UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Technická dokumentácia
SPEKTROSKOPICKÉ DÁTA

Michal Chamula,
Tomáš Bordáč,
Martina Bodišová,
Patrik Fašang

2018

Obsah

1 Úvod

1.1 Podstata dokumentu

Tento dokument popisuje požiadavky zadávateľa na softvér vyvíjaný v projekte Spracovanie spektroskopických dát, návrh systému, popis implementácie, testovacie scenáre a používateľskú príručku.

1.2 Určenie dokumentu

Tento dokument je určený stakeholderom, ktorí na základe popísaných požiadaviek budú softvér vyvíjať. Finálna verzia dokumentu špecifikácia požiadaviek je odsúhlasená zadávateľom.

1.3 Rozsah Systému

Projekt je dynamicky linkovaná knižnica (DLL), ktorá slúži na rátanie komplexných, neanalytických funkcií. Neobsahuje grafické používateľské rozhranie.

1.4 Slovník cudzích pojmov

- **DLL** -(angl. Dynamic Link Library) je množina malých programov, ktorá môže byť použitá viac ako jedným programom v tom istom čase. Táto množina je zväčša uložená v súboroch so suffixom ".dll"

- **Stakeholder-** osoba alebo skupina osôb, ktorá sa podieľa na rovnakom projekte, napr. podnikaní, programovaní, vede a pod.

- **LabVIEW** – (angl. Laboratory Virtual Instrument Engineering Workbench) je vývojové prostredie určené na vizuálne programovanie s podporou čítať DLL knižnice napísané v jazyku c++.

- **Konvolúcia** – matematický operátor spracovávajúci dve funkcie. Je definovaný vzťahom:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\alpha) g(x - \alpha)_{d\alpha}$$

1.5 Referencie

[1] *"Decay time integrals in neutral meson mixing and their efficient evaluation"* - Till Moritz Karbach, Gerhard Raven, Manuel Schiller (CERN - Switzerland, NIKHEF - The Netherlands)

[2] *"An isolated line-shape model to go beyond the Voigt profile in spectroscopic databases and radiative transfer codes"* - N.H. Ngo, D. Lisak, H. Tran, J.-M. Hartmann

[3] *"Efficient computation of some speed-dependent isolated line profiles"* - H. Tran, N.H. Ngo, J.-M. Hartmann

2 Všeobecný popis

2.1 Perspektíva projektu

Projekt bude súčasťou väčšieho celku, ktorý má za úlohu analyzovať spektroskopické dáta. Optická Spektroskopia je oblasť fyziky, zaoberajúca sa štúdiom elektromagnetického žiarenia emitovaného alebo pohlteného vzorkou. Získané informácie sa dajú použiť buď kvalitatívne (charakteristika vnútornej štruktúry vzorky, poprípade prostredia kde sa nachádza), alebo kvantitatívne (určenie koncentrácie známej vzorky).

2.2 Funkcie produktu

Softvér by mal byť schopný v optimálnom čase aplikovať rôzne transformácie na vstupné hodnoty – spektrá a modelovať ich tvar použitím funkcií opisujúcich žiarenie (absorpciu) vzoriek. Medzi tieto funkcie patria: Lorentzova, Gaussova, Voigtova alebo Hartmann–Tran.

2.3 Charakteristika používateľov

Finálny produkt bude využívať oddelenie experimentálnej Fyziky FMFI UK. Z používateľského hľadiska bude produkt použiteľný iba pod vývojovým prostredím LabVIEW.

3 Špecifikácia požiadaviek

3.1 Načítanie vstupného súboru

Systém prečíta vstupný súbor a dáta poukladá do objektov v DLL a pripraví tak namerané dáta na ďalšie spracovanie. Načítavanie dát je už implementované v knižnici DLL, ktorú sme zdedili. LabView načítané dáta zobrazí vo formulároch a nijak ich nemení. Umožňuje tieto dáta len prezerať, prípadne zvoliť transformačnú funkciu, ktorou sa tieto hodnoty prepočítajú a zobrazí počiatočný stav hodnôt, ktoré sme načítali so súboru v jednom formulári a zmenené dáta zvolenou funkciou v druhom formulári.

