametrs.p\_sn = Inlet\_solver.p\_sn(inletparametrs.p, inletparametrs.omega, inletparametrs.omegaV, inletparametrs.m, inletparametrs.J1, inletparametrs.J2, inletparametrs.J3, inletparametrs.V, inletparametrs.W\_sn);

inletparametrs.p\_kn = Inlet\_solver.p\_kn(inletparametrs.p\_sn, inletparametrs.omega, inletparametrs.omegaV, inletparametrs.m, inletparametrs.J2, inletparametrs.V, inletparametrs.W\_sn);

inletparametrs.eta = Inlet\_solver.eta(inletparametrs.p\_sn, inletparametrs.p\_f);

inletparametrs.uk = Inlet\_solver.uk(inletparametrs.u1, inletparametrs.p, inletparametrs.p\_f);

inletparametrs.W\_sn = Inlet\_solver.W\_sn(inletparametrs.W\_km, inletparametrs.S\_sn, inletparametrs.x, inletparametrs.L);

}

public void Initialize(double [] YY, bool IsThis)

{

parametrs.X = YY[0];

parametrs.Y = YY[1];

parametrs.Z = YY[2];

parametrs.V = YY[3];

parametrs.teta = YY[4];

parametrs.psi = YY[5];

parametrs.Omega = YY[6];

}

public void Initialize(double[] YY)

{

inletparametrs.z = YY[0];

inletparametrs.psi = YY[1];

inletparametrs.V = YY[2];

inletparametrs.x = YY[3];

}

public void NextStep(double dt, Externum\_ballistics task)

{

int i;

update(IsThisExternumBallistic);

if (dt < 0) return;

// рассчитать Y1

Y1 = F(t);

for (i = 0; i < Y.Length; i++)

YY[i] = Y[i] + Y1[i] \* (dt / 2.0);

Initialize(YY, IsThisExternumBallistic);

// рассчитать Y2

Y2 = F(t + dt / 2.0);

for (i = 0; i < Y.Length; i++)

YY[i] = Y[i] + Y2[i] \* (dt / 2.0);

Initialize(YY, IsThisExternumBallistic);

// рассчитать Y3

Y3 = F(t + dt / 2.0);

for (i = 0; i < Y.Length; i++)

YY[i] = Y[i] + Y3[i] \* dt;

Initialize(YY, IsThisExternumBallistic);

// рассчитать Y4

Y4 = F(t + dt);

// рассчитать решение на новом шаге

for (i = 0; i < Y.Length; i++)

Y[i] = Y[i] + dt / 6.0 \* (Y1[i] + 2.0 \* Y2[i] + 2.0 \* Y3[i] + Y4[i]);

// рассчитать текущее время

t = t + dt;

}

public void NextStep(double dt, Inlet\_ballistics task)

{

int i;

update();

if (dt < 0) return;

// рассчитать Y1

Y1 = F(t);

for (i = 0; i < Y.Length; i++)

YY[i] = Y[i] + Y1[i] \* (dt / 2.0);

Initialize(YY);

// рассчитать Y2

Y2 = F(t + dt / 2.0);

for (i = 0; i < Y.Length; i++)

YY[i] = Y[i] + Y2[i] \* (dt / 2.0);

Initialize(YY);

// рассчитать Y3

Y3 = F(t + dt / 2.0);

for (i = 0; i < Y.Length; i++)

YY[i] = Y[i] + Y3[i] \* dt;

Initialize(YY);

// рассчитать Y4

Y4 = F(t + dt);

// рассчитать решение на новом шаге

for (i = 0; i < Y.Length; i++)

Y[i] = Y[i] + dt / 6.0 \* (Y1[i] + 2.0 \* Y2[i] + 2.0 \* Y3[i] + Y4[i]);

// рассчитать текущее время

t = t + dt;

}

}

public class Externum\_ballistics : RungeKutta

{

public Externum\_ballistics(uint N) : base(N) { }

/// <summary>

/// Расчёт правых частей

/// </summary>

/// <param name="t"> Время</param>

/// <param name="Y"> Вектор параметров</param>

/// <returns></returns>

public override double[] F(double t)

