**Результат производственной практики Пасько Д. А. за период 26.06.2018.-05.07.2018.**

/// <summary>

/// Класс методов, связанных с вычислением интеграла квадратурами Гаусса-Кронрода

/// </summary>

public static class GaussKronrod

{

static GaussKronrod()

{

GK\_7\_15\_init();

MyGKInit();

List<string> r = new List<string>(); r.Add("");

MasListDinnInfo[0] = new List<string>(r);

MasListDinnInfo[1] = new List<string>(r);

MasListDinnInfo[2] = new List<string>(r);

MasListDinnInfo[3] = new List<string>(r);

}

/// <summary>

/// Процедурная реализация вектор-функции комплексного аргумента

/// </summary>

/// <param name="x">Аргумент</param>

/// <param name="y">Вектор значений</param>

/// <param name="N">Размерность вектора значений</param>

public delegate void Myfunc(Complex x, ref Complex[] y, int N);

/// <summary>

/// Комплексная вектор-функция

/// </summary>

/// <param name="x"></param>

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

/// <returns></returns>

public delegate Complex[] ComplexVectorFunc(Complex x, int N);

/// <summary>

/// Размерность

/// </summary>

static int Nodes = 15;

static bool Key = false;

static double RV\_eps\_step\_increment, norm\_param = 1;

static double[] GK\_nodes, K\_weights, G\_weights;

static double h = 0.1;

static double eps = 0.001;

static void GK\_7\_15\_init()

{

GK\_nodes = new double[Nodes + 1];

K\_weights = new double[Nodes + 1];

G\_weights = new double[Nodes + 1];

GK\_nodes[1] = -0.991455371120813; GK\_nodes[2] = -0.949107912342759;

GK\_nodes[3] = -0.864864423359769; GK\_nodes[4] = -0.741531185599394;

GK\_nodes[5] = -0.586087235467691; GK\_nodes[6] = -0.405845151377397;

GK\_nodes[7] = -0.207784955007898;

GK\_nodes[8] = 0;

GK\_nodes[9] = 0.207784955007898; GK\_nodes[10] = 0.405845151377397;

GK\_nodes[11] = 0.586087235467691; GK\_nodes[12] = 0.741531185599394;

GK\_nodes[13] = 0.864864423359769; GK\_nodes[14] = 0.949107912342759;

GK\_nodes[15] = 0.991455371120813;

K\_weights[1] = 0.022935322010529; K\_weights[2] = 0.063092092629979;

K\_weights[3] = 0.104790010322250; K\_weights[4] = 0.140653259715525;

K\_weights[5] = 0.169004726639267; K\_weights[6] = 0.190350578064785;

K\_weights[7] = 0.204432940075298;

K\_weights[8] = 0.209482141084728;

K\_weights[9] = 0.204432940075298; K\_weights[10] = 0.190350578064785;

K\_weights[11] = 0.169004726639267; K\_weights[12] = 0.140653259715525;

K\_weights[13] = 0.104790010322250; K\_weights[14] = 0.063092092629979;

K\_weights[15] = 0.022935322010529;

G\_weights[1] = 0; G\_weights[2] = 0.129484966168870;

G\_weights[3] = 0; G\_weights[4] = 0.279705391489277;

G\_weights[5] = 0; G\_weights[6] = 0.381830050505119;

G\_weights[7] = 0;

G\_weights[8] = 0.417959183673469;

G\_weights[9] = 0; G\_weights[10] = 0.381830050505119;

G\_weights[11] = 0; G\_weights[12] = 0.279705391489277;

G\_weights[13] = 0; G\_weights[14] = 0.129484966168870;

G\_weights[15] = 0;

RV\_eps\_step\_increment = 1e-8;

}

/// <summary>

/// Интеграл по коченому отрезку

/// </summary>

/// <param name="int\_func">Интегрируемая функция</param>

/// <param name="a">Начало отрезна интегрирования</param>

/// <param name="b">Конец отрезка интегрирования</param>

/// <param name="int\_h">Начальный шаг</param>

/// <param name="eps">Используемая погрешность</param>

/// <param name="ret\_arr">Вектор значений интеграла (результат)</param>

/// <param name="N">Размерность вектора значений</param>

public static void GK\_adaptive\_int(Myfunc int\_func, Complex a, Complex b, double int\_h, double eps, ref Complex[] ret\_arr, int N)

{

double eps\_out = 0;

Complex t\_i\_h, t\_x;

Complex[] ret\_arr\_0 = new Complex[N];

ret\_arr = new Complex[N];

t\_x = new Complex(a);

t\_i\_h = new Complex(int\_h, 0);

if (Math.Abs(Complex.Imag(b) - Complex.Imag(a)) > eps)

{

if (Complex.Imag(b) < Complex.Imag(a))

t\_i\_h = new Complex(0, -int\_h);

else

t\_i\_h = new Complex(0, int\_h);

}

while (Math.Abs((double)(b - t\_x)) > eps)

{

if (Math.Abs((double)(t\_x + t\_i\_h)) > Math.Abs((double)b)) t\_i\_h = b - t\_x;

GK\_int(int\_func, t\_x, t\_x + t\_i\_h, ref ret\_arr\_0, ref eps\_out, N);//if(!Double.IsNaN(ret\_arr\_0[0].Abs))ret\_arr\_0.Show();

if (eps\_out > eps)

{

t\_i\_h = t\_i\_h \* 0.5;

}

else

{

try

{

ret\_arr = Complex.Sum(ret\_arr, ret\_arr\_0);

t\_x = t\_x + t\_i\_h;

//Console.WriteLine(t\_x);

if (eps\_out < 1e-3 \* eps) t\_i\_h = t\_i\_h \* 1.75;

}

catch (Exception e) { throw new Exception(e.Message); }

Vectors v = new Vectors(N);

for (int i = 0; i < v.n; i++) v[i] = Math.Abs((double)ret\_arr\_0[i]);

if ((v.Max) / norm\_param < 2.5e-5) return;

}

}

int\_h = Math.Abs((double)t\_i\_h);

}

/// <summary>

/// Интеграл по коченому отрезку

/// </summary>

/// <param name="int\_func">Интегрируемая функция</param>

/// <param name="a">Начало отрезна интегрирования</param>

/// <param name="b">Конец отрезка интегрирования</param>

/// <param name="int\_h">Начальный шаг</param>

/// <param name="eps">Используемая погрешность</param>

/// <param name="N">Размерность вектора значений</param>

public static Complex[] Integral(Myfunc int\_func, Complex a, Complex b, double int\_h, double eps, int N)

{

Complex[] y = new Complex[0];

try { GK\_adaptive\_int(int\_func, a, b, int\_h, eps, ref y, N); } catch { throw new Exception("тут"); }

return y;

}

/// <summary>

/// Интеграл по прямой

/// </summary>

/// <param name="int\_func">Интегрируемая функция</param>

/// <param name="a">Начало отрезна интегрирования</param>

/// <param name="b">Конец отрезка интегрирования</param>

/// <param name="int\_h">Начальный шаг</param>

/// <param name="eps">Используемая погрешность</param>

/// <param name="N">Размерность вектора значений</param>

public static Complex IntegralInf(ComplexFunc f, Complex a, Complex b, int tt = 100, int n = 15, ComplexFunc delta = null, int maxmult = 12, double vareps = 1e-6, double maxarg = 140)

{

double epss = vareps;

Complex sum = MySuperGaussKronrod(f, a, b, delta, tt, n);//a.Show();b.Show();int\_h.Show();eps.Show();

Complex p = new Complex(sum); if (Double.IsNaN(sum.Re)) $"Такая байда {sum} \t f({a})={f(a)} \t f({b})={f(b)} \t d({a})={delta(a)} \t d({b})={delta(b)}".Show();

Complex ab = b - a;//ab.Show();

int t = 1;

while (t <= maxmult && b.Abs < maxarg)//идти по всем множителям шага

{

//Console.WriteLine($"p={p} t={t} norm={p.Abs}");

Complex s1 = MySuperGaussKronrod(f, a - ab, a, delta, tt, n), s2 = MySuperGaussKronrod(f, b, b + ab, delta, tt, n);

p = s1 + s2;

a -= ab;

b += ab;

ab \*= 2;

sum += p; if (Double.IsNaN(sum.Re)) $"Такая байда {sum} \t f({a})={f(a)} \t f({b})={f(b)} \t d({a})={delta(a)} \t d({b})={delta(b)}".Show();

while (p.Abs > epss && b.Abs < maxarg)//пока добавление больше погрешности

{

//Console.WriteLine($"p={p} t={t} norm={p.Abs}");

s1 = MySuperGaussKronrod(f, a - ab, a, delta, tt, n); s2 = MySuperGaussKronrod(f, b, b + ab, delta, tt, n);

p = s1 + s2;

a -= ab;

b += ab;

sum += p; if (Double.IsNaN(sum.Re)) $"Такая байда {sum} \t f({a})={f(a)} \t f({b})={f(b)} \t d({a})={delta(a)} \t d({b})={delta(b)}".Show();

}

//p = sum;//t.Show();

t++;

}

return sum;

