KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

Edvinas Medišauskas

KOLIAŽU GRĮSTOS FRAKTALINIŲ INTERPOLIACINIŲ FUNKCIJŲ GENERAVIMO PROCEDŪROS SUDARYMAS IR TYRIMAS

Magistro darbas

Vadovas
prof. dr. J. Valantinas

KAUNAS, 2007

KAUNO TECHNOLOGIJOS UNIVERSITETAS

FUNDAMENTALIŲJŲ MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
prof. dr. J.Rimas
2007 06 06

KOLIAŽU GRĮSTOS FRAKTALINIŲ INTERPOLIACINIŲ FUNKCIJŲ GENERAVIMO PROCEDŪROS SUDARYMAS IR TYRIMAS

Taikomosios matematikos magistro baigiamasis darbas

Vadovas
prof. dr. J. Valantinas
2007 06 03

Recenzentas
doc. dr. R. Rindzevičius
2007 06 01

Atliko
FMMM-5 gr. stud.
E. Medišauskas
2007 05 25

KAUNAS, 2007

KVALIFIKCINĖ KOMISIJA

Pirmininkas: Leonas Saulis, profesorius (VGTU)

Sekretorius: Eimutis Valakevičius, docentas (KTU)

Nariai: Algimantas Jonas Aksomaitis, profesorius (KTU)

Vytautas Janilionis, docentas (KTU)

Vidmantas Povilas Pekarskas, profesorius (KTU)

Rimantas Rudzkis, habil.dr., banko „NORD/LB“ vyriausiasis analitikas

Zenonas Navickas, profesorius (KTU)

Arūnas Barauskas, dr., UAB „Elsis“ generalinio direktoriaus pavaduotojas

Medišauskas E. Local collage based generation and analysis of fractal interpolation functions: Master's Thesis in Applied Mathematics / supervisor prof. dr. J. Valantinas; Department of Applied Mathematics, Faculty of Fundamental Sciences, Kaunas University of Technology. – Kaunas, 2007. – 64 p.

SUMMARY

The present work introduces the notion of fractal interpolation functions (FIF), reveals advantages of the application of fractal interpolation to real-world objects and presents necessary procedures for the implementation of the fractal interpolation process itself.

Firstly, the theoretical context needed to describe the notion of a fractal and relevant definitions is presented. Secondly, the detailed description of fractal generating algorithms (deterministic, random iteration) is given. Since the research object represents theoretical and practical aspects of the analysis of fractal interpolation functions, special attention is paid to geometrical fractals, associated with iterated function systems (IFS).

The concept of a fractal interpolation function (FIF) is presented in the work. The author shows that it is possible to generate interpolation functions which are "close" (in the sense of fractal dimension) to the data under processing. The INTERPO software for the generation of fractal interpolation functions has been analysed, and its suggested interactivity has been considered as disadvantage. Finally, this work presents a newly developed methodics for the determination of vertical scaling factors of the affine transformations, what makes the corresponding IFS suitable for the generation of FIF.

During research, a new and necessary software has been developed. The experimental analysis results showed that the new idea, concerning evaluation of vertical scaling factors ("roughness" cofficients of generated fractal interpolation curves), is truly perspective and promissing.

TURINYS

Lentelių sąrašas

Paveikslų sąrašas

ĮVADAS

Realaus fizinio pasaulio objektams (sistemoms) būdingas tam tikras “atsikartojimas savyje”, kai sistemos dalies struktūra primena viso objekto struktūrą. Tokio tipo objektų modeliavimui vien Euklido geometrijos, kai eksperimentinių duomenų analizės pagrindą sudaro globaliai pripažintos elementariosios funkcijos, nebepakanka. Šiuolaikinės grafikos sistemos vis dažniau renkasi fraktalinio modeliavimo priemones.

Šiame darbe aptariamas eksperimentinių duomenų fraktalinio interpoliavimo savitumas, pradžioje pateikiant reikalingų sąvokų bei apibrėžimų bazę (fraktalo sąvoka, fraktalų sintezės būdai, fraktalinė dimensija). Analizės eigoje parodoma, jog įmanoma gauti fraktalines interpoliacines funkcijas „artimas“ (Hausdorfo metrikos prasme) pradiniui objektui. Be to, galima užtikrinti fraktalinio interpoliavimo grafo ir apdorojamų duomenų (objekto) fraktalinių dimensijų atitiktį. Darbe ypatingas dėmesys skiriamas iteruotųjų (afiniųjų) funkcijų sistemos (IFS), reikalingos fraktalinių interpoliacinių funkcijų sintezei, sudarymui. Peržvelgus ir įvertinus šiai sričiai skirtą programinę įrangą, pasiūlytas naujas, eksperimentinių duomenų fraktališkumą įvertinantis, koliažo principu grįstas fraktalinių interpoliacinių funkcijų generavimo algoritmas (metodas), leidžiantis greičiau ir patogiau gauti interpoliavimo rezultatą – fraktalinę interpoliacinę funkciją.

Trečiajame skyriuje pateikiamas siūlomo metodo programinės realizacijos aprašas bei atlikto eksperimento rezultatų analizė.

Darbas pristatytas konferencijoje „Matematika ir matematinis modeliavimas“ (2007) bei išleistas straipsnis šios tematikos pagrindu.

FRAKTALINĖ GEOMETRIJA – NAUJAS MATEMATINIO MODELIAVIMO METODAS

Fraktalinė geometrija – tai šiuolaikinė matematikos šaka, apimanti Euklido geometriją, topologiją, mato bei dinaminių sistemų teorijas. Amerikiečių mokslininkas Benua Mandelbrotas (Benoi B. Mandelbrot) – fraktalų geometrijos pradininkas – šį klasikinės geometrijos plėtinį pavadino gamtos geometrija. Būtent jis rado paaiškinimus įvairiems „patologiniams“, klasikinei matematinei analizei „nepaklūstantiems“, objektams bei reiškiniams (Kantoro dulkės, Peano kreivės, Vejerštraso funkcijos ir kt.). Šio amerikiečių mokslininko dėka fraktalinė geometrija tapo taikomuoju mokslu. Fraktalų panaudojimo sritys nuolat plečiasi. Jeigu prieš dešimt metų didžiausias dėmesys buvo skiriamas realaus pasaulio objektų (debesys, kalnai, jūros paviršius ir pan.) modeliavimui, tai dabar fraktalinės geometrijos, kaip matematinio įrankio, taikymo sričių spektras tapo kur kas platesnis. Tai – duomenų suglaudinimas (kompiuterinė grafika), techninė kainų analizės teorija (ekonomika), kietųjų kūnų paviršių analizė ir sintezė (fizika), biosensorinių tarpusavio poveikių tyrimas (medicina) ir t.t.

Tolimesniuose skyriuose glaustai pristatomas kontekstas, reikalingas fraktalo sąvokai įvesti, pateikiamas fraktalo bei fraktalinės aibės dimensijos apibrėžimai, aptariama fraktalinio interpoliavimo specifika bei supažindinama su koliažo principu – vienu iš kertinių šio darbo elementų. Kadangi tiriamojoje darbo dalyje nagrinėjami teoriniai ir praktiniai fraktalinių interpoliacinių funkcijų analizės aspektai, tai pagrindinį dėmesį skirsime geometriniams fraktalams, gaunamiems taikant iteruotųjų (afiniųjų) funkcijų sistemas.

1.1. ITERUOTŲJŲ FUNKCIJŲ SISTEMOS IR JŲ ATRAKTORIAI (FRAKTALAI)

Pirmiausia pateiksime gerai žinomus erdvės ir metrinės erdvės apibrėžimus, reikalingus skyrelio užsibrėžtoms sąvokoms įvesti.

Erdve vadinama aibė X, kurios elementai (vadinami erdvės taškais) turi tam tikrą bendrą požymį.

Metrine erdve (X, d) vadinama erdvė X su joje apibrėžta realiąja funkcija d: $X\times X\to \mathrm{R}$, vadinama metrika ir nusakančia atstumą tarp bet kurių dviejų erdvės X taškų. Reikalaujama, kad funkcija d tenkintų tokias aksiomas:

$$d(x,y)=d(y,x),\forall x,y\in X, \tag{1.1}$$

$$0<d(x,y)<\infty,\forall x,y\in X, x\neq y, \tag{1.2}$$

$$d(x,x)=0,\forall x\in X, \tag{1.3}$$

$$d(x,y) \le d(x,z) + d(z,y), \forall x,y,z \in X\ . \qquad \textbf{(1.4)}$$

Imkime metrinę erdvę (X,d) ir joje apibrėžtą transformaciją $\varphi: X \to X$. Pastebėsime, jog $\varphi(S) = \{\varphi(x) \mid x \in S\}$, kai $S \subset X$. Be to, transformacija φ yra apgręžiama, kai ji yra abipusiškai vienareikšmė ir $\varphi(X) = X$. Šiuo atveju, galima apibrėžti atvirkštinę transformaciją $\varphi^{-1}: X \to X$ tokią, kad $\varphi^{-1}(y) = x$, ir $x \in X$ yra vienintelis taškas, su kuriuo $\varphi(x) = y$.

Transformacijos $\varphi: X \to X$ *iteracijomis pirmyn* vadinamos transformacijos $\varphi^{0n}: X \to X$, apibrėžiamos lygybėmis:

$$\varphi^{00}(x) = x,\ \varphi^{01}(x) = \varphi(x),\ \varphi^{0(n+1)}(x) = \varphi(\varphi^{on}(x)), \forall n = 1,2,3,\ldots. \qquad \textbf{(1.5)}$$

Jeigu φ yra apgręžiama, tai transformacijos φ *iteracijomis atgal* vadinamos transformacijos $\varphi^{0(-m)}: X \to X$, apibrėžiamos lygybėmis:

$$\varphi^{0(-1)}(x) = \varphi^{-1}(x), \varphi^{0(-m)}(x) = (\varphi^{0m})^{-1}(x), \forall m = 1,2,3,\ldots. \qquad \textbf{(1.6)}$$

Praktiniuose taikymuose svarbiausia akcentuoti ryšį tarp transformacijas apibūdinančių formulių ir jų poveikyje atsirandančių geometrinių pasikeitimų (ištempimų, poslinkių, lenkimų ir pan.) erdvėse. Be to, svarbu tai, kaip transformacijos veikia ne atskirus erdvės *X* taškus, o jos poaibius.

Kalbant apie geometrinius fraktalus, jų prigimtį, paprastai, imamos Euklido metrinės erdvės (R,*d*), (R^2,*d*) arba (R^3,*d*) ir joje veikiančios afiniosios transformacijos.

Bendru atveju, afinioji transformacija, veikianti (tarkime, dvimatėje) Euklido erdvėje (R^2,*d*), žymima ω: $R^2 \to R^2$ ir apibrėžiama lygybe:

$$\omega(x) = \omega(x_1, x_2) = (ax_1 + bx_2 + e, cx_1 + dx_2 + f), \forall x = (x_1, x_2) \in R^2; \qquad \textbf{(1.7)}$$

čia *a, b, c, d, e, f* – realieji skaičiai (afiniosios transformacijos ω parametrai).

Afiniosios transformacijos turi daug svarbių geometrinių ir algebrinių savybių. Jų pagalba galima realizuoti posūkio, atspindžio, panašumo, pražulniąsias ir kitas transformacijas Euklido erdvėje (R^2, *d*).

Dažnai naudojamas ekvivalentus dvimatės afiniosios transformacijos žymėjimas:

$$\omega = \omega(x) = \omega\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} = Ax + T; \qquad \textbf{(1.8)}$$

čia $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ yra realioji antros eilės kvadratinė matrica, o $T = \begin{pmatrix} e \\ f \end{pmatrix}$ - realusis vektorius.

Jeigu su visais *x, y*$\in R^2$ teisinga nelygybė

$$d(\omega(x), \omega(y)) \le s \cdot d(x,y),\ 0 \le s < 1, \qquad \textbf{(1.9)}$$

tai afinioji transformacija ω: $R^2 \to R^2$ vadinama *suspaudžiančiąja*, o skaičius *s* – afiniosios transformacijos *suspaudimo koeficientu*.

Imkime afiniųjų suspaudžiančiųjų transformacijų ω_i: $R^2 \to R^2$ ($i = 1,2,...,N$) rinkinį; atskirų afiniųjų transformacijų suspaudimo koeficientus pažymėkime s_i, $i = 1,2,...,N$. Tada, Euklido erdvė (R^2, d) su joje veikiančių suspaudžiančiųjų afiniųjų transformacijų rinkiniu vadinama *iteruotųjų funkcijų sistema* ir žymima — IFS$\{R^2; \omega_1, \omega_2, \ldots, \omega_N\}$. IFS suspaudimo koeficientu laikomas skaičius $s = \max\{s_1, s_2, \ldots, s_N\}$.

Apibrėžkime dar vieną metrinę erdvę ($H(R^2)$, h) tokiu būdu: $H(R^2)$ – visų netuščių uždarųjų aibės R^2 poaibių aibė; h – metrika, nusakanti atstumą tarp bet kurių dviejų aibės $H(R^2)$ elementų (aibės R^2 poaibių), būtent:

$$h(A,B) = \max\{d(A,B), d(B,A)\};$$
$$\text{čia } d(A,B) = \max_{x\in A}\{\min_{y\in B}\{d(x,y)\}\};\quad d(B,A) = \max_{y\in B}\{\min_{x\in A}\{d(x,y)\}\}. \qquad \textbf{(1.10)}$$

Įvestoji erdvė dažnai vadinama *fraktaline erdve*.

Perkelkime IFS sudarančias afiniąsias transformacijas į erdvę ($H(R^2)$, h). Tada $\omega_i: H(R^2) \to H(R^2)$, apibrėžiama lygybe $\omega_i(B) = \{\omega_i(y) | y \in B\}$ ($\forall B \in H(R^2)$), yra suspaudžiančioji transformacija erdvėje ($H(R^2)$, h), ir jos suspaudimo koeficientas lygus s_i, $i = 1,2,...,N$.

Pagaliau, apibrėžkime dar vieną transformaciją W: $H(R^2) \to H(R^2)$ fraktalinėje erdvėje ($H(R^2)$, h), būtent:

$$W(B) = \omega_1(B) \cup \omega_2(B) \cup \ldots \cup \omega_N(B) = \bigcup_{i=1}^{N} \omega_i(B), \forall B \in H(R^2). \qquad \textbf{(1.11)}$$

Galima įsitikinti, jog pastaroji transformacija taip pat yra suspaudžiančioji, t.y. $d(W(B), W(C)) \le s \cdot h(B,C)$, su visais B, $C \in H(R^2)$; be to, $s = \max\{s_1, s_2, \ldots, s_N\}$.

Vienintelis nejudamasis transformacijos W: $H(R^2) \to H(R^2)$ taškas A ($A \in H(R^2)$) toks, kad

$$A = W(A) = \bigcup_{i=1}^{N} \omega_i(A) = \lim_{n\to\infty} W^{on}(B), \forall B \in H(R^2), \qquad \textbf{(1.12)}$$

vadinamas *IFS atraktoriumi* (arba *fraktalu*).

Kitame skyriuje apžvelgsime populiariausius fraktalų sintezės (generavimo) algoritmus.

1.1.1. FRAKTALŲ SINTEZĖS ALGORITMAI

Žinomi ir praktikoje taikomi įvairūs IFS atraktorių sintezės algoritmai, būtent:

- determinuotasis algoritmas
- atsitiktinių iteracijų algoritmas
- „pabėgimo laiko“ algoritmas.

Detaliau aptarsime pirmuosius du sintezės algoritmus, kadangi jie būtini tiriamąjam darbui, o „pabėgimo laiko“ algoritmo „veikimo mechanizmo“ čia neaprašinėsime.

Determinuotojo fraktalų (IFS atraktorių) generavimo algoritmo „veikimas“ tiesiogiai remiasi IFS atraktoriaus apibrėžimu.

Tarkime, kad $\{R^2; \omega_1, \omega_2, \ldots, \omega_N\}$ yra iteruotųjų funkcijų sistema (IFS); čia ω_i ($i = 1,2,...,N$) yra suspaudžiančiosios afiniosios transformacijos. Parenkame pradinę uždarąją aibę $A_0 \subset R^2$ ($A_0 \in H(R^2)$) ir nuosekliai formuojame aibių seką $\{A_n\}$, būtent:

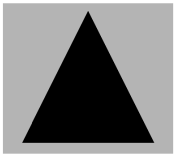
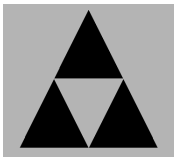
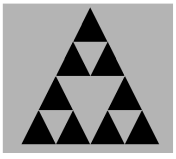
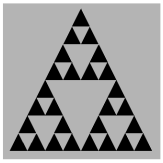
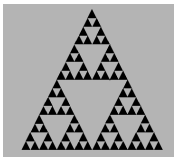
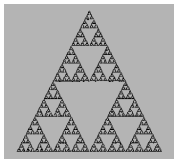
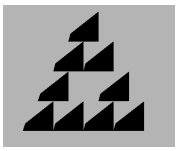
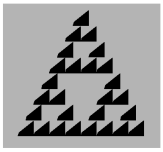
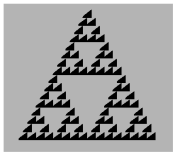
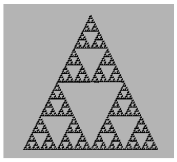
$$A_n = W^{on}(A_0) = W(A_{n-1}) = \bigcup_{i=1}^{N} \omega_i(A_{n-1}), \forall n = 1,2,\ldots. \qquad \textbf{(1.13)}$$

Įrodoma, jog tai Koši seka, kuri konverguoja į IFS atraktorių A. Kai n yra pakankamai didelis skaičius, aibė A_n tampa „artima“ (metrikos h prasme, tuo pačiu ir vizualiai) aibei A (fraktalui, IFS atraktoriui).

Kaip matome, visiškai nesvarbu, kokia imama pradinė uždaroji aibė A_0. Iteracinės procedūros rezultatas yra vienas ir tas pats — IFS atraktorius A. Kitaip tariant, aibė A pilnai nusakoma afiniųjų transformacijų ω_i, ($i = 1,2,...,N$), veikiančių fraktalinėje erdvėje ($H(R^2)$, h), išraiškomis (1.1 lentelė).

1.1 lentelė

Determinuotojo IFS atraktoriaus sintezės algoritmo veikimo rezultatas

***n*=1**	***n*=2**	***n*=3**	***n*=4**	***n*=5**	***n*=8**
					
					

Toliau aptarsime *atsitiktinių iteracijų algoritmą*. Vėlgi, tarkime, kad $\{R^2; \omega_1, \omega_2, \ldots, \omega_N\}$ yra iteruotųjų funkcijų sistema. Kiekvienai suspaudžiančiajai afiniajai transformacijai ω_i ($i = 1,2,...,N$) yra priskiriamas teigiamas skaičius (tikimybė) p_i taip, kad $\sum_{i=1}^{n} p_i = 1$.

Jeigu $\omega_i = \omega_i(x) = \omega_i(x_1, x_2) = (a_i x_1 + b_i x_2 + e_i, c_i x_1 + d_i x_2 + f_i)$ ($i \in \{1,2,...,N\}$), tai apytikslė tikimybės p_i reikšmė randama iš formulės

$$p_i \cong \frac{|a_i d_i - b_i c_i|}{\sum_{j=1}^{N} |a_j d_j - b_j c_j|}. \qquad \textbf{(1.14)}$$

Jeigu su kuria nors reikšme i, $a_i d_i - b_i c_i = 0$, tai p_i prilyginama pakankamai mažam teigiamam skaičiui (pavyzdžiui, $p_i = 0{,}001$).

Toliau, parenkamas pradinis taškas x_0 ($x_0 \in \mathrm{R}^2$). Konstruojama erdvės R^2 taškų seka $\{x_n\}_{n=0}^{\infty}$; čia $x_n \in \{\omega_1(x_{n-1}), \omega_2(x_{n-1}), \ldots, \omega_N(x_{n-1})\}$ ($n = 1,2,\ldots$) ir įvykio, jog x_n įgys reikšmę $\omega_i(x_{n-1})$, tikimybė yra lygi p_i, t.y.

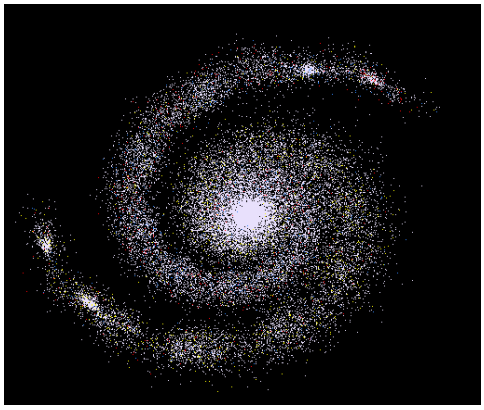
$$P\{x_n = \omega_i\{x_{n-1}\}\} = p_i,\ i \in \{1,2,\ldots,N\}. \qquad \textbf{(1.15)}$$

Atsitiktinių iteracijų algoritmo „veikimas“ yra grindžiamas fraktalo (IFS atraktoriaus) taškams būdinga chaotiška judesio dinamika, [2].

Pateikiame kelis atsitiktinių iteracijų algoritmo pagalba sugeneruotus fraktalų pavyzdžius ir juos atitinkančias IFS:

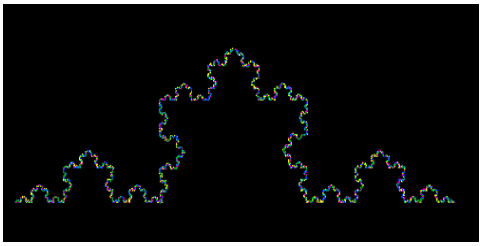
1.2 lentelė

IFS atraktorius „Galaktika“, gautas taikant atsitiktinių iteracijų algoritmą

Gautas fraktalinis vaizdas	Iteruotųjų funkcijų sistema (IFS)
	$IFS\{X;\omega_1,\omega_2,\omega_3,\omega_4,\omega_5\}$ $\omega_1 = \omega_1(x_1,x_2) = (-0.08x_1 + 0.06x_2 - 2.02, 0\cdot x_1 - 0.28x_2 - 2.4)$ $\omega_2 = \omega_2(x_1,x_2) = (-0.05x_1 + 0.20x_2 - 1.51, -0.03x_1 - 0.21x_2 - 3.46)$ $\omega_3 = \omega_3(x_1,x_2) = (-0.05x_1 + 0.23x_2 + 2.04, -0.05x_1 - 0.2x_2 + 2.47)$ $\omega_4 = \omega_4(x_1,x_2) = (-0.01x_1 + 0.30x_2 + 0.67, -0.05x_1 - 0.04x_2 + 3.18)$ $\omega_5 = \omega_5(x_1,x_2) = (0.70x_1 + 0.52x_2 + 0.54, -0.53x_1 + 0.69x_2 + 0.14)$ $x_1, x_2 \in X \subset R^2$ $p_1 = 0.03$, $p_2 = 0.02$, $p_3 = 0.03$, $p_4 = 0.02$, $p_5 = 0.91$

1.3 lentelė

IFS atraktorius „Kocho kreivė“, gautas taikant atsitiktinių iteracijų algoritmą

Gautas fraktalinis vaizdas	Iteruotųjų funkcijų sistema (IFS)
	$IFS\{X;\omega_1,\omega_2,\omega_3,\omega_4,\omega_5\}$ $\omega_1 = \omega_1(x_1,x_2) = (0.34x_1 + 0\cdot x_2 + 2.14, 0\cdot x_1 + 0.34x_2 + 0.02)$ $\omega_2 = \omega_2(x_1,x_2) = (0.17x_1 + 0.29x_2 + 0.55, -0.29x_1 + 0.17x_2 + 0.94)$ $\omega_3 = \omega_3(x_1,x_2) = (0.16x_1 - 0.29x_2 - 0.54, 0.29x_1 + 0.16x_2 + 0.95)$ $\omega_4 = \omega_4(x_1,x_2) = (0.34x_1 + 0\cdot x_2 - 2.15, 0\cdot x_1 + 0.34x_2 + 0.01)$ $x_1, x_2 \in X \subset R^2$ $p_1 = 0.25$, $p_2 = 0.25$, $p_3 = 0.24$, $p_4 = 0.25$

1.4 lentelė

IFS atraktorius „Judesys“, gautas taikant atsitiktinių iteracijų algoritmą

Gautas fraktalinis vaizdas	Iteruotųjų funkcijų sistema (IFS)
	$IFS\{X;\omega_1,\omega_2,\omega_3,\omega_4,\omega_5\}$ $\omega_1=\omega_1(x_1,x_2)=(0.95x_1-0.01,-0.01x_1+0.95x_2)$ $\omega_2=\omega_2(x_1,x_2)=(0.30x_1-0.81x_2+0.7,-0.14x_1-0.16x_2-0.4)$ $\omega_3=\omega_3(x_1,x_2)=(-0.07x_1-0.18x_2-0.17,0.2x_1-0.38x_2+0.1)$ $\omega_4=\omega_4(x_1,x_2)=(-0.19x_1+0.19x_2-0.95,0.01x_1+0.06x_2+0.62)$ $\omega_5=\omega_5(x_1,x_2)=(-0.15x_1+0.14x_2-0.42,0.06x_1+0.02x_2+0.62)$ $x_1,x_2\in X\subset R^2$ $p_1=0.79,\ p_2=0.14,\ p_3=0.05,\ p_4=0.01,\ p_5=0.01$

Šiame darbe, fraktalinių interpoliacinių funkcijų sintezei naudojamas determinuotasis algoritmas, nusileidžiantis atsitiktinių iteracijų algoritmui spartumu, tačiau pasižymintis nuoseklumu bei tikslumu. Kitame skyrelyje pristatysime koliažo principą, kurio idėjomis grįsime visą tolimesnįjį tiriamąjį darbą.