{

double[] F = new double[8];

F[0] = solver.X(parametrs.V, parametrs.teta, parametrs.psi);

F[1] = solver.Y(parametrs.V, parametrs.teta);

F[2] = solver.Z(parametrs.V, parametrs.teta, parametrs.psi);

F[3] = solver.V(parametrs.g, parametrs.teta, parametrs.Cx, parametrs.q, parametrs.Sm, parametrs.m, parametrs.P);

F[4] = solver.teta(parametrs.g, parametrs.teta, parametrs.V, parametrs.Cy, parametrs.q, parametrs.Sm, parametrs.m);

F[5] = solver.psi(parametrs.Cz, parametrs.q, parametrs.Sm, parametrs.m, parametrs.V, parametrs.teta);

F[6] = solver.omega(parametrs.mx\_wx, parametrs.q, parametrs.Sm, parametrs.Length, parametrs.I\_x, parametrs.Mpx);

return F;

}

public List<double[]> CalcExternum(uint N, ExternumParametrs initial\_parametrs, int n)

{

List<double[]> res = new List<double[]>();

// Шаг по времени

double dt = 0.01;

// Объект метода

Externum\_ballistics task = new Externum\_ballistics(N);

// Установим начальные условия задачи

SetInit(0, initial\_parametrs);

// решаем до того как высота снаряда не станет меньше нуля

while (parametrs.Y >= 0 )

{

double[] result = new double[n];

if (parametrs.IsARS == 1)

{

if (t >= parametrs.t\_start && t <= parametrs.t\_start + parametrs.t\_delta)

{

parametrs.P = 11560 / 3;

parametrs.Mpx = solver.Mpx(parametrs.Psigma, parametrs.nu, parametrs.re, parametrs.beta);

}

}

parametrs.P = 0;

parametrs.Mpx = 0;

for (int i = 0; i <= N-1; i++)

{

result[0] = t;

result[i+1] = Y[i];

}

result[N] = parametrs.sigma;

res.Add(result);

NextStep(dt, task);

}

return res;

}

}

public class Inlet\_ballistics : RungeKutta

{

public Inlet\_ballistics(uint N) : base(N) { }

/// <summary>

/// Расчёт правых частей

/// </summary>

/// <param name="t"> Время</param>

/// <param name="Y"> Вектор параметров</param>

/// <returns></returns>

public override double[] F(double t)

{

double[] F = new double[4];

F[0] = Inlet\_solver.z(inletparametrs.uk, inletparametrs.e1, inletparametrs.p);

F[1] = Inlet\_solver.psi(inletparametrs.z, inletparametrs.psiP, inletparametrs.psi, inletparametrs.uk, inletparametrs.e1, inletparametrs.sigma, inletparametrs.k, inletparametrs.S0, inletparametrs.Lambda0, inletparametrs.p);

F[2] = Inlet\_solver.V(inletparametrs.m, inletparametrs.p\_sn, inletparametrs.S\_kn, inletparametrs.eta, inletparametrs.p\_f);

F[3] = Inlet\_solver.x(inletparametrs.V, inletparametrs.eta, inletparametrs.p\_sn, inletparametrs.p\_f);

return F;

}

public List<double[]> CalcInlet(uint N, InletParametrs start\_parametrs, int n)

{

List<double[]> res = new List<double[]>();

// Шаг по времени

double dt = 10e-7;

// Объект метода

Inlet\_ballistics task = new Inlet\_ballistics(N);

// Установим начальные условия задачи

SetInit(0, start\_parametrs);

//Y[3] <= 6322

int iter = 0;

while (Y[3] <= 7.322)

{

iter++;

if (iter%10 == 0)

{

double[] result = new double[n + 10];

result[0] = t;

result[1] = Y[0];

result[2] = Y[1];

result[3] = Y[2];

result[4] = Y[3];

result[5] = inletparametrs.sigma;

result[6] = inletparametrs.p / 1e+6;

result[7] = inletparametrs.p\_sn/1e+6;

result[8] = inletparametrs.p\_kn/1e+6;

result[9] = inletparametrs.eta;

result[10] = inletparametrs.psiP;

result[11] = inletparametrs.T;

result[12] = inletparametrs.W\_km;

result[13] = inletparametrs.W\_sn;

result[14] = inletparametrs.teta;

res.Add(result);

