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

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

namespace Plot3D {
    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(
                "CreateMesh",
                TermType.Function,
                "( function(s), u0, u1, v0, v1, Nu, Nv ) Returns an array representing the x
    , y, and z-coordinates of a parametric surface defined by the function of two variables
    in the first argument.",
                FunctionSections.Unknown,
                true,
                new ArgumentInfo[] {
                    new ArgumentInfo( ArgumentSections.Function ),
                    new ArgumentInfo( ArgumentSections.RealNumber ),
                    new ArgumentInfo( ArgumentSections.RealNumber ),
                    new ArgumentInfo( ArgumentSections.RealNumber ),
                    new ArgumentInfo( ArgumentSections.RealNumber ),
                    new ArgumentInfo( ArgumentSections.RealNumber ),
                    new ArgumentInfo( ArgumentSections.RealNumber ),
                    }
                )
            };
        }

        #endregion

        #region IDisposable Members

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

        #endregion

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

```
        public Term var_u;
        public Term var_v;

        public ОбразКонтекста( Term var_u, Term var_v, Term[] system, ref Store context ) {
            this.context = context;
            this.system = system;

            this.var_u = var_u;
            this.var_v = var_v;
        }
    }

    // Вычисляем текущее значение элементов вектора с выражениями
    public void Функции( double u, double v, double[] vecf, ОбразКонтекста Контекст ) {

        Store context = Контекст.context; // контекст документа
        Term[] Список = Контекст.system; // вектор выражений
        Term перем_u = Контекст.var_u; // обозначение первой переменной
        Term перем_v = Контекст.var_v; // обозначение второй переменной

        // Переводим значения переменных во внутреннее представление SMath
        TNumber m_u = new TNumber(u);
        TNumber m_v = new TNumber(v);

        int n = 0;
        int Размер = Список.Length;
        List<Term> ПреобрСписок = new List<Term>();

        while ( n < Размер ) {
            // Подставляем значение переменной 'u'
            if ( Список[n].Equals( перем_u ) ) {
                ПреобрСписок.AddRange( m_u.obj.ToTerms() );
                n += 1;
            }
            // Подставляем значение переменной 'v'
            else if ( Список[n].Equals( перем_v ) ) {
                ПреобрСписок.AddRange( m_v.obj.ToTerms() );
                n += 1;
            }
            // Подставляем исходные элементы
            else {
                ПреобрСписок.Add( Список[n] );
                n += 1;
            }
        }

        // Пересборка выражения перед вычислением
        // Эту операцию необходимо выполнять, т.к. не все функции определены
        // в численной и/или символьной библиотеке (например: if())
        Term[] Выражение = Decision.Preprocessing( ПреобрСписок.ToArray(), ref context );

        // Вычисляем получившийся вектор
        TMatrix Результат = SMath.Math.Numeric.Expression.Calculate( Выражение, context );

        // Переписываем результат вычисления в массив double[]
        for ( int ii = 0; ii < 3; ii++ ) {
            vecf[ii] = Результат.unit[ ii, 0 ].obj.ToDouble();
        }
    }

    bool IPluginLowLevelEvaluation.ExpressionEvaluation( Term root, Term[][] args, ref Store context, ref Term[] result ) {
        // Если не функция, то прекращаем дальнейший разбор
        if ( root.Type != TermType.Function ) {
            return false;
        }

        else if ( root.Text == "CreateMesh" && root.ChildCount == 7 ) {
            // Делаем предварительный разбор выражений
            // арг* - преобразованные параметры,
```

```
            // args[*] - оригиналы.
            // Дело в том, что args[] содержит передаваемую функцию в виде левой части, т.е. f(x,y),
            // а в арг после разбора запишется уже правая часть, т.е. само "тело" функции f(x,y)
            Term[]
                арг1 = Decision.Preprocessing( args[0], ref context ),
                арг2 = Decision.Preprocessing( args[1], ref context ),
                арг3 = Decision.Preprocessing( args[2], ref context ),
                арг4 = Decision.Preprocessing( args[3], ref context ),
                арг5 = Decision.Preprocessing( args[4], ref context ),
                арг6 = Decision.Preprocessing( args[5], ref context ),
                арг7 = Decision.Preprocessing( args[6], ref context );

            // Шаг 1. Узнаём начальное значение параметра 'u'
            TNumber Число = SMath.Math.Numeric.Expression.Calculate( арг2, context );
