Національний Технічний Університет України “КПІ”

Навчально-науковий комплекс

«Інститут прикладного системного аналізу»

ЛАБОРАТОРНА РОБОТА № 2

З дисципліни: Основи системного аналізу

# Виконали:

# Глузман Марк

Череда Григорій

# група КА-11

Київ 2014

**Задание**

В каждом из вариантов заданы:

* исходные таблицы данных;
* размерности векторов x1, x2, x3 - n1,n2,n3;
* количество выборок n (для расчета);
* количество целевых функций m;
* веса целевых функций;
* метод решения несовместных систем уравнений, в которых число неизвестных не равно числу уравнений;

Наборы P1, P2, P3 – степени полинома Чебышева (Лежандра, Лагера, Эрмита и др.) для x1, x2, x3 – следует подобрать самостоятельно, исходя из критерия минимума максимального отклонения функций:.

Требуется:

1. сформировать целевые функции и вывести на печать

* значения всех промежуточных коэффициентов (λ, a, c) и функций (Ψ,Φ);
* вид полученных функций  через: 1); 2) полиномы Чебышева;

3) в форме обычных многочленов (целесообразно это предусмотреть в файле результатов) как в нормированном, так и в восстановленном виде;

1. построить графики исходных (по выборкам) и полученных функций;
2. оценить погрешность восстановленных функций  по отношению к исходной заданной выборке.

## 4) сделать письменный отчет о выполненной работе, включив листинг программы.

**Варіант 1**

Размерности векторов X1, X2, Х3 – n1=2, n2=2, n3=3;

количество выборок для расчета = 45;

количество целевых функций m=4;

 принимаются равными нормированным значениям ;

метод решения несовместной системы уравнений: **метод сопряженных направлений.**

**Метод спряженних напрямків розв’язку  
СЛНР**

We say that two non-zero vectors *u* and *v* are [conjugate](http://en.wikipedia.org/wiki/Inner_automorphism) (with respect to *A*) if

 \mathbf{u}^\mathrm{T} \mathbf{A} \mathbf{v} = 0. 

Since **A** is symmetric and positive definite, the left-hand side defines an [inner product](http://en.wikipedia.org/wiki/Inner_product_space)

 \langle \mathbf{u},\mathbf{v} \rangle_\mathbf{A} := \langle \mathbf{A} \mathbf{u}, \mathbf{v}\rangle = \langle \mathbf{u},  \mathbf{A}^\mathrm{T} \mathbf{v}\rangle = \langle \mathbf{u}, \mathbf{A}\mathbf{v} \rangle = \mathbf{u}^\mathrm{T} \mathbf{A} \mathbf{v}. 

Two vectors are conjugate if they are orthogonal with respect to this inner product. Being conjugate is a symmetric relation: if **u** is conjugate to **v**, then **v** is conjugate to **u**.

Suppose that P=\{\mathbf{p}_k: \forall i\neq k, i, k \in [1,n], \langle\mathbf{p}_i,\mathbf{p}_k\rangle_{A}=0  \}is a set of *n* mutually conjugate directions. Then Pis a [basis](http://en.wikipedia.org/wiki/Basis_%28linear_algebra%29) of \mathbb{R}^n, so within Pwe can expand the solution \mathbf{x}_*of \mathbf{Ax} = \mathbf{b}:

 \mathbf{x}_* = \sum^{n}_{i=1} \alpha_i \mathbf{p}_i

and we see that

 \mathbf{b}=\mathbf{A}\mathbf{x}_* = \sum^{n}_{i=1} \alpha_i  \mathbf{A} \mathbf{p}_i.

For any \mathbf{p}_k \in P,

 \mathbf{p}_k^\mathrm{T} \mathbf{b}=\mathbf{p}_k^\mathrm{T} \mathbf{A}\mathbf{x}_* = \sum^{n}_{i=1} \alpha_i\mathbf{p}_k^\mathrm{T} \mathbf{A} \mathbf{p}_i=\alpha_k\mathbf{p}_k^\mathrm{T} \mathbf{A} \mathbf{p}_k.

(because \forall i\neq k, p_i, p_kare mutually conjugate)

 \alpha_k = \frac{\mathbf{p}_k^\mathrm{T} \mathbf{b}}{\mathbf{p}_k^\mathrm{T} \mathbf{A} \mathbf{p}_k} = \frac{\langle \mathbf{p}_k, \mathbf{b}\rangle}{\,\,\,\langle \mathbf{p}_k,  \mathbf{p}_k\rangle_\mathbf{A}} = \frac{\langle \mathbf{p}_k, \mathbf{b}\rangle}{\,\,\,\|\mathbf{p}_k\|_\mathbf{A}^2}. 

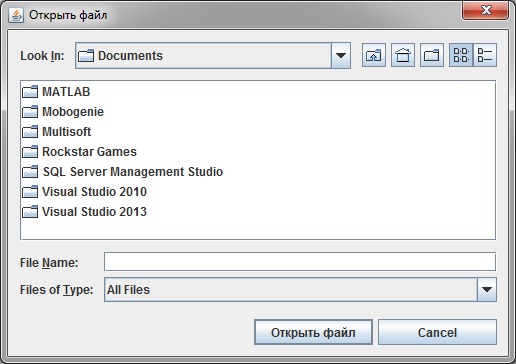
This result is perhaps most transparent by considering the inner product defined above.

This gives the following method for solving the equation **Ax** = **b**: find a sequence of *n* conjugate directions, and then compute the coefficients \scriptstyle \alpha_k.

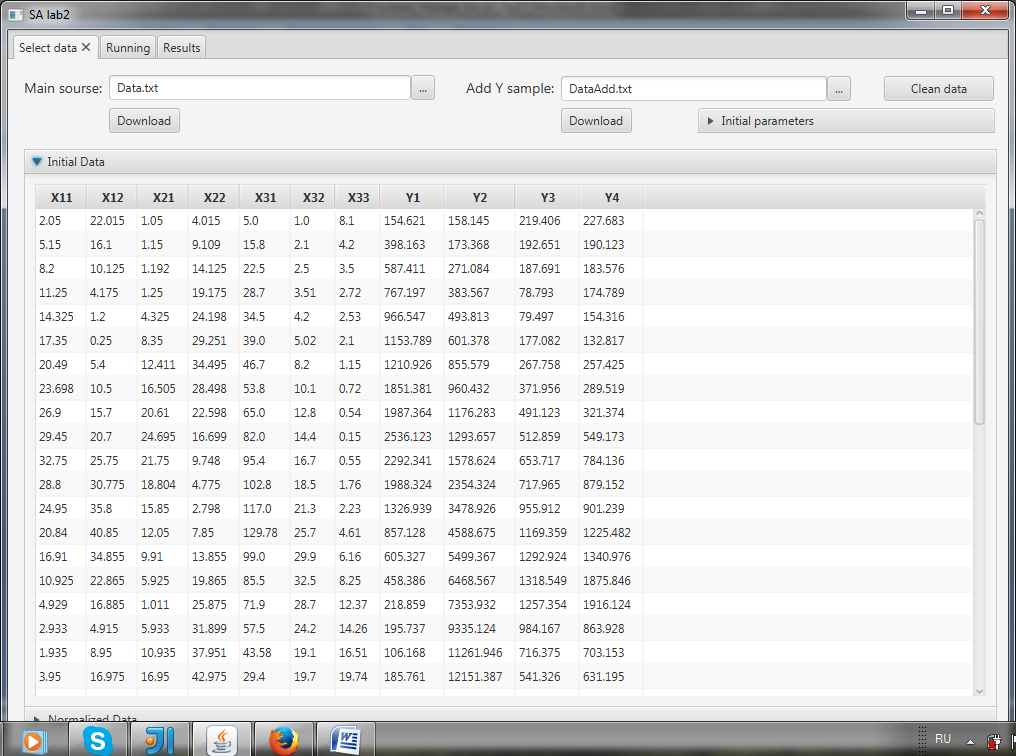
Ми бачимо, що матриця A для даного методу повинна бути симетрична та додатньовизначена. Щоб застосувати цей метод (у нашому випадку наша матриця A зовсім не є симетричною), домножимо рівняння **Ax** = **b** зліва і справа на AT : **AT Ax** = **AT b.** Тепер матриця **AT A** є симетричною і додатньовизначеною.

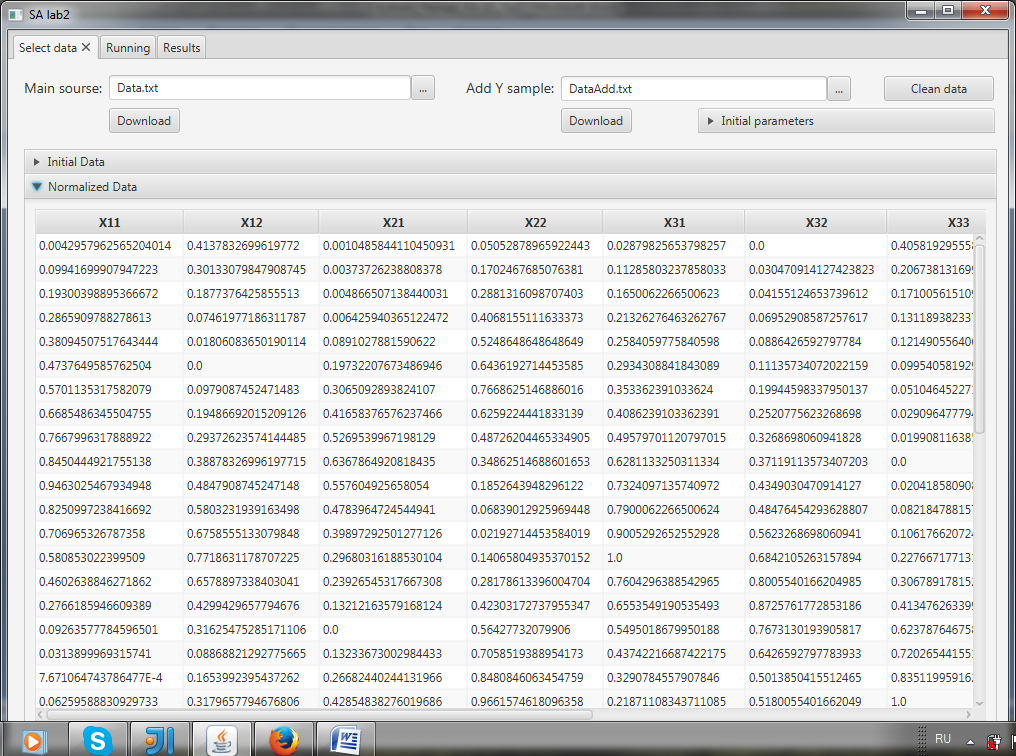
**Інтерфейс користувача**

Тут можемо бачити меню вибору файлу з даними.

****

Тут завантажено дані з файлу

****

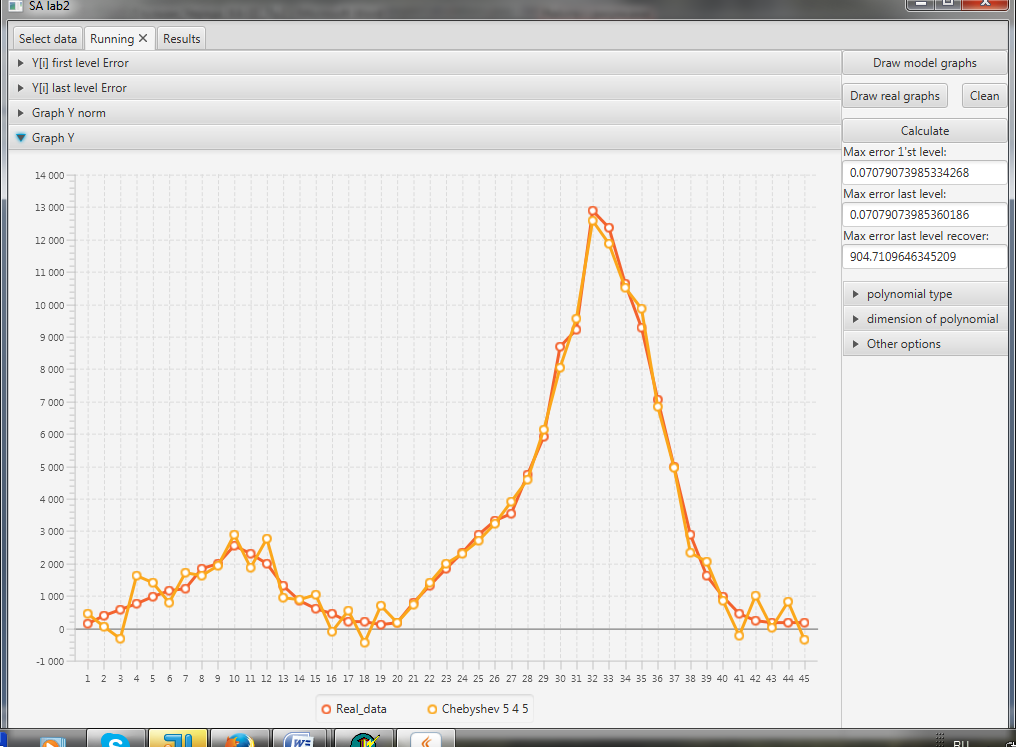
Можна відразу переглянути пораховані нормовані значення:  


У віконечку початкових параметрів можна обрати порядок для кожної із змінних Xi, розмірність вибірки, розмірність змінних.

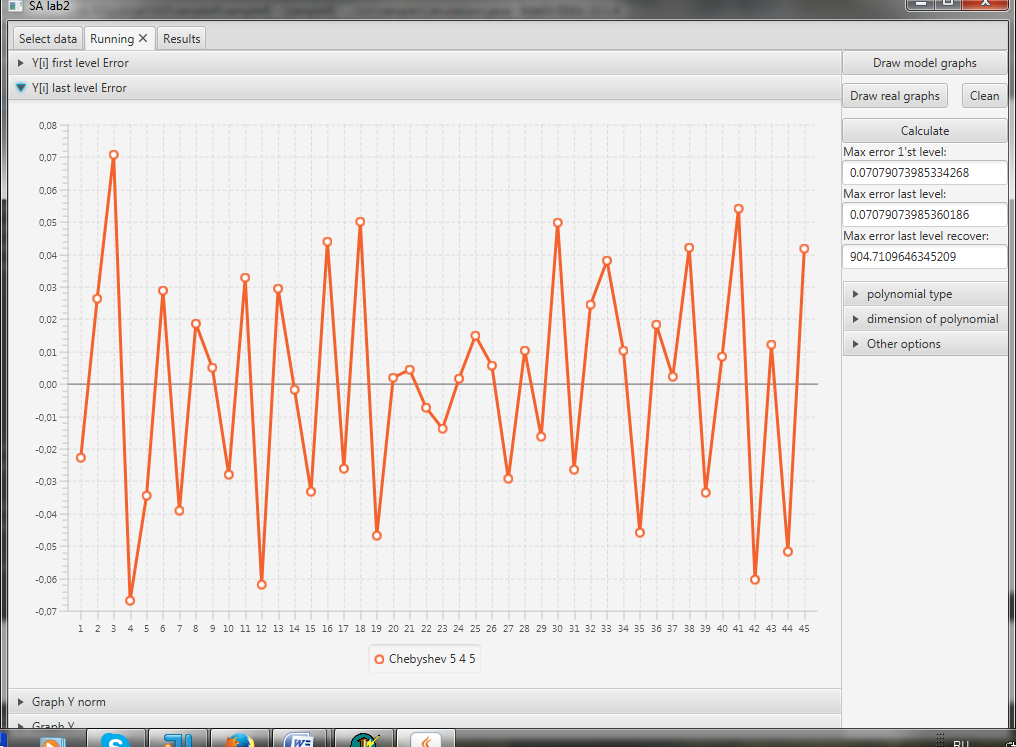
**Виконання відновлення функціональної залежності**

Розглянемо результати роботи на вибірці, спробуємо апроксимувати

Y1

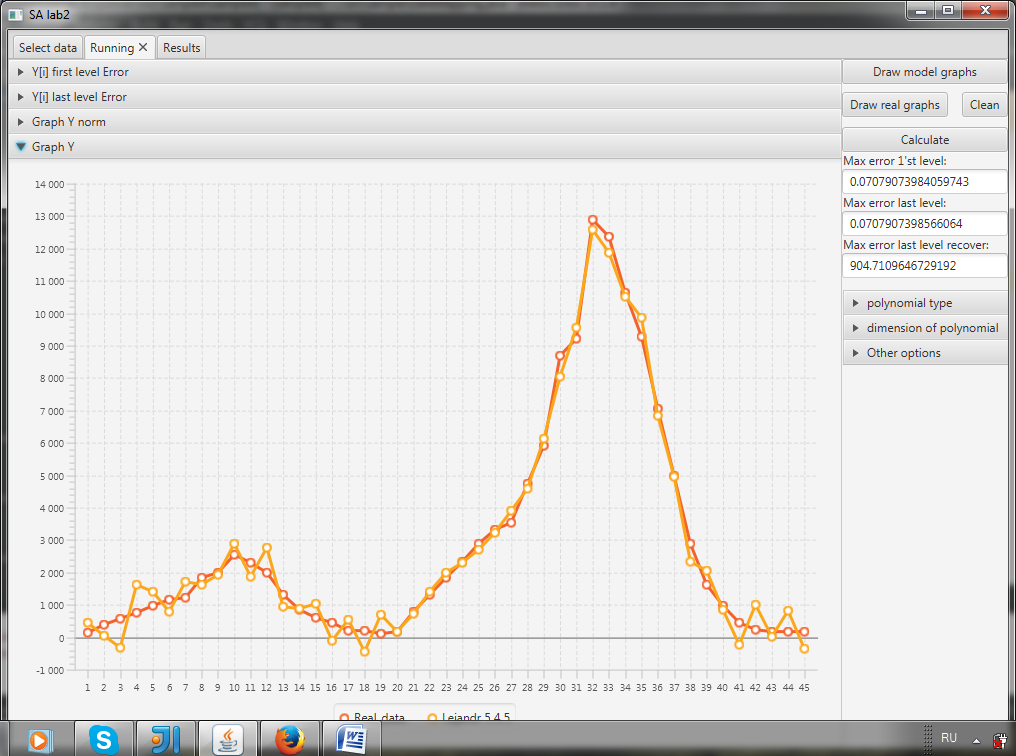


*Рис.1*

В даному випадку використовувалися поліноми Чебишева з порядком для змінних по 5 4 5, ось ми маємо такий результат. Розглянемо вектор похибки:  


