# Program

V této příloze je uveden zdrojový kód programu spolu s komentáři, které vysvětlují, co jednotlivé části programu dělají. Je tady uveden i návod na zkompilování programu.

Program byl vytvořen jako konzolová aplikace v jazyce C#. Pro vytvoření programu bylo použito vývojové prostředí Microsoft Visual Studio 2015 Community Edition.

## Návod na kompilaci

Nejdřív je potřeba vytvořit nový projekt, kam se zkopíruje hlavní třída programu, která obsahuje „static void Main“. Pro každou další třídu programu se vytvoří nový soubor, kam se zkopíruje kód dané třídy programu. Je možnost zkopírovat všechny třídy i do jednoho souboru. V tom případě se do části „namespace DetectionWLF“ v hlavní třídě programu zkopíruje kód všech tříd nacházející se rovněž v části „namespace DetectionWLF“. Na začátek souboru se musí zkopírovat každý použitý obor názvů[1].

Kromě toho je potřeba stáhnout a přidat použité knihovny. Pro načtení souboru byla použita knihovna CSharpFITS[2]. Tato knihovna se stáhne už zkompilovaná (dll formát) a přidá se k projektu standardním způsobem. Pro proložení polynomem byla rovněž použita knihovna[3]. V tomto případě se dá stáhnout zkompilovaná knihovna i zdrojové kódy. V případě stáhnutí zkompilované knihovny je postup stejný jako pro knihovnu CSharpFITS. Pokud se stáhnou zdrojové kódy, tak se jednotlivé třídy knihovny vloží do nových souborů stejně, jako je popsáno pro přidání tříd programu.

## Třída pro načtení souboru

```
using System;
using nom.tam.fits;

namespace DetectionWLF
{
    sealed class Reader
```

[1] anglicky namespace

[2] `https://www.nuget.org/packages/CSharpFITS/`

[3] `http://www.vilipetek.com/2014/12/13/polynomial-fitting-in-c/` [Přístup: 07.05.2017]

```
    {
        static double[,,] data;
        /// <summary>
        /// načítání FITS souboru
        /// </summary>
        /// <param name="filename"> jméno souboru </param>
        /// <returns> pole intenzit </returns>
        public static double[,,] ReadData(string filename)
        {
            /* proměnná typu poskytnutého knihovnou pro
               reprezentací FITS souborů, kde se FITS soubor
               načítá */
            Fits f = new Fits(filename);
            /* zjistí velikost datové kostky */
            int[] dimensions = f.GetHDU(0).Axes;
            Console.Clear();
            Console.WriteLine("Načítání souboru..");
            data = new double[dimensions[1], dimensions[2],
               dimensions[0]];
            /* získání pole intenzit typu Array[]
               reprezentující 3D pole */
            Array[] dataRaw = (Array[])f.GetHDU(0).Kernel;
            /* transformování 3D pole typu Array[] na
               vhodnější typ 3D pole double[,,] */
            for (int i = 0; i < dataRaw.Length; i++)
            {
                Array[] twoDimensional = (Array[])dataRaw[i];
                for (int j = 0; j < twoDimensional.Length; j++)
                {
                    Array oneDimensional = twoDimensional[j];
                    float[] doubled = (float[])oneDimensional;
                    for (int k = 0; k < doubled.Length; k++)
                        data[j, k, i] = doubled[k];
                }
            }
            f.Close();
            return data;
        }
    }
}
```

