Program.cs

```
using System.Security.Principal;
try {

    if (args.Length == 0) throw new ArgumentException("Not found arguments!");
    if (args[0] == "-help") {
        ShowHelp(); return;
    }

    string json = File.ReadAllText(args[1]);
    Data data = JsonConvert.DeserializeObject<Data>(json)!;
    if (data is null) throw new FileNotFoundException("File uncorrected!");

    // Определение функции
    Function.Init(data.N);

    // Генерация сетки
    Generate generator = new Generate(data, Path.GetDirectoryName(args[1])!);
    Grid grid = generator.generate();

    // Трансформация сетки (под синус и косинус) //: Для удобства
    grid = generator.transformation(grid);

    // Решение задачи
    Solve task = new Solve(grid, Path.GetDirectoryName(args[1])!);
    switch(args[2])
    {
        case "--los"     : task.solve(Method.LOS    ); break;
        case "--lu"      : task.solve(Method.LU     ); break;
        case "--bcgstab" : task.solve(Method.BCGSTAB); break;
        default          : task.solve(Method.LOS    ); break;
    };
}
catch (FileNotFoundException ex) {
    WriteLine(ex.Message);
}
catch (ArgumentException ex) {
    ShowHelp();
    WriteLine(ex.Message);
}
```

Solve.cs

```
using System.Diagnostics;
namespace Harmonic;

public class Solve
{
    protected SLAU slau;                                    /// Структура СЛАУ
    protected Grid grid;                                    /// Сетка (узлы, КЭ, краевые)

    private string Path;                                    /// Путь к задаче

    private Stopwatch time;                                 /// Для замера времени

    //: Табличка с окончательным решением
    StringBuilder table_sol;                                /// Табличка с окончательным решением
    StringBuilder table_rate;                               /// Табличка с относительной погрешностью
    //: ------------------------------------------------------------

    //* Конструктор
    public Solve(Grid grid, string path) {
```

```
        this.grid = grid;
        this.Path = path;

        time = new Stopwatch();
    }

    //* Основной метод решения
    public void solve(Method method) {
        portrait();           //? Составление портрета матрицы
        global();             //? Составление глобальной матрицы

        // Новый путь к решению
        Path = method switch
        {
            Method.LOS     => Path + "/output_LOS",
            Method.LU      => Path + "/output_LU",
            Method.BCGSTAB => Path + "/output_BCGSTAB"
        };
        Directory.CreateDirectory(Path);  //? Создание директории с решением

        IMethods issue = method switch    //? Определение метода
        {
            Method.LOS     => new LOS(slau, 1e-16, 10000),
            Method.LU      => new LU(slau),
            Method.BCGSTAB => new BCG(slau, 1e-16, 30000),
            _ => new LOS(slau, 1e-16, 10000)
        };
        issue.Path = this.Path;            //? Установление пути
        time.Start();                      //? Запуск таймера
        slau.q = issue.solve(true);        //? Решение (true - записать итерации)
        time.Stop();                       //? Остановка таймера
        output();                          //? Запись в файл решение
    }

    //* Составление портрета матрицы (ig, jg, выделение памяти)
    private void portrait() {
        Portrait port = new Portrait(grid.Count_Node);

        // Генерируем массивы ig и jg и размерность
        slau.N_el = port.GenPortrait(ref slau.ig, ref slau.jg, grid.Elems);
        slau.N    = grid.Count_Node;

        // Выделяем память
        slau.gl = new Vector(slau.N_el);