*Рис.2*

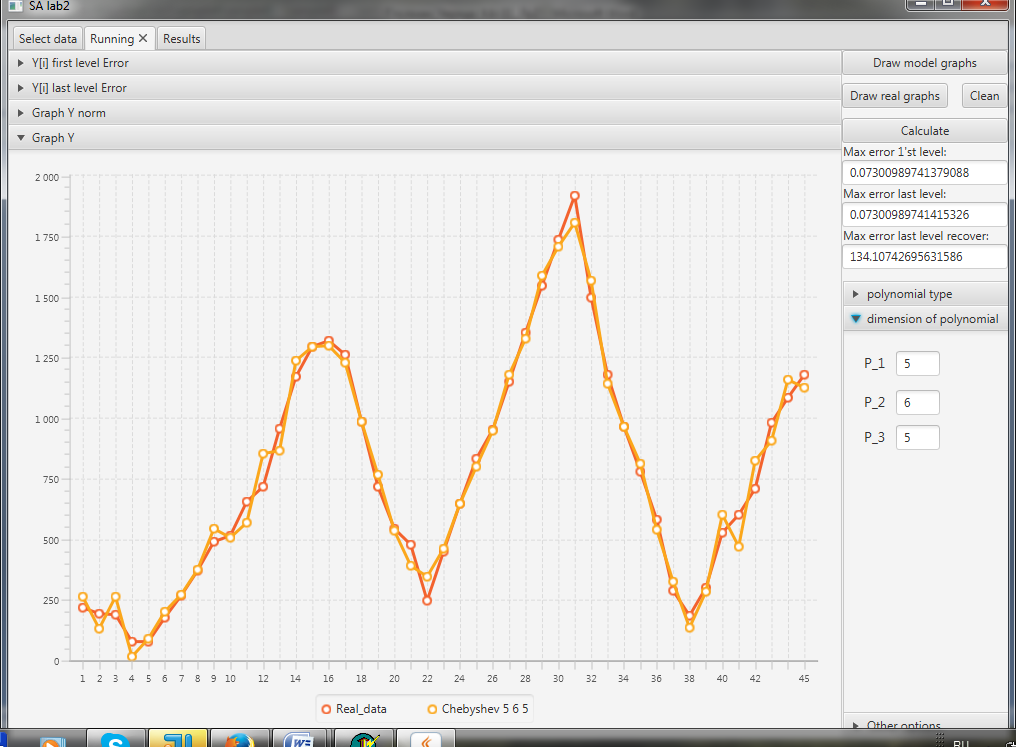
Розглянемо той же випадок для поліномів Лєжандра:



*Рис.3*

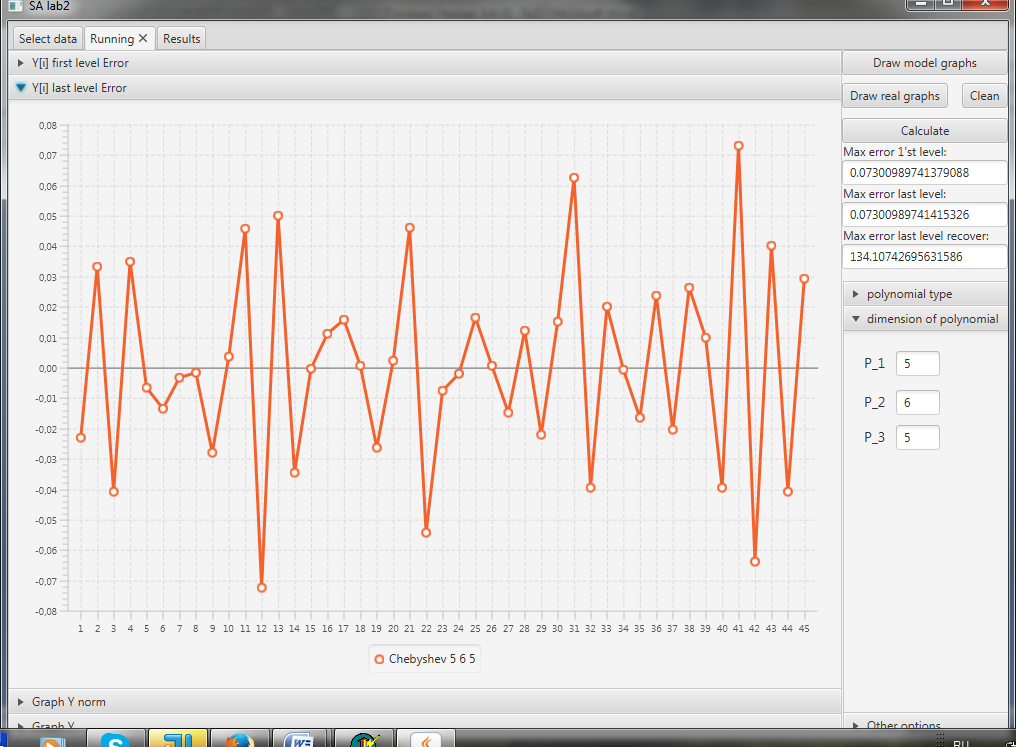
В данному випадку, максимальне відхилення у випадку з поліномами Лежандра є аналогічним (різниця в 8 знаці після коми у ненормованому випадку. Взагалом, для всіх типів поліномів різниця у виконанні програми незначна.

Розглянемо виконнання з поліномами Чебишева на Y3:



*Рис.4*

похибка:



*Рис.5*

Розглянемо результати апроксимації нашої функції.  
Було взято апроксимацію много членом Чебишева з параметрами для відповідних Хі 5 6 5

**Y1**

Полученные коэффциенты при решении задачи

Первый уровень (lambda1)

0,0437 -0,3882 0,1924 0,1006 -0,1032 -0,0248

0,0437 0,8909 0,0685 -0,2762 0,0338 0,0841

Первый уровень (Lambda2)

0,0437 0,2498 0,0173 -0,0282 0,0146 -0,0185 -0,0138

0,0437 -0,3175 -0,0645 -0,1096 0,0801 0,0072 0,0961

Первый уровень (Lambda3)

0,0437 -0,0555 -0,3090 0,2244 0,0996 0,0479

0,0437 -0,4610 -0,2089 0,3498 -0,1717 -0,1768

0,0437 0,1771 -0,1226 0,0568 0,0222 0,0470

Второй уровень (а1)

1,0000 1,0000

Второй уровень (а2)

1,0000 1,0000

Второй уровень (а3)

1,0000 1,0000 1,0000

Третий уровень (С)

1,0000 1,0000 1,0000

psi функции

(0.043738173387211914)\*T0(x10)+(-0.38819861123857685)\*T1(x10)+(0.1924033055944355)\*T2(x10)+(0.1006049366316568)\*T3(x10)+(-0.10318981673376149)\*T4(x10)+(0.043738173387211914)\*T0(x11)+(0.8909083809119096)\*T1(x11)+(0.06847954828442625)\*T2(x11)+(-0.27619495345763023)\*T3(x11)+(0.03380005098788947)\*T4(x11)+(0.043738173387211914)\*T0(x20)+(0.24977638477189293)\*T1(x20)+(0.017310708669663367)\*T2(x20)+(-0.02820011634934037)\*T3(x20)+(0.014623357698126026)\*T4(x20)+(-0.018485588232672465)\*T5(x20)+(0.043738173387211914)\*T0(x21)+(-0.3175272670716032)\*T1(x21)+(-0.06452301184052522)\*T2(x21)+(-0.1096441185168237)\*T3(x21)+(0.08009013905587901)\*T4(x21)+(0.0071774720756551105)\*T5(x21)+(0.043738173387211914)\*T0(x30)+(-0.05546046952023398)\*T1(x30)+(-0.3089701927531968)\*T2(x30)+(0.22442749482557453)\*T3(x30)+(0.0996386384452334)\*T4(x30)+(0.043738173387211914)\*T0(x31)+(-0.46103725705149684)\*T1(x31)+(-0.20888955779662083)\*T2(x31)+(0.3498073845916471)\*T3(x31)+(-0.17166143924090818)\*T4(x31)+(0.043738173387211914)\*T0(x32)+(0.17709395082935087)\*T1(x32)+(-0.12262962463002791)\*T2(x32)+(0.0568295542436866)\*T3(x32)+(0.022242778627288507)\*T4(x32)

Второй уровень функции

Ф1=(1.000000000001015)\*psi1(x10)+(0.9999999999998458)\*psi1(x11)+

Ф2=(0.9999999999975101)\*psi2(x20)+(1.0000000000022786)\*psi2(x21)+

Ф3=(1.0000000000000118)\*psi3(x30)+(0.9999999999982314)\*psi3(x31)+(0.9999999999954737)\*psi3(x32)

Третий уровень функции

1.0000000000001288\*Ф(x1)+1.000000000000166\*Ф(x2)+1.0000000000001856\*Ф(x3)

Получившаяся в результате функция

1.0000000000001288\*(1.000000000001015\*((0.043738173387211914)\*T0(x10)+(-0.38819861123857685)\*T1(x10)+(0.1924033055944355)\*T2(x10)+(0.1006049366316568)\*T3(x10)+(-0.10318981673376149)\*T4(x10)+(0.043738173387211914)\*T0(x11)+(0.8909083809119096)\*T1(x11)+(0.06847954828442625)\*T2(x11)+(-0.27619495345763023)\*T3(x11)+(0.03380005098788947)\*T4(x11)+)+0.9999999999998458\*((0.043738173387211914)\*T0(x10)+(-0.38819861123857685)\*T1(x10)+(0.1924033055944355)\*T2(x10)+(0.1006049366316568)\*T3(x10)+(-0.10318981673376149)\*T4(x10)+(0.043738173387211914)\*T0(x11)+(0.8909083809119096)\*T1(x11)+(0.06847954828442625)\*T2(x11)+(-0.27619495345763023)\*T3(x11)+(0.03380005098788947)\*T4(x11)+)+)1.0000000000001288\*(0.9999999999975101\*((0.043738173387211914)\*T0(x20)+(0.24977638477189293)\*T1(x20)+(0.017310708669663367)\*T2(x20)+(-0.02820011634934037)\*T3(x20)+(0.014623357698126026)\*T4(x20)+(-0.018485588232672465)\*T5(x20)+(0.043738173387211914)\*T0(x21)+(-0.3175272670716032)\*T1(x21)+(-0.06452301184052522)\*T2(x21)+(-0.1096441185168237)\*T3(x21)+(0.08009013905587901)\*T4(x21)+(0.0071774720756551105)\*T5(x21)+)+1.0000000000022786\*((0.043738173387211914)\*T0(x20)+(0.24977638477189293)\*T1(x20)+(0.017310708669663367)\*T2(x20)+(-0.02820011634934037)\*T3(x20)+(0.014623357698126026)\*T4(x20)+(-0.018485588232672465)\*T5(x20)+(0.043738173387211914)\*T0(x21)+(-0.3175272670716032)\*T1(x21)+(-0.06452301184052522)\*T2(x21)+(-0.1096441185168237)\*T3(x21)+(0.08009013905587901)\*T4(x21)+(0.0071774720756551105)\*T5(x21)+)+)1.0000000000001288\*(1.0000000000000118\*((0.043738173387211914)\*T0(x30)+(-0.05546046952023398)\*T1(x30)+(-0.3089701927531968)\*T2(x30)+(0.22442749482557453)\*T3(x30)+(0.0996386384452334)\*T4(x30)+(0.043738173387211914)\*T0(x31)+(-0.46103725705149684)\*T1(x31)+(-0.20888955779662083)\*T2(x31)+(0.3498073845916471)\*T3(x31)+(-0.17166143924090818)\*T4(x31)+(0.043738173387211914)\*T0(x32)+(0.17709395082935087)\*T1(x32)+(-0.12262962463002791)\*T2(x32)+(0.0568295542436866)\*T3(x32)+(0.022242778627288507)\*T4(x32))+0.9999999999982314\*((0.043738173387211914)\*T0(x30)+(-0.05546046952023398)\*T1(x30)+(-0.3089701927531968)\*T2(x30)+(0.22442749482557453)\*T3(x30)+(0.0996386384452334)\*T4(x30)+(0.043738173387211914)\*T0(x31)+(-0.46103725705149684)\*T1(x31)+(-0.20888955779662083)\*T2(x31)+(0.3498073845916471)\*T3(x31)+(-0.17166143924090818)\*T4(x31)+(0.043738173387211914)\*T0(x32)+(0.17709395082935087)\*T1(x32)+(-0.12262962463002791)\*T2(x32)+(0.0568295542436866)\*T3(x32)+(0.022242778627288507)\*T4(x32))+0.9999999999954737\*((0.043738173387211914)\*T0(x30)+(-0.05546046952023398)\*T1(x30)+(-0.3089701927531968)\*T2(x30)+(0.22442749482557453)\*T3(x30)+(0.0996386384452334)\*T4(x30)+(0.043738173387211914)\*T0(x31)+(-0.46103725705149684)\*T1(x31)+(-0.20888955779662083)\*T2(x31)+(0.3498073845916471)\*T3(x31)+(-0.17166143924090818)\*T4(x31)+(0.043738173387211914)\*T0(x32)+(0.17709395082935087)\*T1(x32)+(-0.12262962463002791)\*T2(x32)+(0.0568295542436866)\*T3(x32)+(0.022242778627288507)\*T4(x32)))

Получившаяся в результате восстановленная функция

106.168+12780.075\*(1.0000000000001288\*(1.000000000001015\*((0.043738173387211914)\*T0(x10)+(-0.38819861123857685)\*T1(x10)+(0.1924033055944355)\*T2(x10)+(0.1006049366316568)\*T3(x10)+(-0.10318981673376149)\*T4(x10)+(0.043738173387211914)\*T0(x11)+(0.8909083809119096)\*T1(x11)+(0.06847954828442625)\*T2(x11)+(-0.27619495345763023)\*T3(x11)+(0.03380005098788947)\*T4(x11)+)+0.9999999999998458\*((0.043738173387211914)\*T0(x10)+(-0.38819861123857685)\*T1(x10)+(0.1924033055944355)\*T2(x10)+(0.1006049366316568)\*T3(x10)+(-0.10318981673376149)\*T4(x10)+(0.043738173387211914)\*T0(x11)+(0.8909083809119096)\*T1(x11)+(0.06847954828442625)\*T2(x11)+(-0.27619495345763023)\*T3(x11)+(0.03380005098788947)\*T4(x11)+)+)1.0000000000001288\*(0.9999999999975101\*((0.043738173387211914)\*T0(x20)+(0.24977638477189293)\*T1(x20)+(0.017310708669663367)\*T2(x20)+(-0.02820011634934037)\*T3(x20)+(0.014623357698126026)\*T4(x20)+(-0.018485588232672465)\*T5(x20)+(0.043738173387211914)\*T0(x21)+(-0.3175272670716032)\*T1(x21)+(-0.06452301184052522)\*T2(x21)+(-0.1096441185168237)\*T3(x21)+(0.08009013905587901)\*T4(x21)+(0.0071774720756551105)\*T5(x21)+)+1.0000000000022786\*((0.043738173387211914)\*T0(x20)+(0.24977638477189293)\*T1(x20)+(0.017310708669663367)\*T2(x20)+(-0.02820011634934037)\*T3(x20)+(0.014623357698126026)\*T4(x20)+(-0.018485588232672465)\*T5(x20)+(0.043738173387211914)\*T0(x21)+(-0.3175272670716032)\*T1(x21)+(-0.06452301184052522)\*T2(x21)+(-0.1096441185168237)\*T3(x21)+(0.08009013905587901)\*T4(x21)+(0.0071774720756551105)\*T5(x21)+)+)1.0000000000001288\*(1.0000000000000118\*((0.043738173387211914)\*T0(x30)+(-0.05546046952023398)\*T1(x30)+(-0.3089701927531968)\*T2(x30)+(0.22442749482557453)\*T3(x30)+(0.0996386384452334)\*T4(x30)+(0.043738173387211914)\*T0(x31)+(-0.46103725705149684)\*T1(x31)+(-0.20888955779662083)\*T2(x31)+(0.3498073845916471)\*T3(x31)+(-0.17166143924090818)\*T4(x31)+(0.043738173387211914)\*T0(x32)+(0.17709395082935087)\*T1(x32)+(-0.12262962463002791)\*T2(x32)+(0.0568295542436866)\*T3(x32)+(0.022242778627288507)\*T4(x32))+0.9999999999982314\*((0.043738173387211914)\*T0(x30)+(-0.05546046952023398)\*T1(x30)+(-0.3089701927531968)\*T2(x30)+(0.22442749482557453)\*T3(x30)+(0.0996386384452334)\*T4(x30)+(0.043738173387211914)\*T0(x31)+(-0.46103725705149684)\*T1(x31)+(-0.20888955779662083)\*T2(x31)+(0.3498073845916471)\*T3(x31)+(-0.17166143924090818)\*T4(x31)+(0.043738173387211914)\*T0(x32)+(0.17709395082935087)\*T1(x32)+(-0.12262962463002791)\*T2(x32)+(0.0568295542436866)\*T3(x32)+(0.022242778627288507)\*T4(x32))+0.9999999999954737\*((0.043738173387211914)\*T0(x30)+(-0.05546046952023398)\*T1(x30)+(-0.3089701927531968)\*T2(x30)+(0.22442749482557453)\*T3(x30)+(0.0996386384452334)\*T4(x30)+(0.043738173387211914)\*T0(x31)+(-0.46103725705149684)\*T1(x31)+(-0.20888955779662083)\*T2(x31)+(0.3498073845916471)\*T3(x31)+(-0.17166143924090818)\*T4(x31)+(0.043738173387211914)\*T0(x32)+(0.17709395082935087)\*T1(x32)+(-0.12262962463002791)\*T2(x32)+(0.0568295542436866)\*T3(x32)+(0.022242778627288507)\*T4(x32))))

Полиномиальное представление

-3,211\*X11^1 + 4,136\*X11^2 + 6,287\*X11^3 + -8,698\*X11^4 + 1,996\*X11^5 +

1,693\*X12^1 + -11,455\*X12^2 + 58,134\*X12^3 + -94,652\*X12^4 + 45,738\*X12^5 +

2,466\*X21^1 + -19,449\*X21^2 + 58,894\*X21^3 + -72,991\*X21^4 + 30,435\*X21^5 +

-3,098\*X22^1 + 41,200\*X22^2 + -124,232\*X22^3 + 143,055\*X22^4 + -56,511\*X22^5 +

-6,731\*X31^1 + 40,910\*X31^2 + -102,568\*X31^3 + 105,022\*X31^4 + -36,726\*X31^5 +

16,170\*X32^1 + -96,243\*X32^2 + 204,753\*X32^3 + -179,262\*X32^4 + 55,021\*X32^5 +

5,807\*X33^1 + -31,265\*X33^2 + 71,861\*X33^3 + -71,580\*X33^4 + 25,512\*X33^5 +

+ -0,725

Ошибка восстановленой функции

1012.9420565097541

**Y2**

Полученные коэффциенты при решении задачи

Первый уровень (lambda1)

0,0289 -0,2459 -0,1588 0,0709 0,0473 -0,0041

0,0289 -0,1426 0,0685 0,0728 -0,0465 -0,0389

Первый уровень (Lambda2)