## Třída matematických funkcí

```
using System;
using ViliPetek.LinearAlgebra;

namespace DetectionWLF
{
```

```
sealed class MathHelp
{
    /// <summary>
    /// odčítá od hodnot prvního pole hodnoty druhého pole
    /// </summary>
    /// <param name="y"> původní pole </param>
    /// <param name="y1"> odčítavané pole </param>
    /// <returns> vrátí rozdíl pole rozdílů hodnot
       zadaných polí </returns>
    public static double[] PolySub(double[] y, double[] y1)
    {
        double[] y2 = new double[y.Length];
        for (int i = 0; i < y2.Length; i++)
        {
            y2[i] = y[i] - y1[i];
        }
        return y2;
    }
    /// <summary>
    /// spočítá střední hodnotu zadaného pole
    /// </summary>
    /// <param name="y2"> pole hodnot </param>
    /// <returns> vrátí střední hodnotu </returns>
    public static double ExpectedValue(double[] y2)
    {
        double EX = 0;
        for (int i = 0; i < y2.Length; i++)
        {
            EX = EX + y2[i];
        }
        EX = EX / y2.Length;
        return EX;
    }
    /// <summary>
    /// spočítá standardní odchylku
    /// </summary>
    /// <param name="y2"> pole hodnot </param>
    /// <param name="expectedValue"> střední hodnota
       zadaného pole </param>
    /// <returns> vrátí standardní odchylku </returns>
    public static double StandardDeviation(double[] y2,
       double expectedValue)
    {
        double sigma = 0;
        for (int i = 0; i < y2.Length; i++)
        {
            sigma = sigma + Math.Pow(y2[i] -
               expectedValue, 2);
```

```
            }
            sigma = Math.Sqrt(sigma / (y2.Length-1));
            return sigma;
        }
        /// <summary>
        /// proloží zadané pole polynomem
        /// </summary>
        /// <param name="x"> pole hodnot, na kterých
           funkcionálně zavisí hodnoty z pole "y" </param>
        /// <param name="y"> pole hodnot, které chceme
           proložit polynomem </param>
        /// <param name="degree"> stupeň polynomu </param>
        /// <returns> vrátí pole, kde je vyčislený proložený
           polynom na hodnotách z pole "x" </returns>
        public static double[] Regression(double[] x, double[]
           y, int degree)
        {
            /* proměnná typu poskytnutého knihovnou pro
               proložení polynomem */
            PolyFit fit = new PolyFit(x, y, degree);
            double[] result = fit.Fit(x);
            return result;
        }
    }
}
```

## Třída pro detekci bílých erupcí

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace DetectionWLF
{
    /// <summary>
    /// typ pro uložení 2D souřadnic
    /// </summary>
    struct Coords
    {
        public int X, Y;
        public Coords(int x, int y)
        {
            X = x;
            Y = y;
        }

        public override string ToString()
```

```
    {
        return X + "," + Y;
    }
}

sealed class Detekce
{
    static int MIN_GROUP = 60;      /* min. počet bodů v
       jedné skupině, aby byla zapamatována (v prvním BFS)
       */
    static int MAX_GROUP = 3000;        /* max. počet bodů
       v jedné skupině, aby byla zapamatována (v prvním
       BFS) */
    static int MAX_DIST = 30;       /* max. vzdálenost
       mezi středmi skupin, aby tvořili jednu erupci */
    static int MIN_TIME = 4;        /* min. časový
       interval, který musí nalezená sekvence středů
       trvat, aby byla považována za erupci */
    public static float SIGMA_M = 1.88f;        /* násobek
       standardní odchylky */
    static int MAX_TOTALSEQUENCE = 1000;        /* max.
       počet bodů v jedné sekvenci stredů, aby byla
       zapamatována (v druhém BFS) */

    /// <summary>
    /// Zjistí, jestli pole obsahuje index "x" v prvním
       rozměru a index "y" v druhém rozměru. (Používá se v
       prvním BFS, konkrétně ve funkci processPoint)
    /// </summary>
    /// <param name="array"> trojrozměrné pole (ptáme se
       na indexy jen v prvních dvou rozměrech) </param>
    /// <param name="x"> první index </param>
    /// <param name="y"> druhý index </param>
    /// <returns> jestli pole obsahuje oba indexy, vrátí
       true, jinak false </returns>
    private static bool isInArray(bool[,,] array, int x,
       int y)
    {
        if (x < array.GetLength(0) && x > -1 && y <
           array.GetLength(1) && y > -1)
        {
            return true;
        }
        return false;
    }
    /// <summary>
    /// zjistí vzdálenost mezi zadanými souřadnicemi a
       ověří, jestli je menší než zadaná vzdálenost
```