1.1.2. KOLIAŽO PRINCIPAS

Pagrindinė koliažo idėja (principas) yra ta, kad norėdami gauti iteruotųjų funkcijų sistemą (IFS), kurios atraktorius būtų „artimas“ (panašus) kokiai nors duotai aibei, turime stengtis rasti suspaudžiančiųjų afiniųjų transformacijų rinkinį tokį, kad tos aibės sumažintų vaizdų (kopijų), gautų taikant atskiras afiniąsias transformacijas, sąjunga („koliažas“) būtų kuo „artimesnis“ (Hausdorfo metrikos prasme) pačiai aibei. Lokaliojo koliažo atveju, stengiamės rasti IFS tokią, kurią sudarančios transformacijos veikia jau ne visą duotą aibę, o tam tikrus jos fragmentus (poaibius), t.y. „koliažas“ konstruojamas iš duotos aibės poaibių sumažintų kopijų.

1 teorema (lokalioji koliažo teorema). Tegu $T\in H(\mathrm{R}^2)$ - mus dominantis objektas (aibė); čia $H(\mathrm{R}^2)$ - fraktalinė erdvė, t.y. visų galimų netuščių uždarųjų erdvės R^2 poaibių aibė; $T_1,T_2,...,T_N$ - tam tikri netušti uždarieji aibės T poaibiai.

Jeigu $IFS\{\mathrm{R}^2;\omega_1,\omega_2,...,\omega_N\}$ yra tokia, kad $h(T,\bigcup_{i=1}^{N}\omega_i(T_i))\le\varepsilon$, tai $h(T,A)\le\frac{\varepsilon}{1-s}$; čia: h – Hausdorfo atstumas; $s=\max\{s_1,s_1,...,s_N\}$, kai s_i $(i\in\{1,2,...,N\})$ yra afiniosios transformacijos ω_i suspaudimo koeficientas; $\varepsilon>0$.

Ši lokalioji koliažo teorema plačiai taikoma fraktalinėse skaitmeninių vaizdų efektyvaus kodavimo (suglaudinimo) procedūrose. Tolimesniuose skyreliuose parodysime, jog šią teoremą galima racionaliai panaudoti ir generuojant fraktalines interpoliacines funkcijas su fiziniais objektais (sistemomis) susijusiems duomenims.

1.2. FRAKTALINĖS DIMENSIJOS SĄVOKA

Skaičiai, charakterizuojantys fraktalus, paprastai, vadinami fraktalinėmis dimensijomis. Tai gana svarbios kiekybinės fraktalų charakteristikos. Jos leidžia įvertinti intuityvų supratimą apie tai, kaip tirštai („tankiai“) realaus pasaulio objektas (fraktalas) užpildo erdvę, kurios poaibis jis pats yra. Fraktalinė dimensija – tai objektyvi priemonė fraktalų palyginimui. Interpoliavimo uždavinio kontekste, fraktalinės dimensijos pagalba galima įvertinti gautų fraktalinių interpoliacinių funkcijų kokybę, jų atitiktį interpoliuojamiems duomenims.

Istoriškai, svarbu apibrėžti Hausdorfo ir Bezicovičiaus matą bei dimensiją, tuo pačiu atskleisti su jų apskaičiavimu susijusius sunkumus.

Imkime poaibį $A \subset \mathrm{R}^n$; čia (R^n, d) – n-matė Euklido erdvė. Tegu

$$|A| = \operatorname{diam}(A) = \sup\{d(x,y) | \, x,y \in A\} \tag{1.16}$$

žymi poaibio A diametrą.

Fiksavę teigiamą skaičių $\varepsilon > 0$, imkime skaičią poaibių A_i ($|A_i| \le \varepsilon$) aibę $\{A_i\}$ tokią, kad ji padengtų poaibį A, t.y. $A \subset \bigcup_{i=1}^{\infty} A_i$; rinkinys (aibė) $\{A_i\}$ vadinamas *poaibio A ε-denginiu*.

Toliau, tarkime, kad $p > 0$, ir apibrėžkime dydį $H_\varepsilon^p(A)$ tokiu būdu:

$$H_\varepsilon^p(A) = \inf\left\{\sum_{i=1}^{\infty} |A_i|^p \,\middle|\, \{A_i\} \text{yra poaibio } A\ \varepsilon\text{ - denginys}\right\}. \tag{1.17}$$

Faktiškai, tai minimizavimo uždavinys, kai stengiamasi minimizuoti dengiančiųjų aibių diametrų, pakeltų p-tuoju laipsniu, sumą (pagal visus įmanomus poaibio A ε-denginius). Kita vertus, dydis $H_\varepsilon^p(A)$, kaip funkcija, yra nemažėjantis, kai ε artėja prie nulio. Pereidami prie ribos ($\varepsilon \to 0$), apibrėšime *Hausdorfo ir Bezicovičiaus matą* (poaibiui A) tokiu būdu:

$$H^p(A) = \lim_{\varepsilon \to 0} H_\varepsilon^p(A). \tag{1.18}$$

Dydžio $H^p(A)$ priklausomybės nuo parametro p grafikas rodo, jog yra kritinė p reikšmė, kai $H^p(A)$ „šoka“ nuo ∞ prie 0. Tą kritinį tašką atitinkanti p reikšmė vadinama *Hausdorfo ir Bezicovičiaus dimensija* (žymėsime $D_H(A)$), t.y.

$$D_H(A) = \sup\{p \mid H^p(A) = \infty\} = \inf\{p \mid H^p(A) = 0\}. \quad \textbf{(1.19)}$$

Pastebėsime, jog šią dimensiją apskaičiuoti netgi „paprastoms“ aibėms (fraktalams) yra pakankamai sudėtinga. Vėliau, B. Mandelbrotas šią dimensiją pavadino fraktaline dimensija. Faktiškai, tai nauja erdvės (poaibių) matavimo priemonė. Daugelyje šaltinių, fraktalinės dimensijos sąvokos įvedimui naudojamas šiek tiek supaprastintas požiūris (interpretacija), kurį čia pat ir aptarsime.

Imkime metrinę erdvę (R^n, d). Tegul $A \in \mathrm{H}(\mathrm{R}^n)$. Tarkime, kad $\varepsilon > 0$ ir $N(A,\varepsilon)$ žymi mažiausią uždarųjų rutulių $B(x,\ \varepsilon) = \{y \in \mathrm{R}^n \mid d(x,\ y) \le \varepsilon\}$, sudarančių baigtinį poaibio A denginį, skaičių, t.y. $N(A,\varepsilon)$ – mažiausias sveikasis skaičius toks, kad $A \subset \bigcup_{i=1}^{N(A,\varepsilon)} B(x_i,\varepsilon)$; čia $\{x_1, x_2, \ldots, x_{N(A,\varepsilon)}\} \subset \mathrm{R}^n$.

Fraktalinės dimensijos sąvoka įvedama, remiantis intuityvia idėja, jog aibė A ($A \subset \mathrm{R}^n$) turi fraktalinę dimensiją D, jeigu

$$N(A,\varepsilon) \approx C\varepsilon^{-D}; \quad \textbf{(1.20)}$$

čia: C – tam tikra konstanta; ženklas „≈“ reiškia, jog $\lim_{\varepsilon \to 0}\left(\ln f(\varepsilon)/\ln g(\varepsilon)\right) = 1$, kai $f(\varepsilon) \approx g(\varepsilon)$.

Dabar, išsprendę (1.19) „lygybę“ (D atžvilgiu), gauname, jog:

$$D \approx \frac{\ln N(A,\varepsilon) - \ln C}{\ln \frac{1}{\varepsilon}}. \quad \textbf{(1.21)}$$

Perėję prie ribos, kai $\varepsilon \to 0$, turime:

$$D = \lim_{\varepsilon \to 0} \frac{\ln N(A,\varepsilon)}{\ln \frac{1}{\varepsilon}}. \quad \textbf{(1.22)}$$

Ši riba (jeigu ji egzistuoja ir yra baigtinė) vadinama poaibio A ($A \subset \mathrm{R}^n$) *fraktaline dimensija* ir žymima $D(A) = D$.

Žemiau pateikiame keletą svarbesnių teiginių, skirtų teoriniam ir eksperimentiniam fraktalinės Euklido erdvės poaibių dimensijos nustatymui.

2 teorema. Tegul $A \subset \mathrm{H}(\mathrm{R}^m)$; be to, $\varepsilon_n = Cr^n$ ($C > 0$, $0 < r < 1$), $n = 1,2,\ldots$. Poaibis A turi fraktalinę dimensiją D, apibrėžiamą formule:

$$D(A) = D = \lim_{n \to \infty} \frac{\ln N_n(A)}{\ln \frac{1}{\varepsilon_n}}. \quad \textbf{(1.23)}$$

Ši teorema leidžia tolydųjį kintamąjį ε pakeisti diskrečiuoju kintamuoju ε_n, ko pasekoje supaprastėja skaičiavimo procedūra.

3 teorema. Tarkime, kad $A \subset \mathrm{H}(\mathrm{R}^m)$; be to, erdvė R^m padengta nesusikertančiais m-mačiais „kubiukais“ (dėžutėmis), kurių briaunos ilgis lygus $1/2^n$; tegul $N_n(A)$ žymi „kubiukų“, persidengiančių su poaibiu A, skaičių. Tada

$$D = \lim_{n\to\infty} \frac{\ln N_n(A)}{\ln(2^n)} \qquad \textbf{(1.24)}$$

vadinamas poaibio *A* fraktaline dimensija.

Būtent pastarasis teiginys sudaro pagrindą realaus pasaulio objektų fraktalinės dimensijos eksperimentiniam nustatymui (dimensijos įverčio gavimui).

4 teorema. Jeigu $A, B \in \mathrm{H}(\mathrm{R}^n)$ ir $A \subset B$, tai $D(A) \le D(B)$; be to, $D(A) \ge 0$, $D(B) \le n$.

5 teorema. Jeigu $A, B \in \mathrm{H}(\mathrm{R}^n)$ ir $D(A)$, $D(B)$ bei $D(A \cup B)$ žymi atitinkamai aibių *A, B* ir $A \cup B$ fraktalines dimensijas, tai $D(A \cup B) = D(A)$, kai $D(B) < D(A)$.

Visiškai nejungiąja IFS vadinama IFS$\{\mathrm{R}^n; \omega_1, \omega_2, \ldots \omega_N\}$, tenkinanti sąlygas:

$\omega_i(A) \cap \omega_j(A) = \emptyset$, kai $i \neq j$; A – IFS atraktorius.

6 teorema. Tarkime, kad $A \subset \mathrm{R}^n$ sutampa su visiškai nejungios IFS$\{\mathrm{R}^2; \omega_1, \omega_2, \ldots \omega_N\}$, sudarytos iš afiniųjų panašumo transformacijų, atraktoriumi. Tada, poaibio *A* fraktalinė dimensija *D* yra lygties

$$\sum_{i=1}^{N} s_i^D = 1 \qquad \textbf{(1.25)}$$

sprendinys; čia s_i yra *i*-tosios afiniosios transformacijos suspaudimo koeficientas, $i = 1,2,...,N$.

6 teorema. Tarkime, kad $D(A)$ ir $D_H(A)$ žymi kokio nors aprėžto poaibio $A \subset \mathrm{R}^n$ atitinkamai fraktalinę bei Hausdorfo ir Bezicovičiaus dimensijas. Tada

$$0 \le D_H(A) \le D(A) \le n. \qquad \textbf{(1.26)}$$

Galima būtų teigti, jog $D_H(A)$ „šiek tiek subtiliau“ charakterizuoja poaibį *A*, negu $D(A)$.

Yra ir daugiau požiūrių bei interpretacijų, pateikiant fraktalines Euklido erdvės poaibių dimensijas. Tai – koreliacijos dimensija, informacijos dimensija, Liapunovo dimensija ir panašiai. Plačiau apie jas čia nekalbėsime.

1.3. FRAKTALINĖS INTERPOLIACINĖS FUNKCIJOS, JŲ SPECIFIKA

Elementariosios funkcijos (sinusas, kosinusas, daugianariai ir kt.) sudaro tradicinio eksperimentinių duomenų analizės metodo pagrindą. Tarkime, kad eksperimento metu matuojamos tam tikros realiosios funkcijos $F(x)$ reikšmės. Eksperimento rezultatas – duomenų rinkinys $\{(x_i, F_i) \mid i=0,1,\ldots,N\}$; čia $F_i = F(x_i)$, $i=0, 1, \ldots, N$, ir $x_0 < x_1 < \ldots < x_N$. Šie duomenys pavaizduojami grafiškai (erdvėje R^2) ir analizuojami, t. y. stengiamasi parinkti galimai žemesnio laipsnio daugianarį (atskiru atveju, „laužtę“), kuris „neblogai modeliuotų“ duomenis segmente $[x_0, x_N]$, t.y. kurio grafikas „eitų“ per taškus (x_i, F_i), $i=0,1,\ldots,N$.

Tačiau, grafikos sistemoms kartais keliami šiek tiek didesni reikalavimai, būtent: galimybė modeliuoti realaus pasaulio (fizinius) objektus, tokius kaip debesys, kalnų masyvo profilis, kabantys stalaktitai ir panašiai. Šiems fiziniams objektams (sistemoms) būdinga tai, kad kiekvienos sistemos dalies struktūra tam tikra prasme „atkartoja“ visos sistemos struktūrą. Tai galima aiškinti tuo, jog sistemą įvairiuose jos lygiuose veikiančios ir formuojančios jėgos yra panašios. Natūralu, jog Euklido geometrijos ir elementariųjų funkcijų čia jau nebepakanka.

Pateiksime fraktalinės interpoliacinės funkcijos sąvoką, parodysime, jog minėto tipo duomenims galima parinkti funkcijas, kurios (Hausdorfo metrikos prasme) būtų „artimos“ tiems duomenims; be to, įmanoma užtikrinti, kad fraktalinės interpoliacinės funkcijos grafiko dimensija sutaptų su eksperimento duomenų (grafiko) fraktaline dimensija.

Tarkime, kad eksperimento metu matuojamos tam tikros realiosios funkcijos $F(x)$ reikšmės. Matavimo rezultatas – duomenų aibė $\{(x_i, F_i) \in \mathrm{R}^2 | i = 0,1,...,N\}$; čia $F_i = F(x_i)$, $x_0 < x_1 < ... < x_N$. Tolydžioji funkcija $g : [x_0, x_N] \to \mathrm{R}$ tokia, kad $g(x_i) = F_i$ ($i = 0,1,...,N$), vadinama duomenų aibę $\{(x_i, F_i)\}$ atitinkančia *interpoliacine funkcija*. Taškai (x_i, F_i) ($i = 0,1,...,N$) vadinami *interpoliavimo taškais*.

Žinoma, kad erdvėje R^2 galima sukonstruoti iteruotųjų funkcijų sistemą $IFS\{\mathrm{R}^2; \omega_1, \omega_2, ..., \omega_N\}$, kurią sudarytų afiniosios transformacijos $\omega_1, \omega_2, ..., \omega_N$ ir kurios atraktorius sutaptų su tolydžiosios duomenis interpoliuojančios funkcijos $\varphi : [x_0, x_N] \to \mathrm{R}$ grafiku. Tam pakanka imti pražulniąsias (Oy ašies atžvilgiu) afiniąsias transformacijas:

$$\omega = \omega(x, F) = \begin{pmatrix} r_1 \cos\Theta° & -r_2 \sin 0° \\ r_1 \sin\Theta° & r_2 \cos 0° \end{pmatrix} \begin{pmatrix} x \\ F \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} = \begin{pmatrix} a & 0 \\ c & d \end{pmatrix} \begin{pmatrix} x \\ F \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}; \qquad \textbf{(1.27)}$$

čia $(x, F) \in R^2$, $a, b, c, d, e, f \in \mathrm{R}$.

Afinioji transformacija

$$\omega_i = \omega_i(x, F) = \begin{pmatrix} a_i & 0 \\ c_i & d_i \end{pmatrix} \begin{pmatrix} x \\ F \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \end{pmatrix} \qquad \textbf{(1.28)}$$

atvaizduoja duomenų aibę (nuo x_0 iki x_N) į „juostą“ nuo x_{i-1} iki x_i ($i = 1,2,...,N$). Tai reiškia, jog turi būti tenkinamos sąlygos:

$$\begin{cases} a_i x_0 + e_i = x_{i-1}, \\ a_i x_N + e_i = x_i, \\ c_i x_0 + d_i F_0 + f_i = F_{i-1}, \\ c_i x_N + d_i F_N + f_i = F_i, \end{cases} \qquad \textbf{(1.29)}$$

su visais $i = 1,2,...,N$.

Nesunku pastebėti, jog šioje tiesinių algebrinių lygčių sistemoje vienas kintamasis (tarkime, kad tai vertikalųjį mastelį keičiantis parametras d_i) yra laisvai pasirenkamas. Tada, išsprendę sistemą, gauname:

$$a_i = \frac{x_i - x_{i-1}}{x_N - x_0}, \tag{1.30}$$

$$c_i = \frac{F_i - F_{i-1}}{x_N - x_0} - d_i \frac{F_n - F_0}{x_N - x_0}, \tag{1.31}$$

$$e_i = \frac{x_N x_{i-1} - x_0 x_i}{x_N - x_0}, \tag{1.32}$$

$$f_i = \frac{x_N F_{i-1} - x_0 F_i}{x_N - x_0} - d_i \frac{x_N F_0 - x_0 F_N}{x_N - x_0}. \tag{1.33}$$

Jeigu parinktume $d_i = 0$ ($i = 1,2,...,N$), tai $IFS\{\mathrm{R}^2; \omega_1, \omega_2, ..., \omega_N\}$ atraktorius sutaptų su laužte (dalimis tiesine interpoliacine funkcija). Kitu atveju (esant nenulinėms parametro d_i ($i \in \{1,2,...,N\}$) reikšmėms) – IFS atraktorius išlieka duomenis $\{(x_i, F_i) \in \mathrm{R}^2 | i = 0,1,...,N\}$ interpoliuojančiąja funkcija. Tačiau akivaizdu, jog tokia interpoliuojančių funkcijų generavimo procedūra nėra pati geriausia – retai kada kiekvienas duomenų aibės poaibis, atitinkantis segmentą $[x_{i-1}, x_i]$, būna „panašus“ į visą duomenų aibę; be to, kebli parametrų d_i ($i = 1,2,...,N$) nustatymo specifika.

Nesunku pastebėti, jog parametrai d_i (i=1,2,…,N) įtakoja IFS atraktoriaus (fraktalinės interpoliacinės funkcijos) fraktalinę dimensiją.

Pateikiame be įrodymo vieną (šia prasme labai svarbų) teiginį.

7 teorema. Tarkime, kad IFS$\{\mathrm{R}^2; \omega_1, \omega_2, \ldots, \omega_N\}$ yra susieta su duomenų aibe $\{(x_i, F_i) \in \mathrm{R}^2 | i = 1,2,...,N\}$; be to,

$$\omega_i = \omega_i(x, y) = \begin{pmatrix} a_i & 0 \\ c_i & d_i \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \end{pmatrix}, \; i = 1,2,...,N;$$

čia: koeficientai a_i, c_i, e_i ir f_i apibrėžiami (1.30)-(1.33) formulėmis, $0 \le d_i < 1$, su visais $i = 1,2,...,N$.

Tarkime, kad aibė A yra fraktalinės interpoliacinės funkcijos grafikas (IFS atraktorius).

Jeigu $\sum_{i=1}^{N} |d_i| > 1$ ir interpoliavimo taškai nepriklauso vienai tiesei, tai A turi fraktalinę dimensiją D, kuri yra vienintelis realusis lygties $\sum_{i=1}^{N} |d_i| \cdot a_i^{D-1} = 1$ sprendinys; priešingu atveju, $D(A) = 1$.

Pastebėsime, jog tuo atveju, kai interpoliavimo taškai yra pasiskirstę segmente $[x_0, x_N]$ tolygiai, t.y. $x_i = x_0 + \frac{i}{N}(x_N - x_0),\ i = 0,1,...,N$, koeficientų a_i ($i = 1,2,...,N$) reikšmės lygios $1/N$. Gauname:

$$\sum_{i=1}^{N} |d_i| = N^{D-1}. \tag{1.34}$$

Pastarąją lygtį spręsdami D atžvilgiu, randame, jog

$$D = 1 + \frac{\ln(\sum_{i=1}^{N} | d_i |)}{\ln N}. \tag{1.35}$$

Vadinasi, manipuliuojant parametrų d_i ($i = 1,2,...,N$) reikšmėmis, dydžiui $\sum_{i=1}^{N} |d_i|$ galima priskirti bet kurią reikšmę iš intervalo $[1, N]$. Tuo pačiu, fraktalinės interpoliacinės funkcijos (IFS atraktoriaus A) fraktalinę dimensiją galima prilyginti bet kuriai reikšmei iš intervalo [1, 2]. Tai labai svarbus momentas, kadangi atsiranda galimybė „priderinti“ fraktalines interpoliacinių funkcijų dimensijas prie duomenų aibės (pradinio objekto) fraktalinės dimensijos.

Įdomu ir tai, kad fraktalinė dimensija nepriklauso nuo reikšmių $\{F_i \mid i = 0,1,...,N\}$, išskyrus tą faktą, jog interpoliavimo taškai neturi priklausyti vienai tiesei. Taigi, galima kalbėti apie fraktalinių interpoliacinių funkcijų, turinčių tą pačią dimensiją D, rinkinį. Tam pakanka išpildyti (1.34) sąlygą.

Nors fraktalinės interpoliacinės funkcijos (FIF) buvo formalizuotos daugiau nei prieš dešimtmetį, tačiau tik pastaraisiais metais jas imta plačiau taikyti kompiuterinėje grafikoje, kalbos signalams interpoliuoti, kalno vaizdui, seisminiams duomenims, elektrokardiogramoms modeliuoti ir pan. To pasekoje išaugo metodų, skirtų FIF formavimui, paklausa.

Darbe apžvelgiami paskutiniu metu labiausiai paplitę fraktalinio interpoliavimo programiniai įrankiai bei juose realizuoti algoritmai, siūloma nauja lokaliuoju koliažu grįsta fraktalinių interpoliacinių funkcijų generavimo procedūra.

2. FRAKTALINIŲ INTERPOLIACINIŲ FUNKCIJŲ GENERAVIMO YPATUMAI IR JŲ ANALIZĖ

Šiame skyriuje apžvelgsime priemones, skirtas FIF generavimui, jų problemas bei tobulinimo sprendimus. Pateiksime eksperimentų rezultatus bei jų analizę. Dar prieš pasirenkant programinę įrangą, fiksuokime eksperimento tikslą bei detalizuokime jį veiksmų seka.

Tarkime, turėdami pradinį objektą arba diskretizuotą jo analogą – duomenų aibę, mes norime atlikti kiek galima tikslesnę analizę, t.y. rasti FIF, geriausiai atspindinčią duomenis. Tai gana sudėtingas uždavinys, nes rezultatas, kaip jau buvome užsiminę, priklauso nuo kelių veiksnių, tokių

kaip duomenų aibės atrankos, vertikalųjį mastelį keičiančių koeficientų parinkimo bei visos interpoliavimo metodologijos. Bendru atveju, turėtume realizuoti tokią veiksmų seką:

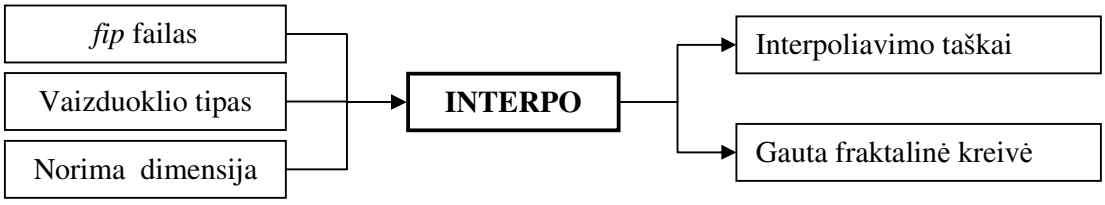
- Pradinio objekto diskretizavimas arba turimų diskrečių duomenų įvedimas;
- Lokaliojo koliažo organizavimas;
- FIF generavimas, remiantis sukauptais duomenimis;
- FIF ir realaus duomenų šaltinio vidutinės kvadratinės paklaidos apskaičiavimas (vizualinis palyginimas).

Programinei įrangai parinkti sunkumų kelia dar ir pradinio objekto tipų įvairovė (geografiniai, ekonominiai, statistiniai, medicininiai ir kitokio pobūdžio duomenys). Atskirų tipų duomenų analizei priemonių sukurta nemažai (Fractan, Market Way Pro, Investor's Dream, SKO), bet vienos universalios priemonės, kurią būtų galima taikyti įvairiarūšei informacijai analizuoti, naudoti neteko. Mūsų numatytą veiksmų seką galima būtų atlikti fraktalinio modeliavimo paketu FRACTAL VISION, kurio sudėtyje galime rasti jau tam tikru universalumu pasižymintį interpoliavimo įrankį INTERPO.

Sekančiuose skyriuose būtent ir apžvelgsime šią programinę įrangą bei pateiksime įrankio INTERPO interpoliavimo interaktyvumui alternatyvą – 1.3 skyriaus idėjomis paremto bei lokaliojo koliažo teorema grindžiamo interpoliavimo metodo programinę realizaciją, kurioje bus pasiūlyta ne tik nauja vertikalųjį mastelį keičiančio d_i parinkimo idėja, bet ir būdas atsižvelgti į duomenų fraktališkumą interpoliavimo metu.