0,0289 0,1713 0,1117 0,0554 0,0048 0,0041 0,0045

0,0289 -0,1524 0,0749 -0,0084 -0,0245 0,0295 -0,0424

Первый уровень (Lambda3)

0,0289 -0,2038 0,0296 -0,1100 -0,0273 0,0179

0,0289 0,3089 -0,2568 -0,0061 0,0291 0,0602

0,0289 0,0441 0,1163 -0,0276 0,0004 -0,0629

Второй уровень (а1)

1,0000 1,0000

Второй уровень (а2)

1,0000 1,0000

Второй уровень (а3)

1,0000 1,0000 1,0000

Третий уровень (С)

1,0000 1,0000 1,0000

psi функции

(0.02892483158060538)\*T0(x10)+(-0.2459239953832973)\*T1(x10)+(-0.15879375000572812)\*T2(x10)+(0.07091215649756324)\*T3(x10)+(0.047324770161718724)\*T4(x10)+(0.02892483158060538)\*T0(x11)+(-0.14257903687734902)\*T1(x11)+(0.06850505389057737)\*T2(x11)+(0.07280965194855385)\*T3(x11)+(-0.04648313343908823)\*T4(x11)+(0.02892483158060538)\*T0(x20)+(0.17130673961416107)\*T1(x20)+(0.11168447711848535)\*T2(x20)+(0.05544000156556074)\*T3(x20)+(0.004823103259512977)\*T4(x20)+(0.004136631279998428)\*T5(x20)+(0.02892483158060538)\*T0(x21)+(-0.15239490728489724)\*T1(x21)+(0.07490960547623698)\*T2(x21)+(-0.008366908062411074)\*T3(x21)+(-0.0244545970371083)\*T4(x21)+(0.029476790626772158)\*T5(x21)+(0.02892483158060538)\*T0(x30)+(-0.20380829050398114)\*T1(x30)+(0.02959154462714906)\*T2(x30)+(-0.10999304628147029)\*T3(x30)+(-0.027316395676368014)\*T4(x30)+(0.02892483158060538)\*T0(x31)+(0.3088816128868494)\*T1(x31)+(-0.25678527046718597)\*T2(x31)+(-0.006063355489488809)\*T3(x31)+(0.0291435460700444)\*T4(x31)+(0.02892483158060538)\*T0(x32)+(0.04406037494560062)\*T1(x32)+(0.11629422919889441)\*T2(x32)+(-0.027637647599549054)\*T3(x32)+(3.977251290206539E-4)\*T4(x32)

Второй уровень функции

Ф1=(1.0000000000006353)\*psi1(x10)+(0.9999999999968111)\*psi1(x11)+

Ф2=(0.9999999999918349)\*psi2(x20)+(1.0000000000011307)\*psi2(x21)+

Ф3=(0.9999999999972914)\*psi3(x30)+(1.0000000000018672)\*psi3(x31)+(1.0000000000014242)\*psi3(x32)

Третий уровень функции

1.000000000000715\*Ф(x1)+1.0000000000000975\*Ф(x2)+0.9999999999987077\*Ф(x3)

Получившаяся в результате функция

1.000000000000715\*(1.0000000000006353\*((0.02892483158060538)\*T0(x10)+(-0.2459239953832973)\*T1(x10)+(-0.15879375000572812)\*T2(x10)+(0.07091215649756324)\*T3(x10)+(0.047324770161718724)\*T4(x10)+(0.02892483158060538)\*T0(x11)+(-0.14257903687734902)\*T1(x11)+(0.06850505389057737)\*T2(x11)+(0.07280965194855385)\*T3(x11)+(-0.04648313343908823)\*T4(x11)+)+0.9999999999968111\*((0.02892483158060538)\*T0(x10)+(-0.2459239953832973)\*T1(x10)+(-0.15879375000572812)\*T2(x10)+(0.07091215649756324)\*T3(x10)+(0.047324770161718724)\*T4(x10)+(0.02892483158060538)\*T0(x11)+(-0.14257903687734902)\*T1(x11)+(0.06850505389057737)\*T2(x11)+(0.07280965194855385)\*T3(x11)+(-0.04648313343908823)\*T4(x11)+)+)1.000000000000715\*(0.9999999999918349\*((0.02892483158060538)\*T0(x20)+(0.17130673961416107)\*T1(x20)+(0.11168447711848535)\*T2(x20)+(0.05544000156556074)\*T3(x20)+(0.004823103259512977)\*T4(x20)+(0.004136631279998428)\*T5(x20)+(0.02892483158060538)\*T0(x21)+(-0.15239490728489724)\*T1(x21)+(0.07490960547623698)\*T2(x21)+(-0.008366908062411074)\*T3(x21)+(-0.0244545970371083)\*T4(x21)+(0.029476790626772158)\*T5(x21)+)+1.0000000000011307\*((0.02892483158060538)\*T0(x20)+(0.17130673961416107)\*T1(x20)+(0.11168447711848535)\*T2(x20)+(0.05544000156556074)\*T3(x20)+(0.004823103259512977)\*T4(x20)+(0.004136631279998428)\*T5(x20)+(0.02892483158060538)\*T0(x21)+(-0.15239490728489724)\*T1(x21)+(0.07490960547623698)\*T2(x21)+(-0.008366908062411074)\*T3(x21)+(-0.0244545970371083)\*T4(x21)+(0.029476790626772158)\*T5(x21)+)+)1.000000000000715\*(0.9999999999972914\*((0.02892483158060538)\*T0(x30)+(-0.20380829050398114)\*T1(x30)+(0.02959154462714906)\*T2(x30)+(-0.10999304628147029)\*T3(x30)+(-0.027316395676368014)\*T4(x30)+(0.02892483158060538)\*T0(x31)+(0.3088816128868494)\*T1(x31)+(-0.25678527046718597)\*T2(x31)+(-0.006063355489488809)\*T3(x31)+(0.0291435460700444)\*T4(x31)+(0.02892483158060538)\*T0(x32)+(0.04406037494560062)\*T1(x32)+(0.11629422919889441)\*T2(x32)+(-0.027637647599549054)\*T3(x32)+(3.977251290206539E-4)\*T4(x32))+1.0000000000018672\*((0.02892483158060538)\*T0(x30)+(-0.20380829050398114)\*T1(x30)+(0.02959154462714906)\*T2(x30)+(-0.10999304628147029)\*T3(x30)+(-0.027316395676368014)\*T4(x30)+(0.02892483158060538)\*T0(x31)+(0.3088816128868494)\*T1(x31)+(-0.25678527046718597)\*T2(x31)+(-0.006063355489488809)\*T3(x31)+(0.0291435460700444)\*T4(x31)+(0.02892483158060538)\*T0(x32)+(0.04406037494560062)\*T1(x32)+(0.11629422919889441)\*T2(x32)+(-0.027637647599549054)\*T3(x32)+(3.977251290206539E-4)\*T4(x32))+1.0000000000014242\*((0.02892483158060538)\*T0(x30)+(-0.20380829050398114)\*T1(x30)+(0.02959154462714906)\*T2(x30)+(-0.10999304628147029)\*T3(x30)+(-0.027316395676368014)\*T4(x30)+(0.02892483158060538)\*T0(x31)+(0.3088816128868494)\*T1(x31)+(-0.25678527046718597)\*T2(x31)+(-0.006063355489488809)\*T3(x31)+(0.0291435460700444)\*T4(x31)+(0.02892483158060538)\*T0(x32)+(0.04406037494560062)\*T1(x32)+(0.11629422919889441)\*T2(x32)+(-0.027637647599549054)\*T3(x32)+(3.977251290206539E-4)\*T4(x32)))

Получившаяся в результате восстановленная функция

8.475+19874.960000000003\*(1.000000000000715\*(1.0000000000006353\*((0.02892483158060538)\*T0(x10)+(-0.2459239953832973)\*T1(x10)+(-0.15879375000572812)\*T2(x10)+(0.07091215649756324)\*T3(x10)+(0.047324770161718724)\*T4(x10)+(0.02892483158060538)\*T0(x11)+(-0.14257903687734902)\*T1(x11)+(0.06850505389057737)\*T2(x11)+(0.07280965194855385)\*T3(x11)+(-0.04648313343908823)\*T4(x11)+)+0.9999999999968111\*((0.02892483158060538)\*T0(x10)+(-0.2459239953832973)\*T1(x10)+(-0.15879375000572812)\*T2(x10)+(0.07091215649756324)\*T3(x10)+(0.047324770161718724)\*T4(x10)+(0.02892483158060538)\*T0(x11)+(-0.14257903687734902)\*T1(x11)+(0.06850505389057737)\*T2(x11)+(0.07280965194855385)\*T3(x11)+(-0.04648313343908823)\*T4(x11)+)+)1.000000000000715\*(0.9999999999918349\*((0.02892483158060538)\*T0(x20)+(0.17130673961416107)\*T1(x20)+(0.11168447711848535)\*T2(x20)+(0.05544000156556074)\*T3(x20)+(0.004823103259512977)\*T4(x20)+(0.004136631279998428)\*T5(x20)+(0.02892483158060538)\*T0(x21)+(-0.15239490728489724)\*T1(x21)+(0.07490960547623698)\*T2(x21)+(-0.008366908062411074)\*T3(x21)+(-0.0244545970371083)\*T4(x21)+(0.029476790626772158)\*T5(x21)+)+1.0000000000011307\*((0.02892483158060538)\*T0(x20)+(0.17130673961416107)\*T1(x20)+(0.11168447711848535)\*T2(x20)+(0.05544000156556074)\*T3(x20)+(0.004823103259512977)\*T4(x20)+(0.004136631279998428)\*T5(x20)+(0.02892483158060538)\*T0(x21)+(-0.15239490728489724)\*T1(x21)+(0.07490960547623698)\*T2(x21)+(-0.008366908062411074)\*T3(x21)+(-0.0244545970371083)\*T4(x21)+(0.029476790626772158)\*T5(x21)+)+)1.000000000000715\*(0.9999999999972914\*((0.02892483158060538)\*T0(x30)+(-0.20380829050398114)\*T1(x30)+(0.02959154462714906)\*T2(x30)+(-0.10999304628147029)\*T3(x30)+(-0.027316395676368014)\*T4(x30)+(0.02892483158060538)\*T0(x31)+(0.3088816128868494)\*T1(x31)+(-0.25678527046718597)\*T2(x31)+(-0.006063355489488809)\*T3(x31)+(0.0291435460700444)\*T4(x31)+(0.02892483158060538)\*T0(x32)+(0.04406037494560062)\*T1(x32)+(0.11629422919889441)\*T2(x32)+(-0.027637647599549054)\*T3(x32)+(3.977251290206539E-4)\*T4(x32))+1.0000000000018672\*((0.02892483158060538)\*T0(x30)+(-0.20380829050398114)\*T1(x30)+(0.02959154462714906)\*T2(x30)+(-0.10999304628147029)\*T3(x30)+(-0.027316395676368014)\*T4(x30)+(0.02892483158060538)\*T0(x31)+(0.3088816128868494)\*T1(x31)+(-0.25678527046718597)\*T2(x31)+(-0.006063355489488809)\*T3(x31)+(0.0291435460700444)\*T4(x31)+(0.02892483158060538)\*T0(x32)+(0.04406037494560062)\*T1(x32)+(0.11629422919889441)\*T2(x32)+(-0.027637647599549054)\*T3(x32)+(3.977251290206539E-4)\*T4(x32))+1.0000000000014242\*((0.02892483158060538)\*T0(x30)+(-0.20380829050398114)\*T1(x30)+(0.02959154462714906)\*T2(x30)+(-0.10999304628147029)\*T3(x30)+(-0.027316395676368014)\*T4(x30)+(0.02892483158060538)\*T0(x31)+(0.3088816128868494)\*T1(x31)+(-0.25678527046718597)\*T2(x31)+(-0.006063355489488809)\*T3(x31)+(0.0291435460700444)\*T4(x31)+(0.02892483158060538)\*T0(x32)+(0.04406037494560062)\*T1(x32)+(0.11629422919889441)\*T2(x32)+(-0.027637647599549054)\*T3(x32)+(3.977251290206539E-4)\*T4(x32))))

Полиномиальное представление

-3,273\*X11^1 + 4,215\*X11^2 + 6,407\*X11^3 + -8,865\*X11^4 + 2,034\*X11^5 +

2,109\*X12^1 + -14,264\*X12^2 + 72,389\*X12^3 + -117,861\*X12^4 + 56,953\*X12^5 +

0,518\*X21^1 + -4,083\*X21^2 + 12,365\*X21^3 + -15,325\*X21^4 + 6,390\*X21^5 +

-3,096\*X22^1 + 41,178\*X22^2 + -124,167\*X22^3 + 142,980\*X22^4 + -56,481\*X22^5 +

-8,228\*X31^1 + 50,009\*X31^2 + -125,380\*X31^3 + 128,380\*X31^4 + -44,894\*X31^5 +

15,081\*X32^1 + -89,762\*X32^2 + 190,963\*X32^3 + -167,190\*X32^4 + 51,315\*X32^5 +

11,634\*X33^1 + -62,632\*X33^2 + 143,958\*X33^3 + -143,395\*X33^4 + 51,106\*X33^5 +

+ -1,066

Ошибка восстановленой функции

738.5076873085679

**Y3**

Полученные коэффциенты при решении задачи

Первый уровень (lambda1)

0,1185 0,3436 0,1686 -0,1822 -0,1009 -0,0041

0,1185 0,4931 -0,1314 -0,1073 0,0766 0,0589

Первый уровень (Lambda2)

0,1185 0,1036 0,1425 0,0612 -0,0295 -0,0526 -0,0621

0,1185 0,4433 -0,2044 0,0177 0,0203 -0,0314 0,0815

Первый уровень (Lambda3)

0,1185 0,3806 -0,0493 0,1651 0,0155 -0,0078

0,1185 0,0464 0,4935 -0,0038 0,0449 -0,1092

0,1185 0,2969 -0,1537 -0,0266 -0,0702 0,1179

Второй уровень (а1)

1,0000 1,0000

Второй уровень (а2)

1,0000 1,0000

Второй уровень (а3)

1,0000 1,0000 1,0000

Третий уровень (С)

1,0000 1,0000 1,0000

psi функции

(0.11854026133867578)\*T0(x10)+(0.3436110512857454)\*T1(x10)+(0.16857829351946466)\*T2(x10)+(-0.18218938427558018)\*T3(x10)+(-0.10088852324901702)\*T4(x10)+(0.11854026133867578)\*T0(x11)+(0.4930503697763787)\*T1(x11)+(-0.13136005051633362)\*T2(x11)+(-0.1072958721718676)\*T3(x11)+(0.07655231269337247)\*T4(x11)+(0.11854026133867578)\*T0(x20)+(0.10356880415488005)\*T1(x20)+(0.1425131536937407)\*T2(x20)+(0.06121999603486069)\*T3(x20)+(-0.029480548056939768)\*T4(x20)+(-0.05255672686278023)\*T5(x20)+(0.11854026133867578)\*T0(x21)+(0.4433244565289512)\*T1(x21)+(-0.20443143386894638)\*T2(x21)+(0.017721757732584494)\*T3(x21)+(0.020256931976982112)\*T4(x21)+(-0.03144698118842064)\*T5(x21)+(0.11854026133867578)\*T0(x30)+(0.38055409950871494)\*T1(x30)+(-0.0492718726297519)\*T2(x30)+(0.16514236115800648)\*T3(x30)+(0.015460887759130385)\*T4(x30)+(0.11854026133867578)\*T0(x31)+(0.0463866930397353)\*T1(x31)+(0.49349686828343253)\*T2(x31)+(-0.003795193931055513)\*T3(x31)+(0.044936577849800034)\*T4(x31)+(0.11854026133867578)\*T0(x32)+(0.29692441368879996)\*T1(x32)+(-0.15370746149835976)\*T2(x32)+(-0.02657735534147176)\*T3(x32)+(-0.07017225603147079)\*T4(x32)

Второй уровень функции

Ф1=(0.9999999999995384)\*psi1(x10)+(1.0000000000013956)\*psi1(x11)+

Ф2=(0.999999999995943)\*psi2(x20)+(1.0000000000008742)\*psi2(x21)+

Ф3=(1.0000000000011313)\*psi3(x30)+(1.0000000000011826)\*psi3(x31)+(0.9999999999999903)\*psi3(x32)

Третий уровень функции

1.0000000000000044\*Ф(x1)+0.999999999999996\*Ф(x2)+0.9999999999999978\*Ф(x3)

Получившаяся в результате функция

1.0000000000000044\*(0.9999999999995384\*((0.11854026133867578)\*T0(x10)+(0.3436110512857454)\*T1(x10)+(0.16857829351946466)\*T2(x10)+(-0.18218938427558018)\*T3(x10)+(-0.10088852324901702)\*T4(x10)+(0.11854026133867578)\*T0(x11)+(0.4930503697763787)\*T1(x11)+(-0.13136005051633362)\*T2(x11)+(-0.1072958721718676)\*T3(x11)+(0.07655231269337247)\*T4(x11)+)+1.0000000000013956\*((0.11854026133867578)\*T0(x10)+(0.3436110512857454)\*T1(x10)+(0.16857829351946466)\*T2(x10)+(-0.18218938427558018)\*T3(x10)+(-0.10088852324901702)\*T4(x10)+(0.11854026133867578)\*T0(x11)+(0.4930503697763787)\*T1(x11)+(-0.13136005051633362)\*T2(x11)+(-0.1072958721718676)\*T3(x11)+(0.07655231269337247)\*T4(x11)+)+)1.0000000000000044\*(0.999999999995943\*((0.11854026133867578)\*T0(x20)+(0.10356880415488005)\*T1(x20)+(0.1425131536937407)\*T2(x20)+(0.06121999603486069)\*T3(x20)+(-0.029480548056939768)\*T4(x20)+(-0.05255672686278023)\*T5(x20)+(0.11854026133867578)\*T0(x21)+(0.4433244565289512)\*T1(x21)+(-0.20443143386894638)\*T2(x21)+(0.017721757732584494)\*T3(x21)+(0.020256931976982112)\*T4(x21)+(-0.03144698118842064)\*T5(x21)+)+1.0000000000008742\*((0.11854026133867578)\*T0(x20)+(0.10356880415488005)\*T1(x20)+(0.1425131536937407)\*T2(x20)+(0.06121999603486069)\*T3(x20)+(-0.029480548056939768)\*T4(x20)+(-0.05255672686278023)\*T5(x20)+(0.11854026133867578)\*T0(x21)+(0.4433244565289512)\*T1(x21)+(-0.20443143386894638)\*T2(x21)+(0.017721757732584494)\*T3(x21)+(0.020256931976982112)\*T4(x21)+(-0.03144698118842064)\*T5(x21)+)+)1.0000000000000044\*(1.0000000000011313\*((0.11854026133867578)\*T0(x30)+(0.38055409950871494)\*T1(x30)+(-0.0492718726297519)\*T2(x30)+(0.16514236115800648)\*T3(x30)+(0.015460887759130385)\*T4(x30)+(0.11854026133867578)\*T0(x31)+(0.0463866930397353)\*T1(x31)+(0.49349686828343253)\*T2(x31)+(-0.003795193931055513)\*T3(x31)+(0.044936577849800034)\*T4(x31)+(0.11854026133867578)\*T0(x32)+(0.29692441368879996)\*T1(x32)+(-0.15370746149835976)\*T2(x32)+(-0.02657735534147176)\*T3(x32)+(-0.07017225603147079)\*T4(x32))+1.0000000000011826\*((0.11854026133867578)\*T0(x30)+(0.38055409950871494)\*T1(x30)+(-0.0492718726297519)\*T2(x30)+(0.16514236115800648)\*T3(x30)+(0.015460887759130385)\*T4(x30)+(0.11854026133867578)\*T0(x31)+(0.0463866930397353)\*T1(x31)+(0.49349686828343253)\*T2(x31)+(-0.003795193931055513)\*T3(x31)+(0.044936577849800034)\*T4(x31)+(0.11854026133867578)\*T0(x32)+(0.29692441368879996)\*T1(x32)+(-0.15370746149835976)\*T2(x32)+(-0.02657735534147176)\*T3(x32)+(-0.07017225603147079)\*T4(x32))+0.9999999999999903\*((0.11854026133867578)\*T0(x30)+(0.38055409950871494)\*T1(x30)+(-0.0492718726297519)\*T2(x30)+(0.16514236115800648)\*T3(x30)+(0.015460887759130385)\*T4(x30)+(0.11854026133867578)\*T0(x31)+(0.0463866930397353)\*T1(x31)+(0.49349686828343253)\*T2(x31)+(-0.003795193931055513)\*T3(x31)+(0.044936577849800034)\*T4(x31)+(0.11854026133867578)\*T0(x32)+(0.29692441368879996)\*T1(x32)+(-0.15370746149835976)\*T2(x32)+(-0.02657735534147176)\*T3(x32)+(-0.07017225603147079)\*T4(x32)))