```
    "maxDist"
/// </summary>
/// <param name="currentCoords"> souřadnice </param>
/// <param name="newCoords"> souřadnice </param>
/// <param name="maxDist"> maximalní vzdálenost
    </param>
/// <returns> je-li vzdálenost mezi souřadnicemi menší
    jako "maxDist", vrátí true, jinak false </returns>
private static bool distance(Coords currentCoords,
    Coords newCoords, int maxDist)
{
    double dist = Math.Sqrt(Math.Pow((currentCoords.X
        - newCoords.X), 2) + Math.Pow((currentCoords.Y
        - newCoords.Y), 2));
    if (dist < maxDist)
        return true;
    else
        return false;
}
/// <summary>
/// spočítá střed souřadnic x nebo y (určí proměnná
    bool)
/// </summary>
/// <param name="listOfCoords"> list souřadnic, u
    kterých chceme spočítat střed </param>
/// <param name="whichCoord"> určuje, jestli se
    spočítá střed x alebo y, true pro x, false pro y
    </param>
/// <returns> vrati zvolený střed souřadnic jako typ
    integer (používá celočiselní dělení) </returns>
private static int countCenter(List<Coords>
    listOfCoords, bool whichCoord)
{
    int xCentre = 0;
    if (whichCoord)
    {
        for (int i = 0; i < listOfCoords.Count; i++)
        {
            xCentre = xCentre + listOfCoords[i].X;
        }
        xCentre = xCentre / listOfCoords.Count();
        return xCentre;
    }
    else
    {
        for (int i = 0; i < listOfCoords.Count; i++)
        {
            xCentre = xCentre + listOfCoords[i].Y;
```

```
            }
            xCentre = xCentre / listOfCoords.Count();
            return xCentre;
        }
    }
    /// <summary>
    /// používá se v prvním BFS
    /// Oveří, jestli byl bod navštívený a jestli je v něm
       zvýšená intenzita. Pokud nebyl navštívený a
       intenzita je zvýšená, označí ho jako navštívený,
       přidá ho do fronty a do aktuální skupiny bodů.
    /// </summary>
    /// <param name="point"> bod, který se ověřuje </param>
    /// <param name="time"> čas, ve kterém se bod ověřuje
       </param>
    /// <param name="visit"> pole, které určuje, jestli
       daný bod byl uz navštívený, pokud byl, je na tomto
       místě true, jinak false </param>
    /// <param name="arrayTF"> pole, kténe má hodnotu true
       v bodech, kde byla zvýšená intenzita, ostatné body
       jsou false </param>
    /// <param name="queue"> fronta </param>
    /// <param name="group"> list souřadnic, kam sa
       přidává bod </param>
    private static void processPoint(Coords point, int
       time, bool[,] visit, bool[,,] arrayTF,
       Queue<Coords> queue, List<Coords> group)
    {
        if (isInArray(arrayTF, point.X, point.Y) &&
           visit[point.X, point.Y] == false &&
           arrayTF[point.X, point.Y, time] == true)
        {
            visit[point.X, point.Y] = true;
            queue.Enqueue(point);
            group.Add(point);
        }
    }
    /// <summary>
    /// první část detekce
    /// zjistí, jestli je v jednotlivých bodech pola
       zvýšena intenzita
    /// </summary>
    /// <param name="data"> 3D pole intenzit </param>
    /// <param name="arrayTF"> ukladá informaci o tom, či
       je v daném bodě pole zvýšena intenzita (true pro
       zvýšenou intenzitu, jinak false) </param>
    /// <param name="i"> index v prvním rozměru 3D pola
       </param>
```