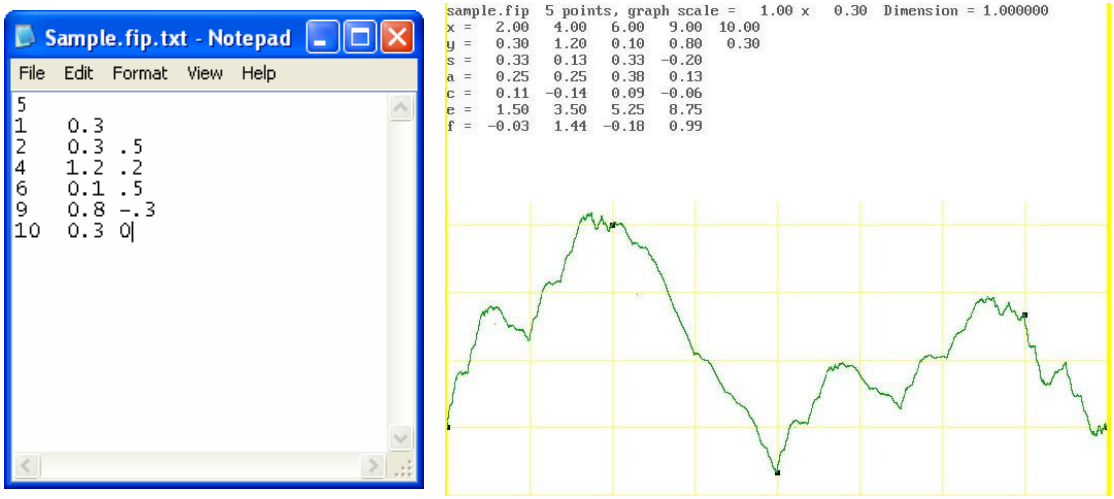
2.1. INTERPO – FRAKTALINIO INTERPOLIAVIMO PRIEMONĖ

Įrankis INTERPO – fraktalinio modeliavimo paketo FRACTAL VISION (2 PRIEDAS) sudėtinė dalis, skirta fraktaliniam interpoliavimui tirti. Startuojant programą, pirmiausiai nurodomas vaizduoklio tipas, po to *fip* formato duomenų failas, su visa reikalinga interpoliavimui informacija, bei fraktalinė (pasirinktina) dimensija. Priemonė ekrane pateikia interpoliuojamą duomenų aibę su kitais įvestais parametrais bei interpoliavimo taškais apribotoje srityje gautą fraktalinę kreivę.



2.1 pav. Interpoliavimo įrankio INTERPO veikimo schema

Duomenų faile nurodoma interpoliavimo taškų kiekis, vaizdavimo tinklelio skalė, jau minėtų taškų koordinatės bei vertikalųjį mastelį įtakojantys parametrai. Pagal pateiktus duomenis, papildomai nurodę dimensiją, gausime atitinkamą fraktalinę kreivę (FIF grafiką):



2.2 pav. Duomenų aibės (Sample.fip) interpoliavimo su INTERPO rezultatas, kai *D*=1,00

2.2 paveiksle pavaizduoto fip failo šifras būtų toks: norima interpoliuoti per 5 taškus, kai vaizdavimo tinklelio žingsnis abscisių ašies kryptimi yra 1, ordinačių – 0.3. Interpoliavimo taškų koordinatės: T_1=(2;0,3), T_2=(4;1,2), T_3=(6;0,1), T_4=(9;0,8) ir T_5=(10;0,3), o vertikalųjį mastelį tarp duotųjų taškų nusakantys parametrai – s_{12}=0,5, s_{23}=0,2, s_{34}=0,5, s_{45}=-0,3; čia s_{ab} – vertikalusis mastelis tarp *a*-tojo ir *b*-tojo taškų.

Tenka pastebėti, kad universalusis (eksperimento duomenų atžvilgiu) įrankis INTERPO turi keletą, eksperimentų efektyviam atlikimui trukdančių, apribojimų bei akivaizdžių vartotojo sąsajos trūkumų, kuriuos ir pabandysime apžvelgti.

2.1.1. INTERPO RIBOTUMAI

Išskirsime keturis pagrindinius INTERPO apribojimus (trūkumus):

1. Pradinė Interpo.c versija riboja duomenų aibės dydį, leisdama interpoliuoti ne daugiau kaip per 100 taškų. Norint atlikti eksperimentą su daugiau duomenų, tenka atrankos būdu gauti duomenų aibės poaibį.
2. Interaktyvi duomenų atranka. Fraktalinės interpoliacinės funkcijos (FIF) atitikimas realiai situacijai bus tuo tikslesnis, kuo tinkamiau bus parinkti interpoliavimo taškai. Kaip juos parinkti? Būtų idealu, kad tai būtų kraštiniai dalių, panašių į visumą, taškai. Jei vizualiai

nustatyti panašumą sunku, galima parinkti kas 10-tą, kas 50-tą ar 500-tąjį tašką, priklausomai nuo to, kiek duomenų turime apskritai, ir stebėti generuojamų FIF kaitą.

3. FIF tinklelio parametrų nurodymas. Tinklelį parenkame priklausomai nuo duomenų aibės taškų koordinatinių pokyčių (tinkamas parinkimas leis matyti visą interpoliacinę kreivę, o ne tik jos dalį).
4. Interaktyvus duomenų fraktalinės dimensijos įvertinimas.

Plačiau apžvelkime ketvirtąjį punktą, itin nepatogų vartotojui bei trukdantį efektyviems eksperimentams atlikti.

2.1.2. INTERPOLIUOJAMŲ DUOMENŲ FRAKTALINĖS DIMENSIJOS ĮVERTINIMAS

Pasirinkta fraktalinės dimensijos reikšmė apsprendžia FIF kreivės vertikalų „grubumą“, o grįžtant prie parametro s_{ab} prasmės (2.1 skyrelis), galima būtų teigti, jog jis vizualiai apsprendžia kreivės „grubumą“ tarp a-tojo ir b-tojo taškų. Pavyzdžiui, jei pirmojo interpoliavimo intervalo parametras s = 0,5, o visų kitų – 0,2, tai kraštinė kairioji kreivės dalis bus aukštesnė ir „grubesnė“ nei likusioji dalis. Neigiama parametro s_{ab} reikšmė nusako atvirkščią kreivės linkį horizontaliosios ašies atžvilgiu.

Įsigilinus į INTERPO įrankio kodą C kalba, buvo išsiaškinta, jog interpoliacinė kreivė sudaroma, 1.3 skyriuje aprašytu metodu skaičiuojant pražulniųjų transformacijų koeficientus a_i, c_i, e_i ir f_i ((1.30-(1.33) formulės), o vertikalųjį mastelį keičiantis koeficientas d_i apskaičiuojamas pagal tokią formulę:

$$d_i = s_{(i+1)(i+2)} \cdot \frac{(N-1)^{D-1}}{\sum_{j=0}^{N-2} \left| s_{(j+1)(j+2)} \right|}, i = 0,1,...N-2; \qquad \textbf{(2.1)}$$

čia s_{ab} – faile nurodytas FIF kreivės grubumas tarp a-tojo ir b-tojo taškų.

Jei parametro s_{ab} reikšmės nežinome (o taip dažniausiai ir būna), INTERPO įrankio autorius pradiniam eksperimentui pataria parinkti visas s_{ab} reikšmes lygias 0,5, vėliau, vizualiai įvertinus gautą rezultatą, reikšmes tikslinti. Kitaip tariant, čia ir išryškėja anksčiau jau ne kartą minėtas INTERPO interaktyvumas, ko pasekoje, negalime momentaliai gauti reikiamos FIF. Šios problemos sprendimui tolimesniuose skyreliuose bus pasiūlytas naujas ir kur kas patogesnis sprendimas su visa programine jo realizacija.

2.2. KOLIAŽU GRĮSTA FRAKTALINIŲ INTERPOLIACINIŲ FUNKCIJŲ GENERAVIMO PROCEDŪRA

2.2.1. STRUKTŪRINĖ SCHEMA

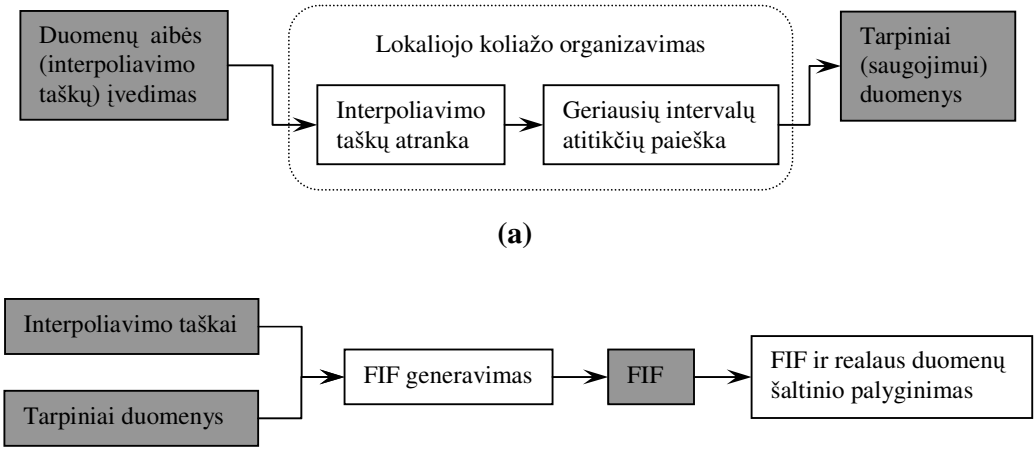
Pagrindinė procedūros idėja yra tokia: fraktalinė interpoliacinė funkcija (FIF) gaunama kaip afiniųjų transformacijų ω_i ($i = 0,1,...,N$) projekcija, kai kiekviena transformacija „veikia“ jau ne visą pradinį objektą (aibę) *T*, o tam tikra tvarka atrinktus jo poaibius.

Tarkime, turime interpoliavimui skirtą duomenų aibę:

$$\left\{(x_i, F_i) \in \mathrm{R}^2 \middle| i = 0,1,...,N\right\}, \tag{2.2}$$

kur $F_i = F(x_i)$, $x_0 < x_1 < ... < x_N$, o N yra pakankamai didelis skaičius.

Tikslas – gauti šią aibę interpoliuojančią FIF, kuri būtų kuo „artimesnė“ (Hausdorfo metrikos *h* arba vidutinės kvadratinės paklaidos (VKP) prasme) realiam duomenų šaltiniui. Procedūrą (iš esmės) sudaro dvi dalys: lokaliojo koliažo organizavimas, kaupiant tarpinę informaciją, bei FIF generavimas (2.3 pav.).



2.3 pav. Struktūrinė FIF generavimo schema:
(a) lokaliojo koliažo organizavimas; (b) FIF generavimas

2.2.2. SU KOLIAŽU SUSIJUSIŲ INTERPOLIAVIMO TAŠKŲ ATRANKA

Iš gausybės duomenų atrankos metodų (atsitiktinės duomenų atrankos, atrankos pagal pasirinktą požymį, intervalo dalijimo pusiau ir kt.) siūlomos idėjos (procedūros) įgyvendinimui buvo pasirinktas

vienas elementariausių matematinių duomenų atrankos metodų - aritmetinės progresijos principu paremta atranka, kai iš duomenų aibės $\{(x_i, F_i) \in \mathrm{R}^2 | i = 0,1,...,N\}$ atrenkamas jos poaibis

$$\{(x_k', F_k') | x_k' = x_{\lambda \cdot k}, F_k' = F_{\lambda \cdot k}; k = 0,1,...,[N/\lambda], \lambda \in \{2,3,...\}\}; \quad \textbf{(2.3)}$$

čia $[x]$ žymi sveikąją skaičiaus x dalį.

Atrinktų interpoliavimo taškų skaičius $[N/\lambda]+1$, kaip matome, priklauso nuo pasirinkto žingsnio λ, kurį didindami (mažindami), gausime mažiau (daugiau) duomenų aibės elementų. Paprastesnei FIF generavimo procedūros realizacijai gauti, atrankos parametrui λ uždėtas apribojimas - $\lambda = 2^n$ ($n = 1,2,...$); be to, kaip parodė vėlesni tyrimai, procedūra „dirba“ efektyviai, kai $[N/\lambda] >> 2$. Pastebėsime, jog atsitiktinis žingsnio λ parinkimas nevisada sudaro sąlygas interpoliavimo metu gauti siektiną rezultatą.

2.2.3. KOLIAŽO ORGANIZAVIMAS

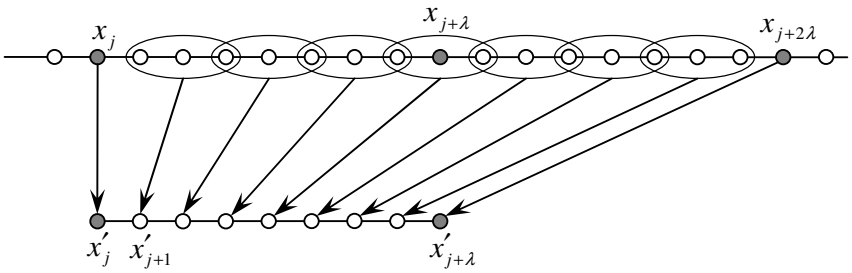
Imkime duomenų aibę $S = \{(x_i, F_i) \in \mathrm{R}^2 | i = 0,1,...,N\}$, kuriai jau fiksuotas su koliažo organizavimu susijęs interpoliavimo taškų atrankos parametras (žingsnis λ), $2 \le \lambda \le [N/2]$.

Koliažo esmė – sukonstruoti $IFS\{\mathrm{R}^2; \omega_1, \omega_2, ..., \omega_N\}$, kurios atraktorius būtų „artimas“ (Hausdorfo metrikos h prasme) nagrinėjamai duomenų aibei S. Tai reiškia, kad turime rasti suspaudžiančiųjų afiniųjų transformacijų $\omega_1, \omega_2, ..., \omega_N$ rinkinį tokį, kad aibės S vaizdų, gautų po pirmosios iteracijos, sąjunga („koliažas“) būtų kuo „artimesnis“ pačiai aibei S, t.y. $h(S, \bigcup_{i=1}^{N} \omega_i(S)) < \varepsilon$. Realiems duomenims tikslinga taikyti lokalųjį koliažą, charakterizuojamą tuo, kad koliažas konstruojamas ne iš visos aibės S kopijų, bet iš atskirų aibės S fragmentų kopijų.

Pirmiausia, visi galimi duomenų aibės S poaibiai, atitinkantys segmentus $[x_j, x_{j+2\lambda}]$ ($j = 0,1,...,\lambda([N/\lambda]-2)$), sudarytus iš $(2\lambda+1)$ gretimų taškų, transformuojami, siekiant gauti dvigubai mažesnius segmentus $[x_j', x_{j+\lambda}']$ atitinkančius poaibius (2.4 pav.):

$$\begin{cases} F_j' = F_j, \\ F_{j+t}' = \dfrac{1}{3} \displaystyle\sum_{\tau=-1}^{1} F_{j+2t+\tau}, t = 1,2,...,\lambda-1, \\ F_{j+\lambda}' = F_{j+2\lambda} \end{cases} \quad \textbf{(2.4)}$$

su visais $j = 0,1,...,\lambda([N/\lambda]-2)$.



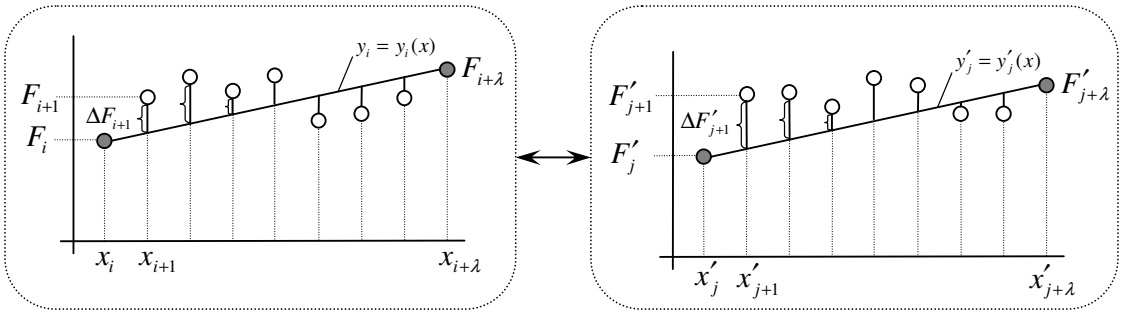
2.4 pav. Duomenų aibės *S* poaibių transformavimas į dvigubai mažesnius poaibius

Antruoju žingsniu, realizuojama pagrindinė lokaliojo koliažo idėja – kiekvienam segmentą $[x_i, x_{i+\lambda}]$ $(i = 0,1,...,[N/\lambda]-1)$ atitinkančiam duomenų aibės poaibiui ieškomas geriausiai (vidutinės kvadratinės paklaidos δ prasme; (2.5) išraiška) jį atitinkantis transformuotas poaibis, susijęs su segmentu $[x'_j, x'_{j+\lambda}]$, $j \in \{0,1,...,\lambda([N/\lambda]-2)\}$, (2.5 pav.), t.y.

$$\delta_{ij} = \sqrt{\frac{1}{\lambda-1}\sum_{t=1}^{\lambda-1}\left[\Delta F_{i+t} - k_i \Delta F'_{j+t}\right]^2} \to \min, \qquad \textbf{(2.5)}$$

kai $\Delta F_{i+t} = F_{i+t} - y_i(x_{i+t})$ ir $\Delta F'_{j+t} = F'_{j+t} - y'_j(x'_{i+t})$; parametras k_i įvedamas tam, kad „pritraukti“ minėtus poabius arčiau vienas kito, ir yra apskaičiuojamas remiantis formule:

$$k_i = \begin{cases} \sum_{t=1}^{\lambda-1} \Delta F_{i+t} \cdot \Delta F'_{j+t} \Big/ \sum_{t=1}^{\lambda-1} (\Delta F'_{j+t})^2, \text{ kai } \sum_{t=1}^{\lambda-1} (\Delta F'_{j+t})^2 \neq 0, \\ 0, \text{ kitu atveju.} \end{cases} \qquad \textbf{(2.6)}$$



2.5 pav. Lokaliojo koliažo organizavimas

Pastebėsime, jog vietoj apskaičiuotų parametro k_i reikšmių (dėl geresnio generuojamos FIF konvergavimo į duomenų šaltinį) naudojamos reikšmės $\hat{k}_i$:

$$\hat{k}_i = \begin{cases} -1, \ kai\, k_i \le -1, \\ k_i, \ kai\ |k_i| \le 1, \\ 1, \ kai\ k_i \ge 1. \end{cases} \quad \textbf{(2.7)}$$

Taigi, tarpinius saugojimui skirtus duomenis sudaro rinkinukai

$$\{< i, j, \hat{k}_i > | i = 0,1,...,[N/\lambda]-1\}. \quad \textbf{(2.8)}$$

Turint pradinę interpoliuojamų taškų aibę $S = \{(x_i, F_i) \in \mathrm{R}^2 | i = 0,1,...,N\}$ ir tarpinius duomenis, galime generuoti FIF.

2.2.4. FRAKTALINĖS INTERPOLIACINĖS FUNKCIJOS GENERAVIMAS

Pastebėsime, jog kiekvieną atitiktį (tarp duomenų aibės poaibio ir geriausiai jį atitinkančio transformuoto poaibio) realizuoja afinioji transformacija. Kitaip tarus, turime lokaliąją $IFS\{\mathrm{R}^2; \omega_1, \omega_2, ..., \omega_N\}$, sudarytą iš $[N/\lambda]-1$ afiniųjų transformacijų ω_i $(i = 1,2,...,N)$, turinčių pavidalą

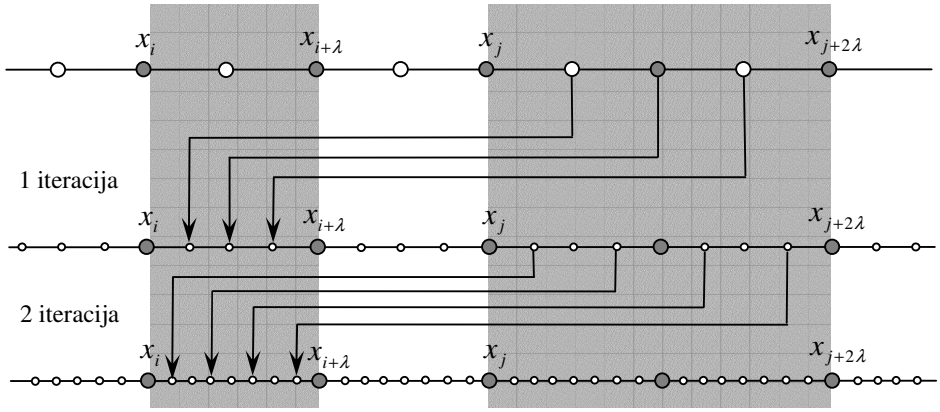
$$\omega_i = \omega_i(x,F) = \begin{pmatrix} a_i & 0 \\ c_i & \hat{k}_i \end{pmatrix} \begin{pmatrix} x \\ F \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \end{pmatrix}; \quad \textbf{(2.9)}$$

čia: $a_i = \dfrac{x_{i+\lambda} - x_i}{x_{j+2\lambda} - x_j} = \dfrac{1}{2}$, $c_i = \dfrac{F_{i+\lambda} - F_i}{x_{j+2\lambda} - x_j} - \hat{k}_i \dfrac{F_{j+2\lambda} - F_j}{x_{j+2\lambda} - x_j}$,

$e_i = \dfrac{x_{j+2\lambda} x_i - x_j x_{i+\lambda}}{x_{j+2\lambda} - x_j}$, $f_i = \dfrac{x_{j+2\lambda} F_i - x_j F_{i+\lambda}}{x_{j+2\lambda} - x_j} - \hat{k}_i \dfrac{x_{j+2\lambda} F_j - x_j F_{j+2\lambda}}{x_{j+2\lambda} - x_j}$,

su visais $i = 0,1,...,[N/\lambda]-1$.

Taigi, kiekviena afinioji transformacija ω_i $(i \in \{0,1,...,[N/\lambda]-1\})$ atvaizduoja duomenų aibės *S* poaibį, atitinkantį segmentą $[x_j, x_{j+2\lambda}]$, į dvigubai mažesnį S poaibį, atitinkantį segmentą $[x_i, x_{i+\lambda}]$, esant atitikčiai $< i, j, \hat{k}_i >$ (2.6 pav.).



2.6 pav. FIF generavimas

Kaip matome (2.6 pav.), pradinė interpoliuojamų taškų aibė yra $S = \left\{(x_i, F_i) \in \mathrm{R}^2 \middle| i = 0,1,...,N\right\}$ ir interpoliavimo taškų atrankos parametras (žingsnis λ) $\lambda = 2$. Pirmosios iteracijos metu afinioji transformacija ω_i, nusakoma rinkiniu $< i, j, \hat{k}_i >$, „veikia“ taškus $\{(x_{j+\tau}, F_{j+\tau}) \middle| \tau = 1,2,..., j + 2\lambda - 1\}$, ko rezultate gauname $(2\lambda - 1)$ taškus segmente $[x_i, x_{i+\lambda}]$ $(i = 0,1,...,[N/\lambda] - 1)$. Turėdami vaizdą $\omega_i(x_{j+\tau}, F_{j+\tau})$ $(\tau \in \{1,2,..., j + 2\lambda - 1\})$, tikriname, ar pradinėje duomenų aibėje *S* yra taškas su abscise $x_{j+\tau}$: jei toks taškas yra, tolimesnėse iteracijose jis nėra apdorojamas. Toliau iteracijos vykdomos, remiantis tokia pagrindine FIF generavimo idėja – kadangi taškai yra vaizduojami į visus segmentus $[x_i, x_{i+\lambda}]$ $(i = 0,1,...,[N/\lambda] - 1)$, tai pasipildo ir segmentas $[x_j, x_{j+2\lambda}]$, kurį sudaro dvigubai mažesni segmentai $[x_j, x_{j+\lambda}]$ bei $[x_{j+\lambda}, x_{j+2\lambda}]$. Gautieji nauji taškai toliau „veikiami“ afiniosios transformacijos ω_i $(i = 0,1,...,[N/\lambda] - 1)$ ir, po *n* iteracijų, jeigu aibės *S* taškų abcisės sutampa su 2^n-tuoju $(n = 1,2,...)$ ekrano O*x* ašies tašku (pikseliu), procesas konverguoja ir gauname fraktalinę interpoliacinę funkciją.

Reikia pastebėti, jog gautoji FIF nėra taškų aibę *S* interpoliuojanti funkcija, nes afinioji transformacija ω_i (2.9 formulė) užtikrina tik vaizduojamo segmento $[x_j, x_{j+2\lambda}]$ kraštinių taškų x_j, $x_{j+2\lambda}$ vaizdų (atitinkamai $\omega_i(x_j), \omega_i(x_{j+2\lambda})$) ir segmento $[x_i, x_{i+\lambda}]$, į kurį vaizduojama, kraštinių taškų x_i, $x_{i+\lambda}$ ordinačių sutapimą ((1.29) sąlygos). To pasekoje, po generavimo procedūros taškai $\{(x_{i+\tau}, F_{i+\tau}) \middle| \tau = 1,2,...,i + \lambda - 1\}$ $(i = 0,1,...,[N/\lambda] - 1)$ nebūtinai priklauso sugeneruotai kreivei (FIF), kuri šiuo atveju yra tik atrinktų interpoliavimo taškų aibę (2.3 formulė) interpoliuojanti FIF. Galutinę FIF, interpoliuojančią taškų aibę *S*, galime rasti perskaičiuojant turimų FIF taškų, priklausančių segmentui $[x_i, x_{i+1}]$ $(i = 0,1,...,N - 1$; $(N+1)$ – interpoliuojamų taškų skaičius), ordinates:

$$\overline{FIF}(g) = FIF(g) + \frac{a(k-g)+bg}{k}, \qquad a = F_i - FIF(0),\ b = F_{i+1} - FIF(k); \tag{2.10}$$

čia: $FIF(g)$ – turimos fraktalinės interpoliacinės funkcijos g-toji reikšmė segmente $[x_i, x_{i+1}]$;

$\overline{FIF}(g)$ – galutinės fraktalinės interpoliacinės f-jos g-toji reikšmė segmente $[x_i, x_{i+1}]$;

$g = 0,1,...,k$, kur $(k+1)$ – segmente $[x_i, x_{i+1}]$ sugeneruotų taškų skaičius ($k = 2^n$, n – generavimo procedūros metu atliktų iteracijų skaičius).