}

/// <summary>

/// Интеграл по отрезку от a до +inf

/// </summary>

/// <param name="f"></param>

/// <param name="a"></param>

/// <param name="b"></param>

/// <param name="n"></param>

/// <param name="maxmult"></param>

/// <param name="vareps"></param>

/// <returns></returns>

public static Complex IntegralHalfInf(ComplexFunc f, double a, double begH, int tt = 100, int n = 15, ComplexFunc delta = null, int maxmult = 12, double vareps = 1e-6, double h = 0.2)

{

double epss = vareps;

Complex sum = MySuperGaussKronrod(f, a, a + begH, delta, tt, n, h);//a.Show();b.Show();int\_h.Show();eps.Show();

Complex p = new Complex(sum); if (Double.IsNaN(sum.Re)) { $"1 Такая байда {sum} \t f({a})={f(a)} \t f({a + begH})={f(a + begH)} \t d({a})={delta(a)} \t d({a + begH})={delta(a + begH)}".Show(); Console.ReadKey(); }

int t = 1; a += begH;

while (t <= maxmult && a < tt)//идти по всем множителям шага

{

//Console.WriteLine($"p={p} t={t} norm={p.Abs}");

p = MySuperGaussKronrod(f, a, a + begH, delta, tt, n, h); if (Double.IsNaN(p.Re)) { $"2 Такая байда {p} \t f({a})={f(a)} \t f({a + begH})={f(a + begH)} \t d({a})={delta(a)} \t d({a + begH})={delta(a + begH)}".Show(); Console.ReadKey(); }

a += begH;

begH \*= 2;

sum += p;

while (p.Abs > epss && a < tt)//пока добавление больше погрешности

{

//Console.WriteLine($"p={p} t={t} norm={p.Abs}");

p = MySuperGaussKronrod(f, a, a + begH, delta, tt, n, h);

if (Double.IsNaN(p.Re) || Double.IsInfinity(p.Re))

{

$"3 При суммировании интегралов промужуточный интеграл равен = {MySuperGaussKronrod(f, a, a + begH, delta, tt, n, h)} \t f({a})={f(a)} \t f({a + begH})={f(a + begH)} \t d({a})={delta(a)} \t d({a + begH})={delta(a + begH)}".Show();

Console.ReadKey();

}

a += begH;

begH \*= 2;

sum += p;

}

//p = sum;//t.Show();

t++;

}

return sum;

}

/// <summary>

/// Определённый интеграл

/// </summary>

/// <param name="f">Интегрируемая функция</param>

/// <param name="a">Начало отрезка интегрирования</param>

/// <param name="b">Конец отрезка интегрирования</param>

/// <returns></returns>

public static double Integral(RealFunc f, double a, double b)

{

Myfunc z = (Complex x, ref Complex[] t, int N) => { t = new Complex[1]; t[0] = f(x.Re); };

Complex[] y = Integral(z, a, b, h, eps, 1);

return y[0].Re;

}

/// <summary>

/// Несобственный интеграл по вещественной оси

/// </summary>

/// <param name="f"></param>

/// <param name="a"></param>

/// <param name="b"></param>

/// <returns></returns>

public static double IntegralInf(ComplexFunc f, double a, double b, ComplexFunc delta = null, int t = 100, int n = 15, double vareps = 1e-6, double maxarg = 150)

{

Complex y = IntegralInf(f, new Complex(a), new Complex(b), t, n, delta, 3, vareps, maxarg);

return y.Re;

}

static void GK\_adaptive\_int\_inf(Myfunc int\_func, Complex a, Complex b, ref double int\_h, ref double eps, ref Complex[] ret\_arr, int N)

{

int i, ipr;

bool it;

double[] temp\_arr = new double[N /\*+ 1\*/];

double t1, eps10, pm, pt, t, int\_h\_1, t\_h;

Complex t\_x\_a, t\_x\_b;

Complex[] ret\_arr\_0 = new Complex[N /\*+ 1\*/];

ret\_arr = new Complex[N /\*+ 1\*/];

t\_x\_b = new Complex(a);

t = Math.Abs((double)(b - a));

eps10 = eps \* 10;

ipr = 0;

it = true;

while (it)

{

t\_x\_a = new Complex(t\_x\_b); t\_x\_b = t\_x\_b + int\_h; t1 = Math.Abs((double)(t\_x\_b - a));

if (t1 > t)

{

t\_x\_b = b;

it = false;

}

int\_h\_1 = int\_h;

GK\_adaptive\_int(int\_func, t\_x\_a, t\_x\_b, int\_h\_1, eps, ref ret\_arr\_0, N);

Vectors v = new Vectors(N);

for (int j = 0; j < v.n; j++) v[j] = Math.Abs((double)ret\_arr\_0[j /\*+ 1\*/]);

if ((v.Min < 1e-10) && (Math.Abs((double)t\_x\_a) > 50.0) && (v.Max < 1e-7)) return;

try

{

if (v.Max / norm\_param < 1e-8) return;

t\_h = Math.Abs((double)(t\_x\_b - t\_x\_a));

temp\_arr = (double[])(v / t\_h);

ret\_arr = Complex.Sum(ret\_arr, ret\_arr\_0);

if (Math.Abs(int\_h) < 10 \* Math.Abs(int\_h\_1))

{

int\_h = 4 \* int\_h;

}

else int\_h = 4 \* int\_h\_1;

// at infinity

pm = 0;

for (i = 0/\*1\*/; i </\*=\*/ N; i++)

{

if (Math.Abs((double)ret\_arr[i]) > 1e-15)

{

pt = Math.Abs((double)(temp\_arr[i] \* t1 / ret\_arr[i]));

if (pt > pm) pm = pt;

}

}

if (pm < eps10)

{

ipr = ipr + 1;

if (ipr > 4) return;

}

else

{

ipr = 0;

}

}

catch { throw new Exception(""); }

}

}

static void GK\_int(Myfunc int\_func, Complex a, Complex b, ref Complex[] ret\_arr, ref double eps\_out, int N)

{

// implicit none;

int i;

Complex[] GK\_nodes\_arb = new Complex[Nodes + 1], K\_weights\_arb = new Complex[Nodes + 1], G\_weights\_arb = new Complex[Nodes + 1];

Complex[,] temp\_arr = new Complex[N + 1, 3];

GK\_nodes\_arb = Complex.Sum(Complex.Mult(0.5 \* (b - a), GK\_nodes), 0.5 \* (b + a));

K\_weights\_arb = Complex.Mult((b - a), (double[])((Vectors)K\_weights \* 0.5));

G\_weights\_arb = Complex.Mult((b - a), (double[])((Vectors)G\_weights \* 0.5));

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

{

int\_func(GK\_nodes\_arb[i], ref ret\_arr, N);

//ret\_arr.Show();

//K\_weights\_arb.Show();

//temp\_arr[0, 0].Show();

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

{

try

{

//Console.WriteLine(j + " <= " + N);

temp\_arr[j, 1] += K\_weights\_arb[i] \* ret\_arr[j - 1];

temp\_arr[j, 2] += G\_weights\_arb[i] \* ret\_arr[j - 1];

}

catch (Exception e) { throw new Exception(e.Message); }

}

}

Vectors v = new Vectors(N);

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

v[j - 1] = Math.Abs((double)(temp\_arr[j, 1] - temp\_arr[j, 2]));

eps\_out = v.Max;

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

ret\_arr[j - 1] = new Complex(temp\_arr[j, 1]);

}

public static List<string>[] MasListDinnInfo = new List<string>[4];

/// <summary>

/// Информация о последнем найденном с помощью DINN5\_GK интеграле

/// </summary>

private static List<string> LastListOfDINN5GK;

/// <summary>

/// Базовый DINN с фортрана

/// </summary>

/// <param name="CF"></param>

/// <param name="t1"></param>

/// <param name="t2"></param>

/// <param name="t3"></param>

/// <param name="t4"></param>

/// <param name="tm"></param>

/// <param name="tp"></param>

/// <param name="eps"></param>

/// <param name="pr"></param>

/// <param name="gr"></param>

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

/// <param name="Rd"></param>

/// <remarks>

/// ! Программа вычисления N интегралов по полубесконечному контуру

///! в случае обратной волны

///!