```
private static void intensityRisen(double[,,] data,
   bool[,,] arrayTF, int i)
{
    for (int j = 0; j < data.GetLength(1); j++)
    {
        double[] intensity = new
           double[data.GetLength(2)];
        double[] time = new double[data.GetLength(2)];
        bool jumpOver = true;
        for (int k = 0; k < data.GetLength(2); k++)
        {
            intensity[k] = data[i, j, k];
            if (intensity[k] != 0)
            {
                jumpOver = false;
            }
            time[k] = k;
        }
        if (jumpOver)
        {
            continue;
        }
        /* proložení polynomem, odčítaní proloženého
           polynomu, ověření zvýšení intenzity a
           případné uložení do "arrayTF" */
        double[] intensityPol =
           MathHelp.Regression(time, intensity, 5);
        intensity = MathHelp.PolySub(intensity,
           intensityPol);
        double expectedValue =
           MathHelp.ExpectedValue(intensity);
        double standardDeviation =
           MathHelp.StandardDeviation(intensity,
           expectedValue);
        for (int k = 0; k < data.GetLength(2); k++)
        {
            if (intensity[k] > expectedValue + SIGMA_M
               * standardDeviation && intensity[k] > 0)
            {
                arrayTF[i, j, k] = true;
            }
        }
    }
}
/// <summary>
/// hlavní funkce pro detekci bílých erupcí
/// </summary>
/// <param name="data"> 3D pole intenzit </param>
```

```
public static void CountDetection(double[,,] data)
{
    List<List<Coords>>[] coords = new
       List<List<Coords>>[data.GetLength(2)];
    List<Coords>[] centers = new
       List<Coords>[data.GetLength(2)];
    bool[,,] arrayTF = new bool[data.GetLength(0),
       data.GetLength(1), data.GetLength(2)];

    /* zjistí počet vláken procesoru, aby případně
       funkce "intensityRisen" běžela paralelně */
    if (Environment.ProcessorCount > 1)
    {
        Console.WriteLine("Analýza dat..");
        Parallel.For(0, data.GetLength(0), i =>
           {intensityRisen(data, arrayTF, i); });
    }
    else
    {
        Console.WriteLine("Analýza dat..");
        for (int i = 0; i < data.GetLength(0); i++)
        {
            intensityRisen(data, arrayTF, i);
        }
    }
    /* zjistí rozměry 3D pole a uloží je, smazání
       velkého 3D pole kvůli paměti */
    int xAxis = data.GetLength(0);
    int yAxis = data.GetLength(1);
    int tAxis = data.GetLength(2);
    data = null;
    /* algoritmus BFS pro hledání skupin blízkých bodů
       se zvýšenou intenzitou */
    Console.WriteLine("Hledání vláken erupce..");
    Queue<Coords> queue = new Queue<Coords>();
    for (int k = 0; k < tAxis; k++)
    {
        coords[k] = new List<List<Coords>>();
        centers[k] = new List<Coords>();
        bool[,] visit = new bool[xAxis, yAxis];
        for (int i = 0; i < xAxis; i++)
        {
            for (int j = 0; j < yAxis; j++)
            {
                int index = 0;
                if (arrayTF[i,j,k] == true &&
                   visit[i,j] == false)
                {
```

```
        coords[k].Add(new List<Coords>());
        index = coords[k].Count - 1;
        visit[i, j] = true;
        queue.Enqueue(new Coords(i, j));
        coords[k][index].Add(new
           Coords(i,j));
    }
    bool altered = false;
    while(queue.Count > 0)
    {
        altered = true;
        Coords current = queue.Dequeue();
        processPoint(new Coords(current.X
           - 1, current.Y), k, visit,
           arrayTF, queue,
           coords[k][index]);
        processPoint(new Coords(current.X
           + 1, current.Y), k, visit,
           arrayTF, queue,
           coords[k][index]);
        processPoint(new Coords(current.X
           , current.Y + 1), k, visit,
           arrayTF, queue,
           coords[k][index]);
        processPoint(new Coords(current.X,
           current.Y - 1), k, visit,
           arrayTF, queue,
           coords[k][index]);
        processPoint(new Coords(current.X
           + 1, current.Y + 1), k, visit,
           arrayTF, queue,
           coords[k][index]);
        processPoint(new Coords(current.X
           + 1, current.Y - 1), k, visit,
           arrayTF, queue,
           coords[k][index]);
        processPoint(new Coords(current.X
           - 1, current.Y + 1), k, visit,
           arrayTF, queue,
           coords[k][index]);
        processPoint(new Coords(current.X
           - 1, current.Y - 1), k, visit,
           arrayTF, queue,
           coords[k][index]);
    }
    /* ověření velikosti nalezené skupiny
       bodů */
    if (altered && (coords[k][index].Count
```