3.2 Zobrazenie načítaných vstupov

3.2.1 Prostredie LabView ponúka dva formuláre súvisiace s načítanými dátami.

3.2.2 Názov prvého formulára je **Data IN** (takto je nazvaný v LabView), ktorý ponúka používateľovi zadať:

3.2.2.1 **Name** - názov datasetu (merania) napr. pondelok, utorok , …

3.2.2.2 Hodnotu **X-in**, kde si zvolíme z ktorého indexu chceme zobraziť X-ovú súradnicu

3.2.2.3 Hodnotu **Y-in**, kde si zvolíme z ktorého indexu chceme zobraziť Y-ovú súradnicu

3.2.2.4 Hodnotu **W-in**, kde si zvolíme z ktorého indexu chceme zobraziť W (chybu merania)

3.2.3 Názov druhého formulára je **Data PAR In** (názov v LabView), ktorý ponúka používateľovi zadať:

3.2.3.1 **Name** - názov datasetu (merania) napr. pondelok, utorok , …

3.2.3.2 **Func.names** - umožní používateľovi zvoliť funkciu, ktorou sa majú vstupné hodnoty upraviť.

3.3 Výstupy do LabView

3.3.1 Používateľ má k dispozícii v LabView ďalší formulár, v ktorom sa zobrazia transformované dáta. Keďže dáta sú uložené v poliach, tak sa zobrazujú iba z jedného indexu, ktorý môže používateľ meniť a tak si prezrieť postupne všetky hodnoty. Až keď používateľ zvolí funkciu a stlačí tlačidlo na transformovanie dát, tieto dáta vo formulári prepočíta podľa zvolenej funkcie a vypíše ich znovu do tohto formulára. Nasledujúci popis počíta s tým, že už bola zvolená funkcia a stlačené tlačidlo na transformáciu dát.

3.3.2 Názov formulára v LabView je **Data_OUT_Fast**, ktorý obsahuje kolónky:

3.3.2.1 **Name** - názov datasetu

3.3.2.2 Hodnotu **X_Out**, kde si zvolíme z ktorého indexu chceme zobraziť X-ovú súradnicu. Táto súradnica je už zmenená zvolenou funkciou.

3.3.2.3 Hodnotu **Y_Out**, kde si zvolíme z ktorého indexu chceme zobraziť Y-ovú súradnicu. Táto súradnica je už zmenená zvolenou funkciou.

3.3.2.4 Hodnotu **W_Out**, kde si zvolíme z ktorého indexu chceme zobraziť W (chybu merania). Táto chyba merania je stále rovnaká. Teda ju DLL nebude nijako meniť.

4 Komunikácia s DLL

DLL bude priamo komunikovať s LabView len cez funkciu fast(), ktorá načíta vstupný súbor. Dáta poukladá do štruktúr a nezmenené ich zobrazí v LabView. Používateľ bude môcť zvoliť funkciu, ktorou bude chcieť dáta transformovať/zmeniť. Ďalej sa budú dáta spracovávať podľa zvolenej funkcie.

5 Funkcie

Funkcie, ktoré si bude môcť používateľ zvoliť sú:

- transformácia x-ovej osi **xt()**,
- transformácia y-ovej osi **yt()**,
- **gauss()**,
- **lorentz()**,
- **voigt()**

Tieto všetky funkcie budú vždy počítať len s jednou hodnotou x, y, w, ktoré máme načítané v štruktúre. Ďalšie parametre, ktoré sú potrebné na výpočet funkcie sú načítané zo súboru a uložené do poľa, obsahujúceho štruktúry Parameters. Štruktúra obsahuje parametre pre všetky funkcie, ale vyplnené sú len tie, ktoré daná funkcia potrebuje (potrebné parametre sú popísané v časti 6. pre jednotlivé funkcie). Výsledné hodnoty X a Y sa zapíšu do nového dvojrozmerného poľa. Výpočet sa vykoná pre každú X-ovú a Y-ovú hodnotu, ktoré vstupujú do funkcií ako vektor.