///! subroutine СF(u, s, n) - подынтегральные функции; u - аргумент(complex16),

///! s(n) - массив значений функций в точке u(complex16),

///! n - число интегралов(integer)

///! [t1, t2],[t3, t4] - участки отклонения контура вниз(real8)

///! [t2, t3] - участок отклонения контура вверх(real8)

///! tm,tp > 0 - величины отклонения контура вниз и вверх(real8)

///! (если нет обратной волны, то следует положить t2 = t3 = t1, tp = 0

///! обход полюсов при этом будет только снизу на участке[t1, t4]

///! с отклонением на tm)

///! eps - отн.погрешность, pr - начальный шаг,

///! N- число интегралов(integer)

///! Rd(N) - выход: массив значений интегралов

/// </remarks>

static void DINN5\_GK(Myfunc CF, double t1, double t2, double t3, double t4, double tm, double tp, double eps, double pr, double gr, int N, out Complex[] Rd)

{

// implicit none

int i;

double int\_h;

Complex a, b;

Complex[] sb = new Complex[N];

GK\_7\_15\_init();

Rd = new Complex[N];

if (t1 < t4)

{

// [0, t1]

a = 0; b = t1; int\_h = 0.25 \* Math.Abs((double)(b - a));

GK\_adaptive\_int(CF, a, b, int\_h, eps, ref sb, N);

Rd = Complex.Sum(Rd, sb);//Rd.Show();

//LastListOfDINN5GK.Add($"\tНа участке [0,t1] GK\_adaptive\_int = {sb[0]}");

if (t3 - t2 < eps)

{ // no inverse poles case

a = b; b = new Complex(t1, -tm); int\_h = 0.25 \* Math.Abs((double)(b - a));

GK\_adaptive\_int(CF, a, b, int\_h, eps, ref sb, N);

Rd = Complex.Sum(Rd, sb);

// LastListOfDINN5GK.Add($"\tНа участке [t1,t1-tm\*i] GK\_adaptive\_int = {sb[0]}");

a = b; b = new Complex(t4, -tm); int\_h = 0.05 \* Math.Abs((double)(b - a));

GK\_adaptive\_int(CF, a, b, int\_h, eps, ref sb, N);

Rd = Complex.Sum(Rd, sb);

//LastListOfDINN5GK.Add($"\tНа участке [t1-tm\*i,t4-tm\*i] GK\_adaptive\_int = {sb[0]}");

a = b; b = t4; int\_h = 0.25 \* Math.Abs((double)(b - a));

GK\_adaptive\_int(CF, a, b, int\_h, eps, ref sb, N);

Rd = Complex.Sum(Rd, sb);

//LastListOfDINN5GK.Add($"\tНа участке [t4-tm\*i,t4] GK\_adaptive\_int = {sb[0]}");

}

else

{

if (t2 - t1 > eps)

{

// first deviation from below

a = b; b = new Complex(t1, -tm); int\_h = 0.5 \* Math.Abs((double)(b - a));

GK\_adaptive\_int(CF, a, b, int\_h, eps, ref sb, N);

Rd = Complex.Sum(Rd, sb);/\*Rd.Show();\*/

//LastListOfDINN5GK.Add($"\tНа участке [t1,-tm\*i] GK\_adaptive\_int = {sb[0]}");

a = b; b = new Complex(t2, -tm); int\_h = 0.5 \* Math.Abs((double)(b - a));

GK\_adaptive\_int(CF, a, b, int\_h, eps, ref sb, N);

Rd = Complex.Sum(Rd, sb);

//LastListOfDINN5GK.Add($"\tНа участке [-tm,t2-tm\*i] GK\_adaptive\_int = {sb[0]}");

}

// diviation from above

a = b; b = new Complex(t2, tp); int\_h = 0.5 \* Math.Abs((double)(b - a));

GK\_adaptive\_int(CF, a, b, int\_h, eps, ref sb, N);

Rd = Complex.Sum(Rd, sb);

//LastListOfDINN5GK.Add($"\tНа участке [t1 либо t2-tm\*i,t2+tp] GK\_adaptive\_int = {sb[0]}");

a = b; b = new Complex(t3, tp); int\_h = 0.5 \* Math.Abs((double)(b - a));

GK\_adaptive\_int(CF, a, b, int\_h, eps, ref sb, N);

Rd = Complex.Sum(Rd, sb);

// LastListOfDINN5GK.Add($"\tНа участке [t2+tp,t3+tp\*i] GK\_adaptive\_int = {sb[0]}");

a = b; b = new Complex(t3, -tm); int\_h = 0.5 \* Math.Abs((double)(b - a));

GK\_adaptive\_int(CF, a, b, int\_h, eps, ref sb, N);

Rd = Complex.Sum(Rd, sb);

//LastListOfDINN5GK.Add($"\tНа участке [t3+tp\*i,t3-tm\*i] GK\_adaptive\_int = {sb[0]}");

// second diviation from below

a = b; b = new Complex(t4, -tm); int\_h = 0.25 \* Math.Abs((double)(b - a));

GK\_adaptive\_int(CF, a, b, int\_h, eps, ref sb, N);

Rd = Complex.Sum(Rd, sb);

//LastListOfDINN5GK.Add($"\tНа участке [t3-tm\*i,t4-tm\*i] GK\_adaptive\_int = {sb[0]}");

a = b; b = t4; int\_h = 0.5 \* Math.Abs((double)(b - a));

GK\_adaptive\_int(CF, a, b, int\_h, eps, ref sb, N);

Rd = Complex.Sum(Rd, sb);

//LastListOfDINN5GK.Add($"\tНа участке [t4-tm\*i,t4] GK\_adaptive\_int = {sb[0]}");

}

Vectors v = new Vectors(N);

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

v[i] = Math.Abs((double)Rd[i]);

norm\_param = v.Max;

// Console.WriteLine(norm\_param);

}

if (gr > t4)

{

a = t4;

b = gr;

int\_h = 0.33; //0.25\*Math.Abs((double)(b-a));

if (Key)

{

GK\_adaptive\_int\_inf(CF, a, b, ref int\_h, ref eps, ref sb, N);

//LastListOfDINN5GK.Add($"\tНа участке gr > t4 GK\_adaptive\_int\_inf = {sb[0]}");

}

else

{

GK\_adaptive\_int(CF, a, b, int\_h, eps \* 10, ref sb, N);

//LastListOfDINN5GK.Add($"\tНа участке gr > t4 GK\_adaptive\_int = {sb[0]}");

}

Rd = Complex.Sum(Rd, sb);

}

//LastListOfDINN5GK.Add("");

//for(int o=0;o<3;o++)

//MasListDinnInfo[o] = new List<string>(MasListDinnInfo[o+1]);

//MasListDinnInfo[3] = LastListOfDINN5GK;

}

public static Complex[] DINN5\_GK(ComplexVectorFunc f, double t1, double t2, double t3, double t4, double tm, double tp, double eps, double pr, double gr, int N)

{

Complex[] Result;

Myfunc ff = (Complex x, ref Complex[] y, int n) => y = f(x, n);

DINN5\_GK(ff, t1, t2, t3, t4, tm, tp, eps, pr, gr, N, out Result);

return Result;

}

/// <summary>

/// Подсчёт несобственного комплексного интеграла от 0 до inf с учётом полюсов

/// </summary>

/// <param name="f">Функция комплексного переменного</param>

/// <param name="t1"></param>

/// <param name="t2"></param>

/// <param name="t3"></param>

/// <param name="t4"></param>

/// <param name="tm">Величина отклонения контура вниз</param>

/// <param name="tp">Величина отклонения контура вверх</param>

/// <param name="eps">Погрешность</param>

/// <param name="pr">Начальный шаг</param>

/// <param name="gr">Верхний предел</param>

/// <remarks>

/// ВЗЯТО С ДОКУМЕНТАЦИИ ОТ ФОРТРАНА

/// ! [t1,t2],[t3,t4] - участки отклонения контура вниз (real8)

///! [t2, t3] - участок отклонения контура вверх(real8)

///! tm,tp > 0 - величины отклонения контура вниз и вверх(real8)

///! (если нет обратной волны, то следует положить t2 = t3 = t1, tp = 0

///! обход полюсов при этом будет только снизу на участке[t1, t4]

///! с отклонением на tm)

/// </remarks>

/// <returns></returns>

public static Complex DINN\_GK(ComplexFunc f, double t1, double t2, double t3, double t4, double tm, double tp = 0, double eps = 1e-4, double pr = 1e-2, double gr = 1e4)

{

ComplexVectorFunc ff = (Complex x, int n) => new Complex[] { f(x) };

return DINN5\_GK(ff, t1, t2, t3, t4, tm, tp, eps, pr, gr, 1)[0];

}

/// <summary>

/// Несобственный интеграл с нулевыми параметрами (если не надо делать обход контура)

/// </summary>

/// <param name="f"></param>

/// <returns></returns>

public static Complex DINN\_GKwith0(ComplexFunc f) => DINN\_GK(f, 0, 0, 0, 0, 0, 0);

/// <summary>

/// Несобственный интеграл по всей оси от конкретно этой функции

/// </summary>

/// <param name="f"></param>

/// <returns></returns>

public static Complex DINN\_GKwith0Full(ComplexFunc f)

{

ComplexFunc f3 = t => f(t) + f(-t);

return DINN\_GKwith0(f3);

}

static void GK\_adaptive\_int\_RealV(Myfunc int\_func, double a, double b, double int\_h, double eps, Complex[] ret\_arr, int N)

{

// implicit none;

double eps\_out = 0, t\_i\_h, t\_x;

Complex[] ret\_arr\_0 = new Complex[N + 1];

t\_x = a;

t\_i\_h = int\_h;

while (b - t\_x > eps)

{

if (t\_x + t\_i\_h > b) t\_i\_h = b - t\_x;

GK\_int\_RealV(int\_func, t\_x, t\_x + t\_i\_h, ret\_arr\_0, eps\_out, N);

if (eps\_out > eps)

{

t\_i\_h = t\_i\_h \* 0.5;

}

else

{

ret\_arr = Complex.Sum(ret\_arr, ret\_arr\_0);

t\_x = t\_x + t\_i\_h;

// Console.WriteLine(t\_x);

if (eps\_out < RV\_eps\_step\_increment) t\_i\_h = t\_i\_h \* 1.5;

}

}

int\_h = t\_i\_h;

}

static void GK\_int\_RealV(Myfunc int\_func, double a, double b, Complex[] ret\_arr, double eps\_out, int N)