```
                    < MIN_GROUP ||
                    coords[k][index].Count > MAX_GROUP))
                {
                    coords[k].Remove(coords[k][index]);
                }
            }
        }
        for(int index = 0; index < coords[k].Count;
            index++)
        {
            centers[k].Add(new Coords(
                countCenter(coords[k][index],true),
                countCenter(coords[k][index],false)));
        }
    }
    /* smazání velkého pole kvůli paměti */
    arrayTF = null;
    /* druhý BFS algoritmus snažící se zjistit, jestli
       se ve dvou po sobě jdoucích časových okamžicích
       v dostatečné blízkosti nachází středy skupin se
       zvýšenou intenzitou */
    Queue<KeyValuePair<int, Coords>> queue2 = new
       Queue<KeyValuePair<int, Coords>>();
    List<KeyValuePair<int, Coords>> sequence_centers =
       new List<KeyValuePair<int, Coords>>();
    HashSet<KeyValuePair<int,Coords>> visited = new
       HashSet<KeyValuePair<int, Coords>>();
    for (int k=0; k < tAxis-1; k++)
    {
        for (int index = 0; index < centers[k].Count;
            index++)
        {
            List<KeyValuePair<int, Coords>>
                currentSequence = new
                List<KeyValuePair<int, Coords>>();
            KeyValuePair<int, Coords> newPoint = new
                KeyValuePair<int, Coords>(k,
                centers[k][index]);
            if (!visited.Contains(newPoint))
            {
                queue2.Enqueue(newPoint);
                visited.Add(newPoint);
                currentSequence.Add(newPoint);
            }
            while(queue2.Count > 0)
            {
                KeyValuePair<int,Coords> current =
                    queue2.Dequeue();
```

```
        if(current.Key >= tAxis - 1)
        {
            queue2.Clear();
            break;
        }
        for (int i_index = 0; i_index <
           centers[current.Key + 1].Count;
           i_index++)
        {
            KeyValuePair<int, Coords>
               nextPoint = new
               KeyValuePair<int,
               Coords>(current.Key + 1,
               centers[current.Key +
               1][i_index]);
            if (!visited.Contains(nextPoint)
               && distance(current.Value,
               centers[current.Key +
               1][i_index], MAX_DIST))
            {
                queue2.Enqueue(new
                   KeyValuePair<int,
                   Coords>(current.Key +
                   1,centers[current.Key +
                   1][i_index]));
                currentSequence.Add(new
                   KeyValuePair<int,
                   Coords>(current.Key + 1,
                   centers[current.Key +
                   1][i_index]));
                visited.Add(nextPoint);
            }
        }
    }
    /* ověřuje, jestli počet bodů aktuální
       sekvenci středů není příliš velký a
       jestli aktuální sekvence trvala
       dostatečně dlouhý časový interval,
       případně se sekvence uloží */
    if (currentSequence.Count == 0)
        continue;
    int numberOfPoints = 0;
    for (int n = 0; n < currentSequence.Count;
       n++)
    {
        numberOfPoints +=
           coords[currentSequence[n].Key].Count;
    }
```