Получившаяся в результате восстановленная функция

78.793+1836.839\*(1.0000000000000044\*(0.9999999999995384\*((0.11854026133867578)\*T0(x10)+(0.3436110512857454)\*T1(x10)+(0.16857829351946466)\*T2(x10)+(-0.18218938427558018)\*T3(x10)+(-0.10088852324901702)\*T4(x10)+(0.11854026133867578)\*T0(x11)+(0.4930503697763787)\*T1(x11)+(-0.13136005051633362)\*T2(x11)+(-0.1072958721718676)\*T3(x11)+(0.07655231269337247)\*T4(x11)+)+1.0000000000013956\*((0.11854026133867578)\*T0(x10)+(0.3436110512857454)\*T1(x10)+(0.16857829351946466)\*T2(x10)+(-0.18218938427558018)\*T3(x10)+(-0.10088852324901702)\*T4(x10)+(0.11854026133867578)\*T0(x11)+(0.4930503697763787)\*T1(x11)+(-0.13136005051633362)\*T2(x11)+(-0.1072958721718676)\*T3(x11)+(0.07655231269337247)\*T4(x11)+)+)1.0000000000000044\*(0.999999999995943\*((0.11854026133867578)\*T0(x20)+(0.10356880415488005)\*T1(x20)+(0.1425131536937407)\*T2(x20)+(0.06121999603486069)\*T3(x20)+(-0.029480548056939768)\*T4(x20)+(-0.05255672686278023)\*T5(x20)+(0.11854026133867578)\*T0(x21)+(0.4433244565289512)\*T1(x21)+(-0.20443143386894638)\*T2(x21)+(0.017721757732584494)\*T3(x21)+(0.020256931976982112)\*T4(x21)+(-0.03144698118842064)\*T5(x21)+)+1.0000000000008742\*((0.11854026133867578)\*T0(x20)+(0.10356880415488005)\*T1(x20)+(0.1425131536937407)\*T2(x20)+(0.06121999603486069)\*T3(x20)+(-0.029480548056939768)\*T4(x20)+(-0.05255672686278023)\*T5(x20)+(0.11854026133867578)\*T0(x21)+(0.4433244565289512)\*T1(x21)+(-0.20443143386894638)\*T2(x21)+(0.017721757732584494)\*T3(x21)+(0.020256931976982112)\*T4(x21)+(-0.03144698118842064)\*T5(x21)+)+)1.0000000000000044\*(1.0000000000011313\*((0.11854026133867578)\*T0(x30)+(0.38055409950871494)\*T1(x30)+(-0.0492718726297519)\*T2(x30)+(0.16514236115800648)\*T3(x30)+(0.015460887759130385)\*T4(x30)+(0.11854026133867578)\*T0(x31)+(0.0463866930397353)\*T1(x31)+(0.49349686828343253)\*T2(x31)+(-0.003795193931055513)\*T3(x31)+(0.044936577849800034)\*T4(x31)+(0.11854026133867578)\*T0(x32)+(0.29692441368879996)\*T1(x32)+(-0.15370746149835976)\*T2(x32)+(-0.02657735534147176)\*T3(x32)+(-0.07017225603147079)\*T4(x32))+1.0000000000011826\*((0.11854026133867578)\*T0(x30)+(0.38055409950871494)\*T1(x30)+(-0.0492718726297519)\*T2(x30)+(0.16514236115800648)\*T3(x30)+(0.015460887759130385)\*T4(x30)+(0.11854026133867578)\*T0(x31)+(0.0463866930397353)\*T1(x31)+(0.49349686828343253)\*T2(x31)+(-0.003795193931055513)\*T3(x31)+(0.044936577849800034)\*T4(x31)+(0.11854026133867578)\*T0(x32)+(0.29692441368879996)\*T1(x32)+(-0.15370746149835976)\*T2(x32)+(-0.02657735534147176)\*T3(x32)+(-0.07017225603147079)\*T4(x32))+0.9999999999999903\*((0.11854026133867578)\*T0(x30)+(0.38055409950871494)\*T1(x30)+(-0.0492718726297519)\*T2(x30)+(0.16514236115800648)\*T3(x30)+(0.015460887759130385)\*T4(x30)+(0.11854026133867578)\*T0(x31)+(0.0463866930397353)\*T1(x31)+(0.49349686828343253)\*T2(x31)+(-0.003795193931055513)\*T3(x31)+(0.044936577849800034)\*T4(x31)+(0.11854026133867578)\*T0(x32)+(0.29692441368879996)\*T1(x32)+(-0.15370746149835976)\*T2(x32)+(-0.02657735534147176)\*T3(x32)+(-0.07017225603147079)\*T4(x32))))

Полиномиальное представление

-5,377\*X11^1 + 6,925\*X11^2 + 10,527\*X11^3 + -14,565\*X11^4 + 3,342\*X11^5 +

2,589\*X12^1 + -17,511\*X12^2 + 88,867\*X12^3 + -144,690\*X12^4 + 69,917\*X12^5 +

3,886\*X21^1 + -30,641\*X21^2 + 92,786\*X21^3 + -114,996\*X21^4 + 47,949\*X21^5 +

-4,492\*X22^1 + 59,746\*X22^2 + -180,158\*X22^3 + 207,454\*X22^4 + -81,950\*X22^5 +

-10,134\*X31^1 + 61,595\*X31^2 + -154,427\*X31^3 + 158,123\*X31^4 + -55,295\*X31^5 +

20,824\*X32^1 + -123,944\*X32^2 + 263,684\*X32^3 + -230,857\*X32^4 + 70,857\*X32^5 +

16,233\*X33^1 + -87,392\*X33^2 + 200,868\*X33^3 + -200,083\*X33^4 + 71,310\*X33^5 +

+ -1,404

Ошибка восстановленой функции

134.10742695631586

**Y4**

Полученные коэффциенты при решении задачи

Первый уровень (lambda1)

0,0762 0,6308 -0,0633 -0,0997 -0,0065 0,0035

0,0762 -0,3862 -0,1693 0,1350 0,0612 -0,0096

Первый уровень (Lambda2)

0,0762 -0,2718 0,1993 0,0662 0,0009 -0,0521 -0,0033

0,0762 0,6056 -0,0709 0,0539 -0,0439 0,0247 -0,0654

Первый уровень (Lambda3)

0,0762 0,4578 0,2013 -0,1000 -0,0776 -0,0151

0,0762 0,4826 0,3951 -0,1682 0,1687 0,0625

0,0762 0,1531 -0,0046 -0,0600 -0,0452 -0,0244

Второй уровень (а1)

1,0000 1,0000

Второй уровень (а2)

1,0000 1,0000

Второй уровень (а3)

1,0000 1,0000 1,0000

Третий уровень (С)

1,0000 1,0000 1,0000

psi функции

(0.07618977519462368)\*T0(x10)+(0.6307688567272897)\*T1(x10)+(-0.06333331312014939)\*T2(x10)+(-0.09968483551998555)\*T3(x10)+(-0.0065046472993230955)\*T4(x10)+(0.07618977519462368)\*T0(x11)+(-0.38623630890605676)\*T1(x11)+(-0.16928790295723478)\*T2(x11)+(0.1349731602350103)\*T3(x11)+(0.06123009458176295)\*T4(x11)+(0.07618977519462368)\*T0(x20)+(-0.27183123706933804)\*T1(x20)+(0.19928806840895613)\*T2(x20)+(0.06618927861505805)\*T3(x20)+(9.354792646060765E-4)\*T4(x20)+(-0.0520932955311805)\*T5(x20)+(0.07618977519462368)\*T0(x21)+(0.6055522960209029)\*T1(x21)+(-0.0708599670482009)\*T2(x21)+(0.05391400055540678)\*T3(x21)+(-0.043903605395200246)\*T4(x21)+(0.024724209175919342)\*T5(x21)+(0.07618977519462368)\*T0(x30)+(0.4577860792741362)\*T1(x30)+(0.20133004947944808)\*T2(x30)+(-0.10001990913201023)\*T3(x30)+(-0.07760325900612691)\*T4(x30)+(0.07618977519462368)\*T0(x31)+(0.48260224924786954)\*T1(x31)+(0.39507436522411)\*T2(x31)+(-0.16816030536694085)\*T3(x31)+(0.1686610104033317)\*T4(x31)+(0.07618977519462368)\*T0(x32)+(0.15308798247902625)\*T1(x32)+(-0.004599890971984626)\*T2(x32)+(-0.059982179373570005)\*T3(x32)+(-0.04518074814312501)\*T4(x32)

Второй уровень функции

Ф1=(1.0000000000018872)\*psi1(x10)+(0.9999999999999966)\*psi1(x11)+

Ф2=(1.0000000000009093)\*psi2(x20)+(1.0000000000005944)\*psi2(x21)+

Ф3=(0.9999999999979274)\*psi3(x30)+(1.000000000001775)\*psi3(x31)+(0.9999999999948079)\*psi3(x32)

Третий уровень функции

0.999999999999999\*Ф(x1)+1.0000000000000568\*Ф(x2)+0.9999999999999899\*Ф(x3)

Получившаяся в результате функция

0.999999999999999\*(1.0000000000018872\*((0.07618977519462368)\*T0(x10)+(0.6307688567272897)\*T1(x10)+(-0.06333331312014939)\*T2(x10)+(-0.09968483551998555)\*T3(x10)+(-0.0065046472993230955)\*T4(x10)+(0.07618977519462368)\*T0(x11)+(-0.38623630890605676)\*T1(x11)+(-0.16928790295723478)\*T2(x11)+(0.1349731602350103)\*T3(x11)+(0.06123009458176295)\*T4(x11)+)+0.9999999999999966\*((0.07618977519462368)\*T0(x10)+(0.6307688567272897)\*T1(x10)+(-0.06333331312014939)\*T2(x10)+(-0.09968483551998555)\*T3(x10)+(-0.0065046472993230955)\*T4(x10)+(0.07618977519462368)\*T0(x11)+(-0.38623630890605676)\*T1(x11)+(-0.16928790295723478)\*T2(x11)+(0.1349731602350103)\*T3(x11)+(0.06123009458176295)\*T4(x11)+)+)0.999999999999999\*(1.0000000000009093\*((0.07618977519462368)\*T0(x20)+(-0.27183123706933804)\*T1(x20)+(0.19928806840895613)\*T2(x20)+(0.06618927861505805)\*T3(x20)+(9.354792646060765E-4)\*T4(x20)+(-0.0520932955311805)\*T5(x20)+(0.07618977519462368)\*T0(x21)+(0.6055522960209029)\*T1(x21)+(-0.0708599670482009)\*T2(x21)+(0.05391400055540678)\*T3(x21)+(-0.043903605395200246)\*T4(x21)+(0.024724209175919342)\*T5(x21)+)+1.0000000000005944\*((0.07618977519462368)\*T0(x20)+(-0.27183123706933804)\*T1(x20)+(0.19928806840895613)\*T2(x20)+(0.06618927861505805)\*T3(x20)+(9.354792646060765E-4)\*T4(x20)+(-0.0520932955311805)\*T5(x20)+(0.07618977519462368)\*T0(x21)+(0.6055522960209029)\*T1(x21)+(-0.0708599670482009)\*T2(x21)+(0.05391400055540678)\*T3(x21)+(-0.043903605395200246)\*T4(x21)+(0.024724209175919342)\*T5(x21)+)+)0.999999999999999\*(0.9999999999979274\*((0.07618977519462368)\*T0(x30)+(0.4577860792741362)\*T1(x30)+(0.20133004947944808)\*T2(x30)+(-0.10001990913201023)\*T3(x30)+(-0.07760325900612691)\*T4(x30)+(0.07618977519462368)\*T0(x31)+(0.48260224924786954)\*T1(x31)+(0.39507436522411)\*T2(x31)+(-0.16816030536694085)\*T3(x31)+(0.1686610104033317)\*T4(x31)+(0.07618977519462368)\*T0(x32)+(0.15308798247902625)\*T1(x32)+(-0.004599890971984626)\*T2(x32)+(-0.059982179373570005)\*T3(x32)+(-0.04518074814312501)\*T4(x32))+1.000000000001775\*((0.07618977519462368)\*T0(x30)+(0.4577860792741362)\*T1(x30)+(0.20133004947944808)\*T2(x30)+(-0.10001990913201023)\*T3(x30)+(-0.07760325900612691)\*T4(x30)+(0.07618977519462368)\*T0(x31)+(0.48260224924786954)\*T1(x31)+(0.39507436522411)\*T2(x31)+(-0.16816030536694085)\*T3(x31)+(0.1686610104033317)\*T4(x31)+(0.07618977519462368)\*T0(x32)+(0.15308798247902625)\*T1(x32)+(-0.004599890971984626)\*T2(x32)+(-0.059982179373570005)\*T3(x32)+(-0.04518074814312501)\*T4(x32))+0.9999999999948079\*((0.07618977519462368)\*T0(x30)+(0.4577860792741362)\*T1(x30)+(0.20133004947944808)\*T2(x30)+(-0.10001990913201023)\*T3(x30)+(-0.07760325900612691)\*T4(x30)+(0.07618977519462368)\*T0(x31)+(0.48260224924786954)\*T1(x31)+(0.39507436522411)\*T2(x31)+(-0.16816030536694085)\*T3(x31)+(0.1686610104033317)\*T4(x31)+(0.07618977519462368)\*T0(x32)+(0.15308798247902625)\*T1(x32)+(-0.004599890971984626)\*T2(x32)+(-0.059982179373570005)\*T3(x32)+(-0.04518074814312501)\*T4(x32)))

Получившаяся в результате восстановленная функция

61.953+1854.171\*(0.999999999999999\*(1.0000000000018872\*((0.07618977519462368)\*T0(x10)+(0.6307688567272897)\*T1(x10)+(-0.06333331312014939)\*T2(x10)+(-0.09968483551998555)\*T3(x10)+(-0.0065046472993230955)\*T4(x10)+(0.07618977519462368)\*T0(x11)+(-0.38623630890605676)\*T1(x11)+(-0.16928790295723478)\*T2(x11)+(0.1349731602350103)\*T3(x11)+(0.06123009458176295)\*T4(x11)+)+0.9999999999999966\*((0.07618977519462368)\*T0(x10)+(0.6307688567272897)\*T1(x10)+(-0.06333331312014939)\*T2(x10)+(-0.09968483551998555)\*T3(x10)+(-0.0065046472993230955)\*T4(x10)+(0.07618977519462368)\*T0(x11)+(-0.38623630890605676)\*T1(x11)+(-0.16928790295723478)\*T2(x11)+(0.1349731602350103)\*T3(x11)+(0.06123009458176295)\*T4(x11)+)+)0.999999999999999\*(1.0000000000009093\*((0.07618977519462368)\*T0(x20)+(-0.27183123706933804)\*T1(x20)+(0.19928806840895613)\*T2(x20)+(0.06618927861505805)\*T3(x20)+(9.354792646060765E-4)\*T4(x20)+(-0.0520932955311805)\*T5(x20)+(0.07618977519462368)\*T0(x21)+(0.6055522960209029)\*T1(x21)+(-0.0708599670482009)\*T2(x21)+(0.05391400055540678)\*T3(x21)+(-0.043903605395200246)\*T4(x21)+(0.024724209175919342)\*T5(x21)+)+1.0000000000005944\*((0.07618977519462368)\*T0(x20)+(-0.27183123706933804)\*T1(x20)+(0.19928806840895613)\*T2(x20)+(0.06618927861505805)\*T3(x20)+(9.354792646060765E-4)\*T4(x20)+(-0.0520932955311805)\*T5(x20)+(0.07618977519462368)\*T0(x21)+(0.6055522960209029)\*T1(x21)+(-0.0708599670482009)\*T2(x21)+(0.05391400055540678)\*T3(x21)+(-0.043903605395200246)\*T4(x21)+(0.024724209175919342)\*T5(x21)+)+)0.999999999999999\*(0.9999999999979274\*((0.07618977519462368)\*T0(x30)+(0.4577860792741362)\*T1(x30)+(0.20133004947944808)\*T2(x30)+(-0.10001990913201023)\*T3(x30)+(-0.07760325900612691)\*T4(x30)+(0.07618977519462368)\*T0(x31)+(0.48260224924786954)\*T1(x31)+(0.39507436522411)\*T2(x31)+(-0.16816030536694085)\*T3(x31)+(0.1686610104033317)\*T4(x31)+(0.07618977519462368)\*T0(x32)+(0.15308798247902625)\*T1(x32)+(-0.004599890971984626)\*T2(x32)+(-0.059982179373570005)\*T3(x32)+(-0.04518074814312501)\*T4(x32))+1.000000000001775\*((0.07618977519462368)\*T0(x30)+(0.4577860792741362)\*T1(x30)+(0.20133004947944808)\*T2(x30)+(-0.10001990913201023)\*T3(x30)+(-0.07760325900612691)\*T4(x30)+(0.07618977519462368)\*T0(x31)+(0.48260224924786954)\*T1(x31)+(0.39507436522411)\*T2(x31)+(-0.16816030536694085)\*T3(x31)+(0.1686610104033317)\*T4(x31)+(0.07618977519462368)\*T0(x32)+(0.15308798247902625)\*T1(x32)+(-0.004599890971984626)\*T2(x32)+(-0.059982179373570005)\*T3(x32)+(-0.04518074814312501)\*T4(x32))+0.9999999999948079\*((0.07618977519462368)\*T0(x30)+(0.4577860792741362)\*T1(x30)+(0.20133004947944808)\*T2(x30)+(-0.10001990913201023)\*T3(x30)+(-0.07760325900612691)\*T4(x30)+(0.07618977519462368)\*T0(x31)+(0.48260224924786954)\*T1(x31)+(0.39507436522411)\*T2(x31)+(-0.16816030536694085)\*T3(x31)+(0.1686610104033317)\*T4(x31)+(0.07618977519462368)\*T0(x32)+(0.15308798247902625)\*T1(x32)+(-0.004599890971984626)\*T2(x32)+(-0.059982179373570005)\*T3(x32)+(-0.04518074814312501)\*T4(x32))))