{

// implicit none;

int i;

double[] GK\_nodes\_arb = new double[Nodes + 1], K\_weights\_arb = new double[Nodes + 1], G\_weights\_arb = new double[Nodes + 1];

Complex[,] temp\_arr = new Complex[N + 1, 3];

GK\_nodes\_arb = (double[])((Vectors)(GK\_nodes) \* 0.5 \* (b - a) + 0.5 \* (b + a));

K\_weights\_arb = (double[])((Vectors)(K\_weights) \* 0.5 \* (b - a));

G\_weights\_arb = (double[])((Vectors)(G\_weights) \* 0.5 \* (b - a));

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

{

int\_func(GK\_nodes\_arb[i], ref ret\_arr, N);

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

{

temp\_arr[j, 1] = temp\_arr[j, 1] + K\_weights\_arb[i] \* ret\_arr[j - 1];

temp\_arr[j, 2] = temp\_arr[j, 2] + G\_weights\_arb[i] \* ret\_arr[j - 1];

}

}

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

ret\_arr[i] = temp\_arr[i, 1] - temp\_arr[i, 2];

Vectors v = new Vectors(N);

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

v[i] = ret\_arr[i].Abs;

eps\_out = (200 \* v.Max) \* 1.5;

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

ret\_arr[i] = temp\_arr[i, 1];

}

private static double[] x15, x21, x31, x41, x51, x61, wgauss15, wgauss21, wgauss31, wgauss41, wgauss51, wgauss61, wkronrod15, wkronrod21, wkronrod31, wkronrod41, wkronrod51, wkronrod61;

private static int ng15=4, ng21=5, ng31=8, ng41=10, ng51=13, ng61=15;

private static void MyGKInit()

{

x15=new double[15];

x21 = new double[21];

x31 = new double[31];

x41 = new double[41];

x51 = new double[51];

x61 = new double[61];

wgauss15 = new double[15];

wgauss21 = new double[21];

wgauss31 = new double[31];

wgauss41 = new double[41];

wgauss51 = new double[51];

wgauss61 = new double[61];

wkronrod15 = new double[15];

wkronrod21 = new double[21];

wkronrod31 = new double[31];

wkronrod41 = new double[41];

wkronrod51 = new double[51];

wkronrod61 = new double[61];

wgauss15[0] = 0.129484966168869693270611432679082;

wgauss15[1] = 0.279705391489276667901467771423780;

wgauss15[2] = 0.381830050505118944950369775488975;

wgauss15[3] = 0.417959183673469387755102040816327;

x15[0] = 0.991455371120812639206854697526329;

x15[1] = 0.949107912342758524526189684047851;

x15[2] = 0.864864423359769072789712788640926;

x15[3] = 0.741531185599394439863864773280788;

x15[4] = 0.586087235467691130294144838258730;

x15[5] = 0.405845151377397166906606412076961;

x15[6] = 0.207784955007898467600689403773245;

x15[7] = 0.000000000000000000000000000000000;

wkronrod15[0] = 0.022935322010529224963732008058970;

wkronrod15[1] = 0.063092092629978553290700663189204;

wkronrod15[2] = 0.104790010322250183839876322541518;

wkronrod15[3] = 0.140653259715525918745189590510238;

wkronrod15[4] = 0.169004726639267902826583426598550;

wkronrod15[5] = 0.190350578064785409913256402421014;

wkronrod15[6] = 0.204432940075298892414161999234649;

wkronrod15[7] = 0.209482141084727828012999174891714;

wgauss21[0] = 0.066671344308688137593568809893332;

wgauss21[1] = 0.149451349150580593145776339657697;

wgauss21[2] = 0.219086362515982043995534934228163;

wgauss21[3] = 0.269266719309996355091226921569469;

wgauss21[4] = 0.295524224714752870173892994651338;

x21[0] = 0.995657163025808080735527280689003;

x21[1] = 0.973906528517171720077964012084452;

x21[2] = 0.930157491355708226001207180059508;

x21[3] = 0.865063366688984510732096688423493;

x21[4] = 0.780817726586416897063717578345042;

x21[5] = 0.679409568299024406234327365114874;

x21[6] = 0.562757134668604683339000099272694;

x21[7] = 0.433395394129247190799265943165784;

x21[8] = 0.294392862701460198131126603103866;

x21[9] = 0.148874338981631210884826001129720;

x21[10] = 0.000000000000000000000000000000000;

wkronrod21[0] = 0.011694638867371874278064396062192;

wkronrod21[1] = 0.032558162307964727478818972459390;

wkronrod21[2] = 0.054755896574351996031381300244580;

wkronrod21[3] = 0.075039674810919952767043140916190;

wkronrod21[4] = 0.093125454583697605535065465083366;

wkronrod21[5] = 0.109387158802297641899210590325805;

wkronrod21[6] = 0.123491976262065851077958109831074;

wkronrod21[7] = 0.134709217311473325928054001771707;

wkronrod21[8] = 0.142775938577060080797094273138717;

wkronrod21[9] = 0.147739104901338491374841515972068;

wkronrod21[10] = 0.149445554002916905664936468389821;

wgauss31[0] = 0.030753241996117268354628393577204;

wgauss31[1] = 0.070366047488108124709267416450667;

wgauss31[2] = 0.107159220467171935011869546685869;

wgauss31[3] = 0.139570677926154314447804794511028;

wgauss31[4] = 0.166269205816993933553200860481209;

wgauss31[5] = 0.186161000015562211026800561866423;

wgauss31[6] = 0.198431485327111576456118326443839;

wgauss31[7] = 0.202578241925561272880620199967519;

x31[0] = 0.998002298693397060285172840152271;

x31[1] = 0.987992518020485428489565718586613;

x31[2] = 0.967739075679139134257347978784337;

x31[3] = 0.937273392400705904307758947710209;

x31[4] = 0.897264532344081900882509656454496;

x31[5] = 0.848206583410427216200648320774217;

x31[6] = 0.790418501442465932967649294817947;

x31[7] = 0.724417731360170047416186054613938;

x31[8] = 0.650996741297416970533735895313275;

x31[9] = 0.570972172608538847537226737253911;

x31[10] = 0.485081863640239680693655740232351;

x31[11] = 0.394151347077563369897207370981045;

x31[12] = 0.299180007153168812166780024266389;

x31[13] = 0.201194093997434522300628303394596;

x31[14] = 0.101142066918717499027074231447392;

x31[15] = 0.000000000000000000000000000000000;

wkronrod31[0] = 0.005377479872923348987792051430128;

wkronrod31[1] = 0.015007947329316122538374763075807;

wkronrod31[2] = 0.025460847326715320186874001019653;

wkronrod31[3] = 0.035346360791375846222037948478360;

wkronrod31[4] = 0.044589751324764876608227299373280;

wkronrod31[5] = 0.053481524690928087265343147239430;

wkronrod31[6] = 0.062009567800670640285139230960803;

wkronrod31[7] = 0.069854121318728258709520077099147;

wkronrod31[8] = 0.076849680757720378894432777482659;

wkronrod31[9] = 0.083080502823133021038289247286104;

wkronrod31[10] = 0.088564443056211770647275443693774;

wkronrod31[11] = 0.093126598170825321225486872747346;

wkronrod31[12] = 0.096642726983623678505179907627589;

wkronrod31[13] = 0.099173598721791959332393173484603;

wkronrod31[14] = 0.100769845523875595044946662617570;

wkronrod31[15] = 0.101330007014791549017374792767493;

wgauss41[0] = 0.017614007139152118311861962351853;

wgauss41[1] = 0.040601429800386941331039952274932;

wgauss41[2] = 0.062672048334109063569506535187042;

wgauss41[3] = 0.083276741576704748724758143222046;

wgauss41[4] = 0.101930119817240435036750135480350;

wgauss41[5] = 0.118194531961518417312377377711382;

wgauss41[6] = 0.131688638449176626898494499748163;

wgauss41[7] = 0.142096109318382051329298325067165;

wgauss41[8] = 0.149172986472603746787828737001969;

wgauss41[9] = 0.152753387130725850698084331955098;

x41[0] = 0.998859031588277663838315576545863;

x41[1] = 0.993128599185094924786122388471320;

x41[2] = 0.981507877450250259193342994720217;

x41[3] = 0.963971927277913791267666131197277;

x41[4] = 0.940822633831754753519982722212443;

x41[5] = 0.912234428251325905867752441203298;

x41[6] = 0.878276811252281976077442995113078;

x41[7] = 0.839116971822218823394529061701521;

x41[8] = 0.795041428837551198350638833272788;

x41[9] = 0.746331906460150792614305070355642;

x41[10] = 0.693237656334751384805490711845932;

x41[11] = 0.636053680726515025452836696226286;

x41[12] = 0.575140446819710315342946036586425;

x41[13] = 0.510867001950827098004364050955251;

x41[14] = 0.443593175238725103199992213492640;

x41[15] = 0.373706088715419560672548177024927;

x41[16] = 0.301627868114913004320555356858592;

x41[17] = 0.227785851141645078080496195368575;

x41[18] = 0.152605465240922675505220241022678;

x41[19] = 0.076526521133497333754640409398838;

x41[20] = 0.000000000000000000000000000000000;

wkronrod41[0] = 0.003073583718520531501218293246031;

wkronrod41[1] = 0.008600269855642942198661787950102;