3. EKSPERIMENTO REZULTATŲ ANALIZĖ

Ankstesniuose skyriuose aprašytai fraktalinių interpoliacinių funkcijų (FIF) generavimo procedūrai, grindžiamai lokaliojo koliažo principu, sukurta programinė realizacija (priemonė), turinti patogią fraktalinio modeliavimo aplinką bei sudaranti puikias sąlygas eksperimentams atlikti. Pirmiausia pateiksime priemonės (įrankio) aprašą su išsamiomis instrukcijomis vartotojui, o sekančiu žingsniu apžvelgsime atlikto eksperimento rezultatus.

3.1. PROGRAMINĖ REALIZACIJA IR INSTRUKCIJA VARTOTOJUI

Programos (priemonės) anotacija bei reikalavimai jos įdiegimui pateikiami 2.1 lentelėje.

2.1 lentelė

Programinės realizacijos anotacija ir diegimo reikalavimai

Produkto pavadinimas	FIF analizė
Kodo sintaksės kalba	.NET C#
Autorius	magistr. Edvinas Medišauskas
Sukūrimo data	2007 m. gegužės mėn.
Reikalavimai	
Operacinė sistema	Windows Server 2003 Microsoft Windows 2000 (rekomenduojama Service Pack 2) Windows XP Professional and Home Edition
Reikalingi komponentai	.NET Framework 1.1 - Microsoft Windows® komponentas, reikalingas kurti ir vykdyti Windows aplikacijas

Pirmiausia, priemonės funkcionalumui atskleisti, apžvelkime pagrindinius modulius (2.2 lentelė), vartotojo formas (2.3 lentelė), jų paskirtį ir žinoma jų tarpusavio sąveikavimą – programos panaudojimo atvejų modelį (2.7 pav.).

2.2 lentelė

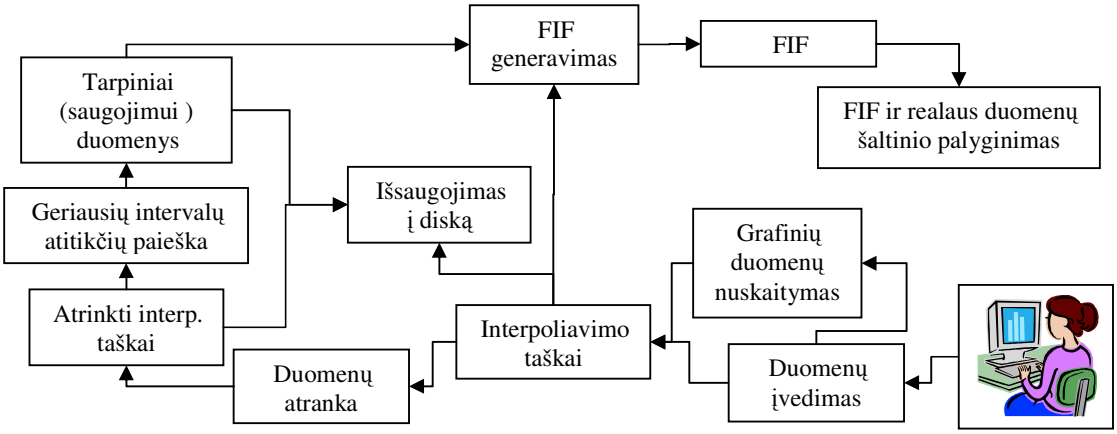
Programinės realizacijos pagrindiniai moduliai, jų paskirtis

Modulio pavadinimas	Paskirtis
Diskretizavimas	Darbo su paveikslu funkcijos: • kreivės ordinatės nuskaitymas; • esminės informacijos paveiksle paruošimas vartotojui; • informacinių ženklų išvedimas paveikslo plote.
Funkcijos	Bendro panaudojimo funkcijos: • taškų masyvų, vidinių programos struktūrų funkcijos; • failo kelio diske funkcijos.
Skaičiavimai	Pagrindinės skaičiavimo procedūros: • iteruotųjų funkcijų sistemos parametrų skaičiavimas; • lokaliojo koliažo organizavimui reikalingi skaičiavimai.
Statikai	Programos nustatymai, bei metodai jiems lengvai prieiti.
IO	Įvesties/išvesties funkcijos: • failo, direktorijos, spalvos įvedimo dialogai; • taškų masyvo ir tarpinių duomenų tekstinių failų, bei programos nustatymų (XML formate) nuskaitymas/išsaugojimas; • taškų masyvo, pranešimo išvedimas į vartotojo sąsają.

2.3 lentelė

Programinės realizacijos pagrindinės vartotojo formos, jų paskirtis

Vartotojo forma	Paskirtis
Duomenų aibės paruošimas	• Išorinių duomenų nuskaitymas; • grafinių duomenų diskretizavimas; • su koliažo organizavimu susijusi duomenų atranka; • sudarytų duomenų aibių išsaugojimo galimybė.
Skaičiavimų monitorius	• Geriausių intervalų atitikčių paieška; • sukauptos informacijos išsaugojimas į tekstinį failą.
FIF generatorius	• Fraktalinės interpoliacinės funkcijos (FIF) generavimas; • gautosios FIF analizė.
Nustatymai	Programos nustatymų langas.

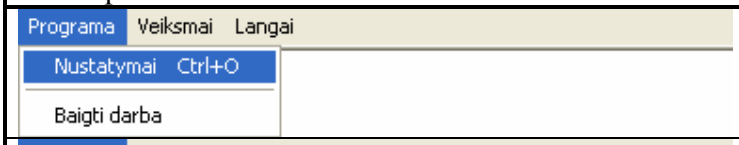
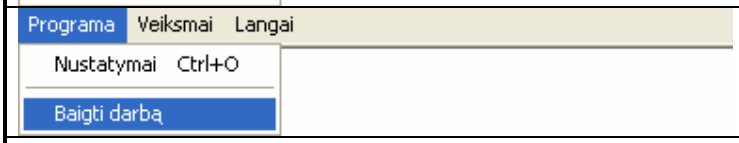
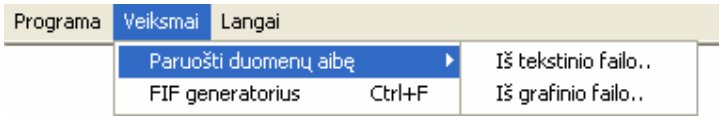
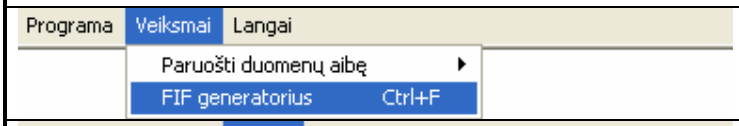
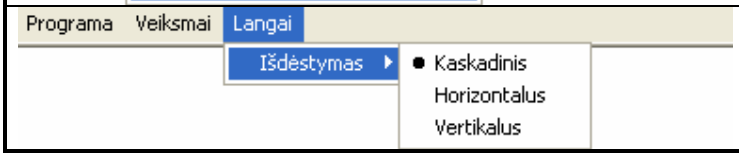


2.7 pav. Programinės realizacijos panaudojimo atvejų modelis

Pastebėsime, jog visas įrankio (paketo) funkcionalumas sutelktas jo moduliuose ir vidinėse formose, t.y. dirbama centralizuotai, nenaudojant jokių pašalinių programų. Apžvelkime pagrindinį paketo meniu:

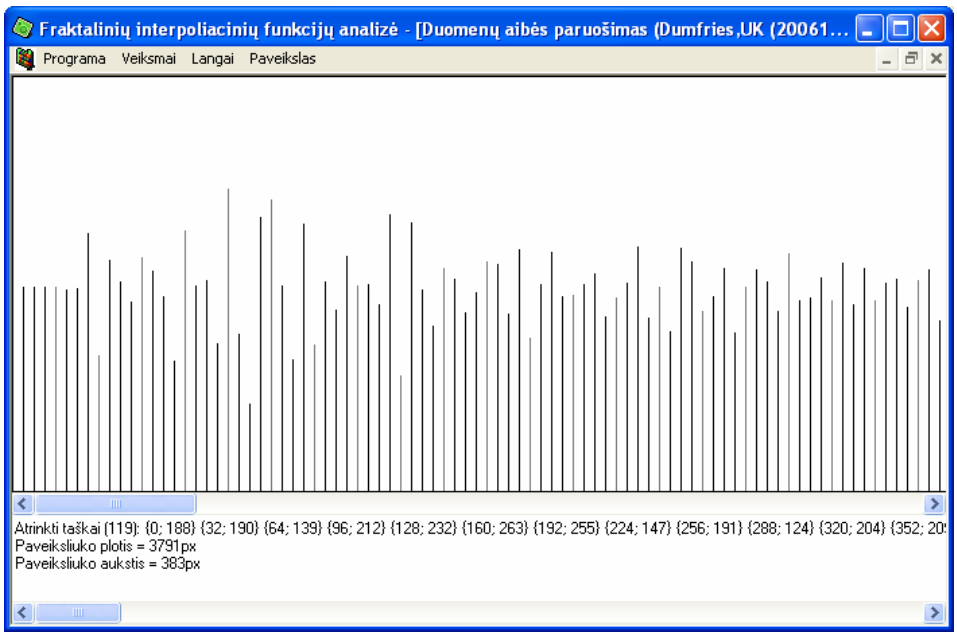
2.4 lentelė

Programinės realizacijos pagrindinis meniu

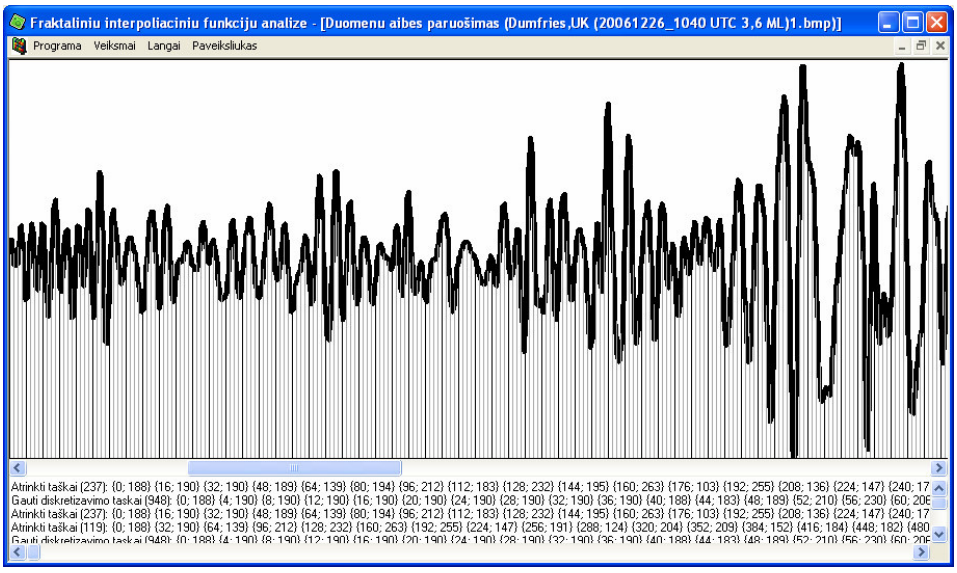
Meniu punktas	Paskirtis
	Iškviečia programinės realizacijos nustatymų langą (2.13 pav.).
	Baigia darbą, uždaro programą.
	Duomenų aibės (interpoliavimo taškų) įvedimas iš iš tekstinio (*.txt) arba grafinio (*.bmp, *.jpg) failų.
	Iškviečia FIF generatoriaus langą (2.12 pav.)
	Programos vidinių langų išdėstymo nustatymas.

Vartotojas pradinę duomenų aibę (interpoliavimo taškus, 2.2 formulė) gali įvesti dviem būdais: rikiuotą arba nerikiuotą taškų aibę iš tekstinio failo (**.txt* formato, 2.9 pav.) arba, tiesiog, iš grafinio failo (**.bmp*, **.jpg* formatai, 2.10 pav.). Pateikus duomenų aibę grafiniu pavidalu, pirmiausia paveikslas diskretizuojamas, realizuojant 2.2.2 skyriuje minėtą aritmetine progresija paremtą taškų atranką, bet tai tik pradinei duomenų aibei atrinkti (ne duomenų aibės poaibiui!). Visai galimai informacijai išgauti (pagal nutylėjimą) pasirenkamas taškų skaičius, lygus pikseliniam grafinio failo pločiui. Tokiu būdu įvedame pradinę duomenų aibę, su kuria vėliau dirbame.

Sekančiu žingsniu vykdoma su lokaliojo koliažo organizavimu susijusi pirminės duomenų aibės (interpoliavimo taškų) atranka, kurios rezultatas – vėlgi ta pačia atranka (kaip ir diskretizavimo atveju) gautas duomenų aibės poaibis. Čia vartotojui leidžiama pasirinkti (2.3) formulėje esantį žingsnio parametrą λ, ir tokiu būdu atrenkami $\lfloor N/\lambda \rfloor + 1$ taškai, kur N – interpoliavimo taškų aibės galia.



2.9 pav. Duomenų aibės paruošimo iš tekstinio failo langas

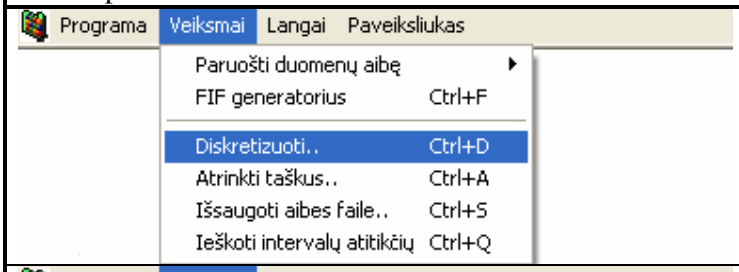
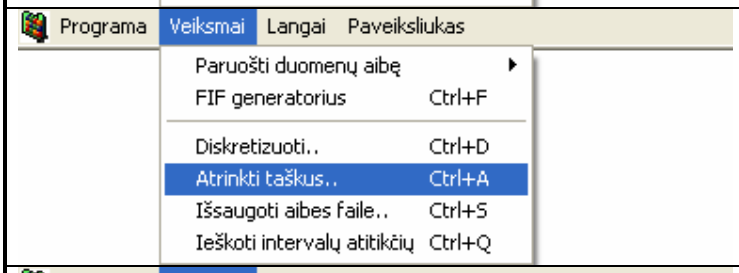
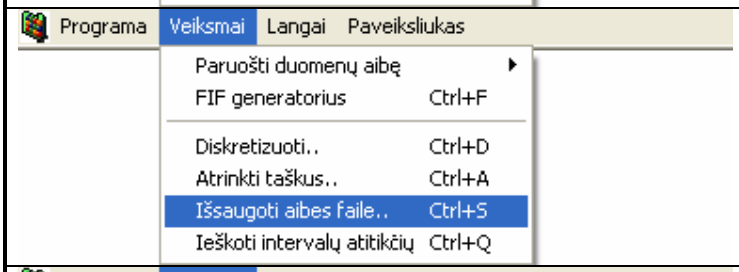
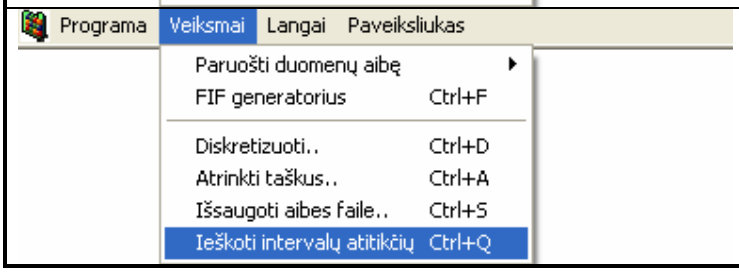


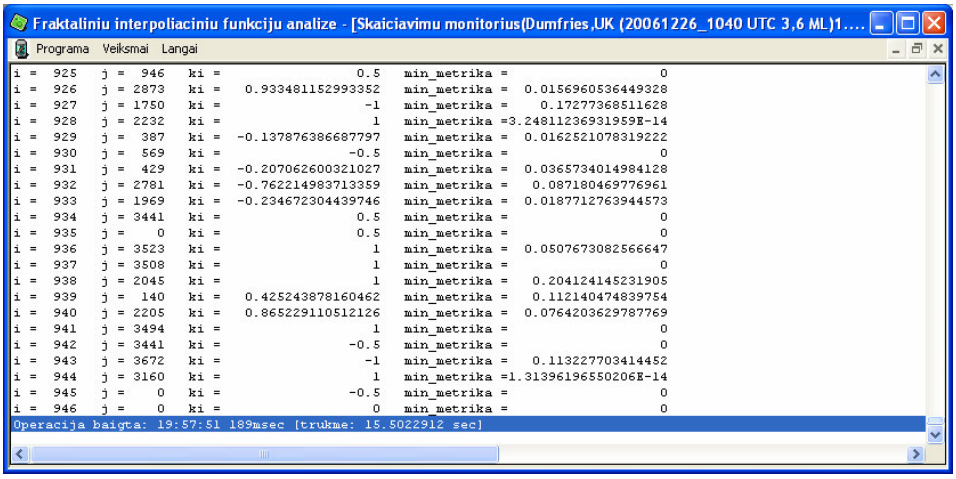
2.10 pav. Duomenų aibės paruošimo iš grafinio failo langas

Duomenų aibės paruošimo langas, kaip ir kiti vidiniai langai, turi savo papildomų meniu punktų, kurie, esant aktyviam langui, integruojami pagrindinėje komandų juostoje (2.5 lentelė):

2.5 lentelė

Duomenų aibės paruošimo lango meniu

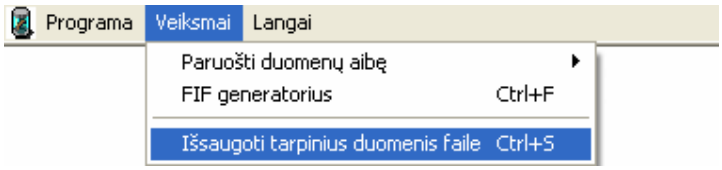
Meniu punktas	Paskirtis
	Grafinio paveikslo diskretizavimo komanda. Įvedant pradinę duomenų aibę iš tekstinio failo ši komanda būna neaktyvi.
	Su lokaliojo koliažo organizavimu susijusios interpoliavimo taškų atrankos komanda.
	Išsaugo sudarytas interpoliuojamų taškų ir atrinktųjų taškų aibes į lokalųjį kietąjį diską tekstiniu formatu.
	Atidaro skaičiavimo monitoriaus langą (2.11 pav.) ir pradeda geriausių intervalų atitikčių paiešką (2.2.3 skyrius).



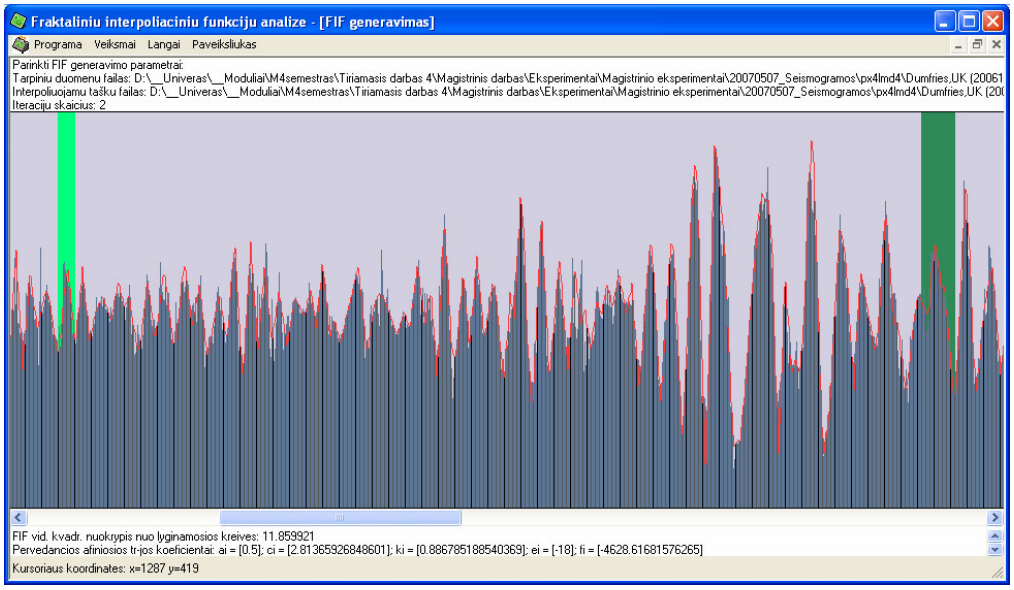
2.11 pav. Skaičiavimo monitoriaus langas

2.6 lentelė

Skaičiavimo monitoriaus lango meniu

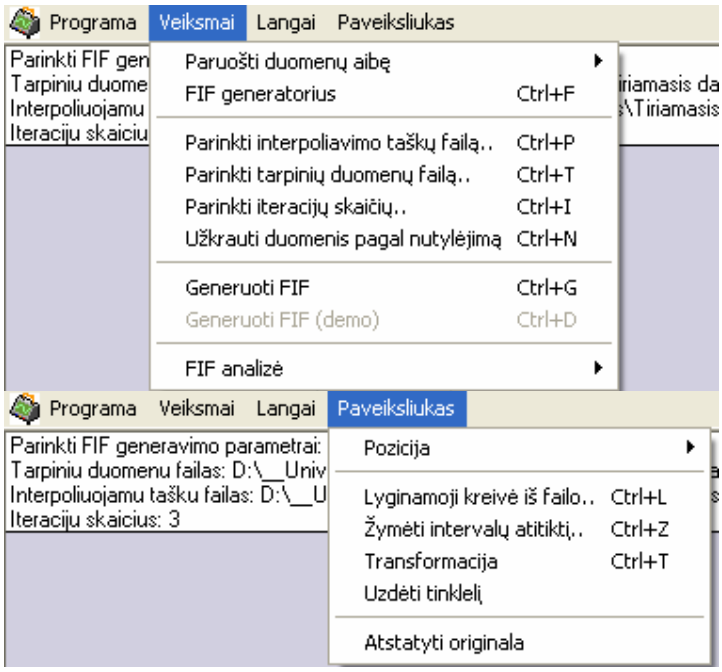
Meniu punktas	Paskirtis
	Išsaugo geriausių intervalų atitikčių paieškos metu rastas atitiktis (2.2.3 skyrius) į lokalųjį kietąjį diską tekstiniu formatu.

Taigi, apžvelgėme paruošiamuosiuos paketo langus, kurių rezultatai – informacija tekstiniuose failuose. Sukauptieji duomenys (interpoliavimo taškai, geriausių intervalų atitikčių rinkinukai) pateikiami vienam iš pagrindinių paketo vidinių langų – FIF generatoriui (2.12 pav.), kuris teikia FIF generavimo bei jos analizės funkcijas.



2.12 pav. FIF generatoriaus langas

Gautos fraktalinės interpoliacinės funkcijos taškai (vaizdumo dėlei) piešiami vertikaliomis linijomis, kurios ilgis žymi konkretaus taško ordinatės dydį. Taipogi, spalvų pagalba išskiriami interpoliuojami taškai bei jų atrinktas poaibis, kuris, kaip jau buvo minėta, paruošiamas lokaliojo koliažo organizavimui. Gautosios FIF analizės priemonės – tai lyginamosios kreivės išvedimas ekrane (vizualinis palyginimas su generavimo rezultatu), atliekamo eksperimento geriausių intervalų atitikčių peržiūra, IFS transformacijų veikimo vizualizacija. Duomenų parinkimas, funkcijos FIF analizei pasiekiamos per FIF generatoriaus lango meniu (2.13 pav., 2.7 lentelė).