Полиномиальное представление

-2,205\*X11^1 + 2,839\*X11^2 + 4,316\*X11^3 + -5,972\*X11^4 + 1,370\*X11^5 +

1,378\*X12^1 + -9,325\*X12^2 + 47,323\*X12^3 + -77,049\*X12^4 + 37,232\*X12^5 +

1,840\*X21^1 + -14,513\*X21^2 + 43,948\*X21^3 + -54,468\*X21^4 + 22,711\*X21^5 +

-1,803\*X22^1 + 23,981\*X22^2 + -72,312\*X22^3 + 83,269\*X22^4 + -32,893\*X22^5 +

-1,953\*X31^1 + 11,872\*X31^2 + -29,765\*X31^3 + 30,477\*X31^4 + -10,658\*X31^5 +

8,840\*X32^1 + -52,618\*X32^2 + 111,943\*X32^3 + -98,007\*X32^4 + 30,081\*X32^5 +

7,880\*X33^1 + -42,420\*X33^2 + 97,501\*X33^3 + -97,120\*X33^4 + 34,614\*X33^5 +

+ -0,788

Ошибка восстановленой функции

85.62993493566728

**Власна виборка.**

Для роботи ми обрали наступну виборку з сайту:  
<https://archive.ics.uci.edu/ml/datasets/Housing>

В перше ця виборка розглядалася у статті:

Harrison, D. and Rubinfeld, D.L.   
'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978

Ціллю дослідження була знайти залежність між цінами та популярністю житла від деяких факторів(наприклад, від забруднення повітря)

Опис данних:

Ця виборка має 13 змінних:

1. CRIM: per capita crime rate by town   
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.   
3. INDUS: proportion of non-retail business acres per town

5. NOX: nitric oxides concentration (parts per 10 million)   
6. RM: average number of rooms per dwelling   
7. AGE: proportion of owner-occupied units built prior to 1940   
8. DIS: weighted distances to five Boston employment centres   
9. RAD: index of accessibility to radial highways   
10. TAX: full-value property-tax rate per $10,000   
11. PTRATIO: pupil-teacher ratio by town   
12. B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town   
13. LSTAT: % lower status of the population   
14. MEDV: Median value of owner-occupied homes in $1000's

Розмірність вибірки – 111 зразків

Для того, щоб більш наочно продемонструвати можливості алгоритма, який запропонований у лабораторній роботі, ми розбили параметри наступним чином:

Множині змінних Х1 (показники, які можливо контролювати при проектуванні або виборі житла) ми віднесли наступні:

6. RM: average number of rooms per dwelling

Множині змінних Х2 (зовнішні фактори, які можливо контролювати, чи ті які не змінюються з часом) ми віднесли наступні:

2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.

3. INDUS: proportion of non-retail business acres per town

7. AGE: proportion of owner-occupied units built prior to 1940

8. DIS: weighted distances to five Boston employment centres

9. RAD: index of accessibility to radial highways

Множині змінних Х3 (зовнішні фактори, які неможливо контролювати або передбачити) ми віднесли наступні:

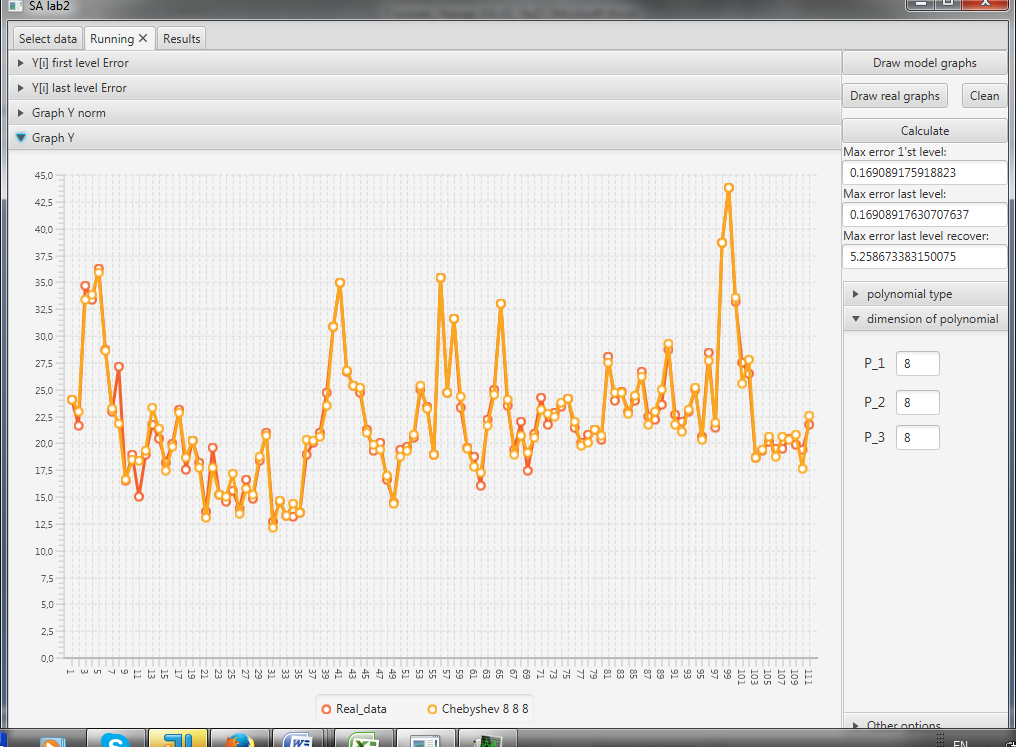
1. CRIM: per capita crime rate by town   
5. NOX: nitric oxides concentration (parts per 10 million)

10. TAX: full-value property-tax rate per $10,000   
11. PTRATIO: pupil-teacher ratio by town   
12. B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town   
13. LSTAT: % lower status of the population

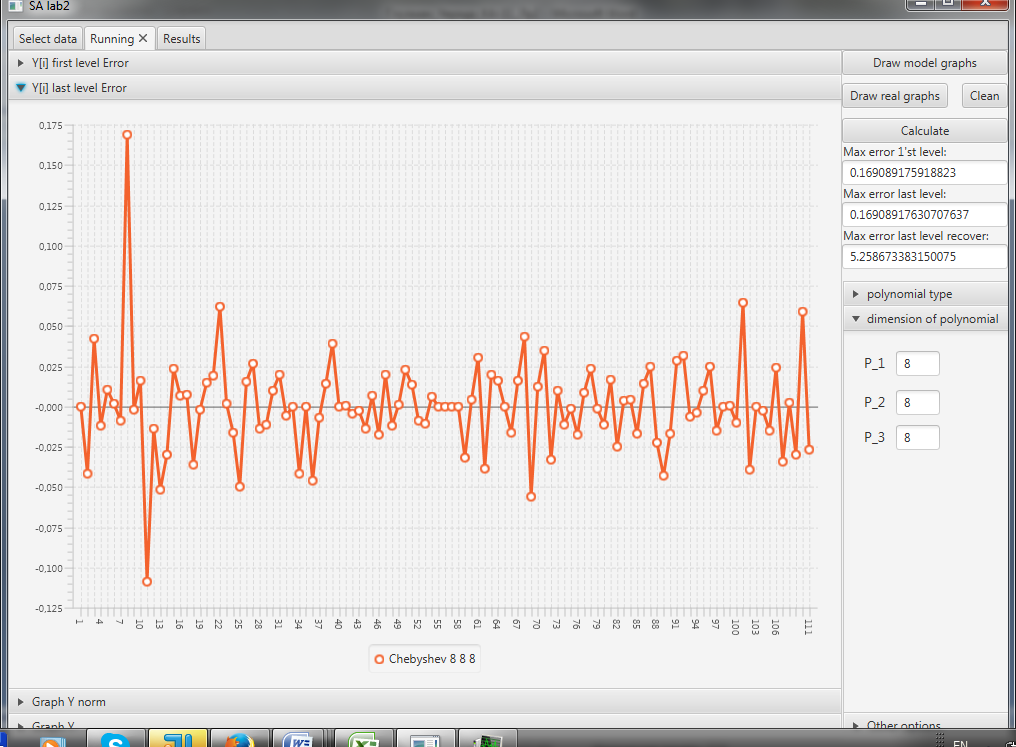
Вихідні параметри:

14. MEDV: Median value of owner-occupied homes in $1000's

Для відновлення функціональної залежності найкраще себе показав поліном 8 ступеня. Поліноми вищих степенів (до 15) давали більшу похибку. Що цікаво, порівняну похибку давали також і поліноми першого першого ступеня (відхилення складає 0.24380090321434697).  
Отже, результати для роботи з поліномами Чебишева порядку 8 для всіх змінних:



Вектор похибки:



**Висновок**

Отже, в нашій роботі була розв’язана задача пошуку функціональної залежності у вигляді узагальнених многочленів, і як критерій пошуку використовувався Чебишевський критерій наближення системи. Згідно з теоремою Вейерштрасса, зі збільшенням степені полінома, наближення моделі повинно покращуватися, але як ми виявили (на власній вибірці) – зі збільшенням від оптимального значення похибка зростатиме.

Для нашої моделі многочлен порядку 5 6 5 для наших змінних вже дає прийнятну похибку. Для власної вибірки оптимальним виявився многочлен 8 порядку для всіх змінних.

Знайдена функціональна залежність дає змогу знайти значення функції як і в середині інтервалу області значення змінних данної вибірки (мається на увазі ті, значення Xi, що не вказані у вибірці), так і дає змогу прогнозування за межі інтервалів області значень змінних у даній вибірці.

**Література по даному питанню:**

Наступні праці присвячені рішенням неузгоджених систем:

В наступних працях описані алгоритми рішення неузгоджених систем в найкращому у значенні Чебишева смислі:

L. V. Vojtíšek

Отыcкание наилучшегo в cмыcле Чебышева решения неcoвмеcтнoй cиcтемы

линейных алгебраичеcких уравнений

Aplikace matematiky, Vol. 11 (1966), No. 3, 232—237

С. И. Зуховицкий

О наилучшем в смысле П. Л. Чебыше-

ва приближении конечной системы несовместных линейных

уравнений, Матем. сб., 1953, том 33(75), номер 2, 327–342

Найсучасніших алгоритм, оснований на методі спряжених градієнтів запропонований у наступній праці:

Van der Vorst, H. A

Iterative Krylov Methods for Large Linear systems. Cambridge University Press, Cambridge. [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [0-521-81828-1](https://en.wikipedia.org/wiki/Special:BookSources/0-521-81828-1). (2003).

З аналізу та аналізу невизнеченостей:

Ronald L. Iman, Jon C. Helton

An Investigation of Uncertainty and Sensitivity Analysis Techniques for Computer Models, Risk Analysis, Volume 8, Issue 1, pages 71–90, March 1988, DOI: 10.1111/j.1539-6924.1988.tb01155.x

da Silva, R.B., Bulska, E., Godlewska-Zylkiewicz, B., Hedrich, M., Majcen, N., Magnusson, B., Marincic, S., Papadakis, I., Patriarca, M., Vassileva, E., Taylor, P.,

Analytical measurement: measurement uncertainty and statistics; [ISBN 978-92-79-23070-7](http://en.wikipedia.org/wiki/Special:BookSources/9789279230707), 2012.

Arnaut, L. R.

Measurement uncertainty in reverberation chambers - I. Sample statistics. Technical report TQE 2, 2nd. ed., National Physical Laboratory, 2008.

Наближення за допомогою поліномів Чебишева різного роду рівнянь та відновлення рівнянь:

Відновлення разривних функцій:

О. В. Жучко, Ю. П. Пытьев,

Восстановление функциональной зависимости

теоретико-возможностными методами, Ж. вычисл. матем. и матем. физ.,

2003, том 43, номер 5, 767–783

Відновлення диференційних рівнянь:

Bogdan Mihaila1 and Ioana Mihaila2

Numerical approximations using Chebyshev polynomial expansions: El-gendi’s method revisited. J. Phys. A: Math. Gen. 35 (2002) 731–746

Bogdan Mihaila

Numerical Approximations Using Chebyshev Polynomial Expansions

Також в процесі роботи над лабораторною роботою ми користувалися наступною літературою:

Використали теорію по методу спряжених напрямків.

Методичний посібник до лабораторних робіт з курсу «Методи оптимізації» А.П. Яковлева, І.Я. Спекторський. – К.: НТУУ «КПІ» ННК «ІПСА» , 2000. – 65 с.

Опис проблематики, методи відновлення залежностей.

Вапник В.Н. Восстановление зависимостей по эмпирическим данным. – М., 1979. – 448 с.

Уточнення рекурсивного алгоритму задання поліномів.

Вікіпедія : сторінки про поліноми Лежандра, Лагера, Ерміта, Чебишева.

Опис алгоритмів наближення функцій.

Никольский С. М. Приближение функций многих переменных и теоремы вложения. – М.: Наука, 1977. – 456 с.

Виведення поліномів Чебишева, Лежандра, Лагера та Ерміта.

Л.А. Люстерник и др. Математический анализ: вычисление элементарных функций.

М.Физматгиз1963.

Лістинг програми:

package sample;

import Jama.Matrix;

import java.io.IOException;

/\*\*

\* Created by Gregory on 17.10.2014.

\*/

enum Polynom {Chebyshev,Lejandr, Lagger,Hermit}

public class Calculations {

//--------- Variables---------------------

// dim = 3 array

int[] PolyOrderJ;

// for Y recovering

double MaxY;

double MinY;

double coef;

Conjugate\_gradient\_method Method;

double[] Ax;

// row number in NormalizedData

int NumberOfSamples;

Polynom type;

// Here is cofficients for a\_i

// dimension n1, n2, n3

double[]A1;

double[]A2;

double[]A3;

double[]AincludeAll;

double[] residualBigF\_System;

double residualBigF\_System\_max;

double [] ResidVecLambda;

double ResidVecLambda\_max;

double[] C;

// the third level of the hierarchy:

double[] BigF1;

double c1;

double[] BigF2;

double c2;

double[] BigF3;

double c3;

// BigF for system;

double[][] BigF\_System;

//for the second method of lambda calculation

double [][] A\_Lambda1;

double [][] A\_Lambda2;

double [][] A\_Lambda3;

// Psi vector

// dimention n1,n2,n3

double[] Psi1, Psi2, Psi3;

//PsiMatrices

double[][] PsMat1, PsMat2, PsMat3;

double[][] PsMat;

// arrays of lambda in each of the three double sum

// n1\*polyOrderJi

double[][] Lambda1, Lambda2, Lambda3;

// Matrix for lambda Linear System A\*lambda=b

double[][] A;

// arrays of polynom value

// n1\*polyOrderJi

double[][] Tp1, Tp2, Tp3;

// востановление

// Y\_model - это востановленно из посчитанныхж

// Y\_true - истинное востановление

double[] Y\_model;

double[] Y\_true;

// -----------------------------------end of variables------------------------------------

// elements of this Rezult will be the arguments of the double sum

public double[][] ElemByElemMatrixMultiplication(double[][] Mat1, double[][] Mat2) {

int RowSize = Mat1.length;

int ColumnSize = Mat1[0].length;

if (RowSize == Mat2.length && ColumnSize == Mat2[0].length) {

double[][] Result = new double[RowSize][ColumnSize];

for (int i = 0; i < RowSize; ++i) {

for (int j = 0; j < ColumnSize; ++j) {

Result[i][j] = Mat1[i][j] \* Mat2[i][j];

}

}

return Result;

} else return null;

}

// это для вычисления двойной суммы, судя подставляется матрица полученная от функции, которая выше

public double ElementSum(double[][] Mat) {

int RowSize = Mat.length;

int ColumnSize = Mat[0].length;

double sum = 0;

for (int i = 0; i < RowSize; ++i) {

for (int j = 0; j < ColumnSize; ++j) {

sum = sum + Mat[i][j];

}

}

return sum;

}

//-----!!! polynom generation

public double Chebyshev(double x, int rang) {

if (rang < 0) return 0;

if (rang == 0) return 1;

if (rang == 1) return -1 + 2 \* x;

return 2 \* (-1 + 2 \* x) \* Chebyshev(x, rang - 1) - Chebyshev(x, rang - 2);

}

public double Lejandr(double x, int rang) {

if (rang < 0) return 0;

if (rang == 0) return 1;

if (rang == 1) return x;

return (1.0/(rang))\*((2\*rang - 1)\*x\*Lejandr(x,rang-1) -

(rang-1)\*Lejandr(x,rang - 2));

}

public double Lagger(double x, int rang) {

if (rang < 0) return 0;

if (rang == 0) return 1;

if (rang == 1) return -x + 1;

return (2 \* rang - 1 - x) \* Lagger(x, rang - 1) -

(rang - 1) \* (rang - 1) \* Lagger(x, rang - 2);

}

public double Hermit(double x, int rang) {

if (rang < 1) return 0;

if (rang == 1) return 2\*x;

if (rang == 2) return 4\*x\*x - 2;

return 2\*x\*Hermit(x,rang - 1) - 2\*(rang - 1)\*Hermit(x, rang - 2);

}

public double [][] Transpose(double [][] Mat){

int RowSize = Mat.length;

int ColumnSize = Mat[0].length;

double[][] MatTrans = new double[ColumnSize][RowSize];

double temp;

for(int i = 0; i < RowSize; ++i){

for(int j = 0; j < ColumnSize; ++j){

MatTrans[j][i] = Mat[i][j];

}

}

return MatTrans;

}

// ------ вычисление матриц с помощью полиномов

double[][] PolynomMatrix(double [][] Xj,int indexQ, int PolyOrder, Polynom type) {

// Here Xj must be transposed, for example Xj has 2\*45 dimensionality

// indexQ - q1 or q2 or q3 or q0 in big formula

int N1 = Xj.length;

double[][] PolyMat = new double[N1][PolyOrder + 1];

switch (type) {

case Chebyshev:

for (int i = 0; i < N1; ++i) {

for (int j = 0; j <= PolyOrder; ++j) {

PolyMat[i][j] = Chebyshev(Xj[i][indexQ], j);

}

}

break;

case Lagger:

for (int i = 0; i < N1; ++i) {

for (int j = 0; j <= PolyOrder; ++j) {

PolyMat[i][j] = Lagger(Xj[i][indexQ], j);

}

}

break;

case Lejandr:

for (int i = 0; i < N1; ++i) {

for (int j = 0; j <= PolyOrder; ++j) {

PolyMat[i][j] = Lejandr(Xj[i][indexQ], j);

}

}

break;

case Hermit:

for (int i = 0; i < N1; ++i) {

for (int j = 0; j <= PolyOrder; ++j) {

PolyMat[i][j] = Hermit(Xj[i][indexQ], j);

}

}

break;

}

return PolyMat;

}

double[] MatrixShapeToVector(double[][] Mat){

int RowSize = Mat.length;

int ColumnSize = Mat[0].length;

int dim = Mat.length \* Mat[0].length;

double[] Vector = new double[dim];

int ind = 0;

for(int i = 0; i < RowSize; ++i){

for(int j = 0; j < ColumnSize; ++j){

Vector[ind] = Mat[i][j];

++ind;

}

}

return Vector;

}

// !!!