wkronrod41[2] = 0.014626169256971252983787960308868;

wkronrod41[3] = 0.020388373461266523598010231432755;

wkronrod41[4] = 0.025882133604951158834505067096153;

wkronrod41[5] = 0.031287306777032798958543119323801;

wkronrod41[6] = 0.036600169758200798030557240707211;

wkronrod41[7] = 0.041668873327973686263788305936895;

wkronrod41[8] = 0.046434821867497674720231880926108;

wkronrod41[9] = 0.050944573923728691932707670050345;

wkronrod41[10] = 0.055195105348285994744832372419777;

wkronrod41[11] = 0.059111400880639572374967220648594;

wkronrod41[12] = 0.062653237554781168025870122174255;

wkronrod41[13] = 0.065834597133618422111563556969398;

wkronrod41[14] = 0.068648672928521619345623411885368;

wkronrod41[15] = 0.071054423553444068305790361723210;

wkronrod41[16] = 0.073030690332786667495189417658913;

wkronrod41[17] = 0.074582875400499188986581418362488;

wkronrod41[18] = 0.075704497684556674659542775376617;

wkronrod41[19] = 0.076377867672080736705502835038061;

wkronrod41[20] = 0.076600711917999656445049901530102;

wgauss51[0] = 0.011393798501026287947902964113235;

wgauss51[1] = 0.026354986615032137261901815295299;

wgauss51[2] = 0.040939156701306312655623487711646;

wgauss51[3] = 0.054904695975835191925936891540473;

wgauss51[4] = 0.068038333812356917207187185656708;

wgauss51[5] = 0.080140700335001018013234959669111;

wgauss51[6] = 0.091028261982963649811497220702892;

wgauss51[7] = 0.100535949067050644202206890392686;

wgauss51[8] = 0.108519624474263653116093957050117;

wgauss51[9] = 0.114858259145711648339325545869556;

wgauss51[10] = 0.119455763535784772228178126512901;

wgauss51[11] = 0.122242442990310041688959518945852;

wgauss51[12] = 0.123176053726715451203902873079050;

x51[0] = 0.999262104992609834193457486540341;

x51[1] = 0.995556969790498097908784946893902;

x51[2] = 0.988035794534077247637331014577406;

x51[3] = 0.976663921459517511498315386479594;

x51[4] = 0.961614986425842512418130033660167;

x51[5] = 0.942974571228974339414011169658471;

x51[6] = 0.920747115281701561746346084546331;

x51[7] = 0.894991997878275368851042006782805;

x51[8] = 0.865847065293275595448996969588340;

x51[9] = 0.833442628760834001421021108693570;

x51[10] = 0.797873797998500059410410904994307;

x51[11] = 0.759259263037357630577282865204361;

x51[12] = 0.717766406813084388186654079773298;

x51[13] = 0.673566368473468364485120633247622;

x51[14] = 0.626810099010317412788122681624518;

x51[15] = 0.577662930241222967723689841612654;

x51[16] = 0.526325284334719182599623778158010;

x51[17] = 0.473002731445714960522182115009192;

x51[18] = 0.417885382193037748851814394594572;

x51[19] = 0.361172305809387837735821730127641;

x51[20] = 0.303089538931107830167478909980339;

x51[21] = 0.243866883720988432045190362797452;

x51[22] = 0.183718939421048892015969888759528;

x51[23] = 0.122864692610710396387359818808037;

x51[24] = 0.061544483005685078886546392366797;

x51[25] = 0.000000000000000000000000000000000;

wkronrod51[0] = 0.001987383892330315926507851882843;

wkronrod51[1] = 0.005561932135356713758040236901066;

wkronrod51[2] = 0.009473973386174151607207710523655;

wkronrod51[3] = 0.013236229195571674813656405846976;

wkronrod51[4] = 0.016847817709128298231516667536336;

wkronrod51[5] = 0.020435371145882835456568292235939;

wkronrod51[6] = 0.024009945606953216220092489164881;

wkronrod51[7] = 0.027475317587851737802948455517811;

wkronrod51[8] = 0.030792300167387488891109020215229;

wkronrod51[9] = 0.034002130274329337836748795229551;

wkronrod51[10] = 0.037116271483415543560330625367620;

wkronrod51[11] = 0.040083825504032382074839284467076;

wkronrod51[12] = 0.042872845020170049476895792439495;

wkronrod51[13] = 0.045502913049921788909870584752660;

wkronrod51[14] = 0.047982537138836713906392255756915;

wkronrod51[15] = 0.050277679080715671963325259433440;

wkronrod51[16] = 0.052362885806407475864366712137873;

wkronrod51[17] = 0.054251129888545490144543370459876;

wkronrod51[18] = 0.055950811220412317308240686382747;

wkronrod51[19] = 0.057437116361567832853582693939506;

wkronrod51[20] = 0.058689680022394207961974175856788;

wkronrod51[21] = 0.059720340324174059979099291932562;

wkronrod51[22] = 0.060539455376045862945360267517565;

wkronrod51[23] = 0.061128509717053048305859030416293;

wkronrod51[24] = 0.061471189871425316661544131965264;

wkronrod51[25] = 0.061580818067832935078759824240055;

wgauss61[0] = 0.007968192496166605615465883474674;

wgauss61[1] = 0.018466468311090959142302131912047;

wgauss61[2] = 0.028784707883323369349719179611292;

wgauss61[3] = 0.038799192569627049596801936446348;

wgauss61[4] = 0.048402672830594052902938140422808;

wgauss61[5] = 0.057493156217619066481721689402056;

wgauss61[6] = 0.065974229882180495128128515115962;

wgauss61[7] = 0.073755974737705206268243850022191;

wgauss61[8] = 0.080755895229420215354694938460530;

wgauss61[9] = 0.086899787201082979802387530715126;

wgauss61[10] = 0.092122522237786128717632707087619;

wgauss61[11] = 0.096368737174644259639468626351810;

wgauss61[12] = 0.099593420586795267062780282103569;

wgauss61[13] = 0.101762389748405504596428952168554;

wgauss61[14] = 0.102852652893558840341285636705415;

x61[0] = 0.999484410050490637571325895705811;

x61[1] = 0.996893484074649540271630050918695;

x61[2] = 0.991630996870404594858628366109486;

x61[3] = 0.983668123279747209970032581605663;

x61[4] = 0.973116322501126268374693868423707;

x61[5] = 0.960021864968307512216871025581798;

x61[6] = 0.944374444748559979415831324037439;

x61[7] = 0.926200047429274325879324277080474;

x61[8] = 0.905573307699907798546522558925958;

x61[9] = 0.882560535792052681543116462530226;

x61[10] = 0.857205233546061098958658510658944;

x61[11] = 0.829565762382768397442898119732502;

x61[12] = 0.799727835821839083013668942322683;

x61[13] = 0.767777432104826194917977340974503;

x61[14] = 0.733790062453226804726171131369528;

x61[15] = 0.697850494793315796932292388026640;

x61[16] = 0.660061064126626961370053668149271;

x61[17] = 0.620526182989242861140477556431189;

x61[18] = 0.579345235826361691756024932172540;

x61[19] = 0.536624148142019899264169793311073;

x61[20] = 0.492480467861778574993693061207709;

x61[21] = 0.447033769538089176780609900322854;

x61[22] = 0.400401254830394392535476211542661;

x61[23] = 0.352704725530878113471037207089374;

x61[24] = 0.304073202273625077372677107199257;

x61[25] = 0.254636926167889846439805129817805;

x61[26] = 0.204525116682309891438957671002025;

x61[27] = 0.153869913608583546963794672743256;

x61[28] = 0.102806937966737030147096751318001;

x61[29] = 0.051471842555317695833025213166723;

x61[30] = 0.000000000000000000000000000000000;

wkronrod61[0] = 0.001389013698677007624551591226760;

wkronrod61[1] = 0.003890461127099884051267201844516;

wkronrod61[2] = 0.006630703915931292173319826369750;

wkronrod61[3] = 0.009273279659517763428441146892024;

wkronrod61[4] = 0.011823015253496341742232898853251;

wkronrod61[5] = 0.014369729507045804812451432443580;

wkronrod61[6] = 0.016920889189053272627572289420322;

wkronrod61[7] = 0.019414141193942381173408951050128;

wkronrod61[8] = 0.021828035821609192297167485738339;

wkronrod61[9] = 0.024191162078080601365686370725232;

wkronrod61[10] = 0.026509954882333101610601709335075;

wkronrod61[11] = 0.028754048765041292843978785354334;

wkronrod61[12] = 0.030907257562387762472884252943092;

wkronrod61[13] = 0.032981447057483726031814191016854;

wkronrod61[14] = 0.034979338028060024137499670731468;

wkronrod61[15] = 0.036882364651821229223911065617136;

wkronrod61[16] = 0.038678945624727592950348651532281;

wkronrod61[17] = 0.040374538951535959111995279752468;

wkronrod61[18] = 0.041969810215164246147147541285970;

wkronrod61[19] = 0.043452539701356069316831728117073;

wkronrod61[20] = 0.044814800133162663192355551616723;

wkronrod61[21] = 0.046059238271006988116271735559374;

wkronrod61[22] = 0.047185546569299153945261478181099;

wkronrod61[23] = 0.048185861757087129140779492298305;

wkronrod61[24] = 0.049055434555029778887528165367238;

wkronrod61[25] = 0.049795683427074206357811569379942;

wkronrod61[26] = 0.050405921402782346840893085653585;

wkronrod61[27] = 0.050881795898749606492297473049805;

wkronrod61[28] = 0.051221547849258772170656282604944;

wkronrod61[29] = 0.051426128537459025933862879215781;

wkronrod61[30] = 0.051494729429451567558340433647099;

int n = 15, ng = ng15;

