**Томас Симпсон**

**(1710 — 1761) — английский математик.От природы пытливый, он много занимался самообразованием. Родился в небогатой семье ткача, и родители планировали ему ту же участь. Уже будучи подростком, он интересовался вопросами астрономии, арифметикой и элементами алгебры.Начиная с 1737 года выходят его сочинения, посвященные вопросам астрономии, математики. Симпсону принадлежат труды по теории вероятности, тригонометрии, он писал учебники по элементарной математике, элементарной геометрии, в которую привнес некоторые новшества, значительные для того времени. Томас Симпсон занимался дифференциальным исчислением, бесконечно малыми величинами, математическим анализом, что находило отражение в его сочинениях.**

**Иога́нн Карл Фри́дрих Га́усс**  ([нем.](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BC%D0%B5%D1%86%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Johann Carl Friedrich Gauß*; [30 апреля](https://ru.wikipedia.org/wiki/30_%D0%B0%D0%BF%D1%80%D0%B5%D0%BB%D1%8F) [1777](https://ru.wikipedia.org/wiki/1777_%D0%B3%D0%BE%D0%B4), [Брауншвейг](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%BD%D1%88%D0%B2%D0%B5%D0%B9%D0%B3) — [23 февраля](https://ru.wikipedia.org/wiki/23_%D1%84%D0%B5%D0%B2%D1%80%D0%B0%D0%BB%D1%8F) [1855](https://ru.wikipedia.org/wiki/1855_%D0%B3%D0%BE%D0%B4), [Гёттинген](https://ru.wikipedia.org/wiki/%D0%93%D1%91%D1%82%D1%82%D0%B8%D0%BD%D0%B3%D0%B5%D0%BD)) — [немецкий](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D1%80%D0%BC%D0%B0%D0%BD%D0%B8%D1%8F) [математик](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), [механик](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%85%D0%B0%D0%BD%D0%B8%D0%BA%D0%B0), [физик](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B7%D0%B8%D0%BA), [астроном](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%82%D1%80%D0%BE%D0%BD%D0%BE%D0%BC) и [геодезист](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BE%D0%B4%D0%B5%D0%B7%D0%B8%D1%81%D1%82). Считается одним из величайших математиков всех времён, «королём математиков». Лауреат [медали Копли](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B4%D0%B0%D0%BB%D1%8C_%D0%9A%D0%BE%D0%BF%D0%BB%D0%B8) (1838), иностранный член [Шведской](https://ru.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B5%D0%B4%D1%81%D0%BA%D0%B0%D1%8F_%D0%BA%D0%BE%D1%80%D0%BE%D0%BB%D0%B5%D0%B2%D1%81%D0%BA%D0%B0%D1%8F_%D0%B0%D0%BA%D0%B0%D0%B4%D0%B5%D0%BC%D0%B8%D1%8F_%D0%BD%D0%B0%D1%83%D0%BA)(1821) и [Российской](https://ru.wikipedia.org/wiki/%D0%A0%D0%BE%D1%81%D1%81%D0%B8%D0%B9%D1%81%D0%BA%D0%B0%D1%8F_%D0%B0%D0%BA%D0%B0%D0%B4%D0%B5%D0%BC%D0%B8%D1%8F_%D0%BD%D0%B0%D1%83%D0%BA) (1824) Академий наук, английского [Королевского общества](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%BD%D0%B4%D0%BE%D0%BD%D1%81%D0%BA%D0%BE%D0%B5_%D0%BA%D0%BE%D1%80%D0%BE%D0%BB%D0%B5%D0%B2%D1%81%D0%BA%D0%BE%D0%B5_%D0%BE%D0%B1%D1%89%D0%B5%D1%81%D1%82%D0%B2%D0%BE).