            double u0 = Число.obj.ToDouble();

            // Шаг 2. Узнаём конечное значение параметра 'u'
            Число = SMath.Math.Numeric.Expression.Calculate( арг3, context );
            double u1 = Число.obj.ToDouble();

            // Шаг 3. Узнаём начальное значение параметра 'v'
            Число = SMath.Math.Numeric.Expression.Calculate( арг4, context );
            double v0 = Число.obj.ToDouble();

            // Шаг 4. Узнаём конечное значение параметра 'v'
            Число = SMath.Math.Numeric.Expression.Calculate( арг5, context );
            double v1 = Число.obj.ToDouble();

            // Шаг 5. Узнаём количество шагов по параметру 'u'
            Число = SMath.Math.Numeric.Expression.Calculate( арг6, context );
            int Nu = ( int ) Число.obj.ToDouble();

            // Шаг 6. Узнаём количество шагов по параметру 'v'
            Число = SMath.Math.Numeric.Expression.Calculate( арг7, context );
            int Nv = ( int ) Число.obj.ToDouble();

            // Шаг 7. Находим дельту по параметру 'u'
            double du = ( u1 - u0 ) / Nu;

            // Шаг 8. Находим дельту по параметру 'v'
            double dv = ( v1 - v0 ) / Nv;

            int Np = 5;

            // Шаг 9. Задаем размер матрицы решений
            TMatrix Решение = new TMatrix( new TNumber[ Nu * ( Np * Nv + Nv - 1 ), 3 ] );
            double[,] Массив = new double[ Nu * ( Np * Nv + Nv - 1 ), 3 ];

            // Создаём буфер для текущих координат x, y и z
            double [] xyz = new double [3];
            ОбразКонтекста Контекст = new ОбразКонтекста( args[0][0], args[0][1], арг1, ref context );

            int ii, jj, n;
            double u, v;

            for ( ii = 0; ii < Nu; ii++ ) {
                u = ii * du + u0;

                for ( jj = 0; jj < Nv; jj++ ) {
                    v = jj * dv + v0;
                    n = ii * ( Np * Nv + Nv - 1 ) + Np * jj;

                    // Точка 1
                    Функции( u, v, xyz, Контекст );
                    Массив[ n, 0 ] = xyz[0];
                    Массив[ n, 1 ] = xyz[1];
                    Массив[ n, 2 ] = xyz[2];

                    // Точка 2
```

```
                        Функции( u, v + dv, xyz, Контекст );
                        Массив[ n + 1, 0 ] = xyz[0];
                        Массив[ n + 1, 1 ] = xyz[1];
                        Массив[ n + 1, 2 ] = xyz[2];

                        // Точка 3
                        Функции( u + du, v + dv, xyz, Контекст );
                        Массив[ n + 2, 0 ] = xyz[0];
                        Массив[ n + 2, 1 ] = xyz[1];
                        Массив[ n + 2, 2 ] = xyz[2];

                        // Точка 4
                        Функции( u + du, v, xyz, Контекст );
                        Массив[ n + 3, 0 ] = xyz[0];
                        Массив[ n + 3, 1 ] = xyz[1];
                        Массив[ n + 3, 2 ] = xyz[2];

                        // Точка 5
                        Функции( u, v, xyz, Контекст );
                        Массив[ n + 4, 0 ] = xyz[0];
                        Массив[ n + 4, 1 ] = xyz[1];
                        Массив[ n + 4, 2 ] = xyz[2];
                    }

                    // Возвращаем точку на позицию (чтобы скрыть переходные линии)
                    for ( int k = 0; k < Nv - 1; k++ ) {
                        n = ii * ( Np * Nv + Nv - 1 ) + Np * jj;
                        Массив[ n + k, 0 ] = Массив[ n - ( k + 2 ) * Np, 0 ];
                        Массив[ n + k, 1 ] = Массив[ n - ( k + 2 ) * Np, 1 ];
                        Массив[ n + k, 2 ] = Массив[ n - ( k + 2 ) * Np, 2 ];
                    }
                }

                n = Nu * ( Np * Nv + Nv - 1 );
                for ( ii = 0; ii < n; ii++ ) {
                    Решение.unit.SetValue( ( TNumber ) Массив[ ii, 0 ], new long[] { ii, 0 }↙
    );
                    Решение.unit.SetValue( ( TNumber ) Массив[ ii, 1 ], new long[] { ii, 1 }↙
    );
                    Решение.unit.SetValue( ( TNumber ) Массив[ ii, 2 ], new long[] { ii, 2 }↙
    );
                }

                result = Решение.ToTerms();

                return true;
            }

            return false;
        }

        #region IPluginMathNumericEvaluation Members

        bool IPluginMathNumericEvaluation.NumericEvaluation( Term value, TNumber[] args, ref↙
    TNumber result ) {
            // Если не функция, то прекращаем дальнейший разбор
            if ( value.Type != TermType.Function ) {
                return false;
            }

            else if ( value.Text == "CreateMesh" && value.ChildCount == 7 ) {
                // Пропускаем
                return true;
            }

            return false;
        }

        #endregion
    }
}
```