// return 1\*3 array

// ExDim = {Rows, Columns, Rows \* Columns}

int[] ExtractDim(double[][] Mat){

int[] ret = new int[3];

ret[0] = Mat.length;

ret[1] = Mat[0].length;

ret[2] = ret[0]\*ret[1];

return ret;

}

void MatrixForLinearEquation(double[][] X1,double[][] X2,double[][] X3){

// DimTpI is required this matrices by my initialization :)

Tp1 = PolynomMatrix(X1, 0, PolyOrderJ[0], type);

Tp2 = PolynomMatrix(X2,0, PolyOrderJ[1],type);

Tp3 = PolynomMatrix(X3,0, PolyOrderJ[2],type);

int[] DimTp1 = ExtractDim(Tp1);

int[] DimTp2 = ExtractDim(Tp2);

int[] DimTp3 = ExtractDim(Tp3);

int RowSizeA = NumberOfSamples;

int ColumnSizeA = DimTp1[2] + DimTp2[2] + DimTp3[2];

A\_Lambda1= new double [RowSizeA][DimTp1[2]];

A\_Lambda2= new double [RowSizeA][DimTp2[2]];

A\_Lambda3= new double [RowSizeA][DimTp3[2]];

// !!!

// Initialization of Matrix A

A = new double[RowSizeA][ColumnSizeA];

// first row initialization

int sum = 0;

for(int i = 0; i < RowSizeA; ++i ){

Tp1 = PolynomMatrix(X1,i, PolyOrderJ[0],type);

Tp2 = PolynomMatrix(X2,i, PolyOrderJ[1],type);

Tp3 = PolynomMatrix(X3,i, PolyOrderJ[2],type);

sum = 0;

for(int j = 0; j < DimTp1[2]; ++j){

A[i][sum] = MatrixShapeToVector(Tp1)[j];

++sum;

A\_Lambda1[i][j]=MatrixShapeToVector(Tp1)[j];

}

for(int j = 0; j < DimTp2[2]; ++j){

A[i][sum] = MatrixShapeToVector(Tp2)[j];

++sum;

A\_Lambda2[i][j]=MatrixShapeToVector(Tp2)[j];

}

for(int j = 0; j < DimTp3[2]; ++j){

A[i][sum] = MatrixShapeToVector(Tp3)[j];

++sum;

A\_Lambda3[i][j]=MatrixShapeToVector(Tp3)[j];

}

}

}

double ScalarF(double [][] X1, double[][] X2, double[][] X3, int indexQ){

// X1, X2, X3 must be transposed, for example X1 has 2\*45 dimensionality

Tp1 = PolynomMatrix(X1,indexQ, PolyOrderJ[0],type);

Tp2 = PolynomMatrix(X2,indexQ, PolyOrderJ[1],type);

Tp3 = PolynomMatrix(X3,indexQ, PolyOrderJ[2],type);

return ElementSum(ElemByElemMatrixMultiplication(Lambda1, Tp1)) +

ElementSum(ElemByElemMatrixMultiplication(Lambda2, Tp2)) +

ElementSum(ElemByElemMatrixMultiplication(Lambda3, Tp3));

}

private void LambdaMatrixCreation(double[] LambdaVec){

int RowSize = Lambda1.length;

int ColumnSize = Lambda1[0].length;

int sum =0;

for(int i = 0; i < RowSize; ++i){

for(int j = 0; j < ColumnSize; ++j){

Lambda1[i][j] = LambdaVec[sum];

++sum;

}

}

RowSize = Lambda2.length;

ColumnSize = Lambda2[0].length;

for(int i = 0; i < RowSize; ++i){

for(int j = 0; j < ColumnSize; ++j){

Lambda2[i][j] = LambdaVec[sum];

++sum;

}

}

RowSize = Lambda3.length;

ColumnSize = Lambda3[0].length;

for(int i = 0; i < RowSize; ++i){

for(int j = 0; j < ColumnSize; ++j){

Lambda3[i][j] = LambdaVec[sum];

++sum;

}

}

}

private double[] CalculateF\_iUsing\_ScalarF\_Function(double [][] X1, double[][] X2, double[][] X3){

double[] yCalc = new double[NumberOfSamples];

for(int i = 0; i < NumberOfSamples; ++i){

yCalc[i] = ScalarF(X1, X2, X3,i);

}

return yCalc;

}

private double[] PsiInitialization(double[][] X1, double[][] X2, double[][] X3,

double[][] Lambda1, double[][] Lambda2, double [][] Lambda3,

int indexQ,int NumberOfPsi){

Psi1 = new double[X1.length];

Psi2 = new double[X2.length];

Psi3 = new double[X3.length];

switch(NumberOfPsi) {

case 0:

Tp1 = PolynomMatrix(X1, indexQ, PolyOrderJ[0], type);

for(int i = 0; i < Psi1.length;++i){

Psi1[i] = PsiElementCalc(Lambda1[i],Tp1[i]);

}

return Psi1;

case 1:

Tp2 = PolynomMatrix(X2, indexQ, PolyOrderJ[1], type);

for(int i = 0; i < Psi2.length;++i){

Psi2[i] = PsiElementCalc(Lambda2[i],Tp2[i]);

}

return Psi2;

case 2:

Tp3 = PolynomMatrix(X3, indexQ, PolyOrderJ[2], type);

for(int i = 0; i < Psi3.length;++i){

Psi3[i] = PsiElementCalc(Lambda3[i],Tp3[i]);

}

return Psi3;

}

return null;

}

public double[][] PsiMatrixCreation(double[][] X1,double[][] X2,double[][] X3,

double[][] Lambda1, double[][] Lambda2, double [][] Lambda3,

int NumberOfPsiMatrix){

// we must solve three different systems on the second level

// it return one of three matrix - defined by NumberOfPsiMatrix

double[][] PsiMatrix;

switch(NumberOfPsiMatrix){

case 0:

PsiMatrix = new double[NumberOfSamples][];

for (int i = 0; i < NumberOfSamples; ++i){

PsiMatrix[i] = PsiInitialization(X1, X2, X3, Lambda1, Lambda2, Lambda3, i, 0);

}

return PsiMatrix;

case 1:

PsiMatrix = new double[NumberOfSamples][];

for (int i=0; i < NumberOfSamples; ++i)

PsiMatrix[i] = PsiInitialization(X1,X2,X3,Lambda1,Lambda2,Lambda3,i,1);

return PsiMatrix;

case 2:

PsiMatrix = new double[NumberOfSamples][];

for(int i = 0; i < NumberOfSamples; ++i){

PsiMatrix[i] = PsiInitialization(X1, X2, X3, Lambda1, Lambda2, Lambda3, i, 2);

}

return PsiMatrix;

}

return null;

}

private double PsiElementCalc(double[] Lambda, double[] TpLambda){

// TpLambda - Урезанное Tp, которое отвечает определенному лямбда

// возвращает елемент Пси по переданным векторам Лямбда и ТпЛямбда

double sum = 0;

for(int i = 0; i < Lambda.length; ++i){

sum = sum + Lambda[i]\*TpLambda[i];

}

return sum;

}

private double[][] BigF\_ForSystem(){

double[][] Mat = new double[3][];

Mat[0] = BigF1;

Mat[1] = BigF2;

Mat[2] = BigF3;

return Transpose(Mat);

}

void A1A2A3\_Extraction\_from\_AIncludeAll(){

int sum = 0;

for(int i=0; i < PsMat1[0].length; ++i){

A1[i] = AincludeAll[sum];

++sum;

}

for(int i=0; i < PsMat2[0].length; ++i){

A2[i] = AincludeAll[sum];

++sum;

}

for(int i=0; i < PsMat3[0].length;++i){

A3[i] = AincludeAll[sum];

++sum;

}

}

double[][] MakeOnePsiMatrix(){

int ColumnSizePsMat1 = PsMat1[0].length;

int ColumnSizePsMat2 = PsMat2[0].length;

int ColumnSizePsMat3 = PsMat3[0].length;

double[][] PsiBigMatrix = new double[NumberOfSamples][ColumnSizePsMat1 + ColumnSizePsMat2 + ColumnSizePsMat3];

for(int i = 0; i < NumberOfSamples; ++i) {

int sum = 0;

for (int j = 0; j < ColumnSizePsMat1; ++j){

PsiBigMatrix[i][sum] = PsMat1[i][j];

++sum;

}

for(int j =0; j < ColumnSizePsMat2; ++j){

PsiBigMatrix[i][sum] = PsMat2[i][j];

++sum;

}

for(int j = 0; j < ColumnSizePsMat3; ++j){

PsiBigMatrix[i][sum] = PsMat3[i][j];

++sum;

}

}

return PsiBigMatrix;

}

double[] Recovered\_Y(double[] Y\_par){

double[] Recov = new double[Y\_par.length];

coef=(MaxY - MinY);

for(int i=0; i < Y\_par.length; ++i){

Recov[i] = Y\_par[i]\*coef+MinY;

}

return Recov;

}

private void BigF\_initialization(){

BigF1 = new double[NumberOfSamples];

BigF2 = new double[NumberOfSamples];

BigF3 = new double[NumberOfSamples];

int ColumnSizePsiMat1 = PsMat1[0].length;

int ColumnSizePsiMat2 = PsMat2[0].length;

int ColumnSizePsiMat3 = PsMat3[0].length;

// the sum from j1 = 0 to j1 = n1 is writing here from

// from lecture 11 page 15

double sum = 0;

for(int i = 0; i < NumberOfSamples; ++i ){

for(int j = 0; j < ColumnSizePsiMat1; ++j){

sum = sum + A1[j]\*PsMat1[i][j];

}

BigF1[i] = sum;

sum = 0;

for(int j = 0; j < ColumnSizePsiMat2; ++j){

sum = sum + A2[j]\*PsMat2[i][j];

}

BigF2[i] = sum;

sum =0;

for(int j = 0; j < ColumnSizePsiMat3; ++j){

sum = sum + A3[j]\*PsMat3[i][j];

}

BigF3[i] = sum;

sum =0;

}

}

public double[] SolveEquation(double[][] A, double []Y, double[] residual, double[] Ax){

double[] Solution = new double[A[0].length];

Method = new Conjugate\_gradient\_method(A, Y, A.length, A[0].length);

Matrix AA = new Matrix(A, A.length, A[0].length);

Matrix x = new Matrix(Method.X,Method.X.length);

Matrix b = new Matrix(Y,Y.length);

Matrix resid = b.minus(AA.times(x));

Matrix AxComputed = AA.times(x);

for(int i = 0; i < A[0].length; ++i){

Solution[i] = x.get(i,0);

}

for (int i = 0; i < Y.length; ++i) {

residual[i] = resid.get(i, 0);

Ax[i] = AxComputed.get(i,0);

}

return Solution;

}

public Calculations(double[][] X1, double [][] X2, double[][] X3, double[] Y, Polynom type, int[] PolyOrder,

int NumSamples, double MaxY, double MinY)

{

// X1, X2, X3, and maybe Y is transposed, for example X1 has 2\*45 dimensionality

PolyOrderJ = PolyOrder;

this.type = type;

NumberOfSamples = NumSamples;

this.MaxY = MaxY;

this.MinY = MinY;

Y\_true = Recovered\_Y(Y);

// здесь формирует матрицу A

MatrixForLinearEquation(X1, X2, X3);

}

public void Lambda\_one\_matrix (double[][] X1, double [][] X2, double[][] X3, double[] Y, boolean SecondLevelTogether)

{

ResidVecLambda = new double[Y.length];

Ax = new double[Y.length];

double[] LambdaVec = SolveEquation(A,Y,ResidVecLambda,Ax);

ResidVecLambda\_max=0;

for (int i=0;i<ResidVecLambda.length;i++)

{

if (Math.abs(ResidVecLambda[i])>ResidVecLambda\_max)

ResidVecLambda\_max=Math.abs(ResidVecLambda[i]);

}

double[][] ResidVecA = new double[3][Y.length];

double[] ResidForBigPsi = new double[Y.length];

Lambda1 = new double[X1.length][PolyOrderJ[0]+1];

Lambda2 = new double[X2.length][PolyOrderJ[1]+1];

Lambda3 = new double[X3.length][PolyOrderJ[2]+1];

LambdaMatrixCreation(LambdaVec);

PsMat1 = PsiMatrixCreation(X1,X2,X3,Lambda1, Lambda2,Lambda3,0);

PsMat2 = PsiMatrixCreation(X1,X2,X3,Lambda1, Lambda2,Lambda3,1);

PsMat3 = PsiMatrixCreation(X1,X2,X3,Lambda1, Lambda2,Lambda3,2);

A1 = SolveEquation(PsMat1,Y, ResidVecA[0],Ax);

A2 = SolveEquation(PsMat2,Y, ResidVecA[1],Ax);

A3 = SolveEquation(PsMat3,Y, ResidVecA[2],Ax);

if(SecondLevelTogether) {

PsMat = MakeOnePsiMatrix();

AincludeAll = SolveEquation(PsMat, Y, ResidForBigPsi,Ax);

A1A2A3\_Extraction\_from\_AIncludeAll();

}

BigF\_initialization();

BigF\_System = BigF\_ForSystem();

residualBigF\_System = new double[Y.length];

// double [3] C = {c1, c2, c3}

C = SolveEquation(BigF\_System, Y, residualBigF\_System,Ax);

residualBigF\_System\_max=0;

for (int i=0;i<residualBigF\_System.length;i++)

{

if (Math.abs(residualBigF\_System[i])>(residualBigF\_System\_max))

residualBigF\_System\_max=Math.abs(residualBigF\_System[i]);

}

Y\_model = Recovered\_Y(Ax);

// Here is situated the reshaping of lambda

try{

DataMapping.WriteToFile("Lambda1.txt", Lambda1);

DataMapping.WriteToFile("Lambda2.txt", Lambda2);

DataMapping.WriteToFile("Lambda3.txt", Lambda3);

Conjugate\_gradient\_method.WriteToFile("ResidForBigPsi.txt", ResidForBigPsi);

// DataMapping.WriteToFile("A.txt", A);

DataMapping.WriteToFile("ResidVecASecondLevel.txt", ResidVecA);

Conjugate\_gradient\_method.WriteToFile("Lambda\_vec.txt", LambdaVec);

Conjugate\_gradient\_method.WriteToFile("C\_vec\_2\_level.txt", C);

Conjugate\_gradient\_method.WriteToFile("Resid\_Vec\_Lambda.txt", ResidVecLambda);

Conjugate\_gradient\_method.WriteToFile("ResidBig\_Fsystem.txt", residualBigF\_System);

}

catch (IOException e) {

e.printStackTrace();

}

}

public void Lambda\_three\_matrix (double[][] X1, double [][] X2, double[][] X3, double[] Y, int i)

{

double [][]A\_Lambda\_i={{0}};

switch(i){

case 1:

A\_Lambda\_i = A\_Lambda1.clone();

break;

case 2:

A\_Lambda\_i = A\_Lambda2.clone();

break;

case 3:

A\_Lambda\_i = A\_Lambda3.clone();

break;

}

Method = new Conjugate\_gradient\_method(A\_Lambda\_i, Y, A\_Lambda\_i.length, A\_Lambda\_i[0].length);

Matrix AA = new Matrix(A\_Lambda\_i, A\_Lambda\_i.length, A\_Lambda\_i[0].length);

Matrix x = new Matrix(Method.X, Method.X.length);

Matrix b = new Matrix(Y, Y.length);

Matrix resid = b.minus(AA.times(x));

double[] ResidVecLambda = new double[Y.length];

// double[] ResidMatLambda = new double[Y.length];

double[] LambdaVec = Method.X;

for (int j = 0; j < Y.length; ++j) {

ResidVecLambda[j] = resid.get(j, 0);

}

// Here is situated the reshaping of lambda

int RowSize = 0;

int ColumnSize = 0;

switch(i){

case 1:

RowSize = X1.length;

ColumnSize = PolyOrderJ[0] + 1;

break;

case 2:

RowSize = X2.length;

ColumnSize = PolyOrderJ[1] + 1;

break;

case 3:

RowSize = X3.length;

ColumnSize = PolyOrderJ[2] + 1;

break;

}

double [][] Lambda = new double[RowSize][ColumnSize];

int sum =0;

for(int k = 0; k < RowSize; ++k){

for(int j = 0; j < ColumnSize; ++j){

Lambda[k][j] = LambdaVec[sum];

++sum;

}

}

switch(i){

case 1:

Lambda1 = Lambda.clone();

break;

case 2:

Lambda2 = Lambda.clone();

break;

case 3:

Lambda3 = Lambda.clone();

break;

}

try {

DataMapping.WriteToFile("Lambda"+Integer.toString(i)+"\_three\_matrix.txt", Lambda);

Conjugate\_gradient\_method.WriteToFile("Lambda"+Integer.toString(i)+"\_three\_vec.txt", LambdaVec);

Conjugate\_gradient\_method.WriteToFile("Resid\_Vec\_Lambda"+Integer.toString(i)+"\_three.txt", ResidVecLambda);

} catch (IOException e) {

e.printStackTrace();

}

}

// ------------- This Code is for Checking the function that the bottom

// They works well

// please, dont remove this comments ;)

/\*

double[] Fi = CalculateF\_iUsing\_ScalarF\_Function(X1,X2,X3);

for(int i = 0; i < Y.length; ++i){

ResidMatLambda[i] = Y[i] - Fi[i];

}

// first row by LambdaVec

// Second By LambdaMat

double[][] Residual = new double[2][Y.length];

Residual[0] = ResidVecLambda;

Residual[1] = ResidMatLambda;

try{

DataMapping.WriteToFile("Residual\_within\_lambda.txt", Residual);

DataMapping.WriteToFile("A.txt",A);

}

catch (IOException e) {

e.printStackTrace();

}\*/

}

package sample; /\*\*

\* Created by Gregory on 12.10.2014.