        slau.gu = new Vector(slau.N_el);
        slau.di = new Vector(slau.N);
        slau.f  = new Vector(slau.N);
        slau.q  = new Vector(slau.N);
    }

    //* Составление глобальной матрицы
    private void global() {

        // Для каждого конечного элемента
        for (int index_fin_el = 0; index_fin_el < grid.Count_Elem; index_fin_el++) {
            // Составление локальной матрицы
            (double[][] local_matrix, Vector local_f) = local(index_fin_el);

            // Добавление в глобальную
            EntryGlobalMatrix(local_matrix, local_f, index_fin_el);
        }

        // Учет только первых краевых
        for (int index_kraev = 0; index_kraev < grid.Count_Kraev; index_kraev++)
            FirstKraev(index_kraev);
    }

    //* Составление локальной матрицы и локального вектора
```

```
private (double[][], Vector) local(int index) {

    // Локальные функции
    int mu(int i) => i % 2;
    int v (int i) => (i/4) % 2;
    int up(int i) => (i/2) % 2;

    double[][] mat = new double[16][];
    mat = mat.Select(n => new double[16]).ToArray();
    Vector vec = new Vector(16);

    // Подсчет lambda, sigma, hi
    double lambda = 0, sigma = 0, hi = 0;
    Vector f_sin = new Vector(8);
    Vector f_cos = new Vector(8);
    for (int i = 0; i < 8; i++) {
        double x = grid.Nodes[grid.Elems[index].Node[2*i]].x;
        double y = grid.Nodes[grid.Elems[index].Node[2*i]].y;
        double z = grid.Nodes[grid.Elems[index].Node[2*i]].z;
        var point = new Vector(new double[]{x, y, z});
        lambda += Lambda(point);
        sigma  += Sigma (point);
        hi     += Hi    (point);

        // Подсчет f
        f_sin[i] = Fs(point);
        f_cos[i] = Fc(point);
    }
    lambda /= 8.0; sigma /= 8.0; hi /= 8.0;

    // Подсчет hx, hy, hz
    double hx = Abs(grid.Nodes[grid.Elems[index].Node[2]].x - grid.Nodes[grid.Elems[index].Node[0]].x);
    double hy = Abs(grid.Nodes[grid.Elems[index].Node[4]].y - grid.Nodes[grid.Elems[index].Node[0]].y);
    double hz = Abs(grid.Nodes[grid.Elems[index].Node[8]].z - grid.Nodes[grid.Elems[index].Node[0]].z);

    // Построение матрицы жесткости и массы
    double[,] G1 = new double[2,2] {{1, -1}, {-1, 1} };
    double[,] M1 = new double[2,2] {{1/3.0, 1/6.0}, {1/6.0, 1/3.0} };

    double[][] G = new double[8][];
    G = G.Select(n => new double[8]).ToArray();
    double[][] M = new double[8][];
    M = M.Select(n => new double[8]).ToArray();

    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++) {
            G[i][j]  = (hy*hz/hx) * G1[mu(i), mu(j)] * M1[v(i), v(j)] * M1[up(i), up(j)];
            G[i][j] += (hx*hz/hy) * M1[mu(i), mu(j)] * G1[v(i), v(j)] * M1[up(i), up(j)];
            G[i][j] += (hy*hx/hz) * M1[mu(i), mu(j)] * M1[v(i), v(j)] * G1[up(i), up(j)];
            G[i][j] *= lambda;
            M[i][j]  = hx*hy*hz * M1[mu(i), mu(j)] * M1[v(i), v(j)] * M1[up(i), up(j)];
        }

    // Построение локальной матрицы
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++) {
            mat[2*i + 1][2*j + 1] = mat[2*i][2*j] = G[i][j] - omega*omega*hi*M[i][j];
            mat[  2*i  ][2*j + 1] = -omega*sigma*M[i][j];
            mat[2*i + 1][  2*j  ] = omega*sigma*M[i][j];
        }

    // Подсчет правой части
    Vector b_sin = new Vector(8);
    Vector b_cos = new Vector(8);
    for (int i = 0; i < 8; i++)
        for (int j = 0; j < 8; j++) {
            b_sin[i] += M[i][j]*f_sin[j];
            b_cos[i] += M[i][j]*f_cos[j];
```

```
        }

    // Построение правой части
    for (int i = 0; i < 8; i++)
        (vec[2*i], vec[2*i + 1]) = (b_sin[i], b_cos[i]);

    return (mat, vec);