// copy nodes

for (int i = n - 1; i >= n / 2; i--)

{

x15[i] = -x15[n - 1 - i];

wkronrod15[i] = wkronrod15[n - 1 - i];

}

// copy Gauss weights

for (int i = ng - 1; i >= 0; i--)

{

wgauss15[n - 2 - 2 \* i] = wgauss15[i];

wgauss15[1 + 2 \* i] = wgauss15[i];

}

for (int i = 0; i <= n / 2; i++)

{

wgauss15[2 \* i] = 0;

}

n = 21; ng = ng21;

// copy nodes

for (int i = n - 1; i >= n / 2; i--)

{

x21[i] = -x21[n - 1 - i];

wkronrod21[i] = wkronrod21[n - 1 - i];

}

// copy Gauss weights

for (int i = ng - 1; i >= 0; i--)

{

wgauss21[n - 2 - 2 \* i] = wgauss21[i];

wgauss21[1 + 2 \* i] = wgauss21[i];

}

for (int i = 0; i <= n / 2; i++)

{

wgauss21[2 \* i] = 0;

}

n = 31; ng = ng31;

// copy nodes

for (int i = n - 1; i >= n / 2; i--)

{

x31[i] = -x31[n - 1 - i];

wkronrod31[i] = wkronrod31[n - 1 - i];

}

// copy Gauss weights

for (int i = ng - 1; i >= 0; i--)

{

wgauss31[n - 2 - 2 \* i] = wgauss31[i];

wgauss31[1 + 2 \* i] = wgauss31[i];

}

for (int i = 0; i <= n / 2; i++)

{

wgauss31[2 \* i] = 0;

}

n = 41; ng = ng41;

// copy nodes

for (int i = n - 1; i >= n / 2; i--)

{

x41[i] = -x41[n - 1 - i];

wkronrod41[i] = wkronrod41[n - 1 - i];

}

// copy Gauss weights

for (int i = ng - 1; i >= 0; i--)

{

wgauss41[n - 2 - 2 \* i] = wgauss41[i];

wgauss41[1 + 2 \* i] = wgauss41[i];

}

for (int i = 0; i <= n / 2; i++)

{

wgauss41[2 \* i] = 0;

}

n = 51; ng = ng51;

// copy nodes

for (int i = n - 1; i >= n / 2; i--)

{

x51[i] = -x51[n - 1 - i];

wkronrod51[i] = wkronrod51[n - 1 - i];

}

// copy Gauss weights

for (int i = ng - 1; i >= 0; i--)

{

wgauss51[n - 2 - 2 \* i] = wgauss51[i];

wgauss51[1 + 2 \* i] = wgauss51[i];

}

for (int i = 0; i <= n / 2; i++)

{

wgauss51[2 \* i] = 0;

}

n = 61; ng = ng61;

// copy nodes

for (int i = n - 1; i >= n / 2; i--)

{

x61[i] = -x61[n - 1 - i];

wkronrod61[i] = wkronrod61[n - 1 - i];

}

// copy Gauss weights

for (int i = ng - 1; i >= 0; i--)

{

wgauss61[n - 2 - 2 \* i] = wgauss61[i];

wgauss61[1 + 2 \* i] = wgauss61[i];

}

for (int i = 0; i <= n / 2; i++)

{

wgauss61[2 \* i] = 0;

}

}

/// <summary>

/// Взятый с alglib метод Гаусса-Кронрода с выбором числа точек

/// </summary>

/// <param name="f"></param>

/// <param name="a"></param>

/// <param name="b"></param>

/// <param name="n"></param>

/// <returns></returns>

public static Complex MySimpleGaussKronrod(ComplexFunc f, Complex a, Complex b, int n = 61)

{

double[] x;

double[] wkronrod;

double[] wgauss;

double eps = 1.0E-32;

int ng = 0;

int[] p1 = new int[0];

int[] p2 = new int[0];

//if (!(n == 15 | n == 21 | n == 31 | n == 41 | n == 51 | n == 61)) throw new Exception("GKQNodesTbl: incorrect N!");

switch (n)

{

case 61:

x = x61;

wkronrod = wkronrod61;

wgauss = wgauss61;

break;

case 15:

x = x15;

wkronrod = wkronrod15;

wgauss = wgauss15;

break;

case 31:

x = x31;

wkronrod = wkronrod31;

wgauss = wgauss31;

break;

case 41:

x = x41;

wkronrod = wkronrod41;

wgauss = wgauss41;

break;

case 51:

x = x15;

wkronrod = wkronrod51;

wgauss = wgauss51;

break;

case 21:

x = x21;

wkronrod = wkronrod21;

wgauss = wgauss21;

break;

default:

throw new Exception("GKQNodesTbl: incorrect N! (n должно быть 15/21/31/41/51/61)");

}

//x = new double[n];

//wkronrod = new double[n];

//wgauss = new double[n];

//if (n == 15)

//{

// ng = 4;

// wgauss[0] = 0.129484966168869693270611432679082;

// wgauss[1] = 0.279705391489276667901467771423780;

// wgauss[2] = 0.381830050505118944950369775488975;

// wgauss[3] = 0.417959183673469387755102040816327;

// x[0] = 0.991455371120812639206854697526329;

// x[1] = 0.949107912342758524526189684047851;

// x[2] = 0.864864423359769072789712788640926;

// x[3] = 0.741531185599394439863864773280788;

// x[4] = 0.586087235467691130294144838258730;

// x[5] = 0.405845151377397166906606412076961;

// x[6] = 0.207784955007898467600689403773245;

// x[7] = 0.000000000000000000000000000000000;

// wkronrod[0] = 0.022935322010529224963732008058970;

// wkronrod[1] = 0.063092092629978553290700663189204;

// wkronrod[2] = 0.104790010322250183839876322541518;

// wkronrod[3] = 0.140653259715525918745189590510238;

// wkronrod[4] = 0.169004726639267902826583426598550;

// wkronrod[5] = 0.190350578064785409913256402421014;

// wkronrod[6] = 0.204432940075298892414161999234649;

// wkronrod[7] = 0.209482141084727828012999174891714;

//}

//if (n == 21)

//{

// ng = 5;

// wgauss[0] = 0.066671344308688137593568809893332;

// wgauss[1] = 0.149451349150580593145776339657697;

// wgauss[2] = 0.219086362515982043995534934228163;

// wgauss[3] = 0.269266719309996355091226921569469;

// wgauss[4] = 0.295524224714752870173892994651338;

// x[0] = 0.995657163025808080735527280689003;

// x[1] = 0.973906528517171720077964012084452;

// x[2] = 0.930157491355708226001207180059508;

// x[3] = 0.865063366688984510732096688423493;

// x[4] = 0.780817726586416897063717578345042;

// x[5] = 0.679409568299024406234327365114874;

// x[6] = 0.562757134668604683339000099272694;

// x[7] = 0.433395394129247190799265943165784;

// x[8] = 0.294392862701460198131126603103866;

// x[9] = 0.148874338981631210884826001129720;

// x[10] = 0.000000000000000000000000000000000;

// wkronrod[0] = 0.011694638867371874278064396062192;

// wkronrod[1] = 0.032558162307964727478818972459390;

// wkronrod[2] = 0.054755896574351996031381300244580;

// wkronrod[3] = 0.075039674810919952767043140916190;

// wkronrod[4] = 0.093125454583697605535065465083366;

// wkronrod[5] = 0.109387158802297641899210590325805;

// wkronrod[6] = 0.123491976262065851077958109831074;

// wkronrod[7] = 0.134709217311473325928054001771707;

// wkronrod[8] = 0.142775938577060080797094273138717;

// wkronrod[9] = 0.147739104901338491374841515972068;

// wkronrod[10] = 0.149445554002916905664936468389821;

//}

//if (n == 31)

//{

// ng = 8;

// wgauss[0] = 0.030753241996117268354628393577204;

// wgauss[1] = 0.070366047488108124709267416450667;

// wgauss[2] = 0.107159220467171935011869546685869;

// wgauss[3] = 0.139570677926154314447804794511028;

// wgauss[4] = 0.166269205816993933553200860481209;

// wgauss[5] = 0.186161000015562211026800561866423;

// wgauss[6] = 0.198431485327111576456118326443839;

// wgauss[7] = 0.202578241925561272880620199967519;

// x[0] = 0.998002298693397060285172840152271;

// x[1] = 0.987992518020485428489565718586613;

// x[2] = 0.967739075679139134257347978784337;

// x[3] = 0.937273392400705904307758947710209;

// x[4] = 0.897264532344081900882509656454496;

// x[5] = 0.848206583410427216200648320774217;

// x[6] = 0.790418501442465932967649294817947;

// x[7] = 0.724417731360170047416186054613938;

// x[8] = 0.650996741297416970533735895313275;