Карл Рунге   
1856 — 1927   
немецкий математик, физик и спектроскопист   
  
разработал методы численного интегрирования систем обыкновенных дифференциальных уравнений — методы Рунге — Кутты   
Исследовал поведение полиномиальной интерполяции при повышении степени полиномов — Феномен Рунге   
  
Мартин Кутта   
1867 — 1944   
немецкий математик   
  
разработал методы численного интегрирования систем обыкновенных дифференциальных уравнений — методы Рунге — Кутты   
  
Леонард Эйлер   
1707-1783  
швейцарский, немецкий и российский математик и механик  
  
Благодаря Эйлеру в математику вошли общая теория рядов, фундаментальная «формула Эйлера» в теории комплексных чисел, операция сравнения по целому модулю, полная теория непрерывных дробей, аналитический фундамент механики, многочисленные приёмы интегрирования и решения дифференциальных уравнений, число e, обозначение i для мнимой единицы, ряд специальных функций и многое другое  
  
Жозеф Лагранж  
1736-1813  
французский математик, астроном и механик  
  
Лагранж внёс существенный вклад во многие области математики, включая вариационное исчисление, теорию дифференциальных уравнений, решение задач на нахождение максимумов и минимумов, теорию чисел (теорема Лагранжа), алгебру и теорию вероятностей. Формула конечных приращений и несколько других теорем названы его именем.

1. Исаак Ньютон   
   1643 — 1727 года   
   английский физик, математик, механик и астроном   
     
   разработал дифференциальное и интегральное исчисление одновременно с Г. Лейбницем   
   мат. анализ: отыскания экстремумов, касательных и нормалей, вычисления радиусов и центров кривизны в декартовых и полярных координатах, отыскания точек перегиба и т. п.
2. Джон Куч Адамс  
   1819-1892  
     
   британский математик и астроном  
   занимался вопросами численного интегрирования дифференциальных уравнений, преподавал в Кембриджском университете
3. Эдуард Артур Милн  
   1896-1950  
     
   английский астрофизик и математик. занимался исчислением дифференциальных уравнений, внём вклад развитие теории переноса излучения в атмосферах звезд

**Лагранж**

public static double toInterpolate(double x)

{

double result = 0.0;

double currentPolinom;

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

{

currentPolinom = 1;

for (int k = 0; k < xArray.Length; k++)

{

if (i != k)

{

currentPolinom \*= (x - xArray[k]) / (xArray[i] - xArray[k]);

}

}

result += yArray[i] \* currentPolinom;

}

return result;

}

**Ньютон**

double f(double x) {

 return   x\*x-(cos(pi\*x));

}

double f1(double x) {

    return   2\*x+(1/x);

}

double f2(double x) {

    return   2+(-1/(x\*x));

}

int main() {

    int n=0;

    double a,b,c,eps;

    cout<<"a="; cin>>a;

    cout<<"b="; cin>>b;

    cout<<"eps="; cin>>eps;

    if(f(a)\*f2(a)>0) c=a;

    else c=b;

    do {

        c=c-f(c)/f1(c);

        n+=1;

    }

    while (fabs(f(c))>=eps);

        cout<<"c="<<c<<"\n";

        cout<<"n="<<n<<"\n";

        getch();

    return 0;

}

**Сплайн**

public interface IFunctionContent

{

double[] X { get; set; } // табличные значения

double[] Y { get; set; } // табличные значения

double[] CoefficientA { get; set; } // нулевой коэффициенты многочлена 3-й степени

double[] CoefficientB { get; set; } // единичный коэффициенты многочлена 3-й степени

double[] CoefficientC { get; set; } // квадратичный коэффициенты многочлена 3-й степени

double[] CoefficientD { get; set; } // кубический коэффициенты многочлена 3-й степени

}

public static class Interpolation

{

static public IFunctionContent FindCoefficients(this IFunctionContent content)

{

int countPoint = content.X.Length;

double[] stepX;

double[] stepYdivStepX;

double[] delta; // прогоночный коэффициент

double[] lambda; // прогоночный коэффициент

stepX = new double[countPoint - 1];

stepYdivStepX = new double[countPoint - 1];

delta = new double[countPoint - 1];

lambda = new double[countPoint - 1];

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

{

if (content.X[i] > content.X[i + 1])

throw new FunctionException("Значения x д.б. в порядке возрастания");

}

for (int i = 1; i < countPoint; i++)

{

stepX[i - 1] = content.X[i] - content.X[i - 1];

if (stepX[i - 1] == 0)

throw new FunctionException("Значения x не должны совпадать");

stepYdivStepX[i - 1] = (content.Y[i] - content.Y[i - 1]) / stepX[i - 1];