6 Popis jednotlivých funkcií

Popis funkcií, ktoré si môže používateľ zvoliť. Nie sú tu popísané funkcie **xt()** a **yt()**, pretože už sú implementované v zdedenom DLL.

6.1 Gauss

Gauss() bude počítať postupne s každou hodnotou X, Y. Tieto hodnoty tvoria vektor v, ktorý vstupuje do funkcie. Parametre pre Γ_D, v_0, $\Delta_0$načíta so svojej sady parametrov. Výsledkom bude vektor so zmenenými hodnotami X a Y, ktoré budú uložené do poľa výsledkov na rovnakom indexe, ako boli pôvodné hodnoty X, Y. Definičný obor je od najmenšie načítaného X, po najvyššie načítané X.

$$P(v) = \frac{1}{|\Gamma_D|}\sqrt{\frac{\ln(2)}{\pi}}\,\exp\{-ln2 * \frac{[v-(v_0+\Delta_0)]^2}{{\Gamma_D}^2}\}$$

6.2 Lorentz

Lorentz() bude počítať postupne s každou hodnotou X, Y. Tieto hodnoty tvoria vektor v, ktorý vstupuje do funkcie. Parametre pre Γ_0, v_0, Δ_0 načíta so svojej sady parametrov. Výsledkom bude vektor so zmenenými hodnotami X a Y, ktoré budú uložené do poľa výsledkov na rovnakom indexe, ako boli pôvodné hodnoty X, Y. Definičný obor je od najmenšie načítaného X, po najvyššie načítané X.

$$P(v - v_0) = \frac{1}{\pi}\frac{|\Gamma_0|}{[(v-(v_0+\Delta_0)]^2+{\Gamma^2}_0}$$

6.3 Voigt

Voigt() bude počítať postupne s každou hodnotou X, Y. Tieto hodnoty tvoria vektor v, ktorý vstupuje do funkcie. Parametre pre Γ_D, v_0, w0, $\Delta_0$načíta so svojej sady parametrov. Výsledkom bude vektor so zmenenými hodnotami X a Y, ktoré budú uložené do poľa výsledkov na rovnakom indexe, ako boli pôvodné hodnoty X, Y. Definičný obor je od najmenšie načítaného X, po najvyššie načítané X.

$$P(v) = \frac{1}{|\Gamma_D|}\sqrt{\frac{\ln(2)}{\pi}} * \mathrm{Re}[faddeeva(z)],$$

$$z = \ln 2\,\frac{\Gamma_0 + i(v_0 + \Delta_0 - x)}{\Gamma_D}$$

7 Návrh

7.1 Špecifikácia vonkajších interfejsov

DLL knižnica vyvíjaná v tomto projekte bude komunikovať s prostredím LabView. Používateľ načíta dáta z grafického rozhrania v programe LabView a nastaví vstupné parametre v tabuľke, ktoré bude vedieť DLL prečítať, spracovať a výsledok ponúknuť používateľovi.

7.2 Vstupný interface

1. Popis vstupného formuláru, Spektrum (obr.1):

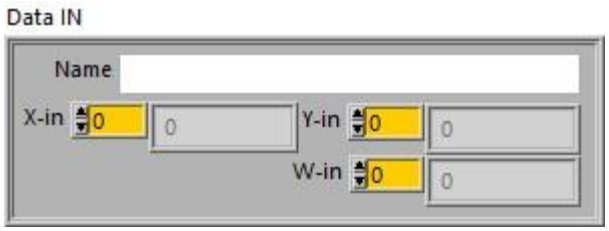
   1.1. Polia:

