```
using System;
using System.Collections.Generic;
using System.Text;

using SMath.Manager;
using SMath.Math;
using SMath.Math.Numeric;

namespace ODESolvers {
    public class Class1: IPluginHandleEvaluation, IPluginLowLevelEvaluation,
    IPluginMathNumericEvaluation {
        TermInfo[] ИнформацияЭлементов;
        AssemblyInfo[] ИнформацияСборки;

        #region IPluginHandleEvaluation Members

        TermInfo[] IPluginHandleEvaluation.TermsHandled {
            get { return this.ИнформацияЭлементов; }
        }

        #endregion

        #region IPlugin Members

        AssemblyInfo[] IPlugin.Dependences {
            get { return this.ИнформацияСборки; }
        }

        void IPlugin.Initialize() {
            this.ИнформацияСборки = new AssemblyInfo[] {
                // Plugin requires to be installed with SMath Studio 0.87 or later version
                new AssemblyInfo( "SMath Studio", new Version( 0, 89 ), new Guid( "a37cba83-
    b69c-4c71-9992-55ff666763bd" ) )
            };

            this.ИнформацияЭлементов = new TermInfo[] {
            new TermInfo(
                    "rkfixed",
                    TermType.Function,
                    "(init, x1, x2, npoints, D) Uses the fourth-order Runge-Kutta fixed-step
     method.",
                    FunctionSections.Unknown,
                    true,
                    new ArgumentInfo[] {
                        new ArgumentInfo( ArgumentSections.ColumnVector ),
                        new ArgumentInfo( ArgumentSections.RealNumber ),
                        new ArgumentInfo( ArgumentSections.RealNumber ),
                        new ArgumentInfo( ArgumentSections.RealNumber ),
                        new ArgumentInfo( ArgumentSections.Function ),
                        }
                    )
            };
        }

        #endregion

        #region IDisposable Members

        void IDisposable.Dispose() {
            // throw new NotImplementedException();
        }

        #endregion

        #region IPluginLowLevelEvaluation Members

        // Вспомогательный образ для работы функции вычисления ДиффСистемы
        public class ОбразКонтекста {
            public Store context;
            public Term[][] args;
            public Term[] system;

            public ОбразКонтекста( Term[][] args, Term[] system, ref Store context ) {
```

```
            this.args = args;
            this.context = context;
            this.system = system;
        }
    }

    // Вычисляем текущее значение элементов ДиффСистемы
    public void ДиффСистема( double t, double[] x, double[] dx, ОбразКонтекста obj ) {
        List<Term> Вых = new List<Term>();

        Term[][] args = obj.args;
        Store context = obj.context;
        Term[] система = obj.system;

        // обозначение первой переменой
        Term перем_t = args[4][0];
        // обозначение второй переменной
        Term перем_x = args[4][1];
        // el()
        Term Элемент = new Term( Functions.El, TermType.Function, 2 );

        // Переводим значения функции в термины SMath
        TMatrix vx = new TMatrix( new TNumber[ x.GetLength(0), 1 ] );
        TNumber nt = new TNumber(t);

        for ( int k = 0; k < x.GetLength(0); k++ ) {
            vx.unit.SetValue( ( TNumber ) x[k], new long[] { k, 0 } );
        }

        int n = 0;
        int m;
        int Размер = система.Length;

        while ( n < Размер ) {
            // Подставляем значение переменной 'x'
            if ( система[n].Equals( перем_x ) && система[ n + 2 ].Equals( Элемент ) ) {
                m = ( int ) SMath.Math.Numeric.Expression.Calculate( new Term[] {
система[ n + 1 ] }, context ).obj.ToDouble();
                Вых.AddRange( vx.unit[ m - 1, 0 ].obj.ToTerms() );
                n += 3;
            }
            // На случай, если с переменной 'x' работают как с вектором
            else if ( система[n].Equals( перем_x ) ) {
                Вых.AddRange( vx.ToTerms() );
                n += 1;
            }
            // Подставляем значение переменной 't'
            else if ( система[n].Equals( перем_t ) ) {
                Вых.AddRange( nt.obj.ToTerms() );
                n += 1;
            }
            // Подставляем исходные элементы
            else {
                Вых.Add( система[n] );
                n += 1;
            }
        }

        // Вычисляем получившийся вектор
        TMatrix вых = SMath.Math.Numeric.Expression.Calculate( Вых.ToArray(), context );

        // Переписываем ответ в терминах double[]
        int r = x.GetLength(0);
        for ( int ii = 0; ii < r; ii++ ) {
            dx[ii] = вых.unit[ ii, 0 ].obj.ToDouble();
        }
    }

    bool IPluginLowLevelEvaluation.ExpressionEvaluation( Term root, Term[][] args, ref
Store context, ref Term[] result ) {
        if ( root.Type == TermType.Function && root.Text == "rkfixed" && root.ChildCount
 == 5 ) {
            Term[]
```

```
                arg1 = Decision.Preprocessing( args[0], ref context ),
                arg2 = Decision.Preprocessing( args[1], ref context ),
                arg3 = Decision.Preprocessing( args[2], ref context ),
                arg4 = Decision.Preprocessing( args[3], ref context ),
                arg5 = Decision.Preprocessing( args[4], ref context );

            // Шаг 1. Узнаём размер вектора начальных условий
            TMatrix НачУсл = SMath.Math.Numeric.Expression.Calculate( arg1, context );
            int КолвоНачУсл = НачУсл.unit.GetLength(0);

            // Шаг 2. Узнаём размер вектора системы
            //TMatrix ВектСистемы = ( TMatrix ) SMath.Math.Numeric.Expression.Calculate(↙
 new Term[] { arg5[ arg5.Length - 3 ] }, context );
            //int КолвоУравн = ВектСистемы.unit.GetLength(0);
            int КолвоУравн = ( int ) SMath.Math.Numeric.Expression.Calculate( new Term[]↙
 { arg5[ arg5.Length - 3 ] }, context ).obj.ToDouble();

            // TODO: Если КолвоНачУсл != КолвоУравн, то ошибка

            // Шаг 3. Узнаём начало интервала
            double НачИнт = SMath.Math.Numeric.Expression.Calculate( arg2, context ).obj↙
.ToDouble();

            // Шаг 4. Узнаём конец интервала
            double КонИнт = SMath.Math.Numeric.Expression.Calculate( arg3, context ).obj↙
.ToDouble();

            // Шаг 5. Узнаём количество шагов
            double КолвоШагов = SMath.Math.Numeric.Expression.Calculate( arg4, context )↙
.obj.ToDouble();

            // Шаг 6. Находим величину шага разбиения
            double Дельта = ( КонИнт - НачИнт ) / КолвоШагов;

            // Шаг 7. Задаем размер матрицы ответов с незав. перем.