}

delta[0] = -stepX[1] / (2 \* (stepX[0] + stepX[1]));

lambda[0] = 1.5 \* (stepYdivStepX[1] - stepYdivStepX[0]) / (stepX[0] + stepX[1]);

for (int i = 2; i < countPoint - 1; i++)

{

delta[i - 1] = -stepX[i] /

(2 \* stepX[i - 1] + 2 \* stepX[i] + stepX[i - 1] \* delta[i - 2]);

lambda[i - 1] = (3 \* stepYdivStepX[i] - 3 \* stepYdivStepX[i - 1] - stepX[i - 1] \* lambda[i - 2]) /

(2 \* stepX[i - 1] + 2 \* stepX[i] + stepX[i - 1] \* delta[i - 2]);

}

content.CoefficientC[countPoint - 2] = 0;

for (int i = countPoint - 2; i > 0; i--)

{

content.CoefficientC[i - 1] = delta[i - 1] \* content.CoefficientC[i] + lambda[i - 1];

}

content.CoefficientD[0] = (content.CoefficientC[0]) / (3 \* stepX[0]);

content.CoefficientB[0] = stepYdivStepX[0] + (2 \* content.CoefficientC[0] \* stepX[0]) / 3;

for (int i = 1; i < countPoint - 1; i++)

{

content.CoefficientD[i] = (content.CoefficientC[i] - content.CoefficientC[i - 1]) / (3 \* stepX[i]);

content.CoefficientB[i] = stepYdivStepX[i] +

(2 \* content.CoefficientC[i] \* stepX[i] + stepX[i] \* content.CoefficientC[i - 1]) / 3;

}

return content;

}

}

**Наименьшие квадраты**

**Эйлер**

public double tryEulerMethod(int segments) //Метод Эйлера

{

double step;

xArray = new double[segments];

yArray = new double[segments];

step = (endX - startX) / segments;

xArray[0] = startX;

yArray[0] = startY;

xArray[segments - 1] = endX;

for (int i = 1; i < segments - 1; i++)

{

xArray[i] = xArray[i - 1] + step;

}

for (int i = 1; i < segments; i++)

{

yArray[i] = yArray[i - 1] + step \* formFunction(xArray[i - 1], yArray[i - 1]);

}

return yArray[segments - 1];

}

**Усовершенствованный Эйлер**

public double tryEulerImprovedMethod(int segments) //Метод Эйлера усовершенствованный

{

double step, firstArgument, secondArgument;

xArray = new double[segments];

yArray = new double[segments];

step = (endX - startX) / segments;

xArray[0] = startX;

yArray[0] = startY;

xArray[segments - 1] = endX;

for (int i = 1; i < segments - 1; i++)

{

xArray[i] = xArray[i - 1] + step;

}

for (int i = 1; i < segments; i++)

{

firstArgument = xArray[i - 1] + step / 2;

secondArgument = yArray[i - 1] + (step / 2) \* formFunction(xArray[i - 1], yArray[i - 1]);

yArray[i] = yArray[i - 1] + step \* formFunction(firstArgument, secondArgument);

}

return yArray[segments - 1];

}

**Рунге-Кутт**

public double tryRungeKuteMethod(int segments)

{

double step, kOne, kTwo, kThree, kFour;

xArray = new double[segments];

yArray = new double[segments];

step = (endX - startX) / segments;

xArray[0] = startX;

yArray[0] = startY;

xArray[segments - 1] = endX;

for (int i = 1; i < segments - 1; i++)

{

xArray[i] = xArray[i - 1] + step;

}

for (int i = 1; i < segments; i++)

{

kOne = formFunction(xArray[i - 1], yArray[i - 1]);

kTwo = formFunction((xArray[i - 1] + step/2),

(yArray[i - 1] + kOne\*step/2));

kThree = formFunction((xArray[i - 1] + step / 2),

(yArray[i - 1] + kTwo \* step / 2));

kFour = formFunction((xArray[i - 1] + step),

(yArray[i - 1] + kThree \* step));

yArray[i] = yArray[i - 1] + step/6 \*(kOne+2\*kTwo+2\*kThree+kFour);

}

return yArray[segments - 1];