   Name: meno dát

   X-in: x-ová súradnica

   Y-in: y-ová súradnica

   W-in: neistota



Obrázok 1: Vstupný formulár

2. Popis vstupného formuláru, Parametre (obr. 2)

   2.1. Polia:

   Param.strings: charakterizujúce mená parametrov a ich vlastnosti (meno, model, nezávislé parametre)

   Param.values: charakterizujúce hodnoty parametrov (hodnota, neistota, škála pre GUI)

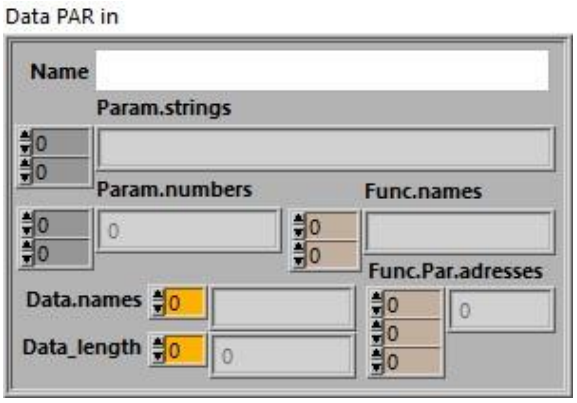
   Func.names: (L je počet transformácií a funkcií modelu – max

6: XT, YT, BL, PK, RF, MC, ktoré treba vypočítať), prvý stĺpec obsahuje meno funkcie (XT, YT, BL, PK, RF, MC), druhý mená „skupín=groups“ v rámci danej funkcie zoradených do jedného stringu: Menno1@Meno2@...@MenoN

Func.par.adresses: 3xLxQ (Q je max počet skupín nachádzajúci sa niektorej z funkcií), prvý inde(page) definuje funkciu podľa poradia v poli Func.names následne každá skupina má jeden riadok, kde prvá hodnota hovorí koľko hodnôt je v danom riadku (adries – poradie v Param.strings a Param.values), poradie adries parametrov je pevne definované pre každú funkciu-skupinu

Data.names: Mená vektorov pred simulovaných dát pre funkciu REF

Data.length: Zodpovedajúca dĺžka(počet bodov) pre vektory pred simulovaných dát (uložené v binárnom súbore na disku)



Obrázok 2: Štruktúra vstupných dát

7.3 Výstupný interface

1. Popis výstupného formuláru, Spektrum (obr. 3):

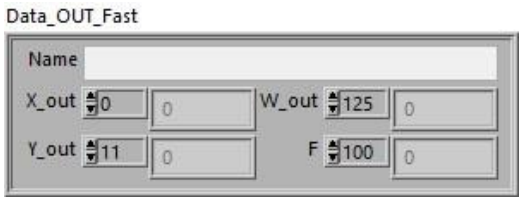
    1. Polia:

        Name: meno dát

        X-in: x-ová súradnica

        Y-in: y-ová súradnica

        W-in: neistota



Obrázok 3: Štruktúra výstupných dát

8 Implementácia

Z LabView je z DLL volaná funkcia fdata_fast(), ktorú začneme upravovať ako prvú. Do jej tela zimplementujeme volanie funkcií na výpočet dát, ktoré sú popísané v katalógu požiadaviek. Volaná metóda má vstupné parametre pointre na 4 štruktúry (TD1 *DataPARin, TD7 *DataIN, TDFast *DataOUT_F, TD10 *Error) a výstupom je typ <int32_t> s ktorým ale nič nerobíme ani neupravujeme, takže ho ani neriešime. Všetky hodnoty s ktorými pracujeme sú iba v štruktúrach. Štruktúry sú pevne definované a nemožno ich meniť. Zavolaním tejto funkcie sa dáta vypočítajú, upravia a pošlú do štruktúry TDFast *DataOUT_F v ktorej zostanú uložené. Po zavolaní metódy sa vypočítané dáta vykreslia v tabuľke Data_OUT_Fast. Funkcie a metódy pre výpočet dát budú dopísané do triedy Transform. Funkcia fdata_fast() už od staršej verzii programu vytvára inštanciu tohto objektu kde sú statické funkcie a metódy takže do jej tela treba implementovať volanie našich metód pre výpočet dát.