}

//* Поиск столбца
private int find(int f, int s) {
    int col = 0;
    for (int i = slau.ig[f]; i < slau.ig[f + 1]; i++) {
        if (slau.jg[i] == s) {
            col = i;
            break;
        }
    }
    return col;
}

//* Занесение матрицы и вектора в глобальную
private void EntryGlobalMatrix(double[][] mat, Vector vec, int index) {
    for (int i = 0; i < 16; i++) {
        int row = grid.Elems[index].Node[i];
        for (int j = 0; j < i; j++) {
            if (row > grid.Elems[index].Node[j]) {
                int col = find(row, grid.Elems[index].Node[j]);
                slau.gl[col] += mat[i][j];
                slau.gu[col] += mat[j][i];
            }
            else {
                int col = find(grid.Elems[index].Node[j], row);
                slau.gu[col] += mat[i][j];
                slau.gl[col] += mat[j][i];
            }
        }
        slau.di[row] += mat[i][i];
        slau.f[row]  += vec[i];
    }
}

//* Учет первых краевых
private void FirstKraev(int index) {
    (int row, double value) = (grid.Kraevs[index].Node[0], grid.Kraevs[index].Value);

    // Стави значения
    slau.di[row] = 1;
    slau.f [row] = value;

    // Зануляем в треугольнике (столбца)
    for (int i = slau.ig[row]; i < slau.ig[row + 1]; i++) {
        slau.f[slau.jg[i]] -= slau.gu[i]*value;
        slau.gl[i] = 0;
        slau.gu[i] = 0;
    }

    // Зануляем в треугольнике (строки)
    for (int i = row + 1; i < slau.N; i++) {
        for (int j = slau.ig[i]; j < slau.ig[i + 1]; j++) {
            if (slau.jg[j] == row) {
                slau.f[i] -= slau.gl[j]*value;
                slau.gl[j] = 0;
                slau.gu[j] = 0;
            }
        }
    }
}
```

```
    //* Запись решения
    private void output() {
        //: Табличка с окончательным решением
        string margin = String.Join("", Enumerable.Repeat("-", 16));

        table_sol = new StringBuilder($"|U{" ",-14} | U`{" ",-12} | |U`- U| {" ",-7}|\n");
        table_sol.Append($"|" + margin + "|" + margin + "|" + margin + "|\n");


        for (int i = 0; i < slau.N; i++) {
            Vector node = new Vector(new double[] { grid.Nodes[i].x, grid.Nodes[i].y, grid.Nodes[i].z });
            if (i % 2 == 0) {
                table_sol.Append($"|{String.Format("{0,16}", slau.q[i].ToString("E6"))}"  +
                                 $"|{String.Format("{0,16}", Us(node).ToString("E6"))}"   +
                                 $"|{String.Format("{0,16}", Abs(Us(node) - slau.q[i]).ToString("E6"))}|\n");
            } else {
                table_sol.Append($"|{String.Format("{0,16}", slau.q[i].ToString("E6"))}"  +
                                 $"|{String.Format("{0,16}", Uc(node).ToString("E6"))}"   +
                                 $"|{String.Format("{0,16}", Abs(Uc(node) - slau.q[i]).ToString("E6"))}|\n");
            }
        }

        table_sol.Append(String.Join("", Enumerable.Repeat("-", 52)) + "\n");

        File.WriteAllText(Path + "/table_sol.txt", table_sol.ToString());
        //: ------------------------------------------------------------

        //: Табличка с погрешностью
        table_rate = new StringBuilder();

        double diffetence_sin, difference_cos, rate_u_sin, rate_u_cos;
        difference_cos = diffetence_sin = rate_u_cos = rate_u_sin = 0;

        for (int i = 0; i < grid.Count_Node / 2; i++) {
            Vector node = new Vector(new double[] { grid.Nodes[2*i].x, grid.Nodes[2*i].y, grid.Nodes[2*i].z });
            double us = Us(node);
            double uc = Uc(node);
            diffetence_sin += Pow((us - slau.q[2*i]), 2);
            difference_cos += Pow((uc - slau.q[2*i + 1]), 2);