            TMatrix Решение = new TMatrix( new TNumber[ ( int ) КолвоШагов + 1,          ↙
КолвоУравн + 1 ] );

            // число переменных
            double[] Перем = new double [ КолвоУравн ];

            // число переменных для 2-го коэф.
            double[] Перем2 = new double [ КолвоУравн ];

            // число переменных для 3-го коэф.
            double[] Перем3 = new double [ КолвоУравн ];

            // число переменных для 4-го коэф.
            double[] Перем4 = new double [ КолвоУравн ];

            // число 1-ых коэф. метода по числу уравнений
            double[] Коэфф1 = new double [ КолвоУравн ];

            // число 2-ых коэф. метода по числу уравнений
            double[] Коэфф2 = new double [ КолвоУравн ];

            // число 3-ых коэф. метода по числу уравнений
            double[] Коэфф3 = new double [ КолвоУравн ];

            // число 4-ых коэф. метода по числу уравнений
            double[] Коэфф4 = new double [ КолвоУравн ];

            // Начальные значения переменных:
            double t = НачИнт;

            for ( int ii = 0; ii < КолвоНачУсл; ii++ ) {
                Перем[ii] = НачУсл.unit[ ii, 0 ].obj.ToDouble();
            }

            // первая точка результата
            Решение.unit.SetValue( ( TNumber ) t, new long[] { 0, 0 } );

            for ( int ii = 1; ii <= КолвоНачУсл; ii++ ) {
```

```
                    Решение.unit.SetValue( ( TNumber ) Перем[ ii - 1 ], new long[] { 0, ii } );
                }

                double [] врм = new double [ КолвоУравн ];
                ОбразКонтекста Контекст = new ОбразКонтекста( args, арг5, ref context );

                // начало цикла итераций
                for ( int ii = 0; ii < ( int ) КолвоШагов; ii++ ) {
                    // 1-й коэфф.
                    ДиффСистема( t, Перем, врм, Контекст );
                    for ( int jj = 0; jj < КолвоУравн; jj++ ) {
                        Коэфф1[jj] = врм[jj] * Дельта;
                        // Находим значения переменных для второго коэф.
                        Перем2[jj] = Перем[jj] + Коэфф1[jj] / 2;
                    }

                    // 2-й коэф.
                    ДиффСистема( t, Перем2, врм, Контекст );
                    for ( int jj = 0; jj < КолвоУравн; jj++ ) {
                        Коэфф2[jj] = врм[jj] * Дельта;
                        // Находим значения переменных для третьго коэф.
                        Перем3[jj] = Перем[jj] + Коэфф2[jj] / 2;
                    }

                    // 3 коэфф.
                    ДиффСистема( t, Перем3, врм, Контекст );
                    for ( int jj = 0; jj < КолвоУравн; jj++ ) {
                        Коэфф3[jj] = врм[jj] * Дельта;
                        // Находим значения переменных для 4 коэф.
                        Перем4[jj] = Перем[jj] + Коэфф3[jj];
                    }

                    // 4 коэфф.
                    ДиффСистема( t, Перем4, врм, Контекст );
                    for ( int jj = 0; jj < КолвоУравн; jj++ ) {
                        Коэфф4[jj] = врм[jj] * Дельта;
                    }

                    // Находим новые значения переменных включая независимую
                    t += Дельта;

                    for ( int k = 0; k < КолвоУравн; k++ ) {
                        Перем[k] = Перем[k] + ( 1.0/6.0 ) * ( Коэфф1[k] + 2 * ( Коэфф2[k] + Коэфф3[k] ) + Коэфф4[k] );
                    }

                    // Результат  итерации:
                    Решение.unit.SetValue( ( TNumber ) t, new long[] { ii + 1, 0 } );

                    for ( int jj = 1; jj <= КолвоНачУсл; jj++ ) {
                        Решение.unit.SetValue( ( TNumber ) Перем[ jj - 1 ], new long[] { ii + 1, jj } );
                    }
                }

                result = Решение.ToTerms();

                return true;
            }

            return true;
        }

        #endregion

        #region IPluginMathNumericEvaluation Members

        bool IPluginMathNumericEvaluation.NumericEvaluation( Term value, TNumber[] args, ref TNumber result ) {
            return false;
        }
```

```
        #endregion
    }
}
```