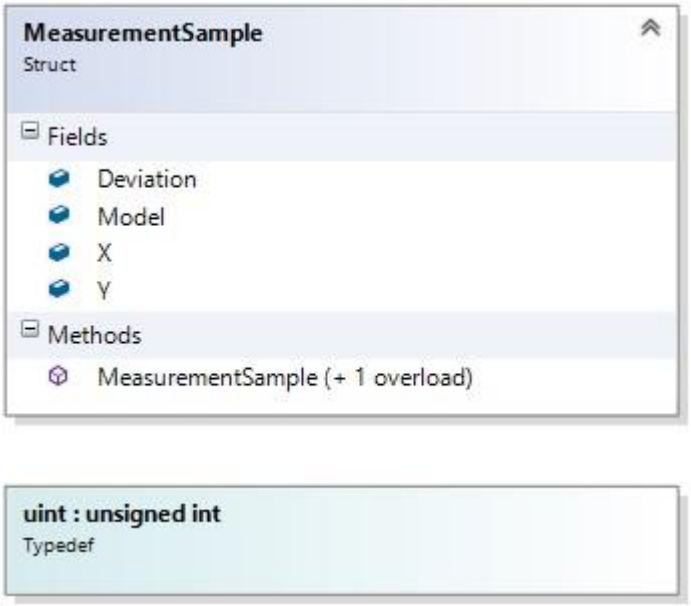
Každá funkcia bude mať vlastnú triedu. Tie vytvoríme v súbore PeakFunkcions.h. Každá funkcia v tomto súbore bude obsahovať inicializáciu. Inicializácia bude funkcia, ktorá do členskej premennej uloží vstupný parameter typu double. Najdôležitejšia funkcia v rámci template bude funkcia Apply, ktorej vstupné parametre budú **in** a **out** a budú typu double. Táto funkcia bude modifikovať vstupné dáta **in**, podľa matematického vzorca popísaného pre každú matematickú funkciu. Prepočítané dáta uloží do **out.**

Súbor PeakFunctions.h bude obsahovať templates pre funkcie, ktoré máme vytvoriť podľa katalógu požiadaviek. Teda tam pribudne template

LorentzFunction. Táto template bude dediť z triedy IFunction. Teda jej doprogramujeme metódu Initialize a Apply. Metóda Initialize dostane ako vstupný parameter double *parameters. Je to smerník na parameter pre funkciu. Tento smerník si uloží do členskej premennej. Metóda Apply bude obsahovať výpočet matematickej funkcie $P(v - v_0) = \frac{1}{\pi} \frac{|\Gamma_0|}{[(v-(v_0+\Delta_0)]^2+\Gamma^2{}_0}$, kde v je vstupný parameter *in*, v0, t0, delta0 sú parametre z členskej premennej mParams. Potom je výpočet: (1/M_PI) * (abs(t0)/v-pow((v0+delta0), 2) + pow(t0, 2)), ktorý uložíme do parametra metódy *out*.