}

**Адамс**

private double function (double x, double y)

{

if (Form1.function== "y'=2\*x")

{

return 2\*x;

}

else

{

return 0.5\*x\*x+y;

}

}

private void AdamsMethod(double[] y1, double step,int n)

{

double[] x = new double[n+1];

x[0] = Form1.StartX;

double[] y = new double[n+1];

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

{

x[i] = x[i - 1] + step;

}

for (int i=0; i<4; i++)

{

y[i] = y1[i];

}

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

{

y[i + 1] = y[i] + step\*((55 / 24) \* function(x[i], y[i]) - (59 / 24) \* function(x[i - 1], y[i - 1]) + (37 / 24) \* function(x[i - 2], y[i - 2]) - (9 / 24) \* function(x[i - 3], y[i - 3]));

}

chart1.Series[0].Points.DataBindXY(x, y);

}

private void step(double accurate, double startX, double endX)

{

int n = 5;

double step = (Math.Abs(endX - startX)) / n;

double[] x = new double[4];

double[] y = new double[4];

y[0] = Form1.StartY;

x[0] = Form1.StartX;

double[] difference = new double[2];

difference[0] = 0;

difference[1] = double.MaxValue;

while (Math.Abs(difference[1] -difference[0])/15 > accurate)

{

for (int j = 0; j < 2; j++)

{

for (int i = 1; i < 4; i++)

{

x[i] = x[i - 1] + step;

double k1, k2, k3, k4;

k1 = function(x[i - 1], y[i - 1]);

k2 = function(x[i - 1] + step / 2, y[i - 1] + step \* k1 / 2);

k3 = function(x[i - 1] + step / 2, y[i - 1] + step \* k2 / 2);

k4 = function(x[i - 1] + step / 2, y[i - 1] + step \* k3);

y[i] = y[i - 1] + step \* (k1 + 2 \* k2 + 2 \* k3 + k4) / 6;

}

if (j == 0)

step = step / 2;

difference[j] = y[3];

}

}

n = (int)Math.Ceiling(Math.Abs(endX - startX) / step);

AdamsMethod(y, step,n);

}

**Предиктор и Корректор(Для Милна)**

Милн с заменой метода SolveMiln

public static DifferentialFunctionContent SolvePredict (this DifferentialFunctionContent content)

{

bool isNext = true;

while (isNext)

{

content.SolveRungeKutt(3);

int count = content.X.Length;

double predict;

double step = Math.Abs((content.X[0] - content.X[count - 1])) / (count - 1);

for (int i = 3; i < count - 1; i++)

{

predict = content.Y[i - 3] + (4 / 3) \* step \*

(2 \* content.function(content.X[i - 2], content.Y[i - 2]) -

content.function(content.X[i - 1], content.Y[i - 1]) +

2 \* content.function(content.X[i], content.Y[i]));

content.Y[i + 1] = content.Y[i - 1] + (step / 3) \*

(content.function(content.X[i - 1], content.Y[i - 1]) +

4 \* content.function(content.X[i], content.Y[i]) +

content.function(content.X[i + 1], predict));

if (CheckPrecision(content.Y[i], predict, content.precision, CoefficientRunge.Miln))

{

if (content.countSplits >= limit)

isNext = true;

break;

}

isNext = false;

}

}

return content;

}

**Милн**

public interface IFunctionContent

{

double[] X { get; set; } // табличные значения

double[] Y { get; set; } // табличные значения

}

public static class DifferentialOperation

{

public static DifferentialFunctionContent SolveMiln(this DifferentialFunctionContent content)

{

bool isNext = true;

while (isNext)

{

content.SolveRungeKutt(3);

int count = content.X.Length;

double previousPredict = content.Y[3];

double predict;

double step = Math.Abs((content.X[0] - content.X[count - 1])) / (count - 1);

double managerController;

isNext = false;

for (int i = 3; i < count - 1; i++)