            rate_u_sin     += Pow(us, 2);
            rate_u_cos     += Pow(uc, 2);
        }

        table_rate.Append($"Общее:   {Sqrt((diffetence_sin + difference_cos) / (rate_u_sin + rate_u_cos)).ToString("E6")}\n
        table_rate.Append($"Синус:   {Sqrt(diffetence_sin / rate_u_sin).ToString("E6")}\n");
        table_rate.Append($"Косинус: {Sqrt(difference_cos / rate_u_cos).ToString("E6")}\n");

        TimeSpan ts = time.Elapsed;
        string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",
            ts.Hours, ts.Minutes, ts.Seconds,
            ts.Milliseconds / 10);
        table_rate.Append($"Время: {elapsedTime}\n");

        File.WriteAllText(Path + "/table_rate.txt", table_rate.ToString());
        //: ------------------------------------------------------------
    }
}
```

LUMethods.cs

```
namespace Harmonic.methods;
public abstract class LUMethods
{
    private SLAU slau;
```

```
protected Vector lgl, lgu, ldi;
protected int N, N_el;

//* Инициализация
protected void Init(SLAU slau) {
    this.slau = slau;
    this.N    = slau.N;
    this.N_el = slau.N_el;
    lgl       = new Vector(N_el);
    lgu       = new Vector(N_el);
    ldi       = new Vector(N);
    Vector.Copy(slau.gl, lgl);
    Vector.Copy(slau.gu, lgu);
    Vector.Copy(slau.di, ldi);
}

//* Инициализация Profile
protected void Init_profile(SLAU slau) {
    this.N    = slau.N;

    int[] newig = new int[this.N + 1];
    for (int i = 1; i <= this.N; i++) {
        if ((slau.ig[i] - slau.ig[i - 1]) > 0)
            newig[i] = newig[i - 1] + (i - slau.jg[slau.ig[i - 1]]);
        else
            newig[i] = newig[i - 1];
    }

    slau.N_el = newig[this.N];
    this.N_el = slau.N_el;

    Vector newgl = new Vector(this.N_el);
    Vector newgu = new Vector(this.N_el);

    for (int i = 0; i < this.N; i++) {
        int col = i - (newig[i + 1] - newig[i]);
        int p = slau.ig[i];
        for(int j = newig[i]; j < newig[i + 1]; j++, col++){
            if(col == slau.jg[p]){
                newgu[j] = slau.gu[p];
                newgl[j] = slau.gl[p];
                p++;
            }
            else
                newgu[j] = newgl[j] = 0;
        }
    }

    slau.ig = new int[this.N + 1];
    slau.gl = new Vector(this.N_el);
    slau.gu = new Vector(this.N_el);
    Array.Copy(newig, slau.ig, newig.Length);
    Vector.Copy(newgl, slau.gl);
    Vector.Copy(newgu, slau.gu);

    this.slau = slau;
}

//* lU-разложение
protected void LU_decomposition() {
    double sumDI, sumGL, sumGU;
    for (int i = 0; i < N; i++) {
        sumDI = 0;
        for (int j = slau.ig[i]; j < slau.ig[i + 1]; j++) {
            sumGL = sumGU = 0;
            int p_s = slau.ig[slau.jg[j]], p_s1 = slau.ig[slau.jg[j] + 1];
            for (int k = slau.ig[i]; k < j; k++)
                for (int p = p_s; p < p_s1; p++)
```

```
                    if (slau.jg[k] == slau.jg[p]) {
                        sumGL += lgl[k]*lgu[p];
                        sumGU += lgl[p]*lgu[k];
                        p_s++;
                    }
            lgl[j] -= sumGL;
            lgu[j]  = (lgu[j] - sumGU) / ldi[slau.jg[j]];
            sumDI  += lgl[j]*lgu[j];
        }
        ldi[i] -= sumDI;
    }
}

//* Решение СЛАУ L^(-1)
protected Vector Solve_L(Vector f) {
    Vector x = new Vector(N);
    double sum;
    for (int i = 0; i < N; i++) {
        sum = 0;
        for (int j = slau.ig[i]; j < slau.ig[i + 1]; j++)
            sum += lgl[j]*x[slau.jg[j]];
        x[i] = (f[i] - sum) / ldi[i];
    }
    return x;
}

//* Решение СЛАУ U^(-1)
protected Vector Solve_U(Vector f) {
    Vector x = new Vector(N);