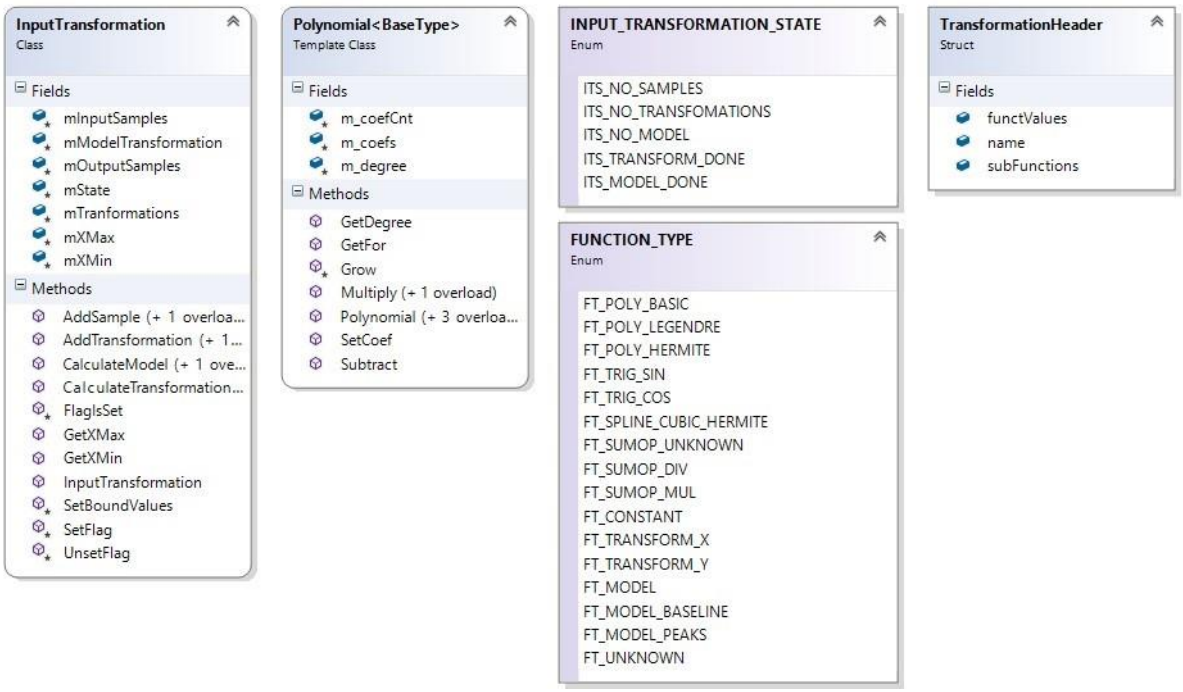
Ako ďalšou template v tomto súbore bude GaussFunction. Bude obsahovať metódu Inicialize, ktorej vstupným parametrom bude smerník na parametre ktoré si uloží do členskej premennej. Druhou metódou bude metóda Apply. Telo metódy Apply bude obsahovať výpočet matematickej funkcie $P(v) = \frac{1}{|\Gamma_D|}\sqrt{\frac{\ln(2)}{\pi}}\exp\{-ln2 * \frac{[v-(v_0+\Delta_0)]^2}{{\Gamma_D}^2}\}$, kde v je vstupný parameter *in*, v0, delta0 a t0 sú parametre uložené v členskej premennej mParams. Potom je vpočet: `(-log(2) * ((v-pow((v0 + delta0), 2)) / pow(t0, 2)))` * (1/t0) * (log(2)/ M_PI) Tento výpočet uložíme do premennej *out*, ktorá je vstupným parametrom do tejto metódy Apply.

Ďalej bude obsahovať template VoightFunction. Bude mať definovanú metódu Inicialize. Táto metóda bude mať jeden parameter mParams. Bude to smerník na parametre pre výpočet matematickej funkcie voight. Tento smerník uloží do členskej premennej. Ďalej bude obsahovať druhú metódu Apply. Tá vo svojom tele vypočíta podľa matematického vzorca dáta a uloží ich do premennej *out*. Táto premenná vstupuje do metódy Apply. Matematický vzorec pre Voight: $P(v) = \frac{1}{|\Gamma_D|}\sqrt{\frac{\ln(2)}{\pi}} * \mathrm{Re}[\mathrm{faddeeva(z)}],\ \mathrm{z} = \ln 2\ \frac{w0+i(v_0+\Delta_0-v)}{\Gamma_D}$, kde v je vstupný parameter *in* do metódy Apply, w0, td, v0, delta0 sú parametre uložené v členskej premennej mParams. Potom je výpočet nasledovný: `z = log(2) * (w0 + i*(v0 + delta0 - v)) / td; first = 1 / abs(td); second = sqrt(log(2) / M_PI); std::complex<double> Mycerf = Cerf::faddeeva(i*z);` Výsledok, ktorý potom zapíšeme na výstup je `first * second * Mycerf.real`.