{

predict = content.Y[i - 3] + (4 / 3) \* step \*

(2 \* content.function(content.X[i - 2], content.Y[i - 2]) -

content.function(content.X[i - 1], content.Y[i - 1]) +

2 \* content.function(content.X[i], content.Y[i]));

managerController = predict +

(28 / 29) \* (content.Y[i] - previousPredict);

content.Y[i + 1] = content.Y[i - 1] + (step / 3) \*

(content.function(content.X[i - 1], content.Y[i - 1]) +

4 \* content.function(content.X[i], content.Y[i]) +

content.function(content.X[i + 1], managerController));

if (CheckPrecision(content.Y[i], predict, content.precision, content.countSplits, CoefficientRunge.Miln))

{

isNext = true;

break;

}

}

}

return content;

}

public static DifferentialFunctionContent SolveRungeKutt(this DifferentialFunctionContent content, int countValue)

{

bool isNext = true;

double[] previousY = new double[countValue];

content.Y = new double[countValue];

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

content.Y[i] = 0;

while (isNext)

{

countSplits \*= 2;

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

previousY[i] = content.Y[i];

content.X = new double[countSplits + 1];

content.Y = new double[countSplits + 1];

content.X[0] = content.xStart;

content.Y[0] = content.yStart;

content.X[countSplits] = content.xEnd;

double step = Math.Abs((content.X[countSplits] - content.X[0])) / countSplits;

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

content.X[i + 1] = content.X[i] + step;

double coefficient1;

double coefficient2;

double coefficient3;

double coefficient4;

double deltaY;

isNext = false;

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

{

coefficient1 = content.function(content.X[i], content.Y[i]);

coefficient2 = content.function(content.X[i] + step / 2,

content.Y[i] + step \* coefficient1 / 2);

coefficient3 = content.function(content.X[i] + step / 2,

content.Y[i] + step \* coefficient2 / 2);

coefficient4 = content.function(content.X[i] + step,

content.Y[i] + step \* coefficient3);

deltaY = step / 6 \* (coefficient1 + 2 \* coefficient2 + 2 \* coefficient3 + coefficient4);

content.Y[i + 1] = content.Y[i] + deltaY;

if (CheckPrecision(content.Y[i + 1], previousY[i], content.precision, content.countSplits, CoefficientRunge.RungeKutt))

{

isNext = true;

break;

}

}

}

return content;

}

private static bool CheckPrecision(double currentValue, double previousValue, double precision, int countSplits, CoefficientRunge coefficient)

{

int order;

switch (coefficient)

{

case CoefficientRunge.RungeKutt: { order = 15; break; }

case CoefficientRunge.Miln: { order = 29; break; }

default: { return false; }

}

double discrepancy = Math.Abs(currentValue - previousValue) / order;

if (discrepancy > precision)

return true;

if (countSplits >= limit)

throw new FunctionException("Невозможно достигнуть указанной точности");

return false;

}

}

Метод Гаусса .

#include <iostream>

using namespace std;

int n, i, j, k;

double d, s;

int main()

{

cout << "Poryadok: " << endl;

cin >> n;

double \*\*a = new double \*[n];

for (i = 0; i <= n; i++)

a[i] = new double [n];

double \*\*a1 = new double \*[n];

for (i = 0; i <= n; i++)

a1[i] = new double [n];

double \*b = new double [n];

double \*x = new double [n];

cout << "Vvedite koefficienty i svobodnye chleny " << endl;

for (i = 1; i <= n; i++)

{

for (j = 1; j <= n; j++)

{

cout << "a[ " << i << "," << j << "]= ";

cin >> a[i][j];

a1[i][j] = a[i][j];

}

cout << "b,[ " << i << "]= ";

cin >> b[i];

}

for (k = 1; k <= n; k++) // прямой ход

{

for (j = k + 1; j <= n; j++)

{

d = a[j][k] / a[k][k]; // формула (1)

for (i = k; i <= n; i++)

{

a[j][i] = a[j][i] - d \* a[k][i]; // формула (2)

}

b[j] = b[j] - d \* b[k]; // формула (3)

}

}

for (k = n; k >= 1; k--) // обратный ход

{

d = 0;

for (j = k + 1; j <= n; j++)

{

s = a[k][j] \* x[j]; // формула (4)

d = d + s; // формула (4)

}

x[k] = (b[k] - d) / a[k][k]; // формула (4)

}

cout << "Korni sistemy: " << endl;

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

cout << "x[" << i << "]=" << x[i] << " " << endl;

return 0;

}