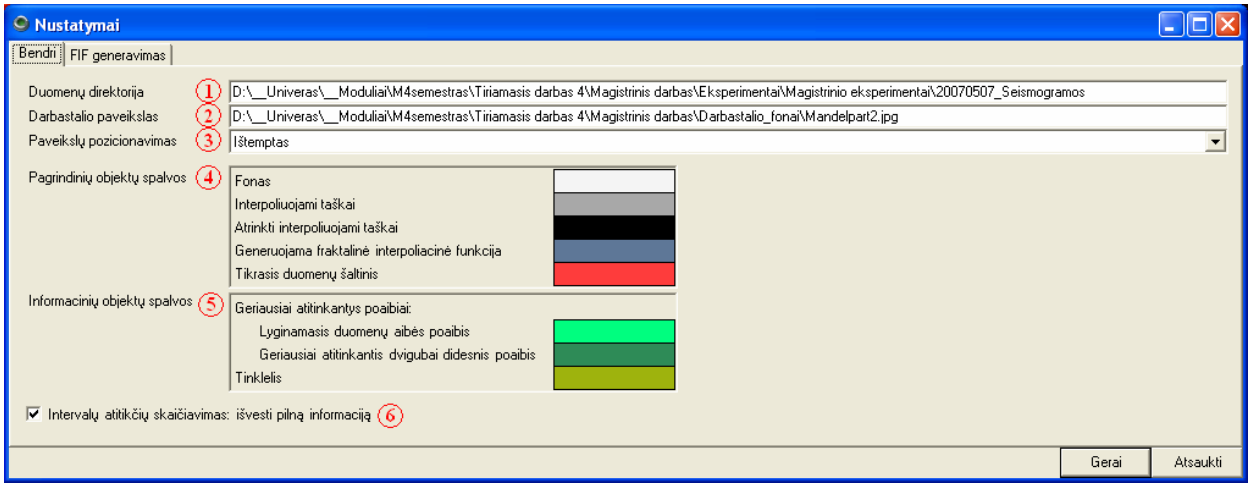
2.13 pav. FIF generatoriaus lango meniu

2.7 lentelė

FIF generatoriaus lango meniu

Meniu punktas	Paskirtis
Grupė „Veiksmai“	
„Parinkti interpoliavimo taškų failą..“	Parinkti interpoliavimo taškų failą (2.2 formulė), išsaugotą duomenų aibės paruošimo lange (2.5 lentelė)
„Parinkti tarpinių duomenų failą..“	Parinkti tarpinių duomenų failą, talpinantį geriausių intervalų atitikčių rinkinukus (2.8 formulė) bei išsaugotą duomenų aibės paruošimo lange (2.5 lentelė)
„Parinkti iteracijų skaičių..“	Parinkti FIF generavimo iteracijų skaičių (2.2.4 skyriaus pabaiga).
„Užkrauti duomenis pagal nutylėjimą..“	Įvesti duomenų failų, nurodytų toliau nagrinėjamame nustatymų lange (2.14 pav.), informaciją.
„Generuoti FIF“	Komanda, pradėti FIF generavimo procedūrą.
Grupė „Paveiksliukas“	
„Lyginamoji kreivė iš failo..“	Leidžia parinkti lyginamąją kreivę iš tekstinio failo, kurio struktūra analogiška interpoliavimo taškų failui, o gretimų taškų abscisių pokyčiai lygūs 1.
„Žymėti intervalų atitiktį..“	Leidžia vizualiai peržvelgti atliekamo eksperimento geriausių intervalų atitikčių rinkinukus (2.8 formulė).
„Transformacija“	Stebėti pervedančiosios afiniosios transformacijos (2.9 formulė) veikimą intervalų peržiūros metu.
„Uždėti tinklelį“	Paveikslo zonai uždedama kvadratinė gardelė su nurodytu pikseliniu žingsniu.
„Atstatyti originalą“	Atkuria originalų generavimo rezultatą, panaikinant analizės metu atliktus paveikslo pakeitimus.

Programinio įrankio nustatymų langas (2.13, 2.14 pav.) turi dvi dalis: bendriems programos nustatymams bei FIF generavimo parametrams įvesti (koreguoti). Kiekvieno nustatymo paskirtis pateikiama 2.8, 2.9 lentelėse.



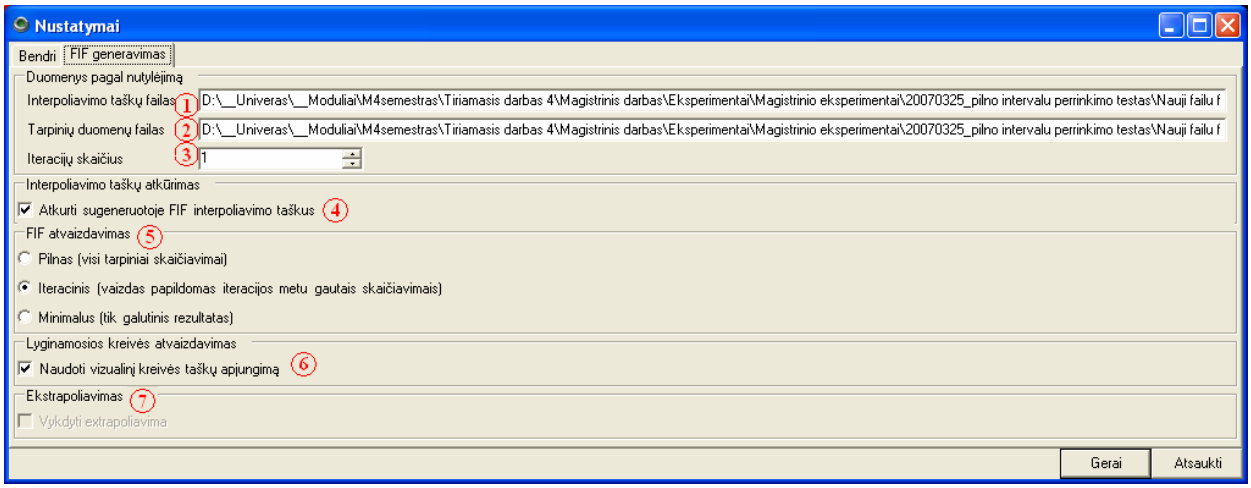
2.13 pav. Paketo nustatymų langas: Bendri programos nustatymai

2.8 lentelė

Paketo nustatymų langas: Bendri programos nustatymai

Nustatymo Nr.	Paskirtis
1	Nurodo darbinę duomenų direktoriją *
2	Pagrindinio lango fono paveikslas *
3	Nutylėtasis programos darbinių paveikslų pozicionavimas
4	Pagrindinių objektų spalvų parinkimas **
5	Informacinių objektų spalvų parinkimas **
6	Nurodo, ar reikalinga pilna skaičiavimų informacija, vykdant geriausių intervalų atitikčių paiešką (2.2.3 skyrius), skaičiavimo monitoriaus lange (2.11 pav.)

* - parinkimo dialogas iškviečiamas dvigubu pelės paspaudimu
** - parinkimo dialogas iškviečiamas paspaudus pelę keičiamos spalvos zonoje



2.14 pav. Paketo nustatymų langas: FIF generavimas

2.9 lentelė

Paketo nustatymų langas: FIF generavimas

Nustatymo Nr.	Paskirtis
1	Nurodo pagal nutylėjimą naudotiną interpoliavimo taškų failą FIF generavimo metu *
2	Nurodo pagal nutylėjimą naudotiną tarpinių duomenų failą FIF generavimo metu *
3	Nurodo pagal nutylėjimą naudotiną FIF generavimo iteracijų skaičių
4	Nurodo, ar bus taikoma tarpinės FIF korekcija (2.10 formulė), gaunant galutinę FIF
5	FIF generavimo proceso vizualizavimo tipas (neįtakoja galutinio rezultato)
6	Nurodo, ar parinktos lyginamosios kreivės taškams (sugeneruotos FIF analizės metu) bus taikoma vizualinė korekcija
7	Ekstrapoliavimo vykdymas (neaktyvus nustatymas, pilnai nerealizuotas)

* - parinkimo dialogas iškviečiamas dvigubu pelės paspaudimu

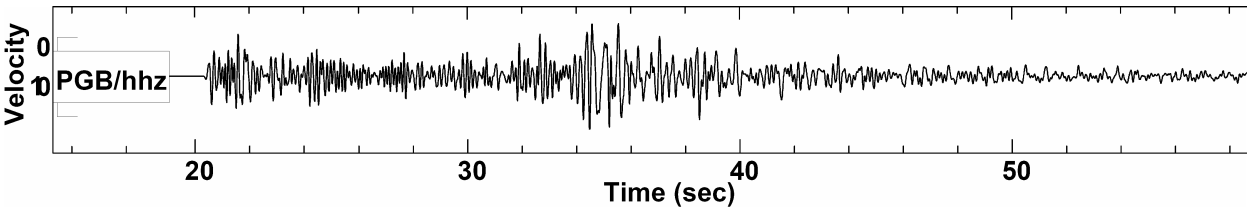
Apžvelgę programinės realizacijos galimybes bei jos panaudojimo specifiką, pereikime prie sekančio svarbaus žingsnio šiame skyriuje – ekperimento atlikimo bei gautų rezultatų analizės.

3.2. EKSPERIMENTAS

Tiriamojo darbo metu nagrinėjamos realaus pasaulio objektų (sistemų) modeliavimo problemos. Įvertinus fraktališkumo (t.y. „atsikartojimo savyje“) savybę, siūlomas vienas iš galimų sprendimų – koliažu grįsta fraktalinių interpoliacinių funkcijų generavimo procedūra (2.2 skyrius), kurios efektyvumui patikrinti atliksime eksperimentą.

2006 metų gruodžio 26 dieną 10 valandą 40 minučių (UTC laiku) pietinėje Škotijos dalyje tarp Damfrio (Dumfries) ir Galovėjaus (Galloway) miestų nugriaudė rekordinis šiame regione 3,5 balo pagal Richterio (Richter) skalę stiprumo žemės drebėjimas. Nors šiame regione pastoviai jaučiamas seisminis aktyvumas, tačiau tokios žemės vibracijos vietiniai gyventojai nepatyrė nuo 1888 metų, kada stiprumas siekė 3,4 balo pagal Richterio skalę. Žinoma, kaip ir kiti tokio pobūdžio įvykiai, jie buvo nuodugniai ištirti mokslininkų, pateikta gausybė išsamių ataskaitų bei prognozių.

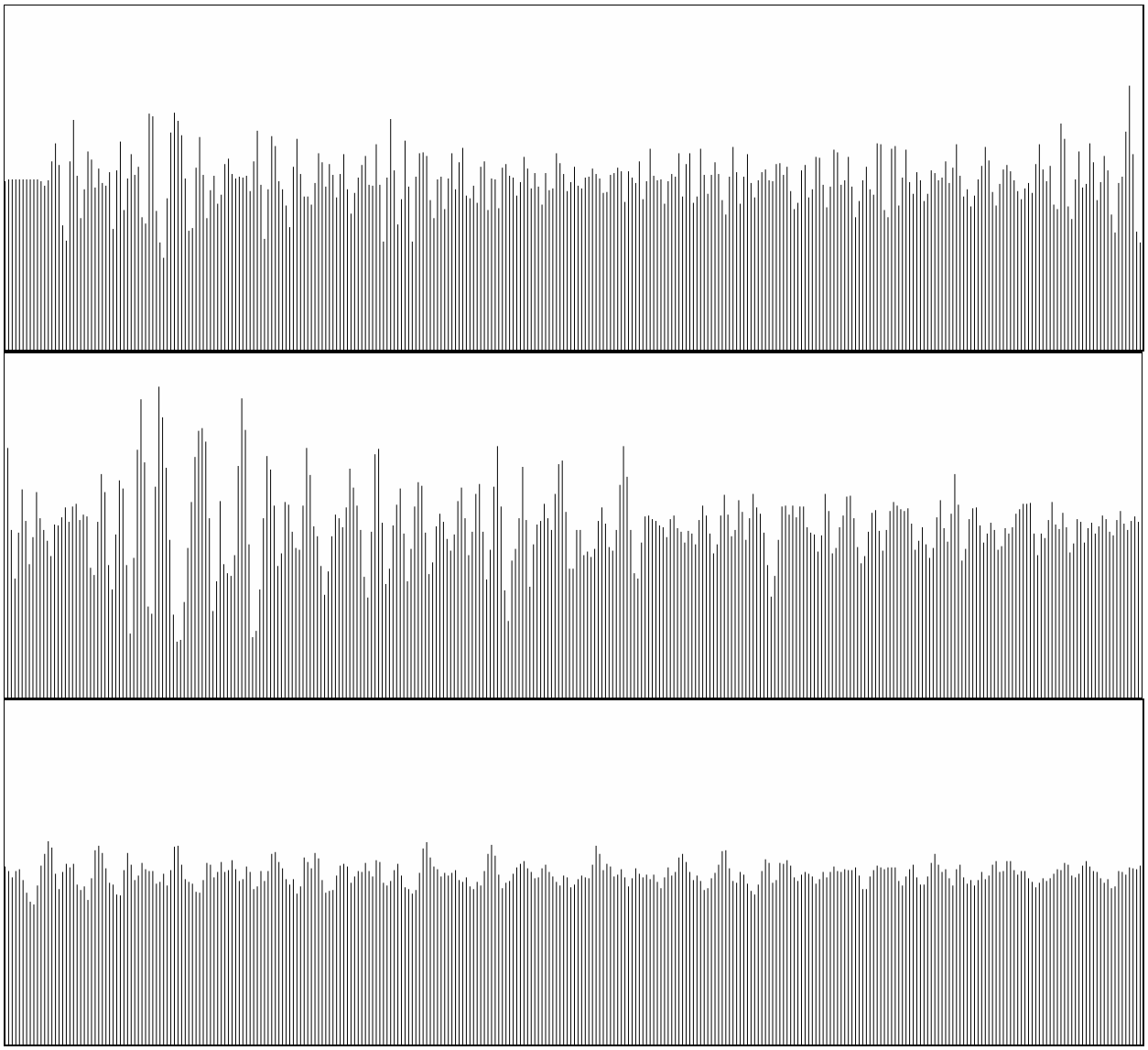
Jau senokai pastebėta, kad seisminį aktyvumą aprašančios kreivės – seismogramos – ypatingai pasižymi fraktališkumo savybe [3], todėl eksperimento duomenims naudosime minėto žemės drebėjimo seismogramą:



2.15 pav. Damfrio žemės drebėjimo (2006.12.26, 10:40) seismograma

Duomenų objektą (2.15 pav.) turime grafiniame (**.bmp* formato, 3792 x 402 pikselio matmenų) faile, kuriame talpinama praktiškai viso įvykio nuo 20-osios iki 59-osios sekundžių seismogramos kreivė. Kadangi ordinačių ašies matavimo vienetai ir jų mastelis nėra itin aktualūs eksperimentui, plačiau čia jų neaptarsime.

Pirmiausia, duomenų aibės sudarymui diskretizuojame paveikslą, nuskaitydami seismogramos reikšmes (ordinates) kas 4-tame abscisių ašies taške (pikselyje) (2.16 pav.):

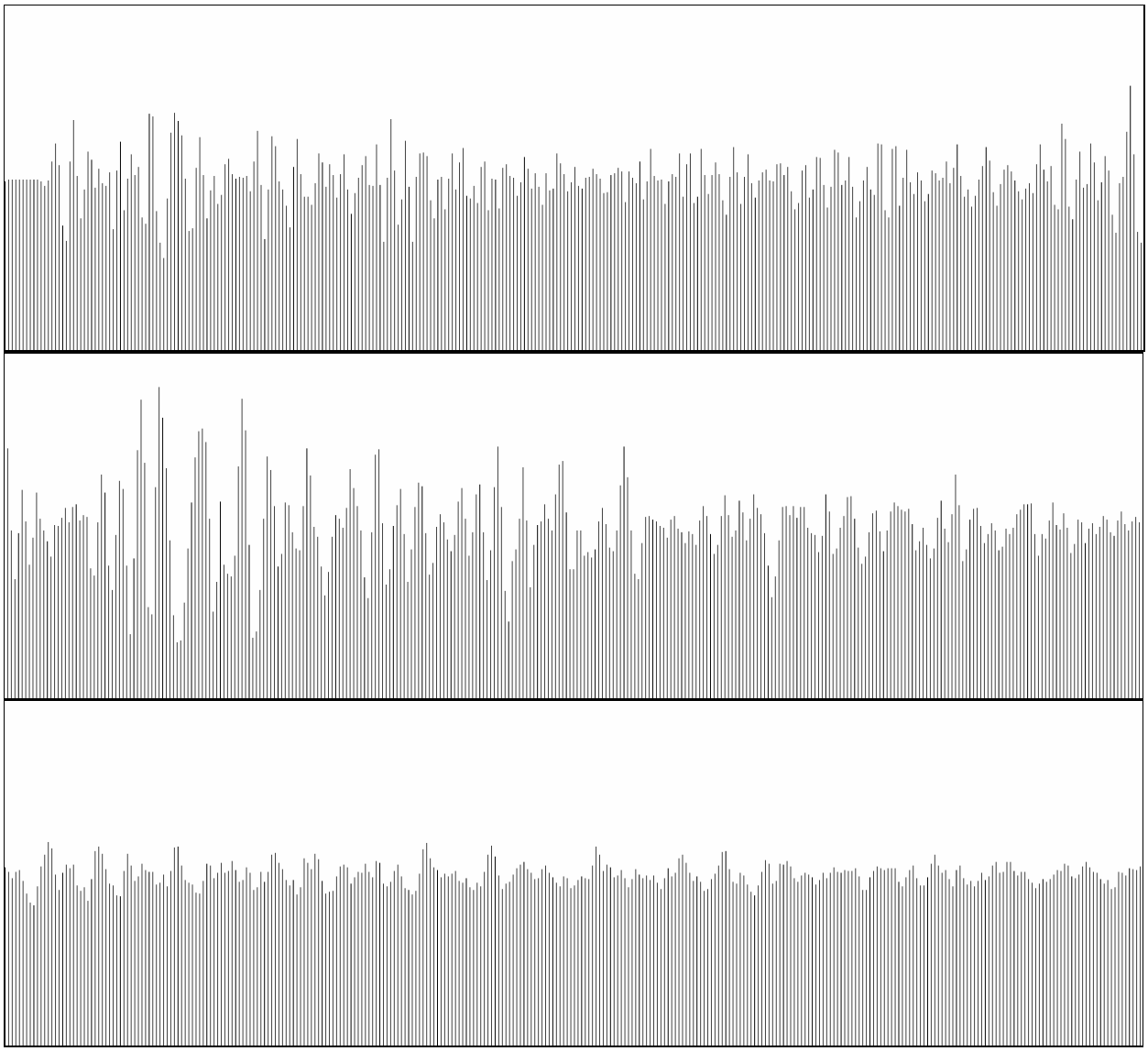


2.16 pav. Seismogramos duomenys (kas 4-tas taškas; $N = 948$)

Pastebėsime, kad šiame (2.16 paveiklas) ir tolimesniuose eksperimento paveiksluose informacija dažniausiai pateikiama suskaidyta į tris dalis, kurios žymi to paties objekto tęstines dalis (žiūrint iš viršaus į apačią).

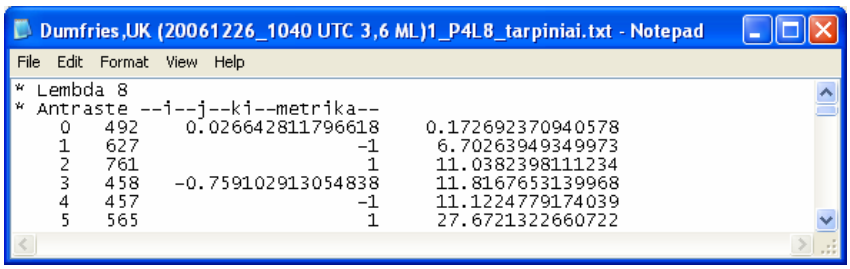
Kaip matome, pirmojo etapo metu (2.16 paveiklas) pagaminome pradinius eksperimento duomenis (2.2 formulė), t.y. interpoliavimo taškų (R^2 plokštumoje) aibę, kurios galia šiuo atveju yra N=948. Žinoma, šioje vietoje būtina pabrėžti, jog siūloma procedūra (2.2 skyrius) naudinga tik tuo atveju, jei dėl tam tikrų priežasčių negaunamas pilnas duomenų objektas (seismograma), o turima tik dalis informacijos (pvz., mūsų suformuotoji duomenų aibė).

Antruoju žingsniu vykdome su lokaliuoju koliažu susijusią interpoliuojamų duomenų atranką (2.2.2 skyrius), parinkdami atrankos parametrą $\lambda = 2^3 = 8$ (2.3 formulė):



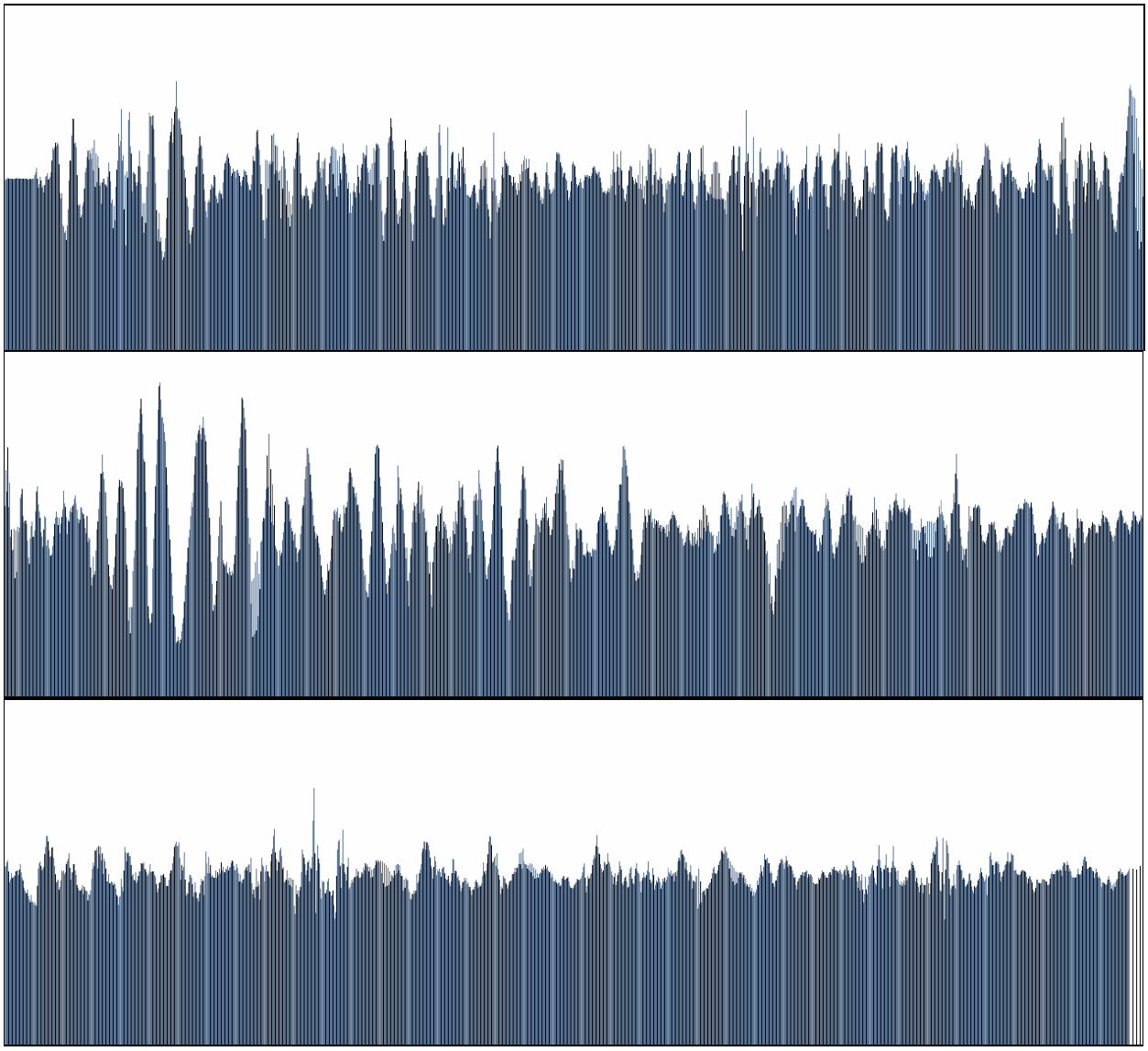
2.17 pav. Seismogramos duomenų atranka ($N = 948$, $\lambda = 8$)

Tamsesnės spalvos linijos (2.17 pav.) žymi atrinktojo poaibio (2.3 formulė), kurio galia [N/ λ]+1 = 119, taškus. Sekančiu žingsniu pereinama prie 2.2.3 skyriuje aprašyto koliažo organizavimo, kurio metu kaupiami duomenys apie geriausiai atitinkančius interpoliuojamų duomenų poaibius:



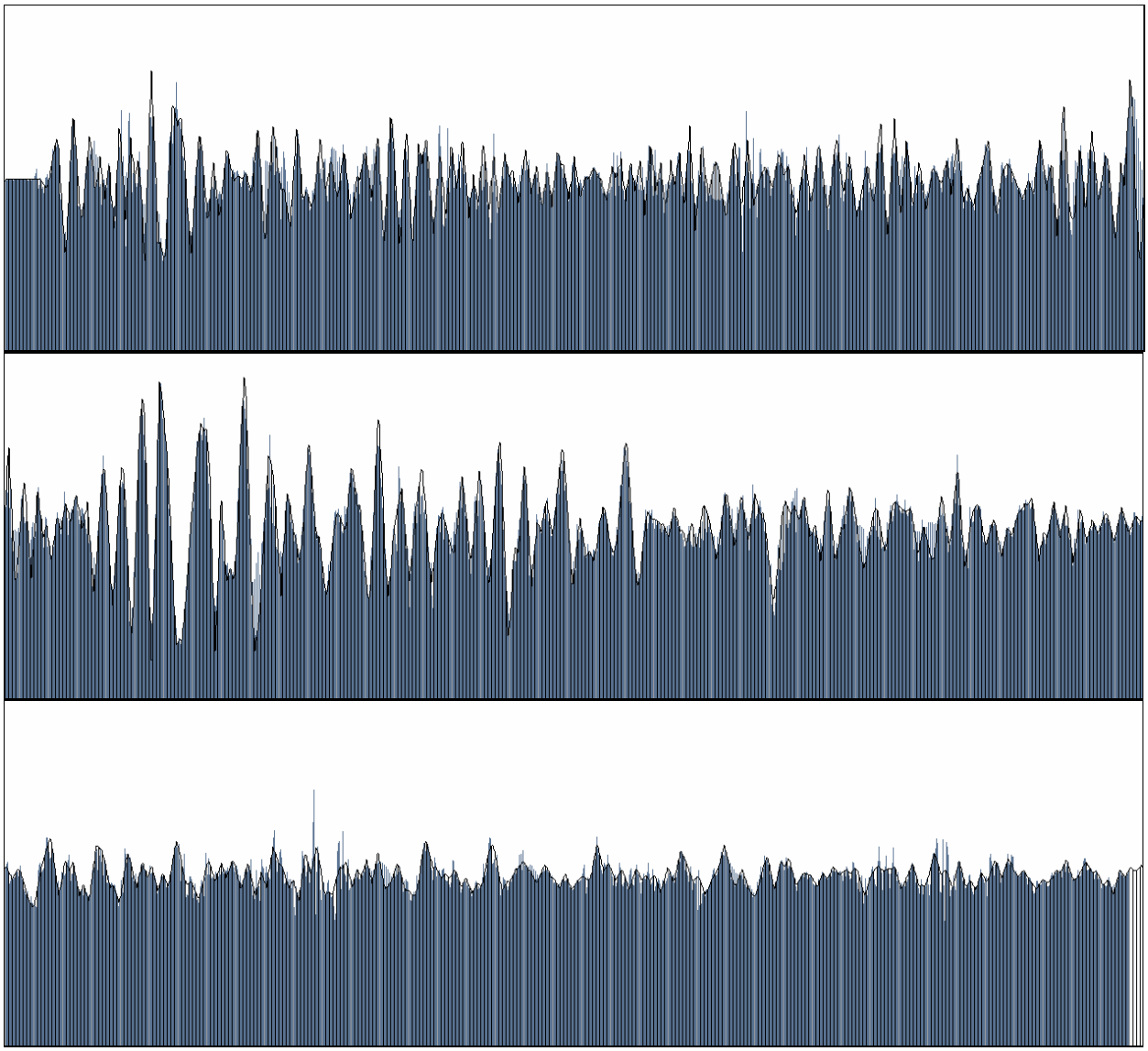
2.18 pav. Seismogramos tarpiniai (geriausių atitikčių) duomenys