}

if (inletparametrs.psi > inletparametrs.psiP)

{

int i = 0;

}

NextStep(dt,task);

}

return res;

}

}

}

# Файл «Optimization.cs»

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Externum\_ballistics

{

public static class ArrayExtensions

{

public static double[] GetCopy(this double[] x)

{

var copy = new double[x.Length];

for (int i = 0; i < x.Length; i++)

{

copy[i] = x[i];

}

return copy;

}

}

public class Optimization

{

ExternumParametrs parametrs = new ExternumParametrs();

Externum\_ballistics test = new Externum\_ballistics(8);

double[] x0 = {52, 22};

int n = 9;

int j = 0;

double[] x0\_ = new double[2];

double[] x1 = new double[2];

double[] y = new double[2];

double[] delta = { 1, 0.5 };

double gamma = 0.7;

double eps = 0.001;

double[] answer = new double[3];

bool IsAccuracyReached = false;

public double[] CoordinateSearchDetectionAlgorithm(double[] x)

{

while (j < 2)

{

y = Plus(x, delta);

if (f(y) > f(x))

{

x = y.GetCopy();

}

else

{

y = Minus(x, delta);

if (f(y) > f(x))

{

x = y.GetCopy();

}

else

{

j++;

}

}

}

return x;

}

public double[] Optimize()

{

while (IsAccuracyReached == false)

{

Step1();

}

answer[0] = x1[0];

answer[1] = x1[1];

answer[2] = f(x1);

return answer;

}

public double[] Step1()

{

x0\_ = CoordinateSearchDetectionAlgorithm(x0);

if (x0\_ != x0)

{

Step3();

}

else

{

Step2();

}

return x0\_;

}

public void Step2()

{

double u = norma(delta);

if (norma(delta) < eps)

{

x1 = x0.GetCopy();

IsAccuracyReached = true;

}

else

{

delta = product(delta, gamma);

Step1();

}

}

public double[] Step3()

{

x1 = Move(x0\_, x0);

Step4();

return x1;

}

public void Step4()

{

x1 = CoordinateSearchDetectionAlgorithm(x1);

if (f(x1) > f(x0\_))

{

x0 = x0\_.GetCopy();

x0\_ = x1.GetCopy();

Step3();

}

else

{

x0 = x1.GetCopy();

Step1();

}

}

public double[] Move(double[] x0\_, double[] x0)

{

return Minus(product(x0\_, 2).ToArray(), x0);

}

public double[] Minus(double[] x, double[] delta)

{

for (int i = 0; i < x.Length; i++)

{

x[i] = x[i] - delta[i];

}

return x;

}

public double[] product(double[] delta, double gamma)

{

var y = new double[delta.Length];

for (int i = 0; i < delta.Length; i++)

{

y[i] = delta[i] \* gamma;

}

return y;

}

public double[] Plus(double[] x, double[] delta)

{

var y = new double[x.Length];

for (int i = 0; i < x.Length; i++)

{

y[i] = x[i] + delta[i];

}

return y;

}

public double norma(double[] x)

{

double sum = 0;

for (int i = 0; i < x.Length; i++)

{

sum += x[i];

}

sum = Math.Sqrt(sum);

return sum;

}

public double f(double[] x)

{

parametrs = parametrs.Get\_Initial\_Conditions(parametrs);

parametrs.teta = x[0];

parametrs.t\_start = x[1];

List<double[]> result = new List<double[]>();

result = test.CalcExternum(8, parametrs, n);

int last = result.Count - 1;

return result[last][1];