```
            if (numberOfPoints > MAX_TOTALSEQUENCE)
            {
                for (int n = 0; n <
                   currentSequence.Count; n++)
                {
                    KeyValuePair<int, Coords> point =
                       new KeyValuePair<int,
                       Coords>(currentSequence[n].Key,
                       currentSequence[n].Value);
                    visited.Remove(point);
                }
                continue;
            }
            if (currentSequence[currentSequence.Count
               - 1].Key - currentSequence[0].Key >=
               MIN_TIME)
            {
                for (int n = 0; n <
                   currentSequence.Count; n++)
                {
                    KeyValuePair<int, Coords> point =
                       new KeyValuePair<int,
                       Coords>(currentSequence[n].Key,
                       currentSequence[n].Value);
                    sequence_centers.Add(point);
                }
            }
            else
            {
                for(int n = 0; n <
                   currentSequence.Count; n++)
                {
                    KeyValuePair<int, Coords> point =
                       new KeyValuePair<int,
                       Coords>(currentSequence[n].Key,
                       currentSequence[n].Value);
                    visited.Remove(point);
                }
            }
        }
    }
    /* výpis do souboru */
    Console.WriteLine("Vytváření výstupního
       souboru..");
    for (int m = 0; m < sequence_centers.Count(); m++)
    {
        int position =
           centers[sequence_centers[m].Key].IndexOf(
```

```
                    sequence_centers[m].Value);
                for (int a = 0; a <
                    coords[sequence_centers[m].Key][position].Count();
                    a++)
                {
                    Writer.Write(
                        coords[sequence_centers[m].Key][position][a].Y,
                        coords[sequence_centers[m].Key][position][a].X,
                        sequence_centers[m].Key);
                }
            }
            Writer.Close();
        }
    }
}
```

## Třída pro výpis do výstupního souboru

```
using System.IO;

namespace DetectionWLF
{
    sealed class Writer
    {
        public static StreamWriter MyStreamWriter;
        /// <summary>
        /// vypíše souřadnice do výstupního souboru, oddělené
           mezerou
        /// </summary>
        /// <param name="x"> první prostorová souřadnice
           </param>
        /// <param name="y"> druhá prostorová souřadnice
           </param>
        /// <param name="t"> časová souřadnice </param>
        public static void Write(int x, int y, int t)
        {
            MyStreamWriter.WriteLine(x + " " + y + " " + t);
        }
        public static void Close()
        {
            MyStreamWriter.Close();
        }
    }
}
```

## Hlavní třída programu

```
using System;
using System.IO;

namespace DetectionWLF
{
    class Program
    {
        static void Main(string[] args)
        {
            /* nastavení zvyklostí používaných v dané zemi
               (formát datumu, čísel,...), použito aby program
               používal desetinnou bodku */
            System.Threading.Thread.CurrentThread.CurrentCulture
               =
               System.Globalization.CultureInfo.InvariantCulture;
            /* načtení ze standardního vstupu */
            if (args.Length == 0)
            {
                Console.WriteLine("Jméno souboru: ");
                string filename = Console.ReadLine();
                Console.WriteLine("Násobek standardní
                   odchylky: ");
                string error = Console.ReadLine();
                if (error.Trim() != "")
                /* změna hodnoty násobku standardní odchylky
                   "SIGMA_M" na základě vstupu od uživatele */
                {
                    Detekce.SIGMA_M = float.Parse(error);
                }
                string filenameWOExtension =
                   Path.GetFileNameWithoutExtension(filename);
                Writer.MyStreamWriter = new
                   StreamWriter(string.Format("{0}-out-{1}.txt",
                   filenameWOExtension, Detekce.SIGMA_M));
                double[,,] data = Reader.ReadData(filename);
                Detekce.CountDetection(data);
            }
            /* načtení z příkazového řádku  */
            else
            {
                string filename =
                   Path.GetFileNameWithoutExtension(args[0]);
                /* změna hodnoty násobku standardní odchylky
                   "SIGMA_M" na základě vstupu od uživatele */
                if (args.Length > 1)
                {
```

```
                    Detekce.SIGMA_M = float.Parse(args[1]);
                }
                Writer.MyStreamWriter = new
                   StreamWriter(string.Format("{0}-out-{1}.txt",
                   filename, Detekce.SIGMA_M));
                double[,,] data = Reader.ReadData(args[0]);
                Detekce.CountDetection(data);
            }
        }
    }
}
```