    Vector f_copy = new Vector(N);
    Vector.Copy(f, f_copy);
    for (int i = N - 1; i >= 0; i--) {
        x[i] = f_copy[i] / ldi[i];
        for (int j = slau.ig[i]; j < slau.ig[i + 1]; j++)
            f_copy[slau.jg[j]] -= lgu[j]*x[i];
    }
    return x;
}


//* lU-разложение Profile
protected void LU_decomposition_profile() {
    for(int i = 0; i < slau.N; i++){
        int i0 = slau.ig[i];
        int i1 = slau.ig[i + 1];
        int j = i - (i1 - i0);
        double sd = 0;
        for(int m = i0; m < i1; m++,j++){
            double sl = 0;
            double su = 0;
            int j0 = j < 0 ? -33686019 : slau.ig[j];
            int j1 = slau.ig[j + 1];
            int mi = i0;
            int mj = j0;
            int kol_i = m - i0;
            int kol_j = j1 - j0;
            int kol_r = kol_i - kol_j;
            if(kol_r < 0) mj -= kol_r;
            else mi += kol_r;
            for(; mi<m; mi++, mj++){
                sl += slau.gl[mi]*slau.gu[mj];
                su += slau.gu[mi]*slau.gl[mj];
            }
            slau.gl[m] = slau.gl[m] - sl;
            slau.gu[m] = j < 0 ? 0 : (slau.gu[m] - su) / slau.di[j];
            sd += slau.gl[m]*slau.gu[m];
        }
        slau.di[i] = slau.di[i] - sd;
```

```
        }
    }

    //* Прямой ход
    protected Vector normalRunning(Vector f) {
        Vector y = new Vector(slau.N);
        for (int i = 0; i < slau.N; i++) {
            int i0 = slau.ig[i],
                i1 = slau.ig[i + 1],
                j = i - (i1 - i0);
            double sum = 0;
            for (int k = i0; k < i1; k++, j++)
                sum += j < 0 ? 0 : y[j] * slau.gl[k];
            y[i] = (f[i] - sum) / slau.di[i];
        }
        return y;
    }

    //* Обратный ход
    protected Vector reverseRunning(Vector y) {
        Vector x = new Vector(slau.N);
        for (int i = slau.N - 1; i >= 0; i--) {
            int i0 = slau.ig[i],
                i1 = slau.ig[i + 1],
                j = i - (i1 - i0);
            double xi = y[i];
            for (int k = i0; k < i1; k++, j++) {
                if (j >= 0)
                    y[j] -= xi * slau.gu[k];
            }
            x[i] = xi;
        }
        return x;
    }
}
```

LU.cs

```
namespace Harmonic.methods;
public class LU : LUMethods, IMethods
{
    private SLAU slau;                      /// Матрица

    public string Path { get; set; }        /// Путь к задаче

    public LU(SLAU slau) { this.slau = slau; }

    public Vector solve(bool log = false) {
        Init_profile(slau);                                 //? СЛАУ в профильный формат из разреженного
        LU_decomposition_profile();                         //? LU-разложение
        Vector q = reverseRunning(normalRunning(slau.f));   //? Решение прямой и обратный ход
        return q;
    }

    public string GetName() => "LU";
}
```

LOS.cs

```
namespace Harmonic.methods;
public class LOS : LUMethods, IMethods
{
    private SLAU slau;                      /// Матрица
    protected double EPS;                   /// Точность решения
    protected int maxIter;                  /// Максимальное количество итераций

    public string Path { get; set; }        /// Путь к задаче

    //: Табличка с итерациями и невязкой
    StringBuilder table_iter;               /// Табличка каждой итерации и невязки
    //: ----------------------------------------------------

    public LOS(SLAU slau, double EPS, int maxIter) {
        this.slau    = slau;
        this.EPS     = EPS;
        this.maxIter = maxIter;
    }

    public Vector solve(bool log = false) {
        //: Таблички
        if (log == true)