}

}

}

# Материалы аудиовизуальных отображений, порождаемых программой для ЭВМ

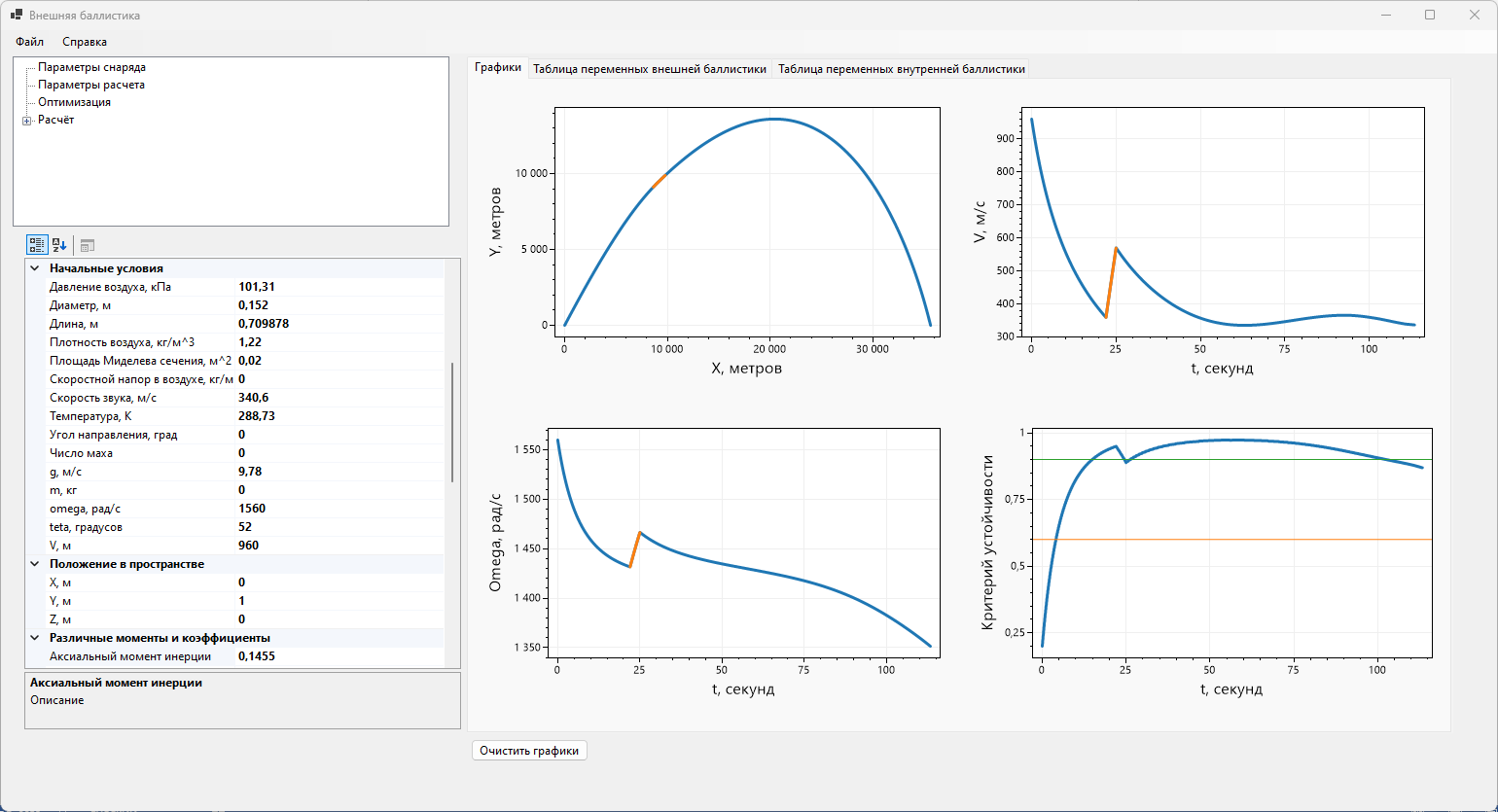


Рисунок 1 – Результаты расчёта задачи внешней баллистики

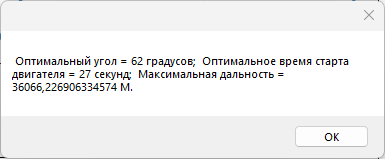


Рисунок 2 – Результаты решения задачи оптимизации.