\*/

import java.io.\*;

import java.util.Scanner;

public class DataMapping {

double [][] NormalizedData;

double [][] RealData;

int DimX1;

int DimX2;

int DimX3;

int DimY;

int NumSamples;

double[] MaxArray;

double[] MinArray;

public double Max(double[] array){

int len = array.length;

double max = array[0];

for(int i = 1; i < len; ++i ){

if (max < array[i]) max = array[i];

}

return max;

}

public double Min(double[] array){

int len = array.length;

double min = array[0];

for(int i = 1; i < len; ++i ){

if (min > array[i]) min = array[i];

}

return min;

}

public double [][] Transpose(double [][] Mat){

int RowSize = Mat.length;

int ColumnSize = Mat[0].length;

double[][] MatTrans = new double[ColumnSize][RowSize];

double temp;

for(int i = 0; i < RowSize; ++i){

for(int j = 0; j < ColumnSize; ++j){

MatTrans[j][i] = Mat[i][j];

}

}

return MatTrans;

}

public double[][] Normalize(double[][] Mat){

int RowSize = Mat.length;

int ColumnSize = Mat[0].length;

MaxArray = new double[ColumnSize];

MinArray = new double[ColumnSize];

//We will return this matrix, but it will not be transposed

// we want maximum within fixed column

double[][] HelpMat = Transpose(Mat);

// Process of normalization

for(int i = 0; i < ColumnSize; ++i){

// The max min versus columns is needed here.

MaxArray[i] = Max(HelpMat[i]);

MinArray[i] = Min(HelpMat[i]);

for(int j = 0; j < RowSize; ++j ){

HelpMat[i][j] = (HelpMat[i][j] - MinArray[i])/ (MaxArray[i] - MinArray[i]);

}

}

return Transpose(HelpMat);

}

public static void WriteToFile(String name, double[][] Mat)throws FileNotFoundException, IOException {

PrintWriter fout = new PrintWriter(new FileWriter(name));

//fout.write(this.getSize());

int ColumnLength = Mat[0].length;

int RowLength = Mat.length;

for ( int i = 0; i < RowLength; ++i ) {

for ( int j = 0; j < ColumnLength; ++j ) {

fout.print(String.format("%.4f", Mat[i][j]));

fout.write(" ");

}

fout.write("\n");

}

fout.close();

}

public DataMapping(String FileName, int DimX1, int DimX2, int DimX3, int DimY, int FileLength ){

this.NumSamples = FileLength;

this.DimX1 = DimX1;

this.DimX2 = DimX2;

this.DimX3 = DimX3;

this.DimY = DimY;

int ColumnSize = DimX1 + DimX2 + DimX3 + DimY;

int RowSize = FileLength;

String str;

NormalizedData = new double[RowSize][ColumnSize];

RealData = new double[RowSize][ColumnSize];

try{

File InFile = new File(FileName);

Scanner s = new Scanner(InFile);

for(int i=0; i < RowSize; ++i) {

for (int j = 0; j < ColumnSize; ++j) {

//str = s.next();

// NormalizedData[i][j] = Double.parseDouble(str);

NormalizedData[i][j] = s.nextDouble();

RealData[i][j]=NormalizedData[i][j];

//s.next().toDouble

}

//a1[i] = s.nextInt();

}

} catch (FileNotFoundException e) {

e.printStackTrace();

}

NormalizedData = Normalize(NormalizedData);

try{

WriteToFile("NormData.txt", NormalizedData);

}

catch (IOException e) {

e.printStackTrace();

}

}

public DataMapping(String FileName1, String FileName2,int DimX1, int DimX2, int DimX3, int DimY, int FileLength ){

this.NumSamples = FileLength;

this.DimX1 = DimX1;

this.DimX2 = DimX2;

this.DimX3 = DimX3;

this.DimY = DimY;

int ColumnSize = DimX1 + DimX2 + DimX3 + DimY;

int RowSize = FileLength;

String str;

NormalizedData = new double[RowSize][ColumnSize];

RealData = new double[RowSize][ColumnSize];

try{

File InFile1 = new File(FileName1);

Scanner s1 = new Scanner(InFile1);

File InFile2 = new File(FileName2);

Scanner s2 = new Scanner(InFile2);

for(int i=0; i < RowSize; ++i) {

for (int j = 0; j < ColumnSize; ++j) {

if (j<ColumnSize-1) {

NormalizedData[i][j] = s1.nextDouble();

RealData[i][j] = NormalizedData[i][j];

}

else

{

NormalizedData[i][j] = s2.nextDouble();

RealData[i][j] = NormalizedData[i][j];

}

}

//a1[i] = s.nextInt();

}

} catch (FileNotFoundException e) {

e.printStackTrace();

}

NormalizedData = Normalize(NormalizedData);

try{

WriteToFile("NormData.txt", NormalizedData);

}

catch (IOException e) {

e.printStackTrace();

}

}

}

package sample;

import javafx.beans.property.SimpleStringProperty;

import javafx.beans.value.ObservableValue;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.event.ActionEvent;

import javafx.scene.chart.LineChart;

import javafx.scene.chart.XYChart;

import javafx.scene.control.\*;

import javafx.scene.control.TableColumn.CellDataFeatures;

import javafx.util.Callback;

import javax.swing.\*;

import java.io.\*;

import java.util.Arrays;

public class ControllerSample {

public Polynom type\_now;

public TextField text\_file\_input;

public TableView table\_data\_input;

public TextField box\_y;

public TextField box\_x2;

public TextField box\_x3;

public TextField box\_x1;

public TextField box\_length;

public TextField Text\_res;

public double[][] X1;

public double[][] X2;

public double[][] X3;

public double[][] Y;

public double[] MaxY;

public double[] MinY;

public RadioButton Radio1;

public RadioButton Radio2;

public RadioButton Radio3;

public RadioButton Radio4;

public TextField box\_p1;

public TextField box\_p2;

public TextField box\_p3;

public TableView data\_real;

public LineChart y0\_first\_level;

public LineChart Y0\_last\_level;

public TextField Y\_order;

public ToggleButton SecondLevelTogether;

public LineChart Graph\_Y;

public TextArea Text\_results;

public TextField Text\_Y;

public Initial\_data data\_in;

public Button box\_down\_main;

public Button box\_down\_y;

//public DataMapping DaMa;

public Calculations LambdaDefinition;

public TextField Text\_err1;

public TextField Text\_err\_res;

public TextField Text\_err\_res\_recover;

public LineChart Graph\_Y\_norm;

public void open\_file\_input(javafx.event.ActionEvent actionEvent) {

final JFileChooser fileopen = new JFileChooser();

int ret = fileopen.showOpenDialog(null);

if (ret == JFileChooser.APPROVE\_OPTION) {

File file\_input = fileopen.getSelectedFile();

text\_file\_input.setText(file\_input.getPath());

}

}

public void test(ActionEvent actionEvent) {

if ( Radio1.selectedProperty().getValue())

{

type\_now=Polynom.Chebyshev;

}

if ( Radio2.selectedProperty().getValue())

{

type\_now=Polynom.Lejandr;

}

if ( Radio3.selectedProperty().getValue())

{

type\_now=Polynom.Lagger;

}

if ( Radio4.selectedProperty().getValue())

{

type\_now=Polynom.Hermit;

}

data\_in.add(type\_now, Integer.parseInt(box\_p1.getText()), Integer.parseInt(box\_p2.getText()), Integer.parseInt(box\_p3.getText()));

int Y\_ord=Integer.parseInt(Y\_order.getText())-1;

int[] Order = {data\_in.p1, data\_in.p2, data\_in.p3};

LambdaDefinition = new Calculations(X1, X2, X3, Y[Y\_ord],data\_in.type, Order, Y[Y\_ord].length, MaxY[Y\_ord],MinY[Y\_ord]);

LambdaDefinition.Lambda\_one\_matrix(X1, X2, X3, Y[Y\_ord],SecondLevelTogether.selectedProperty().getValue());

// LambdaDefinition.Lambda\_three\_matrix(X1, X2, X3, Y[Y\_ord], 1);

// LambdaDefinition.Lambda\_three\_matrix(X1, X2, X3, Y[Y\_ord], 2);

// LambdaDefinition.Lambda\_three\_matrix(X1, X2, X3, Y[Y\_ord], 3);

double Y\_error\_rec=0;

for (int i=0;i<LambdaDefinition.Y\_model.length;i++)

{

if(Math.abs(LambdaDefinition.Y\_model[i]-LambdaDefinition.Y\_true[i])>Y\_error\_rec)

Y\_error\_rec=Math.abs(LambdaDefinition.Y\_model[i]-LambdaDefinition.Y\_true[i]);

}

Text\_err\_res\_recover.setText(Double.toString(Y\_error\_rec));

Text\_err1.setText(Double.toString(LambdaDefinition.ResidVecLambda\_max));

Text\_err\_res.setText(Double.toString(LambdaDefinition.residualBigF\_System\_max));

Write\_Results(Text\_res.getText(), LambdaDefinition, data\_in, Y\_error\_rec);

Write\_to\_text(Text\_res.getText());

}

public void DistributeData(double[][] Mat, int DimX1, int DimX2, int DimX3, int DimY, int NumSamples) {

X1 = new double[DimX1][NumSamples];

for (int i = 0; i < DimX1; ++i) {

for (int j = 0; j < NumSamples; ++j) {

X1[i][j] = Mat[j][i];

}

}

X2 = new double[DimX2][NumSamples];

for (int i = 0; i < DimX2; ++i) {

for (int j = 0; j < NumSamples; ++j) {

X2[i][j] = Mat[j][DimX1 + i];

}

}

X3 = new double[DimX3][NumSamples];

for (int i = 0; i < DimX3; ++i) {

for (int j = 0; j < NumSamples; ++j) {

X3[i][j] = Mat[j][DimX1 + DimX2 + i];

}

}

Y = new double[DimY][NumSamples];

for (int i = 0; i < DimY; ++i) {

for (int j = 0; j < NumSamples; ++j) {

Y[i][j] = Mat[j][DimX1 + DimX2 + DimX3 + i];

}

}

}

public void OnRadio1(ActionEvent actionEvent) {

if ( !Radio1.selectedProperty().getValue())

{

Radio1.setSelected(true);

type\_now=Polynom.Chebyshev;

}

else

{

Radio2.setSelected(false);

Radio3.setSelected(false);

Radio4.setSelected(false);

}

}

public void OnRadio2(ActionEvent actionEvent) {

if ( !Radio2.selectedProperty().getValue())

{

Radio2.setSelected(true);

type\_now=Polynom.Lejandr;

}

else

{

Radio1.setSelected(false);

Radio3.setSelected(false);

Radio4.setSelected(false);

}

}

public void OnRadio3(ActionEvent actionEvent) {

if ( !Radio3.selectedProperty().getValue())

{

Radio3.setSelected(true);

type\_now=Polynom.Lagger;

}

else

{

Radio2.setSelected(false);

Radio1.setSelected(false);

Radio4.setSelected(false);

}

}

public void OnRadio4(ActionEvent actionEvent) {

if ( !Radio4.selectedProperty().getValue())

{

Radio4.setSelected(true);

type\_now=Polynom.Hermit;

}

else

{

Radio2.setSelected(false);

Radio3.setSelected(false);

Radio1.setSelected(false);

}

}

public void OnButtonRes(ActionEvent actionEvent) {

JFileChooser fileopen = new JFileChooser();

int ret = fileopen.showDialog(null, "Cохранить");

if (ret == JFileChooser.APPROVE\_OPTION) {

File file\_input = fileopen.getSelectedFile();

Text\_res.setText(file\_input.getPath());

}

}

public void OnClean(ActionEvent actionEvent) {

y0\_first\_level.getData().removeAll();

y0\_first\_level.getData().clear();

Graph\_Y.getData().removeAll();

Graph\_Y.getData().clear();

Y0\_last\_level.getData().removeAll();

Y0\_last\_level.getData().clear();

Graph\_Y\_norm.getData().removeAll();

Graph\_Y\_norm.getData().clear();

}

public void Write\_to\_text (String file\_name)

{ try {

File file = new File( file\_name );

BufferedReader br = new BufferedReader (

new InputStreamReader(

new FileInputStream( file ), "UTF-8"

)

);

String line = null;

while ((line = br.readLine()) != null) {

Text\_results.setText(Text\_results.getText()+line+"\n");

}

br.close();

}

catch (IOException e) {

e.printStackTrace();

}

}

public void Write\_Results (String file\_rez\_name,Calculations LambdaDefinition, Initial\_data data\_in, double err )

{

try {

PrintWriter fout = new PrintWriter(new FileWriter(file\_rez\_name));

fout.write("Полученные коэффциенты при решении задачи\n");

fout.write("Первый уровень (lambda1)\n");

File file = new File( "Lambda1.txt" );

BufferedReader br = new BufferedReader (

new InputStreamReader(

new FileInputStream( file ), "UTF-8"

)

);

String line = null;

while ((line = br.readLine()) != null) {

fout.println( line );

}

br.close();

fout.write("Первый уровень (Lambda2)\n");

file = new File( "Lambda2.txt" );

br = new BufferedReader (

new InputStreamReader(

new FileInputStream( file ), "UTF-8"

)

);

line = null;

while ((line = br.readLine()) != null) {

fout.println( line );

}

br.close();

fout.write("Первый уровень (Lambda3)\n");

file = new File( "Lambda3.txt" );

br = new BufferedReader (

new InputStreamReader(

new FileInputStream( file ), "UTF-8"

)

);

line = null;

while ((line = br.readLine()) != null) {

fout.println( line );

}

br.close();

int ColumnLength = LambdaDefinition.A1.length;

fout.write("Второй уровень (а1)\n");

for (int j = 0; j < ColumnLength; ++j) {

fout.print(String.format("%.4f", LambdaDefinition.A1[j]));

fout.write(" ");

}

fout.write("\n");

ColumnLength = LambdaDefinition.A1.length;

fout.write("Второй уровень (а2)\n");

for (int j = 0; j < ColumnLength; ++j) {

fout.print(String.format("%.4f", LambdaDefinition.A2[j]));

fout.write(" ");

}

fout.write("\n");

ColumnLength = LambdaDefinition.A3.length;

fout.write("Второй уровень (а3)\n");

for (int j = 0; j < ColumnLength; ++j) {

fout.print(String.format("%.4f", LambdaDefinition.A3[j]));

fout.write(" ");

}

fout.write("\n");

ColumnLength = LambdaDefinition.C.length;

fout.write("Третий уровень (С)\n");

for (int j = 0; j < ColumnLength; ++j) {

fout.print(String.format("%.4f", LambdaDefinition.C[j]));

fout.write(" ");

}

fout.write("\n");

String s1="",s2="",s3="";

for (int i=0;i<data\_in.length\_of\_x1;i++)

for (int j=0;j<data\_in.p1;j++)

s1=s1+"("+Double.toString(LambdaDefinition.Lambda1[i][j])+")"+"\*T"+Integer.toString(j)+"(x1"+Integer.toString(i)+")+";

for (int i=0;i<data\_in.length\_of\_x2;i++)

for (int j=0;j<data\_in.p2;j++)

s2=s2+"("+Double.toString(LambdaDefinition.Lambda2[i][j])+")"+"\*T"+Integer.toString(j)+"(x2"+Integer.toString(i)+")+";

for (int i=0;i<data\_in.length\_of\_x3;i++)

for (int j=0;j<data\_in.p3;j++)

s3=s3+"("+Double.toString(LambdaDefinition.Lambda3[i][j])+")"+"\*T"+Integer.toString(j)+"(x3"+Integer.toString(i)+")+";

s3=s3.substring(0, s3.length() - 1);

String s1z=s1+s2+s3;

fout.write("psi функции\n");

fout.write(s1z);

String s12="",s22="",s32="";

for (int i=0;i<data\_in.length\_of\_x1;i++)

s12=s12+"("+Double.toString(LambdaDefinition.A1[i])+")"+"\*psi1(x1"+Integer.toString(i)+")+";

for (int i=0;i<data\_in.length\_of\_x2;i++)

s22=s22+"("+Double.toString(LambdaDefinition.A2[i])+")"+"\*psi2(x2"+Integer.toString(i)+")+";

for (int i=0;i<data\_in.length\_of\_x3;i++)

s32=s32+"("+Double.toString(LambdaDefinition.A3[i])+")"+"\*psi3(x3"+Integer.toString(i)+")+";

s32=s32.substring(0, s32.length() - 1);

// String s12z=s12+s22+s32;

// fout.write("\n");

fout.write("Второй уровень функции\n");

fout.write("Ф1="+s12+"\n");

fout.write("Ф2="+s22+"\n");

fout.write("Ф3="+s32+"\n");

fout.write("\n");

String s14="";

fout.write("Третий уровень функции\n");

for (int i=0;i<3;i++)

s14=s14+Double.toString(LambdaDefinition.C[i])+"\*Ф(x1)+";

s14=s14.substring(0, s14.length() - 1);

fout.write(s14);

fout.write("\n");

fout.write("Получившаяся в результате функция\n");

String s10="", s20="", s30="";

for (int i=0;i<data\_in.length\_of\_x1;i++)

s10=s10+Double.toString(LambdaDefinition.A1[i])+"\*("+s1+")+";

for (int i=0;i<data\_in.length\_of\_x2;i++)

s20=s20+Double.toString(LambdaDefinition.A2[i])+"\*("+s2+")+";

for (int i=0;i<data\_in.length\_of\_x3;i++)

s30=s30+Double.toString(LambdaDefinition.A3[i])+"\*("+s3+")+";

s30=s30.substring(0, s30.length() - 1);

s10=Double.toString(LambdaDefinition.C[0])+"\*("+s10+")"+Double.toString(LambdaDefinition.C[0])+"\*("+s20+")"+Double.toString(LambdaDefinition.C[0])+"\*("+s30+")";

fout.write(s10+"\n");

fout.write("Получившаяся в результате восстановленная функция\n");

s10=Double.toString(LambdaDefinition.MinY)+"+"+Double.toString(LambdaDefinition.coef)+"\*("+s10+")";

fout.write(s10+"\n");

fout.write("Ошибка восстановленой функции\n");

s10=Double.toString(err);

fout.write(s10+"\n");

fout.close();

}

catch (IOException e) {

e.printStackTrace();

}

}

public void open\_file\_input\_y(ActionEvent actionEvent) {

JFileChooser fileopen = new JFileChooser();

int ret = fileopen.showDialog(null, "Открыть файл");

if (ret == JFileChooser.APPROVE\_OPTION) {

File file\_input = fileopen.getSelectedFile();

Text\_Y.setText(file\_input.getPath());