            table_iter = new StringBuilder($"Iter{" ", 5} Nev{" ", 12}\n");
        //: ----------------------------------------------------

        Vector r   = new Vector(slau.N);
        Vector z   = new Vector(slau.N);
        Vector Az  = new Vector(slau.N);
        Vector LAU = new Vector(slau.N);
        Vector p   = new Vector(slau.N);

        double f_norm = Sqrt(Scalar(slau.f, slau.f));

        // LU-разложение
        Init(slau);
        LU_decomposition();

        r = Solve_L(slau.f - slau.Mull(slau.q));
        z = Solve_U(r);
        p = Solve_L(slau.Mull(z));
```

```
        double alpha, betta, Eps = 0;
        int Iter = 0;
        do {
            betta  = Scalar(p, p);
            alpha  = Scalar(p, r) / betta;
            slau.q = slau.q + alpha * z;
            r      = r - alpha * p;
            LAU    = Solve_L(slau.Mull(Solve_U(r)));
            betta  = -Scalar(p, LAU) / betta;
            z      = Solve_U(r) + betta * z;
            p      = LAU + betta * p;
            Eps    = Sqrt(Scalar(r, r)) / f_norm;

            //: Табличка
            if (log == true) {
                table_iter.Append($"{String.Format("{0,4}", ++Iter)}"  +
                                  $"{String.Format("{0,19}",  Eps.ToString("E6"))}\n");
            }
            //: ------------------------------------------------
        } while(Iter < maxIter &&
                Eps  > EPS);


        //: Запись таблички в файл
        if (log == true)
            File.WriteAllText(Path + "/table_iter.txt", table_iter.ToString());
        //: ------------------------------------------------

        return slau.q;
    }

    public string GetName() => "LOS";
}
```

IMethods.cs

```
namespace Harmonic.methods;
public class BCG : LUMethods, IMethods
{
    private SLAU slau;                      /// Матрица

    private double EPS;                     /// Точность
    private int maxIter;                    /// Максимальное количество итераций

    public string Path { get; set; }        /// Путь к задаче

    //: Табличка с итерациями и невязкой
    StringBuilder table_iter;               /// Табличка каждой итерации и невязки
    //: ------------------------------------------------------

    public BCG(SLAU slau, double EPS, int maxIter) {
        this.slau    = slau;
        this.EPS     = EPS;
        this.maxIter = maxIter;
    }

    public Vector solve(bool log = false) {
        //: Таблички
        if (log == true)
            table_iter = new StringBuilder($"Iter{" ", 5} Nev{" ", 12}\n");
        //: ------------------------------------------------

        Vector r0   = new Vector(slau.N);
        Vector rk   = new Vector(slau.N);
```

```
        Vector z    = new Vector(slau.N);
        Vector LAUz = new Vector(slau.N);
        Vector p    = new Vector(slau.N);
        Vector LAUp = new Vector(slau.N);
        Vector r_p  = new Vector(slau.N);

        double f_norm = Sqrt(Scalar(slau.f, slau.f));

        // LU-разложение
        Init(slau);
        LU_decomposition();

        r0 = Solve_L(slau.f - slau.Mull(slau.q));
        Vector.Copy(r0, rk);
        Vector.Copy(r0, z);

        int Iter = 0;
        double Eps = 0, alpha, gamma, betta;
        do {
            betta  = Scalar(rk, r0);
            LAUz   = Solve_L(slau.Mull(Solve_U(z)));
            alpha  = betta / Scalar(r0, LAUz);
            p      = rk - alpha*LAUz;
            LAUp   = Solve_L(slau.Mull(Solve_U(p)));
            gamma  = Scalar(p, LAUp) / Scalar(LAUp, LAUp);
            slau.q = slau.q + alpha*z + gamma*p;
            Vector.Copy(rk, r_p);
            rk     = p - gamma*LAUp;