Nurodę interpoliavimo taškų aibę, sukauptus tarpinius duomenis bei maksimalų iteracijų skaičių (šiuo atveju $n = \log_2 4 = 2$), 2.2.4 skyriuje aprašytu metodu generuojame fraktalinę interpoliacinę funkciją (FIF):



2.19 pav. Sugeneruotoji FIF ($N = 948$, $\lambda = 8$)

Gautoji fraktalinė interpoliacinė funkcija (2.19 pav.) – tai kreivė, „einanti“ per eksperimento pradžioje sudarytos duomenų aibės (kas 4-tas pikselis) taškus. Reikia pastebėti, kad šiuo atveju tarp kiekvienos gretimų interpoliavimo taškų poros, buvo sugeneruota po 3 naujus taškus: vienas pirmosios iteracijos metu ir likę 2 – antrosios. Modeliavimo kokybei nusakyti, vizualiai palyginkime sugeneruotąją FIF su tikruoju duomenų šaltiniu (seismograma):



2.19 pav. Sugeneruotos FIF ($N = 948$ **,** $\lambda = 8$ **) palyginimas su seismograma**

Vizualiai kreivės „gana“ panašios (2.19 pav.), o seismogramos duomenys tarp interpoliavimo taškų modeliuojami „pakankamai“ gerai. Norint tiksliau įvertinti modeliavimo kokybę, galima apskaičiuoti vidutinę kvadratinę paklaidą (VKP) tarp lyginamųjų kreivių:

$$\delta = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(FIF(i) - X(i))^2}\ ; \qquad \textbf{(2.11)}$$

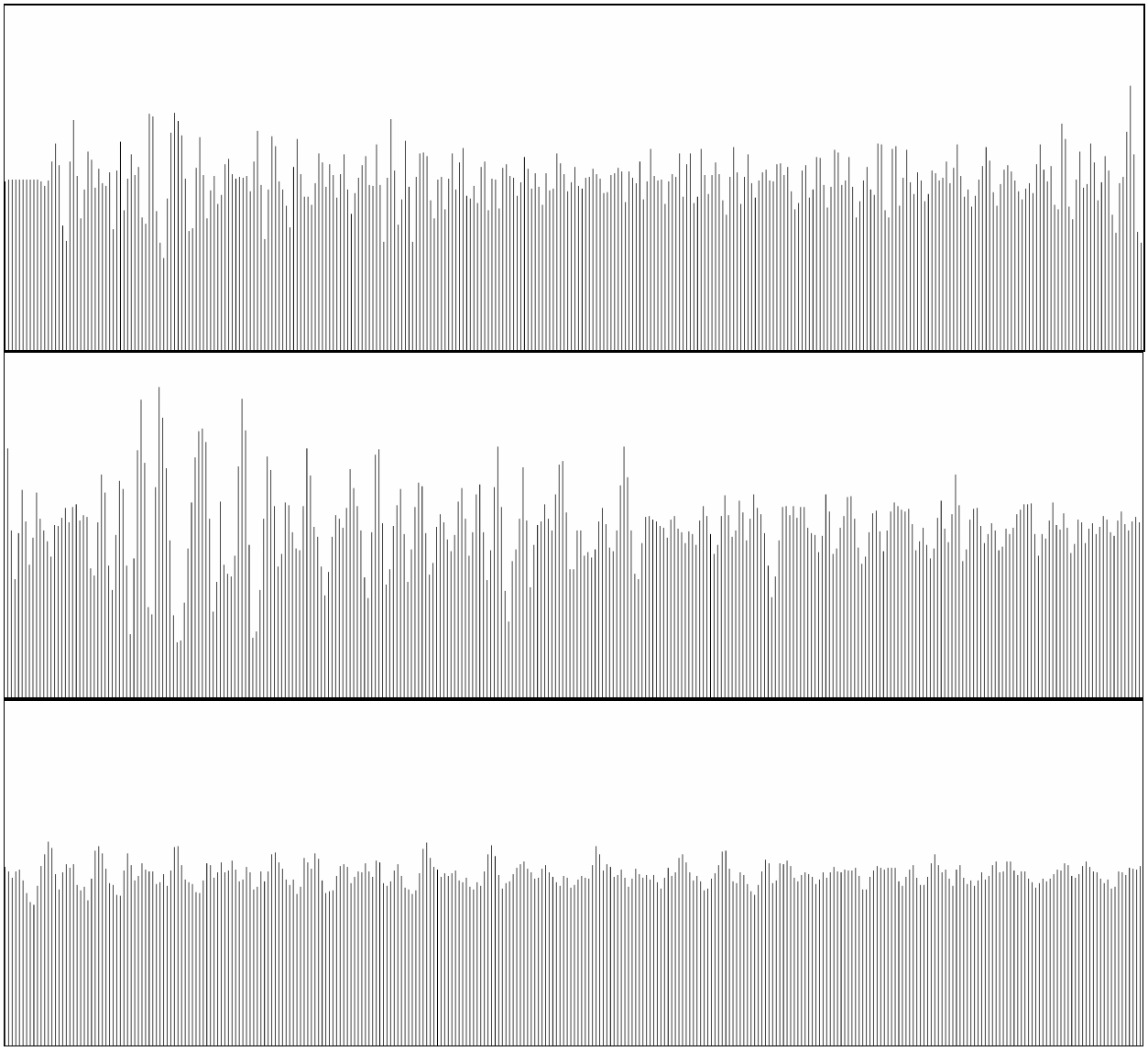
čia: $FIF(i)$ ir $X(i)$ - atitinkamai sugeneruotos FIF ir lyginamosios kreivės reikšmės i-tajame taške, o N – lyginamąją kreivę aprašančios taškų aibės galia.

Be to, šiame kontekste tam tikrą prasmę turėtų ir santykinis (procentinis) įvertis:

$$\alpha = [\delta / (\max_{1 \le i \le N} X(i) - \min_{1 \le i \le N} X(i))] \cdot 100\%\ ; \qquad \textbf{(2.12)}$$

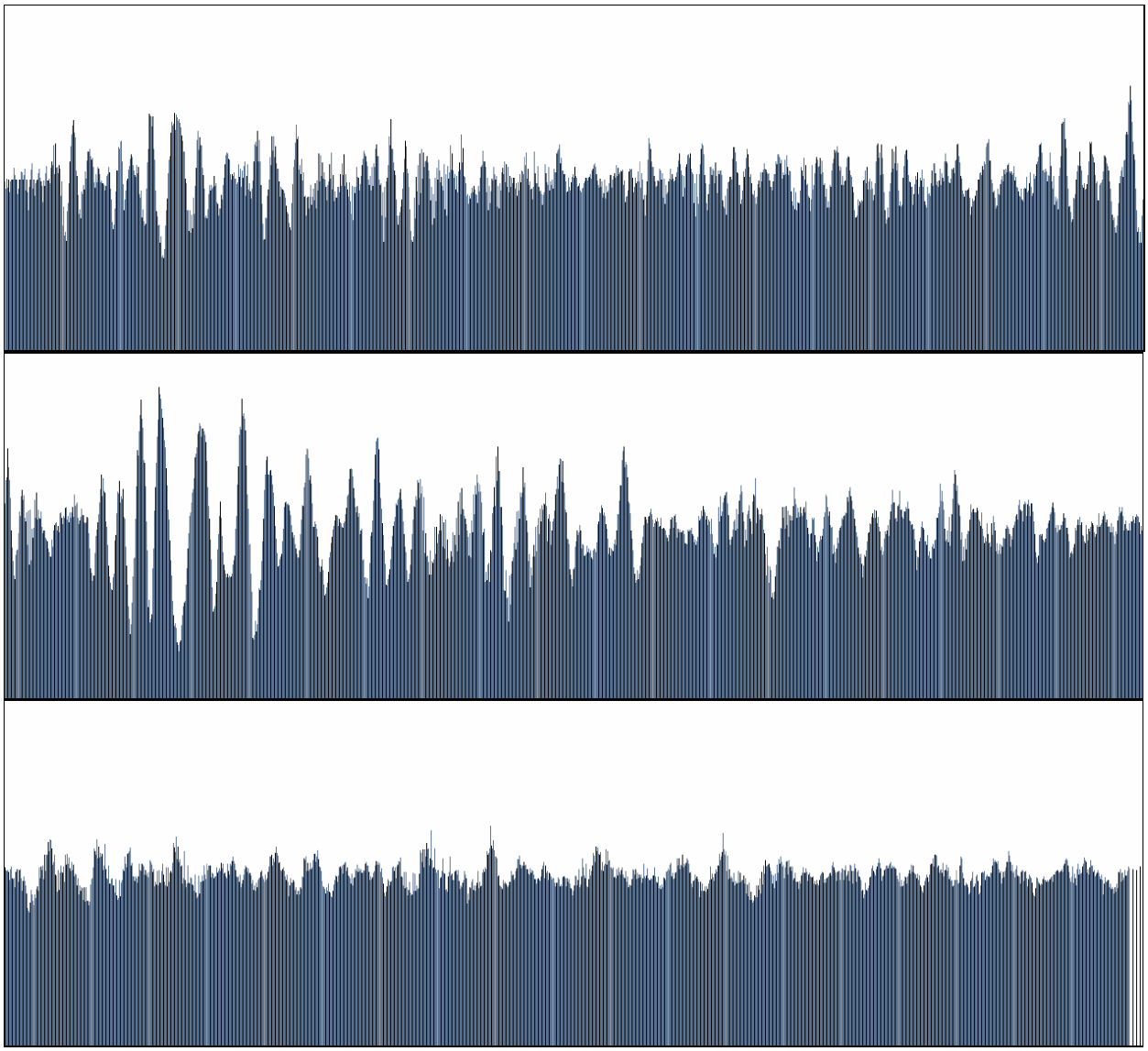
Šiuo atveju, suskaičiuotieji įverčiai $\delta = 12{,}779$, $\alpha = 4{,}1\%$.

Dabar pabandykime pakeisti interpoliavimo taškų atrankos parametrą λ, kurio didinimas reikš, jog mes mažiname atrinktųjų interpoliavimo taškų skaičių (2.2.2 skyrius). Imkime $\lambda = 16$, ir iš tos pačios sudarytosios duomenų aibės (2.16 pav.) atrinkime naują poaibį (2.20 pav.):



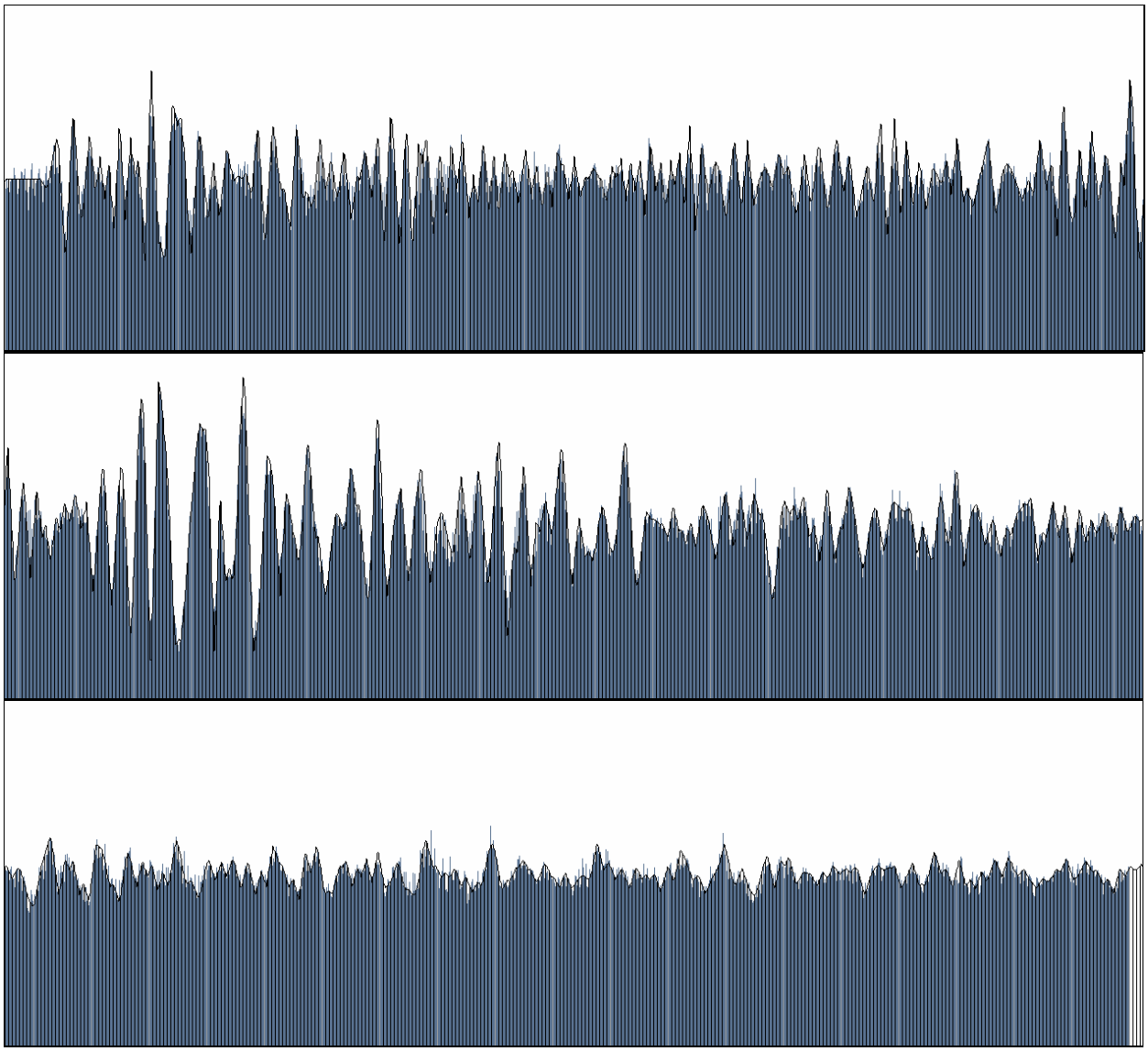
2.20 pav. Seismogramos duomenų atranka ($N = 948$ **,** $\lambda = 16$ **)**

Kadangi naudojame tą pačią duomenų aibę (interpoliuojamus taškus), tai gauname 60 atrinktų interpoliavimo taškų (2.20 pav.), o tai jau beveik perpus mažiau nei anksčiau atliktos atrankos ($\lambda = 8$) metu. Analogiškai kartojame geriausių atitikčių paiešką ir, sukaupę tarpinius (saugojimui) duomenis, generuojame naują FIF (2.21 pav.).



2.21 pav. Sugeneruotoji FIF ($N = 948$, $\lambda = 16$)

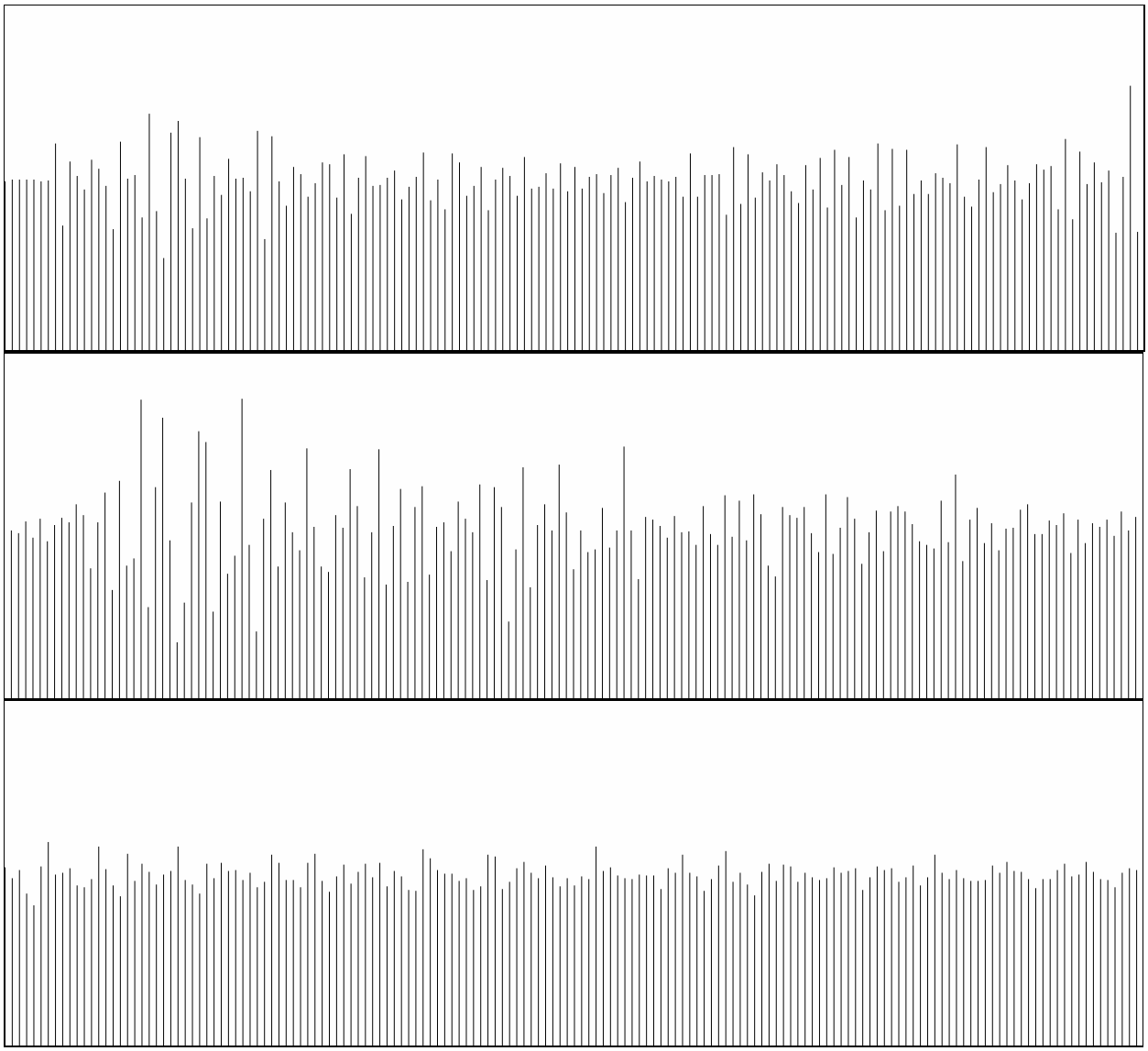
Gautąją fraktalinę interpoliacinę funkciją (2.21 pav.) taip pat lyginkime su tikruoju duomenų šaltiniu – seismograma, ir įvertinkime modeliavimo kokybę.



2.22 pav. Sugeneruotos FIF ($N = 948$, $\lambda = 16$) palyginimas su seismograma

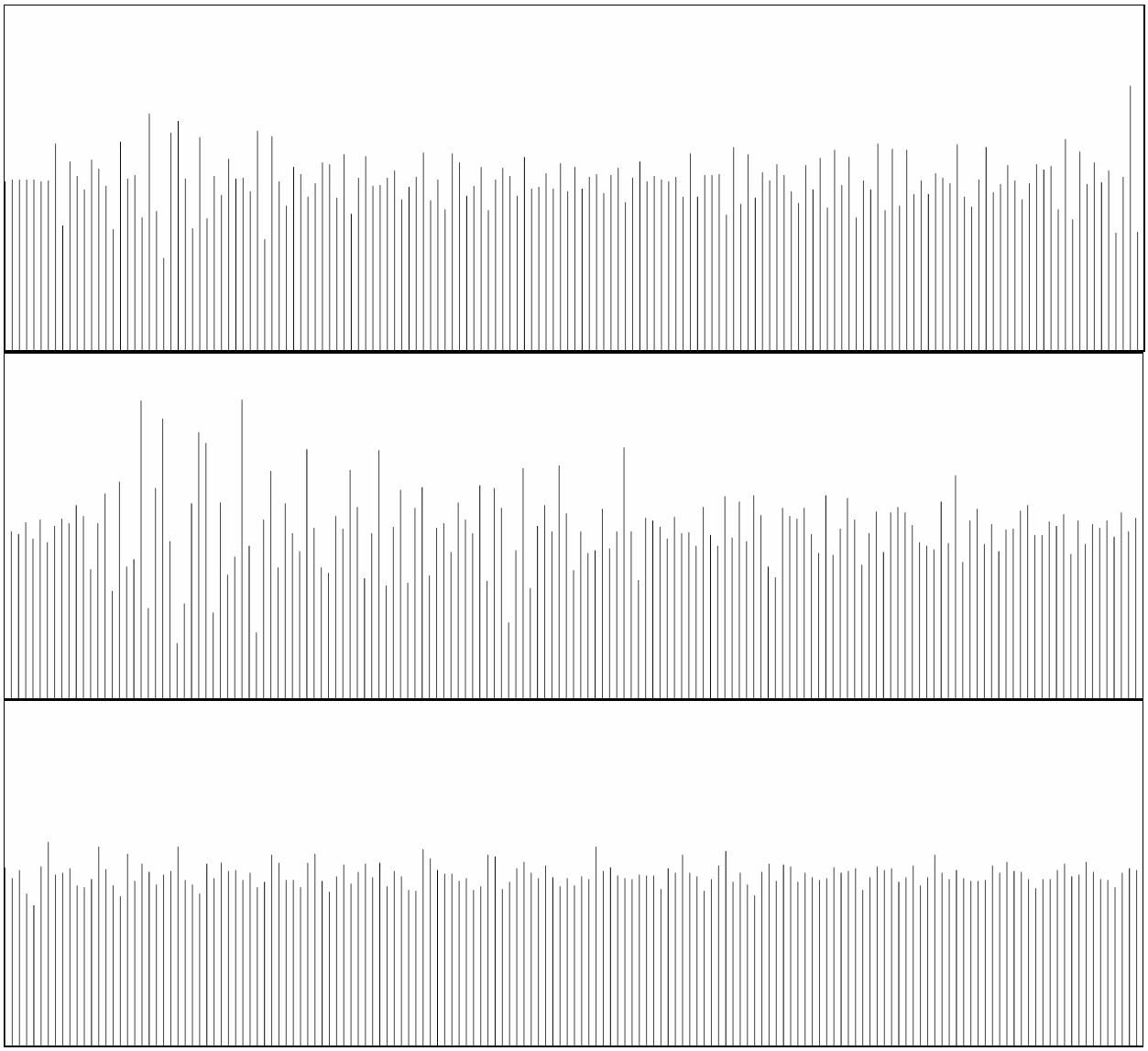
Vizualusis kreivių palyginimas „nelabai“ skiriasi nuo 2.19 paveiksle kreivių sutapatinimo, o vidutinė kvadratinė paklaida sumažėjo ($\delta = 10{,}606$, $\alpha = 3{,}4\%$). Galima daryti išvadą, jog sprendžiant interpoliavimo uždavinį, netikslinga apsiriboti viena fiksuota parametro λ reikšme.