}

}

public void ONbox\_down\_main(ActionEvent actionEvent) {

data\_real.getColumns().removeAll();

table\_data\_input.getColumns().removeAll();

data\_real.getColumns().clear();

table\_data\_input.getColumns().clear();

ObservableList<String[]> data1 = FXCollections.observableArrayList();

ObservableList<String[]> data = FXCollections.observableArrayList();

data1.removeAll();

table\_data\_input.setItems(data1);

data.removeAll();

data\_real.setItems(data);

data\_real.getColumns().removeAll();

table\_data\_input.getColumns().removeAll();

data\_in = new Initial\_data(Integer.parseInt(box\_length.getText()), Integer.parseInt(box\_x1.getText()), Integer.parseInt(box\_x2.getText()), Integer.parseInt(box\_x3.getText()), Integer.parseInt(box\_y.getText()));

int RowSize = data\_in.length\_of\_sample;

int ColumnSize = data\_in.column\_sum;

DataMapping DaMa = new DataMapping(text\_file\_input.getText(), data\_in.length\_of\_x1, data\_in.length\_of\_x2, data\_in.length\_of\_x3, data\_in.length\_of\_y, data\_in.length\_of\_sample);

MaxY = new double[data\_in.length\_of\_y];

MinY = new double[data\_in.length\_of\_y];

int step1 = data\_in.length\_of\_x1 + data\_in.length\_of\_x2 + data\_in.length\_of\_x3;

int step2 = step1 + data\_in.length\_of\_y;

for(int i = step1; i < step2; ++i){

MaxY[i - step1] = DaMa.MaxArray[i];

MinY[i - step1] = DaMa.MinArray[i];

}

String[][] staffArray = new String[RowSize + 1][ColumnSize];

for (int j = 0; j < data\_in.length\_of\_x1; ++j)

staffArray[0][j] = "X1" + Integer.toString(j + 1);

for (int j = 0; j < data\_in.length\_of\_x2; ++j)

staffArray[0][data\_in.length\_of\_x1 + j] = "X2" + Integer.toString(j + 1);

for (int j = 0; j < data\_in.length\_of\_x3; ++j)

staffArray[0][data\_in.length\_of\_x1 + data\_in.length\_of\_x2 + j] = "X3" + Integer.toString(1 + j);

for (int j = 0; j < data\_in.length\_of\_y; ++j)

staffArray[0][data\_in.length\_of\_x1 + data\_in.length\_of\_x3 + data\_in.length\_of\_x2 + j] = "Y" + Integer.toString(j + 1);

for (int i = 0; i < RowSize; ++i) {

for (int j = 0; j < ColumnSize; ++j)

staffArray[i + 1][j] = String.valueOf(DaMa.RealData[i][j]);

}

// StackPane root = new StackPane();

data = FXCollections.observableArrayList();

data.addAll(Arrays.asList(staffArray));

data.remove(0);//remove titles from data

for (int i = 0; i < staffArray[0].length; i++) {

TableColumn tc = new TableColumn(staffArray[0][i]);

final int colNo = i;

tc.setCellValueFactory(new Callback<CellDataFeatures<String[], String>, ObservableValue<String>>() {

@Override

public ObservableValue<String> call(CellDataFeatures<String[], String> p) {

return new SimpleStringProperty((p.getValue()[colNo]));

}

});

data\_real.getColumns().add(tc);

}

data\_real.setItems(data);

String[][] staffArray1 = new String[RowSize + 1][ColumnSize];

for (int j = 0; j < data\_in.length\_of\_x1; ++j)

staffArray1[0][j] = "X1" + Integer.toString(j + 1);

for (int j = 0; j < data\_in.length\_of\_x2; ++j)

staffArray1[0][data\_in.length\_of\_x1 + j] = "X2" + Integer.toString(j + 1);

for (int j = 0; j < data\_in.length\_of\_x3; ++j)

staffArray1[0][data\_in.length\_of\_x1 + data\_in.length\_of\_x2 + j] = "X3" + Integer.toString(1 + j);

for (int j = 0; j < data\_in.length\_of\_y; ++j)

staffArray1[0][data\_in.length\_of\_x1 + data\_in.length\_of\_x3 + data\_in.length\_of\_x2 + j] = "Y" + Integer.toString(j + 1);

for (int i = 0; i < RowSize; ++i) {

for (int j = 0; j < ColumnSize; ++j)

staffArray1[i + 1][j] = String.valueOf(DaMa.NormalizedData[i][j]);

}

data1 = FXCollections.observableArrayList();

data1.addAll(Arrays.asList(staffArray1));

data1.remove(0);//remove titles from data

for (int i = 0; i < staffArray1[0].length; i++) {

TableColumn tc = new TableColumn(staffArray1[0][i]);

final int colNo = i;

tc.setCellValueFactory(new Callback<CellDataFeatures<String[], String>, ObservableValue<String>>() {

@Override

public ObservableValue<String> call(CellDataFeatures<String[], String> p) {

return new SimpleStringProperty((p.getValue()[colNo]));

}

});

table\_data\_input.getColumns().add(tc);

}

table\_data\_input.setItems(data1);

// X\_I and Y initialization

DistributeData(DaMa.NormalizedData, DaMa.DimX1, DaMa.DimX2, DaMa.DimX3, DaMa.DimY, DaMa.NumSamples);

try {

DataMapping.WriteToFile("X1.txt", X1);

DataMapping.WriteToFile("X2.txt", X2);

DataMapping.WriteToFile("X3.txt", X3);

DataMapping.WriteToFile("Y.txt", Y);

} catch (IOException e) {

e.printStackTrace();

}

// box\_down\_main.setDisable(true);

box\_down\_y.setDisable(false);

}

public void Onbox\_down\_y(ActionEvent actionEvent) {

data\_real.getColumns().removeAll();

table\_data\_input.getColumns().removeAll();

data\_real.getColumns().clear();

table\_data\_input.getColumns().clear();

ObservableList<String[]> data1 = FXCollections.observableArrayList();

ObservableList<String[]> data = FXCollections.observableArrayList();

data1.removeAll();

table\_data\_input.setItems(data1);

data.removeAll();

data\_real.setItems(data);

data\_real.getColumns().removeAll();

table\_data\_input.getColumns().removeAll();

box\_y.setText(Integer.toString(Integer.parseInt(box\_y.getText()) + 1));

data\_in = new Initial\_data(Integer.parseInt(box\_length.getText()), Integer.parseInt(box\_x1.getText()), Integer.parseInt(box\_x2.getText()), Integer.parseInt(box\_x3.getText()), Integer.parseInt(box\_y.getText()));

int RowSize = data\_in.length\_of\_sample;

int ColumnSize = data\_in.column\_sum;

DataMapping DaMa = new DataMapping(text\_file\_input.getText(),Text\_Y.getText(), data\_in.length\_of\_x1, data\_in.length\_of\_x2, data\_in.length\_of\_x3, data\_in.length\_of\_y, data\_in.length\_of\_sample);

MaxY = new double[data\_in.length\_of\_y];

MinY = new double[data\_in.length\_of\_y];

int step1 = data\_in.length\_of\_x1 + data\_in.length\_of\_x2 + data\_in.length\_of\_x3;

int step2 = step1 + data\_in.length\_of\_y;

for(int i = step1; i < step2; ++i){

MaxY[i - step1] = DaMa.MaxArray[i];

MinY[i - step1] = DaMa.MinArray[i];

}

String[][] staffArray = new String[RowSize + 1][ColumnSize];

for (int j = 0; j < data\_in.length\_of\_x1; ++j)

staffArray[0][j] = "X1" + Integer.toString(j + 1);

for (int j = 0; j < data\_in.length\_of\_x2; ++j)

staffArray[0][data\_in.length\_of\_x1 + j] = "X2" + Integer.toString(j + 1);

for (int j = 0; j < data\_in.length\_of\_x3; ++j)

staffArray[0][data\_in.length\_of\_x1 + data\_in.length\_of\_x2 + j] = "X3" + Integer.toString(1 + j);

for (int j = 0; j < data\_in.length\_of\_y; ++j)

staffArray[0][data\_in.length\_of\_x1 + data\_in.length\_of\_x3 + data\_in.length\_of\_x2 + j] = "Y" + Integer.toString(j + 1);

for (int i = 0; i < RowSize; ++i) {

for (int j = 0; j < ColumnSize; ++j)

staffArray[i + 1][j] = String.valueOf(DaMa.RealData[i][j]);

}

// StackPane root = new StackPane();

// ObservableList<String[]>

data = FXCollections.observableArrayList();

data.removeAll();

data.addAll(Arrays.asList(staffArray));

data.remove(0);//remove titles from data

for (int i = 0; i < staffArray[0].length; i++) {

TableColumn tc = new TableColumn(staffArray[0][i]);

final int colNo = i;

tc.setCellValueFactory(new Callback<CellDataFeatures<String[], String>, ObservableValue<String>>() {

@Override

public ObservableValue<String> call(CellDataFeatures<String[], String> p) {

return new SimpleStringProperty((p.getValue()[colNo]));

}

});

data\_real.getColumns().add(tc);

}

data\_real.setItems(data);

String[][] staffArray1 = new String[RowSize + 1][ColumnSize];

for (int j = 0; j < data\_in.length\_of\_x1; ++j)

staffArray1[0][j] = "X1" + Integer.toString(j + 1);

for (int j = 0; j < data\_in.length\_of\_x2; ++j)

staffArray1[0][data\_in.length\_of\_x1 + j] = "X2" + Integer.toString(j + 1);

for (int j = 0; j < data\_in.length\_of\_x3; ++j)

staffArray1[0][data\_in.length\_of\_x1 + data\_in.length\_of\_x2 + j] = "X3" + Integer.toString(1 + j);

for (int j = 0; j < data\_in.length\_of\_y; ++j)

staffArray1[0][data\_in.length\_of\_x1 + data\_in.length\_of\_x3 + data\_in.length\_of\_x2 + j] = "Y" + Integer.toString(j + 1);

for (int i = 0; i < RowSize; ++i) {

for (int j = 0; j < ColumnSize; ++j)

staffArray1[i + 1][j] = String.valueOf(DaMa.NormalizedData[i][j]);

}

// ObservableList<String[]>

data1 = FXCollections.observableArrayList();

data1.removeAll();

data1.addAll(Arrays.asList(staffArray1));

data1.remove(0);//remove titles from data

for (int i = 0; i < staffArray1[0].length; i++) {

TableColumn tc = new TableColumn(staffArray1[0][i]);

final int colNo = i;

tc.setCellValueFactory(new Callback<CellDataFeatures<String[], String>, ObservableValue<String>>() {

@Override

public ObservableValue<String> call(CellDataFeatures<String[], String> p) {

return new SimpleStringProperty((p.getValue()[colNo]));

}

});

table\_data\_input.getColumns().add(tc);

}

table\_data\_input.setItems(data1);

// X\_I and Y initialization

DistributeData(DaMa.NormalizedData, DaMa.DimX1, DaMa.DimX2, DaMa.DimX3, DaMa.DimY, DaMa.NumSamples);

try {

DataMapping.WriteToFile("X1.txt", X1);

DataMapping.WriteToFile("X2.txt", X2);

DataMapping.WriteToFile("X3.txt", X3);

DataMapping.WriteToFile("Y.txt", Y);

} catch (IOException e) {

e.printStackTrace();

}

box\_down\_y.setDisable(true);

}

public void Onbox\_clean\_data(ActionEvent actionEvent) {

// table\_data\_input = new TableView();

data\_real.getColumns().removeAll();

table\_data\_input.getColumns().removeAll();

data\_real.getColumns().clear();

table\_data\_input.getColumns().clear();

ObservableList<String[]> data1 = FXCollections.observableArrayList();

ObservableList<String[]> data = FXCollections.observableArrayList();

data1.removeAll();

table\_data\_input.setItems(data1);

box\_down\_main.setDisable(false);

box\_down\_y.setDisable(true);

data.removeAll();

data\_real.setItems(data);

data\_real.getColumns().removeAll();

table\_data\_input.getColumns().removeAll();

}

public void OnButton\_Draw\_real(ActionEvent actionEvent) {

XYChart.Series series = new XYChart.Series();

series = new XYChart.Series();

series.setName("Real\_data");

for (int i=0; i<Y[0].length;i++)

series.getData().add(new XYChart.Data(Integer.toString(i+1), LambdaDefinition.Y\_true[i]));

Graph\_Y.getData().add(series);

series = new XYChart.Series();

series = new XYChart.Series();

series.setName("Real\_data");

for (int i=0; i<Y[0].length;i++)

series.getData().add(new XYChart.Data(Integer.toString(i+1), Y[Integer.parseInt(Y\_order.getText())-1][i]));

Graph\_Y\_norm.getData().add(series);

}

public void OnButton\_Draw\_model(ActionEvent actionEvent) {

XYChart.Series series = new XYChart.Series();

series.setName(data\_in.name\_of\_graph);

for (int i=0; i<Y[0].length;i++)

series.getData().add(new XYChart.Data(Integer.toString(i+1), LambdaDefinition.ResidVecLambda[i]));

y0\_first\_level.getData().add(series);

series = new XYChart.Series();

series.setName(data\_in.name\_of\_graph);

for (int i=0; i<Y[0].length;i++)

series.getData().add(new XYChart.Data(Integer.toString(i+1), LambdaDefinition.residualBigF\_System[i]));

Y0\_last\_level.getData().add(series);

series = new XYChart.Series();

series.setName(data\_in.name\_of\_graph);

for (int i=0; i<Y[0].length;i++)

series.getData().add(new XYChart.Data(Integer.toString(i+1), LambdaDefinition.Y\_model[i]));

Graph\_Y.getData().add(series);

series = new XYChart.Series();

series.setName(data\_in.name\_of\_graph);

for (int i=0; i<Y[0].length;i++)

series.getData().add(new XYChart.Data(Integer.toString(i+1), LambdaDefinition.Ax[i]));

Graph\_Y\_norm.getData().add(series);

}

}

package sample;

/\*\*

\* Created by Марк on 14.10.2014.

\*/

public class Initial\_data {

int length\_of\_sample=45;

int length\_of\_x1=2;

int length\_of\_x2=2;

int length\_of\_x3=3;

int length\_of\_y=4;

int column\_sum=11;

public String name\_of\_graph;

Polynom type;

int p1=3;

int p2=3;

int p3=3;

public Initial\_data(int ls, int l1, int l2, int l3, int ly,Polynom type1, int p11, int p12, int p13 )

{

length\_of\_sample=ls;

length\_of\_x1=l1;

length\_of\_x2=l2;

length\_of\_x3=l3;

length\_of\_y=ly;

column\_sum=length\_of\_x1+length\_of\_x2+length\_of\_x3+length\_of\_y;

type=type1;

p1=p11;

p2=p12;

p3=p13;

if ( type1==Polynom.Chebyshev)

{

name\_of\_graph="Chebyshev "+Integer.toString(p11)+" "+Integer.toString(p12)+" "+Integer.toString(p13);

}

if ( type1==Polynom.Lejandr)

{

name\_of\_graph="Lejandr "+Integer.toString(p11)+" "+Integer.toString(p12)+" "+Integer.toString(p13);

}

if (type1==Polynom.Lagger)

{

name\_of\_graph="Lagger "+Integer.toString(p11)+" "+Integer.toString(p12)+" "+Integer.toString(p13);

}

if (type1==Polynom.Hermit)

{

name\_of\_graph="Hermit "+Integer.toString(p11)+" "+Integer.toString(p12)+" "+Integer.toString(p13);

}

}

public void add(Polynom type1, int p11, int p12, int p13 )

{

column\_sum=length\_of\_x1+length\_of\_x2+length\_of\_x3+length\_of\_y;

type=type1;

p1=p11;

p2=p12;

p3=p13;

if ( type1==Polynom.Chebyshev)

{

name\_of\_graph="Chebyshev "+Integer.toString(p11)+" "+Integer.toString(p12)+" "+Integer.toString(p13);

}

if ( type1==Polynom.Lejandr)

{

name\_of\_graph="Lejandr "+Integer.toString(p11)+" "+Integer.toString(p12)+" "+Integer.toString(p13);

}

if (type1==Polynom.Lagger)

{

name\_of\_graph="Lagger "+Integer.toString(p11)+" "+Integer.toString(p12)+" "+Integer.toString(p13);

}

if (type1==Polynom.Hermit)

{

name\_of\_graph="Hermit "+Integer.toString(p11)+" "+Integer.toString(p12)+" "+Integer.toString(p13);

}

}

public Initial\_data(int ls, int l1, int l2, int l3, int ly)

{

length\_of\_sample=ls;

length\_of\_x1=l1;

length\_of\_x2=l2;

length\_of\_x3=l3;

length\_of\_y=ly;

column\_sum=length\_of\_x1+length\_of\_x2+length\_of\_x3+length\_of\_y;

}

public void set\_sample(int i)

{

length\_of\_sample=i;

}

public void set\_x3(int i)

{

length\_of\_x3=i;

}

public void set\_x1(int i)

{

length\_of\_x1=i;

}

public void set\_x2(int i)

{

length\_of\_x2=i;

}

public void set\_y(int i)

{

length\_of\_y=i;

}

}

package sample;

/\*\*

\* Created by Марк on 18.10.2014.

\*/

import Jama.\*;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.lang.Math;

public class Conjugate\_gradient\_method {

int dim1;

int dim2;

Matrix A;

// Matrix ATA=A.transpose().times(A);

Matrix b;

Matrix x;

double[] X;

public Conjugate\_gradient\_method(double[][] F, double[] l, int dim1, int dim2) {

A = new Matrix(F, dim1, dim2);

b = new Matrix(l, dim1);

x = new Matrix(dim2,1);

double rsnew = 0;

Matrix ATA = (A.transpose().times(A)).copy();

Matrix ATb=(A.transpose().times(b)).copy();

Matrix r = ATb.minus(ATA.times(x)).copy();

Matrix p = r.copy();

double rsold = (r.transpose().times(r)).get(0, 0);

for (int i = 0; i < 10e4; i++) {

Matrix Ap = ATA.times(p);

double alpha = rsold / (p.transpose().times(Ap).get(0, 0));

x = p.times(alpha).plus(x);

r = r.minus(Ap.times(alpha));

rsnew = (r.transpose().times(r)).get(0, 0);

if (Math.sqrt(rsnew) < 1e-10) {

break;

}

p = p.times(rsnew / rsold).plus(r);

rsold = rsnew;

}

X = new double[dim2];

for (int i = 0; i < dim2; i++)

X[i] = x.get(i, 0);

}

public static void WriteToFile(String name, double[] X) throws FileNotFoundException, IOException {

PrintWriter fout = new PrintWriter(new FileWriter(name));

//fout.write(this.getSize());

int ColumnLength = X.length;

for (int j = 0; j < ColumnLength; ++j) {

fout.print(String.format("%.4f", X[j]));

fout.write(" ");

}

fout.write("\n");

fout.close();

}

}