// x[9] = 0.570972172608538847537226737253911;

// x[10] = 0.485081863640239680693655740232351;

// x[11] = 0.394151347077563369897207370981045;

// x[12] = 0.299180007153168812166780024266389;

// x[13] = 0.201194093997434522300628303394596;

// x[14] = 0.101142066918717499027074231447392;

// x[15] = 0.000000000000000000000000000000000;

// wkronrod[0] = 0.005377479872923348987792051430128;

// wkronrod[1] = 0.015007947329316122538374763075807;

// wkronrod[2] = 0.025460847326715320186874001019653;

// wkronrod[3] = 0.035346360791375846222037948478360;

// wkronrod[4] = 0.044589751324764876608227299373280;

// wkronrod[5] = 0.053481524690928087265343147239430;

// wkronrod[6] = 0.062009567800670640285139230960803;

// wkronrod[7] = 0.069854121318728258709520077099147;

// wkronrod[8] = 0.076849680757720378894432777482659;

// wkronrod[9] = 0.083080502823133021038289247286104;

// wkronrod[10] = 0.088564443056211770647275443693774;

// wkronrod[11] = 0.093126598170825321225486872747346;

// wkronrod[12] = 0.096642726983623678505179907627589;

// wkronrod[13] = 0.099173598721791959332393173484603;

// wkronrod[14] = 0.100769845523875595044946662617570;

// wkronrod[15] = 0.101330007014791549017374792767493;

//}

//if (n == 41)

//{

// ng = 10;

// wgauss[0] = 0.017614007139152118311861962351853;

// wgauss[1] = 0.040601429800386941331039952274932;

// wgauss[2] = 0.062672048334109063569506535187042;

// wgauss[3] = 0.083276741576704748724758143222046;

// wgauss[4] = 0.101930119817240435036750135480350;

// wgauss[5] = 0.118194531961518417312377377711382;

// wgauss[6] = 0.131688638449176626898494499748163;

// wgauss[7] = 0.142096109318382051329298325067165;

// wgauss[8] = 0.149172986472603746787828737001969;

// wgauss[9] = 0.152753387130725850698084331955098;

// x[0] = 0.998859031588277663838315576545863;

// x[1] = 0.993128599185094924786122388471320;

// x[2] = 0.981507877450250259193342994720217;

// x[3] = 0.963971927277913791267666131197277;

// x[4] = 0.940822633831754753519982722212443;

// x[5] = 0.912234428251325905867752441203298;

// x[6] = 0.878276811252281976077442995113078;

// x[7] = 0.839116971822218823394529061701521;

// x[8] = 0.795041428837551198350638833272788;

// x[9] = 0.746331906460150792614305070355642;

// x[10] = 0.693237656334751384805490711845932;

// x[11] = 0.636053680726515025452836696226286;

// x[12] = 0.575140446819710315342946036586425;

// x[13] = 0.510867001950827098004364050955251;

// x[14] = 0.443593175238725103199992213492640;

// x[15] = 0.373706088715419560672548177024927;

// x[16] = 0.301627868114913004320555356858592;

// x[17] = 0.227785851141645078080496195368575;

// x[18] = 0.152605465240922675505220241022678;

// x[19] = 0.076526521133497333754640409398838;

// x[20] = 0.000000000000000000000000000000000;

// wkronrod[0] = 0.003073583718520531501218293246031;

// wkronrod[1] = 0.008600269855642942198661787950102;

// wkronrod[2] = 0.014626169256971252983787960308868;

// wkronrod[3] = 0.020388373461266523598010231432755;

// wkronrod[4] = 0.025882133604951158834505067096153;

// wkronrod[5] = 0.031287306777032798958543119323801;

// wkronrod[6] = 0.036600169758200798030557240707211;

// wkronrod[7] = 0.041668873327973686263788305936895;

// wkronrod[8] = 0.046434821867497674720231880926108;

// wkronrod[9] = 0.050944573923728691932707670050345;

// wkronrod[10] = 0.055195105348285994744832372419777;

// wkronrod[11] = 0.059111400880639572374967220648594;

// wkronrod[12] = 0.062653237554781168025870122174255;

// wkronrod[13] = 0.065834597133618422111563556969398;

// wkronrod[14] = 0.068648672928521619345623411885368;

// wkronrod[15] = 0.071054423553444068305790361723210;

// wkronrod[16] = 0.073030690332786667495189417658913;

// wkronrod[17] = 0.074582875400499188986581418362488;

// wkronrod[18] = 0.075704497684556674659542775376617;

// wkronrod[19] = 0.076377867672080736705502835038061;

// wkronrod[20] = 0.076600711917999656445049901530102;

//}

//if (n == 51)

//{

// ng = 13;

// wgauss[0] = 0.011393798501026287947902964113235;

// wgauss[1] = 0.026354986615032137261901815295299;

// wgauss[2] = 0.040939156701306312655623487711646;

// wgauss[3] = 0.054904695975835191925936891540473;

// wgauss[4] = 0.068038333812356917207187185656708;

// wgauss[5] = 0.080140700335001018013234959669111;

// wgauss[6] = 0.091028261982963649811497220702892;

// wgauss[7] = 0.100535949067050644202206890392686;

// wgauss[8] = 0.108519624474263653116093957050117;

// wgauss[9] = 0.114858259145711648339325545869556;

// wgauss[10] = 0.119455763535784772228178126512901;

// wgauss[11] = 0.122242442990310041688959518945852;

// wgauss[12] = 0.123176053726715451203902873079050;

// x[0] = 0.999262104992609834193457486540341;

// x[1] = 0.995556969790498097908784946893902;

// x[2] = 0.988035794534077247637331014577406;

// x[3] = 0.976663921459517511498315386479594;

// x[4] = 0.961614986425842512418130033660167;

// x[5] = 0.942974571228974339414011169658471;

// x[6] = 0.920747115281701561746346084546331;

// x[7] = 0.894991997878275368851042006782805;

// x[8] = 0.865847065293275595448996969588340;

// x[9] = 0.833442628760834001421021108693570;

// x[10] = 0.797873797998500059410410904994307;

// x[11] = 0.759259263037357630577282865204361;

// x[12] = 0.717766406813084388186654079773298;

// x[13] = 0.673566368473468364485120633247622;

// x[14] = 0.626810099010317412788122681624518;

// x[15] = 0.577662930241222967723689841612654;

// x[16] = 0.526325284334719182599623778158010;

// x[17] = 0.473002731445714960522182115009192;

// x[18] = 0.417885382193037748851814394594572;

// x[19] = 0.361172305809387837735821730127641;

// x[20] = 0.303089538931107830167478909980339;

// x[21] = 0.243866883720988432045190362797452;

// x[22] = 0.183718939421048892015969888759528;

// x[23] = 0.122864692610710396387359818808037;

// x[24] = 0.061544483005685078886546392366797;

// x[25] = 0.000000000000000000000000000000000;

// wkronrod[0] = 0.001987383892330315926507851882843;

// wkronrod[1] = 0.005561932135356713758040236901066;

// wkronrod[2] = 0.009473973386174151607207710523655;

// wkronrod[3] = 0.013236229195571674813656405846976;

// wkronrod[4] = 0.016847817709128298231516667536336;

// wkronrod[5] = 0.020435371145882835456568292235939;

// wkronrod[6] = 0.024009945606953216220092489164881;

// wkronrod[7] = 0.027475317587851737802948455517811;

// wkronrod[8] = 0.030792300167387488891109020215229;

// wkronrod[9] = 0.034002130274329337836748795229551;

// wkronrod[10] = 0.037116271483415543560330625367620;

// wkronrod[11] = 0.040083825504032382074839284467076;

// wkronrod[12] = 0.042872845020170049476895792439495;

// wkronrod[13] = 0.045502913049921788909870584752660;

// wkronrod[14] = 0.047982537138836713906392255756915;

// wkronrod[15] = 0.050277679080715671963325259433440;

// wkronrod[16] = 0.052362885806407475864366712137873;

// wkronrod[17] = 0.054251129888545490144543370459876;

// wkronrod[18] = 0.055950811220412317308240686382747;

// wkronrod[19] = 0.057437116361567832853582693939506;

// wkronrod[20] = 0.058689680022394207961974175856788;

// wkronrod[21] = 0.059720340324174059979099291932562;

// wkronrod[22] = 0.060539455376045862945360267517565;

// wkronrod[23] = 0.061128509717053048305859030416293;

// wkronrod[24] = 0.061471189871425316661544131965264;

// wkronrod[25] = 0.061580818067832935078759824240055;

//}

//if (n == 61)

//{

// ng = 15;

// wgauss[0] = 0.007968192496166605615465883474674;

// wgauss[1] = 0.018466468311090959142302131912047;

// wgauss[2] = 0.028784707883323369349719179611292;

// wgauss[3] = 0.038799192569627049596801936446348;

// wgauss[4] = 0.048402672830594052902938140422808;

// wgauss[5] = 0.057493156217619066481721689402056;

// wgauss[6] = 0.065974229882180495128128515115962;

// wgauss[7] = 0.073755974737705206268243850022191;

// wgauss[8] = 0.080755895229420215354694938460530;

// wgauss[9] = 0.086899787201082979802387530715126;

// wgauss[10] = 0.092122522237786128717632707087619;