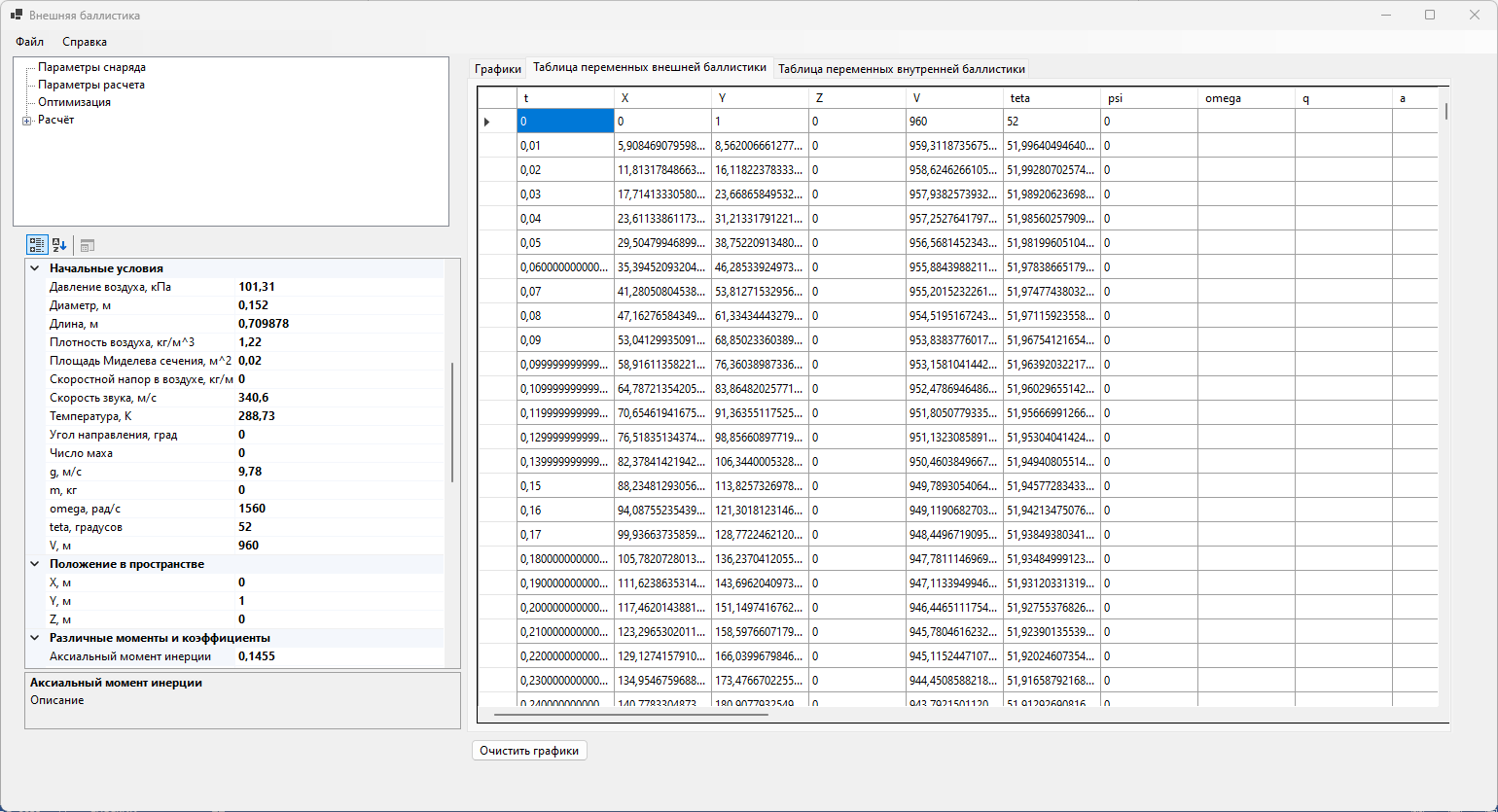


Рисунок 3 – Таблица переменных, полученных при решении задачи внешней баллистики