Antroje eksperimento dalyje sudarykime naują interpoliuojamų taškų aibę, nuskaitydami minėto Damfrio žemės drebėjimo seismogramos (2.15 pav.) reikšmes kas 8-tąjį abscisės pikselį (2.23 pav):



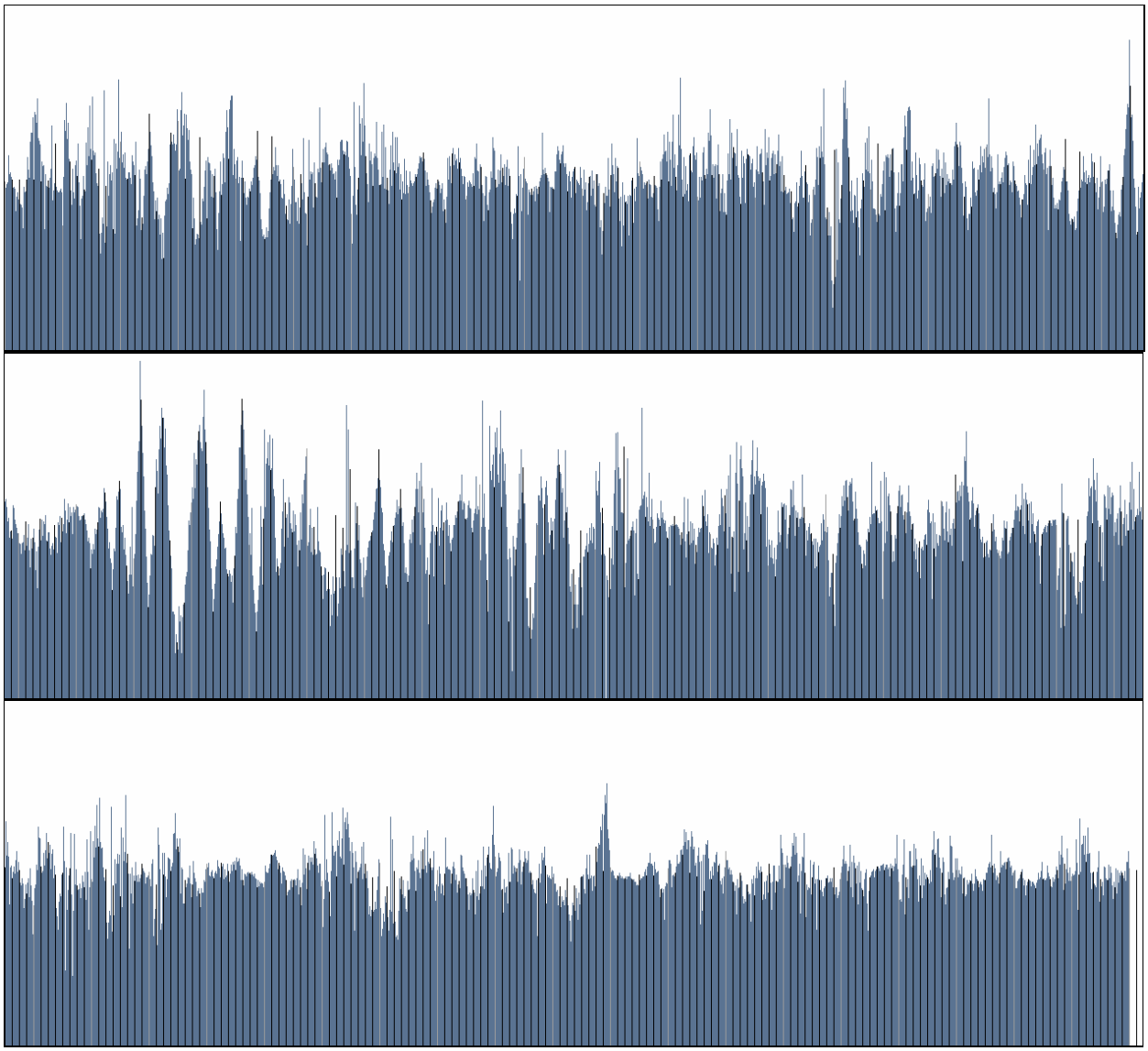
2.23 pav. Seismogramos duomenys (kas 8-tas taškas; $N = 474$)

Dabar gautą interpoliavimų taškų aibę (2.23 pav.) sudaro $N = 474$ taškai. Su šia naująja aibe atlikime dar du eksperimentus, organizuodami koliažą su tomis pačiomis atrankos parametro reikšmėmis, t.y. $\lambda = 8$ ir $\lambda = 16$.



2.24 pav. Seismogramos duomenų atranka ($N = 474$ **,** $\lambda = 8$ **)**

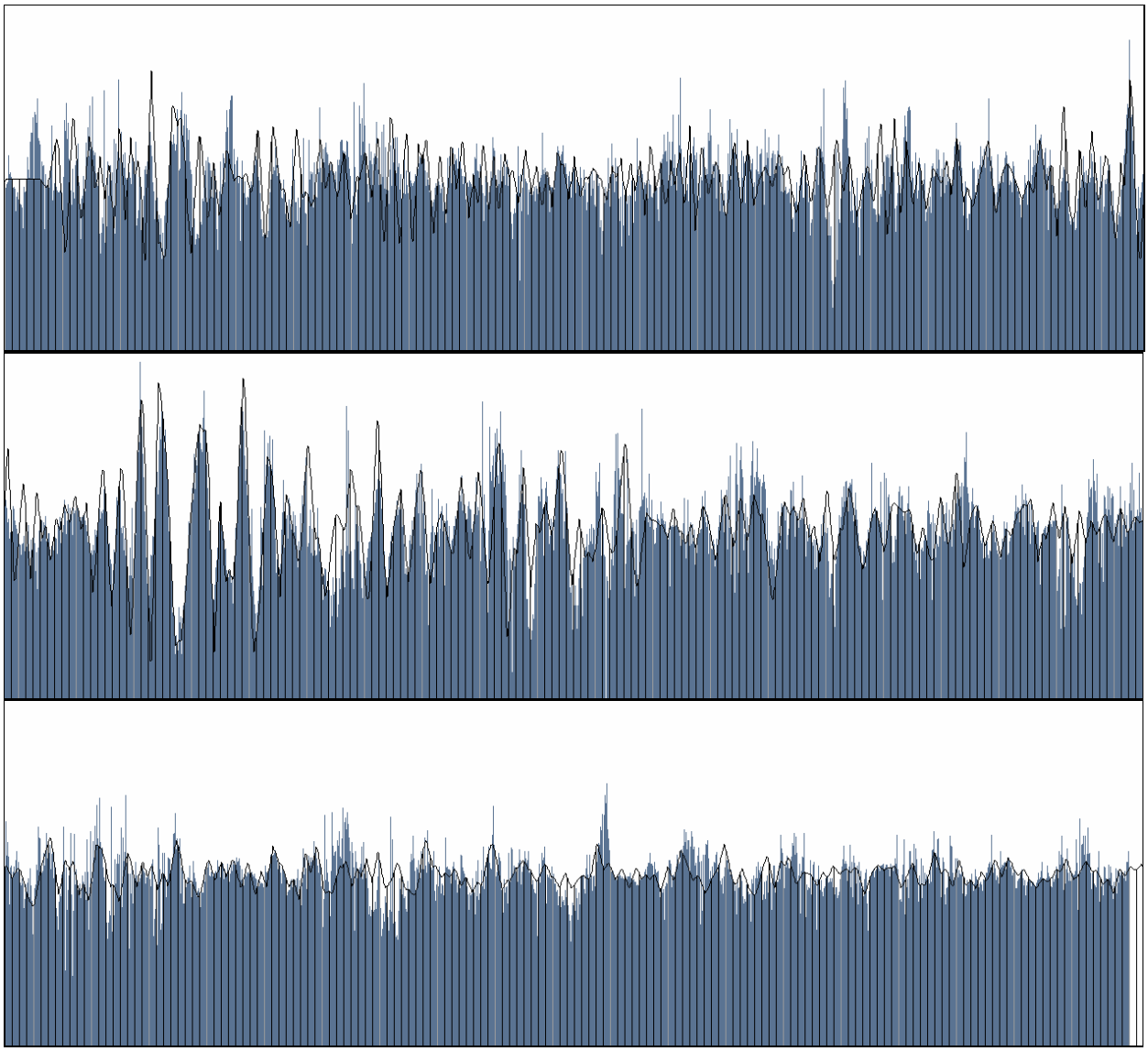
Parinkus atrankos parametrą $\lambda = 8$ (2.24 pav.), gauta 60 atrinktų interpoliavimo taškų. Atlikus analogiškus veiksmus, sugeneruota nauja FIF (2.24 pav.):



2.24 pav. Sugeneruotoji FIF, $N = 474$**,** $\lambda = 8$

Generavimo procedūrai šįkart prireikė jau trijų iteracijų ($n = \log_2 8 = 3$), nes turimą interpoliuojamų duomenų aibę sudaro kas 8-toji seismogramos reikšmė. Tarp kiekvienos gretimų interpoliavimo taškų poros, buvo sugeneruota po 7 naujus taškus: vienas pirmosios iteracijos metu, du – antrosios ir likę 4 – trečiosios.

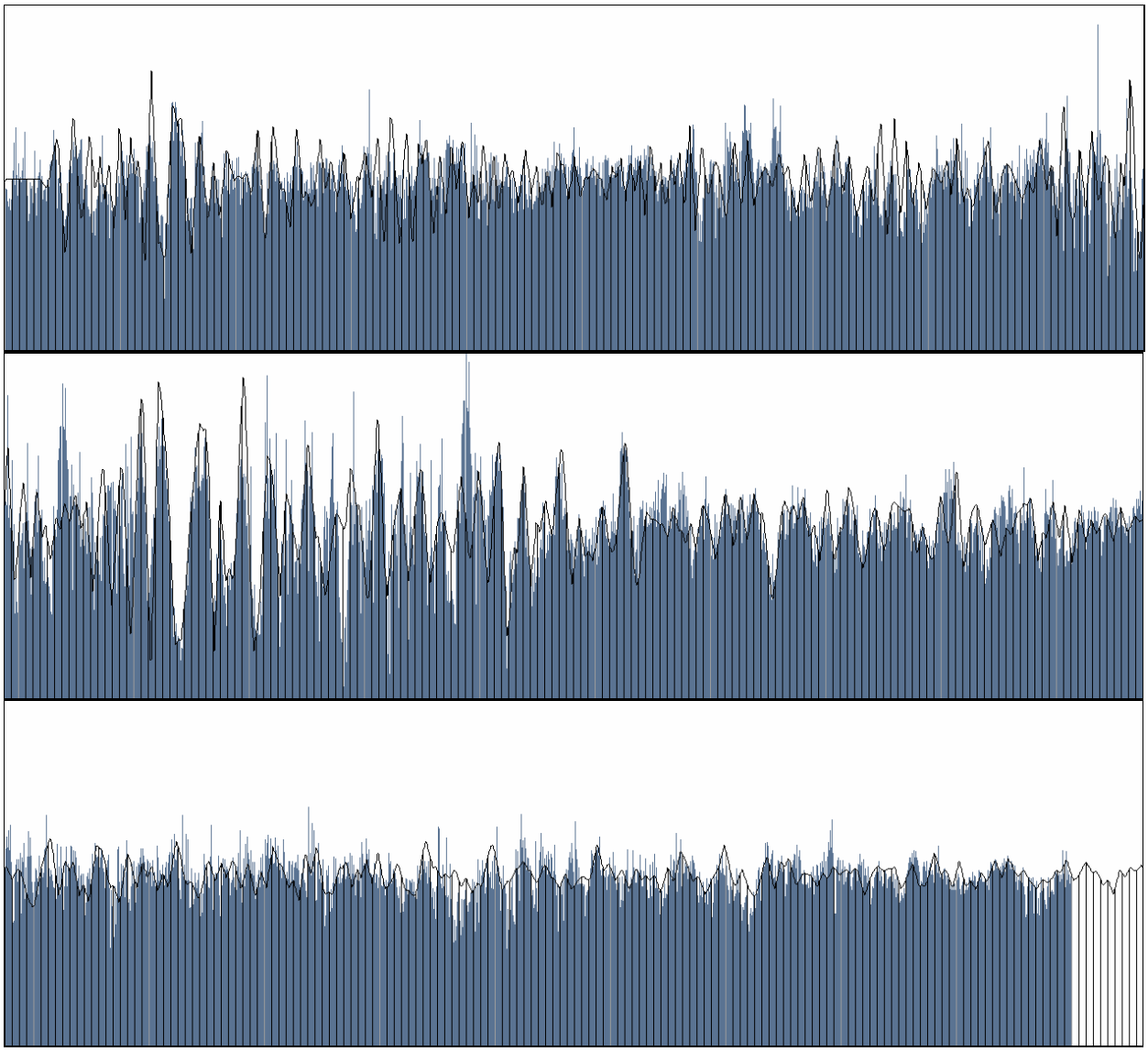
Gautąją fraktalinę interpoliacinę funkciją (2.24 pav.), vėlgi, lyginkime su tikruoju duomenų šaltiniu (2.25 pav.).



2.25 pav. Sugeneruotos FIF ($N = 474$, $\lambda = 8$) palyginimas su seismograma

Lyginant kreives (2.25 pav.) akivaizdžiai matome, jog sugeneruota FIF jau daugiau skiriasi nuo tikrojo duomenų šaltinio, ką patvirtina ir VKP ($\delta = 32{,}512$, $\alpha = 10{,}4\%$). Tai natūralu, nes interpoliavimo taškų skaičius taip pat sumažėjęs ($N = 474$). Nepaisant nuokrypių, gautoji FIF vis tik išlaiko seismogramos kreivės tendencijas.

Paskutiniame eksperimento etape, iš turimos duomenų aibės (2.23 pav.) atrinkime interpoliavimo taškus jau su parametru $\lambda = 16$ ir po analogiškų veiksmų palyginkime kreives (2.26 pav.).



2.26 pav. Sugeneruotos FIF ($N = 474$, $\lambda = 16$) **palyginimas su seismograma**

Vidutinė kvadratinė paklaida (VKP) šiuo atveju $\delta = 32{,}254$ bei $\alpha = 10{,}3\%$.

IŠVADOS

1. Fraktalinio interpoliavimo priemonių praktinis panaudojimas išlieka problematiškas vien dėl to, kad nenumatytos (nėra žinomos) procedūros, išryškinančios realaus pasaulio objektams būdingos "atsikartojimo savyje" savybės.
2. Darbe pasiūlytas naujas realaus (fizinio) pasaulio objektų fraktalinio interpoliavimo metodas (priemonė) bei jo programinė realizacija. Siūlomos priemonės esmė - apdorojamoje duomenų aibėje ieškomi labiausiai vienas kitą atitinkantys (panašūs) poaibiai, taip konstruojant lokaliąją iteruotųjų funkcijų sistemą (lokalųjį koliažą). Gautosios iteruotųjų funkcijų sistemos atraktorius tapatinamas su duomenų aibę atitinkančia (fraktaline) interpoliacine funkcija. Siūlomos interpoliavimo priemonės (algoritmo) privalumas tas, jog įvertinama vidinė interpoliuojamos duomenų aibės struktūra.
3. Skaičiavimų, susijusių su preliminariu duomenų aibės formavimu bei fraktalinių interpoliacinių funkcijų generavimu, sudėtingumas mažai priklauso nuo apdorojamos duomenų aibės apimties. Sudarytąjį fraktalinio interpoliavimo metodą tikslinga taikyti tais atvejais, kai interpoliavimui skirta duomenų aibė yra arba sudėtingų (didelių kaštų reikalaujančių) eksperimentų rezultatas, arba duomenų „nuskaitymas" dideliu dažniu yra techniškai neįmanomas.
4. Atlikti eksperimentai parodė, jog siūlomas koliažu grįstas fraktalinio interpoliavimo metodas veikia efektyviai bei yra perspektyvus. Metodo praktinio panaudojimo sritys – realaus pasaulio objektų modeliavimas, fizikinių signalų apdorojimas ir pan.

LITERATŪRA

1. Barnsley, M. F. / *Fractals everywhere*, Second edition, San Diego, CA, USA, Cambridge, 1988.
2. Valantinas, J. / *Fraktalinė geometrija*, Kaunas, Lithuania, KTU, 1999.
3. Oliver, D. Fractals in The Real World / *Fractal Vision*, Sams Publishing, Indiana, USA, 1992.
4. Mandelbrot, B.B. / *The Fractal Geometry of Nature*, 1977.
5. Peitgen, H., Richter, P. / *The Beauty of Fractals*, Springer-Verlag, Berlin, New York, 1986.
6. Image Analyzer, [žiūrėta: 2007-03-20] – Prieiga per internetą: <http://meesoft.logicnet.dk/Analyzer/plugins/>

1 PRIEDAS

PROGRAMINĖS REALIZACIJOS KODAS

Pateikiame pagrindinius programinės realizacijos modulių kodus.

Modulis „Diskretizavimas“:

```
using System;
using System.Collections;
using System.Drawing;

namespace Kodas
{
        public class Diskretizavimas
        {
                #region Kintamieji
                Bitmap k_bmap;            //darbinis bitmapas
                int F_max;
                int x_max;
                Color k_col;              //bakgroundo spalva
                #endregion
                #region Pagalbiniu klasiu kintamieji
                Funkcijos                         _fn;
                #endregion
                #region Konstruktorius
                public Diskretizavimas(Image img) : this(new Bitmap(img))
                {
                }
                public Diskretizavimas(Bitmap btm)
                {
                        k_bmap = btm;
                        k_col = k_bmap.GetPixel(0,0);
                        F_max = k_bmap.Height - 1;
                        x_max = k_bmap.Width - 1;
                        _fn = new Funkcijos();
                }
                #endregion
                #region Properciai
                public Bitmap BtmDarbinis
                {
                        get {return k_bmap as Bitmap;}
                }
                #endregion
                #region Metodai
                public int gauti_F(int x)
                {
                        if (x == 462 || x == 0)
                        {
                                int kk = 8;
                                kk = kk++;
                        }
                        int F = F_max;
                        while (k_col.Equals(k_bmap.GetPixel(x,F)) && F > 0) F--;
                        int storis = 0; //FIF "storis"
                        if (x != x_max && x != 0)
                                while (!k_col.Equals(k_bmap.GetPixel(x, F-storis))) storis++;
                        int rez = F_max - F + storis/2;
                        return rez;
                }

                public void pavaizduoti_linija(int x, int FF, Color line_color)
                {
                        int F = F_max;
                        int aukstis = 0;
                        while (aukstis < FF && F > 0)
                        {
                                k_bmap.SetPixel(x,F, line_color);
                                F--;
                                aukstis++;
                        }
                }
                public bool pavaizduoti_kreive(ArrayList kreives_taskai, bool bSuApjungimu, Color col, bool
                                                                                        bTestiPrieTrukio)
                {
                        ArrayList kr_taskai = _fn.Rikiuoti_pagal_x((ArrayList)kreives_taskai.Clone());
                        int x_prev = -1;
                        int y_prev = 0;
                        for (int ii=0; ii<kr_taskai.Count; ii++)
                        {
                                int x = (int)Math.Round(_fn.Gauti_Masyvo_x(kr_taskai, ii));
                                int y = F_max - (int)Math.Round(_fn.Gauti_Masyvo_F(kr_taskai, ii));
                                if (y > F_max || y < 0) continue;
                                k_bmap.SetPixel(x,y, col);
                                if (bSuApjungimu && ii != 0)
                                {
                                        if (x-x_prev != 1)
```

```
                        {
                                if (bTestiPrieTrukio)
                                {
                                        x_prev = x;
                                        y_prev = y;
                                        continue;
                                }
                                else
                                        return false;
                        }
                        int delta_y = y-y_prev;
                        if (delta_y != 0)
                        {
                                int artinimo_koef = delta_y > 0 ? 1 : -1;
                                int cursor = y_prev + artinimo_koef;
                                while (cursor != y)
                                {
                                        k_bmap.SetPixel(x,cursor, col);
                                        cursor += artinimo_koef;
                                }
                        }
                }
                x_prev = x;
                y_prev = y;
        }
        return true;
}
public void uzdeti_fona(Color fono_spalva)
{
        for (int y = 0; y < k_bmap.Height; y++)
                for (int x = 0; x < k_bmap.Width; x++)
                        k_bmap.SetPixel(x, y, fono_spalva);
}
public void pavaizduoti_VertJuosta(int x1, int x2, Color juostos_color)
{
        for (int ii = x1; ii <= x2; ii++)
        {
                int y = 0;
                while (y < k_bmap.Height && k_bmap.GetPixel(ii, y) == k_col)
                        k_bmap.SetPixel(ii, y++, juostos_color);
        }
}
public Bitmap apkirpti_apacia() //apkarpymas is apacios
{
        int F = 0;
        try
        {
                for (F = F_max; F >= 0; F--)
                        for (int xx = 0; xx <= x_max; xx++)
                                if (!k_col.Equals(k_bmap.GetPixel(xx,F)))
                                {
                                        k_bmap = k_bmap.Clone(new Rectangle(0, 0, k_bmap.Width,
                                                F), new System.Drawing.Imaging.PixelFormat());
                                        return k_bmap;
                                }
                return k_bmap;
        }
        catch(Exception)
        {return k_bmap;}
        finally
        {
                F_max = F - 1;
        }
}
public Bitmap apkirpti_kaire() //apkarpymas is kaires
{
        int x = 0;
        try
        {
                for (x = 0; x <= x_max; x++)
                        for (int FF = F_max; FF >= 0; FF--)
                                if (!k_col.Equals(k_bmap.GetPixel(x,FF)))
                                {
                                        k_bmap = k_bmap.Clone(new Rectangle(x, 0, x_max - x,
                                         k_bmap.Height), new System.Drawing.Imaging.PixelFormat());
                                        return k_bmap;
                                }
                return k_bmap;
        }
        catch(Exception)
        {return k_bmap;}
        finally
        {
                x_max = x_max - x - 1;
        }
}
public Bitmap apkirpti_desine() //apkarpymas is desines
{
        int h = 0;
        try
        {
```

```
                                for (h = x_max; h >= 0; h--)
                                        for (int FF = F_max; FF >= 0; FF--)
                                                if (!k_col.Equals(k_bmap.GetPixel(h,FF)))
                                                {
                                                        k_bmap = k_bmap.Clone(new Rectangle(0, 0, h+1,
                                                 k_bmap.Height), new System.Drawing.Imaging.PixelFormat());
                                                        return k_bmap;
                                                }
                                        return k_bmap;
                                }
                                catch(Exception)
                                {return k_bmap;}
                                finally
                                {
                                        x_max = h;
                                }
                        }
                        public Bitmap apkirpti_virsu()
                        {
                                int F = 0;
                                try
                                {
                                        for (F = 0; F <= F_max; F++)
                                                for (int xx = 0; xx <= x_max; xx++)
                                                if (!k_col.Equals(k_bmap.GetPixel(xx,F)))
                                                {
                                                        k_bmap = k_bmap.Clone(new Rectangle(0, F, k_bmap.Width,
                                                 k_bmap.Height - F), new System.Drawing.Imaging.PixelFormat());
                                                        return k_bmap;
                                                }
                                        return k_bmap;
                                }
                                catch(Exception)
                                {return k_bmap;}
                                finally
                                {
                                        F_max = k_bmap.Height - F - 1;
                                }
                        }
                        #endregion
                }
}
```

Modulis „Funkcijos“:

```
using System;
using System.Collections;
using System.Windows.Forms;
using Formos;

namespace Kodas
{
        public class Funkcijos
        {
                #region Tasku masyvo funkcijos
                public ArrayList Pildyti_Tasku_Masyva(ArrayList arr, double x, double F)
                {
                        arr.Add(new double[2] {x,F});
                        return arr;
                }

                public void Kopijuoti_Masyvo_Taska(ArrayList aFrom, int indFrom, ArrayList aTo)
                {
                        aTo.Add(Gauti_Masyvo_Taska(aFrom, indFrom));
                }
                public double[] Gauti_Masyvo_Taska(ArrayList arr, int ind)
                {
                        return (double[])arr[ind];
                }
                public double Gauti_Masyvo_x(ArrayList arr, int ind)
                {
                        return ((double[])arr[ind])[0];
                }
                public double Gauti_Masyvo_F(ArrayList arr, int ind)
                {
                        return ((double[])arr[ind])[1];
                }
                public void Keisti_Masyvo_x(ArrayList arr, int ind, double newValue)
                {
                        ((double[])arr[ind])[0] = newValue;
                }
                public void Keisti_Masyvo_F(ArrayList arr, int ind, double newValue)
                {
                        ((double[])arr[ind])[1] = newValue;
                }
                public double Gauti_Max_F(ArrayList arr)
                {
                        double max = 0;
                        for(int ii = 0; ii < arr.Count; ii++)
                                max = Math.Max(max, Gauti_Masyvo_F(arr,ii));
```

```
        return max;
}
public ArrayList Rikiuoti_pagal_x(ArrayList arr)
{
        ArrayList sorted = new ArrayList();
        int ilgis = arr.Count;
        for (int tt = 0; tt < ilgis; tt++)
        {
                int ind = 0;  //min x indeksas masyve
                double x = Gauti_Masyvo_x(arr,0); //min x reiksme

                for (int ii = 0; ii < arr.Count; ii++)
                {
                        if (x > Gauti_Masyvo_x(arr,ii))
                        {
                                x = Gauti_Masyvo_x(arr,ii);
                                ind = ii;
                        }
                }
                sorted = Pildyti_Tasku_Masyva(sorted, x, Gauti_Masyvo_F(arr, ind));
                arr.RemoveAt(ind);
        }
        return sorted;
}

public ArrayList Pildyti_Tarpinius_Duomenis(ArrayList arr, int i, int j, double ki)
{
        arr.Add(new object[3] {i,j,ki});
        return arr;
}
public int Gauti_tarp_i(ArrayList arr, int ind)
{
        return (int)((object[])arr[ind])[0];
}

public int Gauti_tarp_j(ArrayList arr, int ind)
{
        return (int)((object[])arr[ind])[1];
}
public double Gauti_tarp_ki(ArrayList arr, int ind)
{
        return (double)((object[])arr[ind])[2];
}

public bool Egzistuoja_Masyve(ArrayList arr, double x)
{
        for (int kk = 0; kk < arr.Count; kk++)
        {
                if (Gauti_Masyvo_x(arr,kk) == x)
                        return true;
        }
        return false;
}
public int Gauti_Tasko_Indeksa_Is_Kaires(ArrayList arr, double x, int nuo, int iki, out bool
                                                                                        bSutampa)
{
        //arr - surikiuotas pagal x tasku masyvas
        bSutampa = false;
        for (int kk = nuo; kk < iki; kk++)
        {
                double intStart  = Gauti_Masyvo_x(arr,kk);
                double intEnd    = Gauti_Masyvo_x(arr,kk+1);
                if (x >= intStart && x < intEnd)
                {
                        bSutampa = (x == intStart);
                        return kk;
                }
        }
        return -1;
}

public int Gauti_IteracijuSkaiciu(ArrayList arr)
{
        if (arr.Count < 2) return 0;
        int pxTarpas = (int)(Gauti_Masyvo_x(arr,1) - Gauti_Masyvo_x(arr,0));
        double n = Math.Log(pxTarpas,2);
        return (int)(n+0.01);
}
#endregion
#region Failo kelio funkcijos
public string Modifikuoti_failo_kelia(string f_kelias, string priedas)
{
        try
        {
                string f_dir = f_kelias.Substring(0, f_kelias.LastIndexOf("\\"));
                string f_vrd = f_kelias.Substring(f_kelias.LastIndexOf("\\"));
                string mod_f_vrd = f_vrd.Substring(0, f_vrd.IndexOf(".")) + priedas;
                string mod_f_kelias = f_dir + mod_f_vrd;
                return mod_f_kelias;
        }
        catch (Exception ex)
```

```
                {
                        Error.Process(ex);
                        return "";
                }
        }
        public string Gauti_failo_varda(string f_kelias, bool ar_su_pletiniu)

        {
                if (!ar_su_pletiniu) //grazina failo varda be pletinio is viso kelio
                {
                        string f_vrd = f_kelias.Substring(f_kelias.LastIndexOf("\\") + 1);
                        return f_vrd.Substring(0, f_vrd.IndexOf("."));
                }
                else return f_kelias.Substring(f_kelias.LastIndexOf("\\") + 1);

        }

        #endregion
    }
}
```