            betta  = (alpha*Scalar(rk, r0)) / (gamma*betta);
            z      = rk + betta*r_p - betta*gamma*LAUz;
            Eps    = Sqrt(Scalar(rk, rk)) / f_norm;

            //: Табличка
            if (log == true) {
                table_iter.Append($"{String.Format("{0,4}", ++Iter)}"  +
                                  $"{String.Format("{0,19}",  Eps.ToString("E6"))}\n");
            }
            //: ------------------------------------------------
        } while (Iter < maxIter &&
                 Eps  > EPS);

        //: Запись таблички в файл
        if (log == true)
            File.WriteAllText(Path + "/table_iter.txt", table_iter.ToString());
        //: ------------------------------------------------
        slau.q = Solve_U(slau.q);
        return slau.q;
    }

    public string GetName() => "BCGSTAB";
}
```

Iteration.cs

```
namespace Harmonic.methods;
public class BCG : LUMethods, IMethods
{
    private SLAU slau;                      /// Матрица

    private double EPS;                     /// Точность
    private int maxIter;                    /// Максимальное количество итераций

    public string Path { get; set; }        /// Путь к задаче

    //: Табличка с итерациями и невязкой
```

```
    StringBuilder table_iter;               /// Табличка каждой итерации и невязки
    //: ----------------------------------------------------

    public BCG(SLAU slau, double EPS, int maxIter) {
        this.slau    = slau;
        this.EPS     = EPS;
        this.maxIter = maxIter;
    }

    public Vector solve(bool log = false) {
        //: Таблички
        if (log == true)
            table_iter = new StringBuilder($"Iter{" ", 5} Nev{" ", 12}\n");
        //: ----------------------------------------------------

        Vector r0   = new Vector(slau.N);
        Vector rk   = new Vector(slau.N);
        Vector z    = new Vector(slau.N);
        Vector LAUz = new Vector(slau.N);
        Vector p    = new Vector(slau.N);
        Vector LAUp = new Vector(slau.N);
        Vector r_p  = new Vector(slau.N);

        double f_norm = Sqrt(Scalar(slau.f, slau.f));

        // LU-разложение
        Init(slau);
        LU_decomposition();

        r0 = Solve_L(slau.f - slau.Mull(slau.q));
        Vector.Copy(r0, rk);
        Vector.Copy(r0, z);

        int Iter = 0;
        double Eps = 0, alpha, gamma, betta;
        do {
            betta  = Scalar(rk, r0);
            LAUz   = Solve_L(slau.Mull(Solve_U(z)));
            alpha  = betta / Scalar(r0, LAUz);
            p      = rk - alpha*LAUz;
            LAUp   = Solve_L(slau.Mull(Solve_U(p)));
            gamma  = Scalar(p, LAUp) / Scalar(LAUp, LAUp);
            slau.q = slau.q + alpha*z + gamma*p;
            Vector.Copy(rk, r_p);
            rk     = p - gamma*LAUp;
            betta  = (alpha*Scalar(rk, r0)) / (gamma*betta);
            z      = rk + betta*r_p - betta*gamma*LAUz;
            Eps    = Sqrt(Scalar(rk, rk)) / f_norm;

            //: Табличка
            if (log == true) {
                table_iter.Append($"{String.Format("{0,4}", ++Iter)}"  +
                                  $"{String.Format("{0,19}",  Eps.ToString("E6"))}\n");
            }
            //: ----------------------------------------------------
        } while (Iter < maxIter &&
                 Eps  > EPS);

        //: Запись таблички в файл
        if (log == true)
            File.WriteAllText(Path + "/table_iter.txt", table_iter.ToString());
        //: ----------------------------------------------------
        slau.q = Solve_U(slau.q);
        return slau.q;
    }

    public string GetName() => "BCGSTAB";
}
```