// wgauss[11] = 0.096368737174644259639468626351810;

// wgauss[12] = 0.099593420586795267062780282103569;

// wgauss[13] = 0.101762389748405504596428952168554;

// wgauss[14] = 0.102852652893558840341285636705415;

// x[0] = 0.999484410050490637571325895705811;

// x[1] = 0.996893484074649540271630050918695;

// x[2] = 0.991630996870404594858628366109486;

// x[3] = 0.983668123279747209970032581605663;

// x[4] = 0.973116322501126268374693868423707;

// x[5] = 0.960021864968307512216871025581798;

// x[6] = 0.944374444748559979415831324037439;

// x[7] = 0.926200047429274325879324277080474;

// x[8] = 0.905573307699907798546522558925958;

// x[9] = 0.882560535792052681543116462530226;

// x[10] = 0.857205233546061098958658510658944;

// x[11] = 0.829565762382768397442898119732502;

// x[12] = 0.799727835821839083013668942322683;

// x[13] = 0.767777432104826194917977340974503;

// x[14] = 0.733790062453226804726171131369528;

// x[15] = 0.697850494793315796932292388026640;

// x[16] = 0.660061064126626961370053668149271;

// x[17] = 0.620526182989242861140477556431189;

// x[18] = 0.579345235826361691756024932172540;

// x[19] = 0.536624148142019899264169793311073;

// x[20] = 0.492480467861778574993693061207709;

// x[21] = 0.447033769538089176780609900322854;

// x[22] = 0.400401254830394392535476211542661;

// x[23] = 0.352704725530878113471037207089374;

// x[24] = 0.304073202273625077372677107199257;

// x[25] = 0.254636926167889846439805129817805;

// x[26] = 0.204525116682309891438957671002025;

// x[27] = 0.153869913608583546963794672743256;

// x[28] = 0.102806937966737030147096751318001;

// x[29] = 0.051471842555317695833025213166723;

// x[30] = 0.000000000000000000000000000000000;

// wkronrod[0] = 0.001389013698677007624551591226760;

// wkronrod[1] = 0.003890461127099884051267201844516;

// wkronrod[2] = 0.006630703915931292173319826369750;

// wkronrod[3] = 0.009273279659517763428441146892024;

// wkronrod[4] = 0.011823015253496341742232898853251;

// wkronrod[5] = 0.014369729507045804812451432443580;

// wkronrod[6] = 0.016920889189053272627572289420322;

// wkronrod[7] = 0.019414141193942381173408951050128;

// wkronrod[8] = 0.021828035821609192297167485738339;

// wkronrod[9] = 0.024191162078080601365686370725232;

// wkronrod[10] = 0.026509954882333101610601709335075;

// wkronrod[11] = 0.028754048765041292843978785354334;

// wkronrod[12] = 0.030907257562387762472884252943092;

// wkronrod[13] = 0.032981447057483726031814191016854;

// wkronrod[14] = 0.034979338028060024137499670731468;

// wkronrod[15] = 0.036882364651821229223911065617136;

// wkronrod[16] = 0.038678945624727592950348651532281;

// wkronrod[17] = 0.040374538951535959111995279752468;

// wkronrod[18] = 0.041969810215164246147147541285970;

// wkronrod[19] = 0.043452539701356069316831728117073;

// wkronrod[20] = 0.044814800133162663192355551616723;

// wkronrod[21] = 0.046059238271006988116271735559374;

// wkronrod[22] = 0.047185546569299153945261478181099;

// wkronrod[23] = 0.048185861757087129140779492298305;

// wkronrod[24] = 0.049055434555029778887528165367238;

// wkronrod[25] = 0.049795683427074206357811569379942;

// wkronrod[26] = 0.050405921402782346840893085653585;

// wkronrod[27] = 0.050881795898749606492297473049805;

// wkronrod[28] = 0.051221547849258772170656282604944;

// wkronrod[29] = 0.051426128537459025933862879215781;

// wkronrod[30] = 0.051494729429451567558340433647099;

//}

//// copy nodes

//for (int i = n - 1; i >= n / 2; i--)

//{

// x[i] = -x[n - 1 - i];

// wkronrod[i] = wkronrod[n - 1 - i];

//}

//// copy Gauss weights

//for (int i = ng - 1; i >= 0; i--)

//{

// wgauss[n - 2 - 2 \* i] = wgauss[i];

// wgauss[1 + 2 \* i] = wgauss[i];

//}

//for (int i = 0; i <= n / 2; i++)

//{

// wgauss[2 \* i] = 0;

//}

//alglib.tsort.tagsort(ref x, n, ref p1, ref p2);

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

//{

// double tmp = wkronrod[i];

// wkronrod[i] = wkronrod[p2[i]];

// wkronrod[p2[i]] = tmp;

// tmp = wgauss[i];

// wgauss[i] = wgauss[p2[i]];

// wgauss[p2[i]] = tmp;

//}

ComplexFunc t = (Complex r) => (a + (r + 1) / 2 \* (b - a));

Complex sumKR = new Complex(0), sumGS = new Complex(0);

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

{

Complex tmp = f(t(x[i]));if(tmp==null)tmp.Show();

if (Double.IsNaN(tmp.Re) || Double.IsInfinity(tmp.Re)) throw new Exception($"Значение функции Nan или Inf во время интегрирования: f({t(x[i])}) = {f(t(x[i]))}");//Console.WriteLine($"Значение функции Nan или Inf во время интегрирования: f({t(x[i])}) = {f(t(x[i]))}");

sumKR += wkronrod[i] \* tmp;

sumGS += wgauss[i] \* tmp;

}

//if ((sumGS - sumKR).Abs > sumKR.Abs / 1000)

// return MySimpleGaussKronrod(f, a, a + (b-a) / 2, n) + MySimpleGaussKronrod(f,a+(b-a)/2,b,n);

//sumKR /= 2;//.Show();

//sumGS /= 2;//.Show();

return sumKR / 2 \* (b - a);

}

/// <summary>

/// Взятый с alglib метод Гаусса-Кронрода с выбором числа точек

/// </summary>

/// <param name="f"></param>

/// <param name="a"></param>

/// <param name="b"></param>

/// <param name="n"></param>

/// <returns></returns>

public static double MySimpleGaussKronrod(RealFunc f, double a, double b, int n = 61)

{

return MySimpleGaussKronrod((Complex t) => f(t.Re), new Complex(a), new Complex(b), n).Re;

}

/// <summary>

/// Метод Гаусса-Кронрода, который вместо отрезка делает обход контура, если на отрезке есть полюса

/// </summary>

/// <param name="f">Интегрируемая функция</param>

/// <param name="a">Начало отрезка интегрирования</param>

/// <param name="b">Конец отрезка интегрирования</param>

/// <param name="delta">Знаменатель интегрируемой функции</param>

/// <param name="t">Предполагаемый радиус отрезка, вне которого полюсов нет</param>

/// <param name="n">Число узлов интегрировани</param>

/// <param name="h">Отклонение контура</param>

/// <returns></returns>

public static Complex MySuperGaussKronrod(ComplexFunc f, Complex a, Complex b, ComplexFunc delta = null, int t = 100, int n = 61, double h = 0.2)

{

if (delta == null || Math.Max(b.Re, -a.Re) > t) return MySimpleGaussKronrod(f, a, b, n);//если знаменатель неизвестен либо отрезок вне зоны нахождения предполагаемых полюсов, интегрировать по-обычному

if (a.Im != 0 && a.Im != b.Im) return MySimpleGaussKronrod(f, a, b, n);//если отрезок не на вещественной оси, интегрировать

if (a.Re \* b.Re < 0 || delta(0) != 0) return MySuperGaussKronrod(f, a, 0, delta, t, n, h) + MySuperGaussKronrod(f, 0, b, delta, t, n, h);//если концы отрезка по обе стороны от 0, разбить на 2

var tmp = Optimization.Neu(delta, a, b);//tmp.Show();//найти корни на отрезке (надо знать, существуют или нет)

if (a.Re < 0) h \*= -1;//если отрезок слева от 0, брать обход снизу

if (tmp == null) return MySimpleGaussKronrod(f, a, b, n);//если нет полюсов, решать по-обычному

else

return MySimpleGaussKronrod(f, a, a + h, n) + MySimpleGaussKronrod(f, a + h, b + h, n) + MySimpleGaussKronrod(f, b + h, b, n);//иначе обойти контур

}

/// <summary>

/// Метод Гаусса-Кронрода, использующий параллельные вычисления за счёт разбиения отрезка интегрирования на несколько частей

/// </summary>

/// <param name="f">Интегрируемая функция</param>

/// <param name="a">Начало отрезка интегрирования</param>

/// <param name="b">Конец отрезка интегрирования</param>

/// <param name="n">Число точек в методе</param>

/// <param name="count">На сколько отрезков разбиватся исходный отрезок</param>

/// <returns></returns>

public static Complex ParallelGaussKronrod(ComplexFunc f, Complex a, Complex b, int n=61, int count=5)

{

Complex[] mas = new Complex[count];

Complex step = (b - a) / (count - 1);

Parallel.For(1, count,i=>mas[i]=MySimpleGaussKronrod(f,a+(i-1)\*step,a+i\*step,n));

Array.Sort(mas);

Complex sum = 0;

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

sum += mas[i];

return sum;

}

}