Modulis „Skaičiavimai“:

```
using System;
using System.Collections;

namespace Kodas
{
    public class Skaiciavimai
    {
        #region Pagalbines klases
        Funkcijos _fn;
        #endregion
        #region Konstruktorius
        public Skaiciavimai()
        {
                _fn = new Funkcijos();
        }
        #endregion
        #region Metodai

        public ArrayList Atrinkti_poaibi(ArrayList S, int par)
        {
                ArrayList rez = new ArrayList();
                int i = 0, j = 0;
                while (j < S.Count)
                {
                        _fn.Pildyti_Tasku_Masyva(rez, _fn.Gauti_Masyvo_x(S, j), _fn.Gauti_Masyvo_F(S, j));
                        i++;
                        j = par * i;
                }
                //paimamas ir paskutinis taskas del atraktoriaus generavimo FV ribotumu
                /*if (j - par != S.Count-1)
                        _fn.Pildyti_Tasku_Masyva(rez, _fn.Gauti_Masyvo_x(S, S.Count-1),
                                                                _fn.Gauti_Masyvo_F(S, S.Count-1));
                */
                return rez;
        }
        public double Tieses_reiksme(ArrayList Sat, double x, int indSt, int indEnd)
        {
                //i=1,2,3,.. Apskaiciuojama tieses, einancios per indSt ir indEnd masyvo taskus, reikšme
                                                                                                taške x
                //y = kx + b
                double delta_F = _fn.Gauti_Masyvo_F(Sat, indEnd) - _fn.Gauti_Masyvo_F(Sat, indSt);
                double delta_x = _fn.Gauti_Masyvo_x(Sat, indEnd) - _fn.Gauti_Masyvo_x(Sat, indSt);
                return (delta_F/delta_x) * (x - _fn.Gauti_Masyvo_x(Sat, indSt)) + _fn.Gauti_Masyvo_F(Sat,
                                                                                                indSt);
        }
        public ArrayList Apskaiciuoti_Di(ArrayList m, ArrayList sgm2, int N, double D)
        {
                ArrayList rez = new ArrayList();
                double sum = 0;
                for (int tt = 0; tt < N; tt++)
                {
                        sum = sum + (double)sgm2[tt];
                }
                double k = Math.Pow((double)N, D-1) / sum;
                for (int ii = 0; ii < N; ii++)
                {
                        double d = Math.Sign((double)m[ii]) * k * (double)sgm2[ii];
                        rez.Add(d);
                        //rez.Add(0.0001);
                }
                return rez;
        }


        public Hashtable Apsk_parametrus(ArrayList k_Sat, ArrayList aTarp)
        {
                Hashtable htPars = new Hashtable();
                ArrayList ai = new ArrayList();
```

```
        ArrayList ci = new ArrayList();
        ArrayList aki = new ArrayList();
        ArrayList ei = new ArrayList();
        ArrayList fi = new ArrayList();

        for(int ii=0; ii<aTarp.Count; ii++)
        {
                int             j       = _fn.Gauti_tarp_j(aTarp,ii);
                double  ki      = _fn.Gauti_tarp_ki(aTarp,ii);

                double  xi      = _fn.Gauti_Masyvo_x(k_Sat, ii);
                double  Fi      = _fn.Gauti_Masyvo_F(k_Sat, ii);
                double  xi1     = _fn.Gauti_Masyvo_x(k_Sat, ii+1);
                double  Fi1     = _fn.Gauti_Masyvo_F(k_Sat, ii+1);

                double  xj      = _fn.Gauti_Masyvo_x(k_Sat, j);
                double  Fj      = _fn.Gauti_Masyvo_F(k_Sat, j);
                double  xj2     = _fn.Gauti_Masyvo_x(k_Sat, j+2);
                double  Fj2     = _fn.Gauti_Masyvo_F(k_Sat, j+2);


                ai.Add((xi1 - xi)/(xj2 - xj));
                ci.Add((Fi1 - Fi)/(xj2 - xj) - ki * ((Fj2 - Fj)/(xj2 - xj)));
                aki.Add(ki);
                ei.Add((xj2 * xi - xj * xi1)/(xj2 - xj));
                fi.Add((xj2 * Fi - xj * Fi1)/(xj2 - xj) - ki * ((xj2 * Fj - xj * Fj2)/(xj2 - xj)));
        }
        htPars["ai"] = ai;
        htPars["ci"] = ci;
        htPars["ki"] = aki;
        htPars["ei"] = ei;
        htPars["fi"] = fi;
        return htPars;
}
public Hashtable Apsk_parametrus_pilnas(ArrayList k_S, ArrayList aTarp, int lembda)
{
        Hashtable htPars = new Hashtable();
        ArrayList ai = new ArrayList();
        ArrayList ci = new ArrayList();
        ArrayList aki = new ArrayList();
        ArrayList ei = new ArrayList();
        ArrayList fi = new ArrayList();

        for(int ii=0; ii<aTarp.Count; ii++)
        {
                int             i       = _fn.Gauti_tarp_i(aTarp,ii);
                int             j       = _fn.Gauti_tarp_j(aTarp,ii);
                double  ki      = _fn.Gauti_tarp_ki(aTarp,ii);

                double  xi      = _fn.Gauti_Masyvo_x(k_S, i*lembda);
                double  Fi      = _fn.Gauti_Masyvo_F(k_S, i*lembda);
                double  xi1     = _fn.Gauti_Masyvo_x(k_S, (i+1)*lembda);
                double  Fi1     = _fn.Gauti_Masyvo_F(k_S, (i+1)*lembda);

                double  xj      = _fn.Gauti_Masyvo_x(k_S, j);
                double  Fj      = _fn.Gauti_Masyvo_F(k_S, j);
                double  xj2     = _fn.Gauti_Masyvo_x(k_S, j+2*lembda);
                double  Fj2     = _fn.Gauti_Masyvo_F(k_S, j+2*lembda);


                ai.Add((xi1 - xi)/(xj2 - xj));
                ci.Add((Fi1 - Fi)/(xj2 - xj) - ki * ((Fj2 - Fj)/(xj2 - xj)));
                aki.Add(ki);
                ei.Add((xj2 * xi - xj * xi1)/(xj2 - xj));
                fi.Add((xj2 * Fi - xj * Fi1)/(xj2 - xj) - ki * ((xj2 * Fj - xj * Fj2)/(xj2 - xj)));
        }
        htPars["ai"] = ai;
        htPars["ci"] = ci;
        htPars["ki"] = aki;
        htPars["ei"] = ei;
        htPars["fi"] = fi;
        return htPars;
}
public ArrayList Apskaiciuoti_mi(ArrayList S, ArrayList Sat, int par)
{
        ArrayList rez = new ArrayList();
        double mi;
        int i = 1, j = 0;
        while (j < S.Count)
        {
                double nuok_sum = 0;
                for (int s = 0; s <= par; s++)    //skaiciuojam nuokrypiu suma
                {
                        int ind = par*(i-1)+s;
                        double tasko_F = _fn.Gauti_Masyvo_F(S, ind);
                        double tieses_F = Tieses_reiksme(Sat, _fn.Gauti_Masyvo_x(S, ind), i-1, i);
                        double nuokrypis = tasko_F - tieses_F;
                        nuok_sum = nuok_sum + nuokrypis;
                }
                mi = nuok_sum / (par + 1);
                rez.Add(mi);
```

```
                        i++;
                        j = par * i;
                }
                return rez;
        }
        public ArrayList Apskaiciuoti_SIGMAKVi(ArrayList S, ArrayList Sat, ArrayList m_arr, int par)
        {
                ArrayList rez = new ArrayList();
                double sigma_kvi;
                int i = 1, j = 0;
                while (j < S.Count)
                {
                        double sum = 0;
                        for (int s = 0; s <= par; s++)    //skaiciuojam nuokrypiu kvadratu - mi^2 suma
                        {
                                int ind = par*(i-1)+s;
                                double tasko_F = _fn.Gauti_Masyvo_F(S, ind);
                                double tieses_F = Tieses_reiksme(Sat, _fn.Gauti_Masyvo_x(S, ind), i-1, i);
                                double nuokrypio_kv = Math.Pow(tasko_F - tieses_F, 2);
                                double posumis = nuokrypio_kv - Math.Pow((double)m_arr[i-1],2);
                                sum = sum + posumis;
                        }
                        sigma_kvi = sum / (par + 1);
                        rez.Add(sigma_kvi);
                        i++;
                        j = par * i;
                }
                return rez;
        }

        public ArrayList ApjungtiIntervala(ArrayList S, int j, int lembda)
        {
                //apjungia paduoto masyvo poaibio nuo j-tojo iki (j+2*lembda)-ojo reiksmes ir padaro
                                                                        dvigubai trumpesni masyva

                //patikriname ar turim poaibi S aibeje
                if (j+2*lembda > S.Count-1)
                        return null;

                //patikriname ar turim poaibi S aibeje atmetus galinius taskus, kur jau nesusidaro lembda
                                                                                        zingsnis
                //cia galioja tik pilname perrinkime!
                if (j+2*lembda > (int)((S.Count-1)/lembda)*lembda)
                        return null;

                ArrayList arr = new ArrayList(lembda+1);

                arr.Add(S[j]);
                for (int t = 1; t < lembda; t++)
                {
                        double F = 0;
                        for (int r = -1; r <= 1; r++)
                                F += _fn.Gauti_Masyvo_F(S,j+2*t+r);
                        F = ((double)1/3)*F;

                        _fn.Pildyti_Tasku_Masyva(arr, _fn.Gauti_Masyvo_x(S,j+t), F);
                }
                _fn.Pildyti_Tasku_Masyva(arr, _fn.Gauti_Masyvo_x(S,j+lembda),
                                                                _fn.Gauti_Masyvo_F(S,j+2*lembda));
                return arr;
        }
        #endregion
    }
}
```

Modulis „Statikai“:

```
using System;
using System.Collections;

namespace Kodas
{
    public class Statikai
    {
        #region Kintamieji
        static string             _rootDir = "";
        static Hashtable _options = null;
        #endregion
        #region Properciai
        public static string RootDir
        {
                set {_rootDir = value;}
                get {return _rootDir;}
        }
        public static Hashtable Nustatymai
        {
                set {_options = value;}
                get {return _options;}
        }
        #endregion
```

```
                #region Public metodai
                public static string Gauti_StrNustatyma(string vardas)
                {
                        return _options.ContainsKey(vardas) ? _options[vardas].ToString() : "";
                }
                public static int Gauti_IntNustatyma(string vardas, int defReiksme)
                {
                        return _options.ContainsKey(vardas) ? Convert.ToInt32(_options[vardas]) : defReiksme;
                }
                public static double Gauti_DoubleNustatyma(string vardas, double defReiksme)
                {
                        return _options.ContainsKey(vardas) ? Convert.ToDouble(_options[vardas]) : defReiksme;
                }
                public static bool Gauti_BoolNustatyma(string vardas, bool defReiksme)
                {
                        return _options.ContainsKey(vardas) ? Convert.ToBoolean(_options[vardas]) : defReiksme;
                }
                #endregion
        }
}
```

Modulis „IO“:

```
using System;
using System.IO;
using System.Data;
using System.Drawing;
using System.Collections;
using System.Xml;
using System.Windows.Forms;
using Formos;

namespace Kodas
{
        public class IO
        {
                #region Kintamieji
                OpenFileDialog           _openFile_dlg;    //failo parinkimo dialogas
                SaveFileDialog           _saveFile_dlg;    //failo issaugojimo dialogas
                FolderBrowserDialog _openFold_dlg;//direktorijos parinkimo dialogas
                ColorDialog                      _openColor_dlg;  //spalvos parinkimo dialogas
                PapildomaFailoInfo       _pfi;                    //paskutinio nuskaityto failo info
                #endregion
                #region Pagalbines klases
                Funkcijos _fn;
                #endregion
                #region Konstruktorius
                public IO()
                {
                        _fn = new Funkcijos();
                }
                #endregion
                #region Properciai
                public PapildomaFailoInfo PFI
                {
                        get {return _pfi;}
                }
                #endregion
                #region Metodai

                public string Ivedimo_dialogas(string title, string filtras)
                {
                        _openFile_dlg = new OpenFileDialog();
                        _openFile_dlg.Filter = filtras;
                        string inicDir = Statikai.Gauti_StrNustatyma("Duomenu_direktorija");
                        if (inicDir == "")
                                inicDir = Statikai.RootDir;
                        _openFile_dlg.InitialDirectory = inicDir;
                        _openFile_dlg.Title = title;
                        if (_openFile_dlg.ShowDialog() == DialogResult.OK)
                        {
                                if(_openFile_dlg.FileName != "")
                                {
                                        return _openFile_dlg.FileName;
                                }
                        }
                        return "";
                }
                public string Issaugojimo_dialogas(string title, string filtras)
                {
                        _saveFile_dlg = new SaveFileDialog();
                        _saveFile_dlg.Filter = filtras;
                        string inicDir = Statikai.Gauti_StrNustatyma("Duomenu_direktorija");
                        if (inicDir == "")
                                inicDir = Statikai.RootDir;
                        _saveFile_dlg.InitialDirectory = inicDir;
                        _saveFile_dlg.Title = title;
                        if (_saveFile_dlg.ShowDialog() == DialogResult.OK)
                        {
                                if(_saveFile_dlg.FileName != "")
```

```
                {
                        return _saveFile_dlg.FileName;
                }
        }
        return "";
}
public string Ivedimo_dialogas_Dir(string title)
{
        _openFold_dlg = new FolderBrowserDialog();
        _openFold_dlg.SelectedPath = Statikai.RootDir;
        _openFold_dlg.Description = title;
        if (_openFold_dlg.ShowDialog() == DialogResult.OK)
        {
                if(_openFold_dlg.SelectedPath != "")
                        return _openFold_dlg.SelectedPath;
        }
        return "";
}
public Color Ivedimo_dialogas_Color(Color initColor)
{
        _openColor_dlg = new ColorDialog();
        _openColor_dlg.Color = initColor;
        _openColor_dlg.FullOpen = true;
        if (_openColor_dlg.ShowDialog() == DialogResult.OK)
        {
                return _openColor_dlg.Color;
        }
        return Color.Empty;
}


public ArrayList Nuskaityti_taskus_f(string f_kelias)
{
        ArrayList tsk_arr = new ArrayList();
        try
        {
                using (StreamReader sr = new StreamReader(f_kelias))
                {
                        _pfi = new PapildomaFailoInfo();
                        string line = "";
                        while ((line = sr.ReadLine()) != null && line != "")
                        {
                                string[] str_arr = line.Split(new char[1]{' '});
                                if (str_arr[0] == "*")
                                {
                                        #region Papildomos info nuskaitymas
                                        switch (str_arr[1])
                                        {
                                                case "DarbPlotas[W/H]":
                                                        _pfi.DarbPlotas = new
                                 Size(Convert.ToInt16(str_arr[2]), Convert.ToInt16(str_arr[3]));
                                                        break;
                                        }
                                        #endregion
                                }
                                else
                                {
                                        double x = Convert.ToDouble(str_arr[0]);
                                        double F = Convert.ToDouble(str_arr[1]);
                                        tsk_arr = _fn.Pildyti_Tasku_Masyva(tsk_arr, x, F);
                                }
                        }
                }
        }
        catch (System.IO.DirectoryNotFoundException)
        {
                MessageBox.Show("Nerasta nurodyta direktorija!");
                return null;
        }
        catch (Exception ex)
        {
                Error.Process(ex);
                return null;
        }
        return tsk_arr;
}
public ArrayList Nuskaityti_tarpinius_f(string f_kelias)
{
        ArrayList aTarp = new ArrayList();
        try
        {
                using (StreamReader sr = new StreamReader(f_kelias))
                {
                        _pfi = new PapildomaFailoInfo();
                        string line = "";
                        while ((line = sr.ReadLine()) != null && line != "")
                        {
                                string[] str_arr = line.Split(new char[1]{' '});
                                if (str_arr[0] == "*")
                                {
                                        #region Papildomos info nuskaitymas
```

```
                                        switch (str_arr[1])
                                        {
                                                case "Lembda":
                                                        _pfi.Lembda =
                                                                Convert.ToInt32(str_arr[2]);
                                                        break;
                                        }
                                        #endregion
                                }
                                else
                                {
                                        int             i       = -1;
                                        int             j       = -1;
                                        double  ki      = double.MinValue;
                                        int nKelintas = 0;

                                        foreach (string s in str_arr)
                                        {
                                                if (s != "")
                                                {
                                                        switch(nKelintas)
                                                        {
                                                                case 0:
                                                                        i = Convert.ToInt32(s);
                                                                        break;
                                                                case 1:
                                                                        j = Convert.ToInt32(s);
                                                                        break;
                                                                case 2:
                                                                        ki =
                                                                         Convert.ToDouble(s);
                                                                        break;
                                                                default:
                                                                        break;

                                                        }
                                                        nKelintas++;
                                                }
                                        }
                                        if(nKelintas < 3)
                                                throw new Exception("Faile blogi duomenys!");
                                        aTarp = _fn.Pildyti_Tarpinius_Duomenis(aTarp, i, j, ki);
                                }
                        }
                }
        }
        catch (System.IO.DirectoryNotFoundException)
        {
                MessageBox.Show("Nerasta nurodyta direktorija!");
                return null;
        }
        catch (Exception ex)
        {
                Error.Process(ex);
                return null;
        }
        return aTarp;
}
public Hashtable Nuskaityti_NustatymusXML(string xmlPath)
{
        try
        {
                Hashtable opts = new Hashtable();
                XmlDocument doc = new XmlDocument();
                doc.Load(xmlPath);
                XmlNode root = doc.DocumentElement;
                foreach (XmlNode nd in root.ChildNodes)
                        opts[nd.Name] = nd.InnerText;
                return opts;
        }
        catch (Exception ex)
        {
                Error.Process(ex);
                return null;
        }
}


public bool Issaugoti_taskus_f(ArrayList tsk_arr, String f_vardas, System.Drawing.Size size)
{
        try
        {
                using (StreamWriter sw = new StreamWriter(f_vardas, false))
                {
                        if (size != System.Drawing.Size.Empty)
                                sw.WriteLine("* DarbPlotas[W/H] "+size.Width.ToString()+"
                                                                "+size.Height.ToString());
                        sw.WriteLine("* Antraste --x--y--");
                        string line = "";
                        for (int ii = 0; ii < tsk_arr.Count; ii++)
                        {
```

```
                                line = _fn.Gauti_Masyvo_x(tsk_arr, ii).ToString() + " ";
                                line += _fn.Gauti_Masyvo_F(tsk_arr, ii).ToString();
                                sw.WriteLine(line);
                        }
                }
                return true;
        }
        catch (Exception ex)
        {
                Error.Process(ex);
                return false;
        }
}
public bool Issaugoti_IFS(ArrayList A, ArrayList C, ArrayList D, ArrayList E, ArrayList F, string
                                                                                        f_kelias)
{
        string f_vrd = _fn.Gauti_failo_varda(f_kelias, false);
        try
        {
                using (StreamWriter sw = new StreamWriter(f_kelias, false))
                {
                        sw.WriteLine(f_vrd + " {");
                        string line = "";
                        double tikimybe = System.Math.Round((double)1/A.Count, 2);
                        for (int ii = 0; ii < A.Count; ii++)
                        {
                                double a = System.Math.Round((double)A[ii],2);
                                double b = 0;
                                double c = System.Math.Round((double)C[ii],2);
                                //double d = System.Math.Round((double)D[ii] * (-1), 2);
                                double d = System.Math.Round((double)D[ii],2);
                                //double d = 0;
                                double e = System.Math.Round((double)E[ii],2);
                                double f = System.Math.Round((double)F[ii],2);

                                line = String.Format("{0} {1} {2} {3} {4} {5} {6}", a.ToString(),
                 b.ToString(), c.ToString(), d.ToString(), e.ToString(), f.ToString(), tikimybe);
                                line = line.Replace(",", ".");
                                sw.WriteLine(line);
                        }
                        sw.WriteLine("}");
                }
                return true;
        }
        catch (Exception ex)
        {
                Error.Process(ex);
                return false;
        }
}
public bool Issaugoti_tarpinius_f(ArrayList aIntAtitiktys, string f_kelias, int lembda)
{
        try
        {
                using (StreamWriter sw = new StreamWriter(f_kelias, false))
                {
                        sw.WriteLine("* Lembda "+lembda.ToString());
                        sw.WriteLine("* Antraste --i--j--ki--metrika--");
                        int ii = 0;
                        foreach (object oAtitiktis in aIntAtitiktys)
                        {
                                sw.WriteLine(String.Format("{0} {1} {2} {3}",
                                 ii.ToString().PadLeft(5,' '),
                                 ((object[])oAtitiktis)[0].ToString().PadLeft(5,' '),
                                 ((object[])oAtitiktis)[1].ToString().PadLeft(20,' '),
                                 ((object[])oAtitiktis)[3].ToString().PadLeft(20,' ')));
                                ii++;
                        }
                }
                return true;
        }
        catch (Exception ex)
        {
                Error.Process(ex);
                return false;
        }
}
public void Issaugoti_NustatymusXML()
{
        try
        {
                string failas = Statikai.RootDir+"\\Nustatymai.xml";
                XmlDocument doc = new XmlDocument();
                doc.Load(failas);
                XmlNode root = doc.DocumentElement;

                IDictionaryEnumerator en = Statikai.Nustatymai.GetEnumerator();
                while (en.MoveNext())
                {
                        XmlNodeList xnl = doc.GetElementsByTagName(en.Key.ToString());
                        if (xnl.Count == 1)
```

```
                                        xnl[0].InnerText = en.Value.ToString();
                                else
                                {
                                        XmlElement elem = doc.CreateElement(en.Key.ToString());
                                        elem.InnerText = en.Value.ToString();
                                        root.AppendChild(elem);
                                }
                        }
                        doc.Save(failas);
                }
                catch (Exception ex)
                {
                        Error.Process(ex);
                }
        }
        public bool Issaugoti_Paveiksla(string f_kelias, Image img)
        {
                try
                {
                        img.Save(f_kelias);
                        return true;
                }
                catch (Exception ex)
                {
                        Error.Process(ex);
                        return false;
                }
        }

        public void Isvesti_aibe(ListBox lb, ArrayList arr, string comm)
        {
                string eilute = comm;
                for (int ii = 0; ii < arr.Count; ii++)
                {
                        eilute += System.Math.Round((double)arr[ii], 2) + "     ";
                }
                lb.Items.Insert(0, eilute);
        }
        public void Isvesti_pranesima(ListBox lb, string pranesimas)
        {
                lb.Items.Insert(0, pranesimas);
        }
        public void Isvesti_tasku_aibe(ListBox lb, ArrayList arr, string comm)
        {
                string eilute = comm;
                for (int ii = 0; ii < arr.Count; ii++)
                {
                        eilute += "{" + System.Math.Round(_fn.Gauti_Masyvo_x(arr, ii), 2) + "; " +
                                        System.Math.Round(_fn.Gauti_Masyvo_F(arr, ii), 2) + "} ";
                }
                lb.Items.Insert(0, eilute);
        }
        #endregion
}
public class PapildomaFailoInfo
{
        #region Kintamieji
        private Size     _sDarbPlotas     = Size.Empty;    //staciakampis, kuriame talpinami taskai
        private int              _lembda                  = -1;                    //interpoliavimo tasku
                                                           atrankos parametras koliazo organizavimui
        #endregion
        #region Properciai
        public Size DarbPlotas
        {
                get {return _sDarbPlotas;}
                set {_sDarbPlotas = value;}
        }
        public int Lembda
        {
                get {return _lembda;}
                set {_lembda = value;}
        }
        #endregion
}
}
```

2 PRIEDAS
FRAKTALINIO MODELIAVIMO PRIEMONIŲ RINKINYS FRACTAL VISION

FRACTAL VISION – tai programinis paketas, kurio pagrindinė paskirtis yra fraktalų generavimas. Čia galima rasti darbe aptartų determinuotojo bei atsitiktinių iteracijų fraktalų sintezės metodų (1.1.1 skyrius) realizaciją, fraktalinės dimensijos skaičiavimo „kubiukų“ metodu (1.2 skyrius; 3 teorema) galimybę.

FRACTAL VISION paketo trumpa anotacija

Produkto pavadinimas:	Fractal Vision	
Autorius:	Dick Oliver, Cedar Software, RR 1 Box 5140, Morrisville VT 05661 USA	
Sukūrimo data:	1990 m. balandžio mėn.	
Pagrindiniai paketo įrankiai		
Failas **FV.exe** - fraktalinis modeliavimas bei pagalbinės funkcijos		
LOAD	Picture	duomenų įvedimas iš failo, turinčio *pcx* formatą
	IFS codes	iteruotųjų funkcijų sistemos iš tekstinio failo įvedimo galimybė
SAVE	Picture	gauto rezultato išsaugojimas faile, *pcx* formatu
	IFS codes	IFS, generuojančios rezultatą, išsaugojimas tekstiniame faile
DRAW	determinuotasis fraktalo generavimo algoritmas	
PAINT	atsitiktinių iteracijų fraktalo generavimo algoritmas	
EXTRAS	Fractal Dim	fraktalinės dimensijos skaičiavimas „kubiukų“ metodu
Failas **INTERPO.exe** – interaktyvus fraktalinis interpoliavimas		