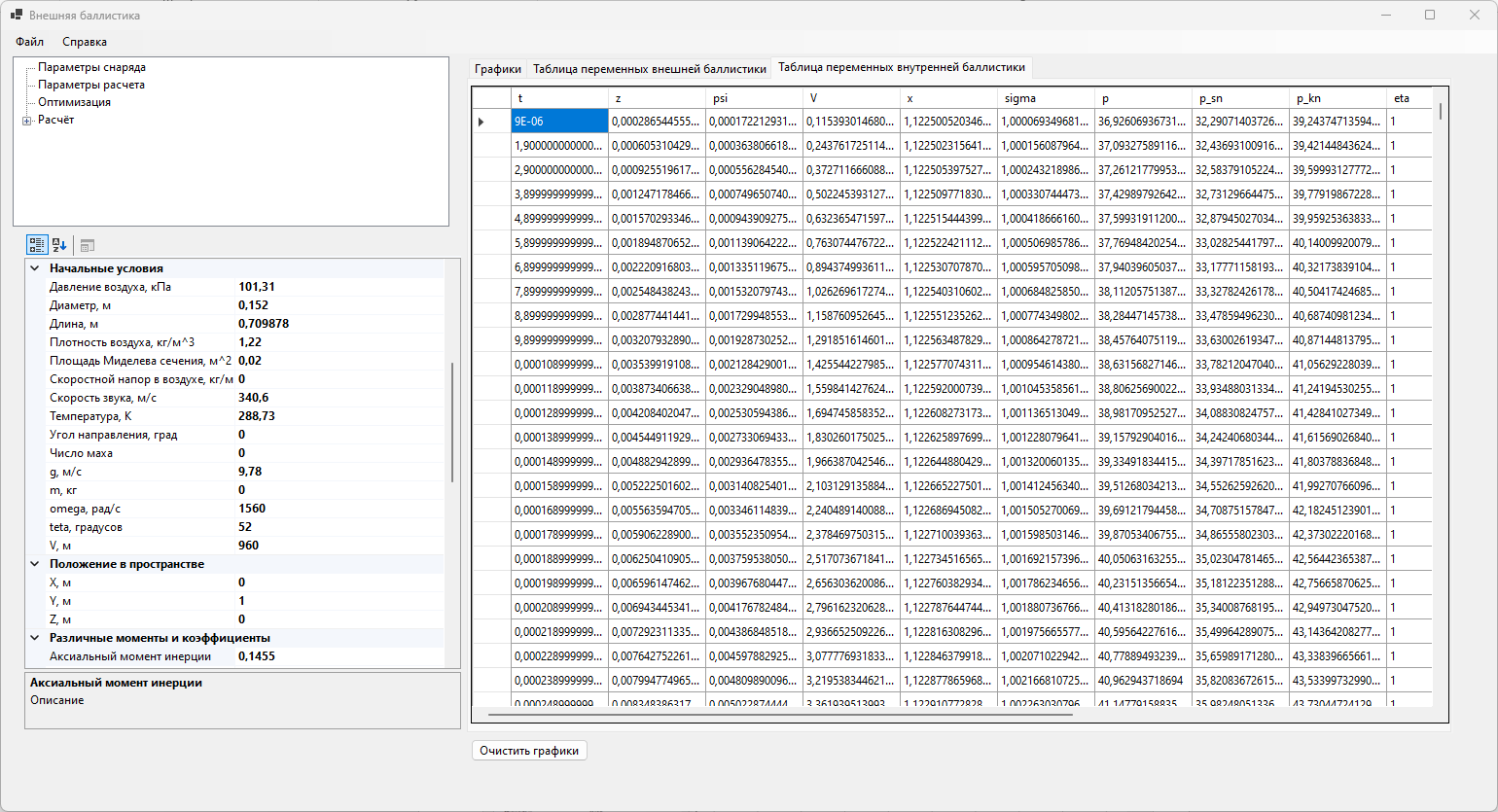


Рисунок 4 – Таблица переменных, полученных при решении задачи внутренней баллистики