Obr. 4. Štruktúra MeasurementSample

MeasurementSample je štruktúra definovaná v časti DataAnalysis súbor *CommonTypes.h*, slúži na uchovanie hodnôt pre každý bod. Každý bod načítaný zo vstupného súboru je objekt tejto štruktúry. Členské premenné X, Y sú pre x-ovú a y-ovú súradnicu, Deviation uchováva chybu merania. V LabView je táto chyba merania označená „W“ a Model je výsledkom prepočítanej funkcie. Ak nie je zvolená žiadna funkcia, tento model je 0 a na výstupnom grafe sa nič nevykreslí. Jej konštruktor naplní tieto členské premenné. Konštruktor je preťažený, teda ho vieme použiť v našom prípade dvoma spôsobmi. Ak nepoužijeme žiadne vstupné údaje, premenné sa nastavia na nulu. Toto sa použije pri inicializácii výstupu. Pri načítaní vstupného súboru potrebujeme naplniť tieto premenné už známymi hodnotami. Použije sa konštruktor s troma vstupnými údajmi (double X, double Y, double Deviation). Model zostáva na 0 a pre vstup sa nezmení.



Obr. 5 diagram triedy InputTransformation, Polynomial, enumeračné typy a TransformationHeader

TransformationHeader je štruktúra, ktorá uchováva informáciu o transformácii. Má členské premenné *Buffer<Buffer<double>> functValues*, *Bufer<string> subFunctions* a *string name*.

InputTransformation je trieda, ktorá sa stará o načítanie vstupných dát, ich transformáciu a prepočet podľa matematických funkcií. Metódy v tejto triede ako jedným z parametrov dostávajú smerník na výstup. K tomuto výstupu má prístup LabView a číta z neho dáta. Preto má tento výstup pevne alokované miesto v pamäti už od začiatku, aby s LabView vedel komunikovať.

Členské premenné:

```
vector< MeasurementSample > mInputSamples;
   je vektor, ktorý obsahuje objekty typu MeasurementSample. Sú tu uložené
   všetky vstupné dáta. (x-ová, y-ová suradnica a chyba merania)
vector< MeasurementSample > mOutputSamples;
   je vektor, ktorý obsahuje objekty typu MeasurementSample. Tu sa budú
   ukladať prepočítané výstupné dáta.
vector< shared_ptr< IFunction<MeasurementSample> > > mTranformations;
   je vektor, ktorý obsahuje všetky transformácie, ktoré sa môžu vykonať podľa
   toho, čo sa zvolí v prostredí LabView
vector< shared_ptr< IFunction<MeasurementSample> > > mModelTransformation;
    je vektor, ktorý obsahuje všetky modely, ktoré sa môžu vykonať podľa toho,
    čo sa zvolí v prostredí LabView. Model je prepočet súradníc poľa
    matematickej funkcie.
double mXMax;
double mXMin;
int mState;
```

CalculateModel alokuje miesto v pamäti pre výstup. Z toho miesta bude LabView čítať dáta a vykresľovať na graf.

AddSample je preťažená metóda. Na vstupe môže byť Objekt typu MeasurementSample. Ten pridá do vektora mInputSamples. Alebo dostaneme na vstupe tri konštanty (double x, double y, double uncertainty), kde x, y sú súradnice bodu a uncertainty je chyba merania (v LabView označená ako W). Z týchto hodnôt vytvorí objekt typu MeasurementSample a pridá ho do vektroa mInputSamples.

AddTransformation vstupný parameter je typu *TransforamtionHeader*. Na základe tohoto parametra vytvorí a pripraví konkrétny objekt typu *IFunction*. Podľa typu zo vstupného parametra uloží tento objekt do vektora *mTransformation* alebo *mModelTransformation*. Ak je meno v *TransformationHeadr* `FT_MODEL_BASELINE` alebo `FT_MODEL_PEAKS` uloží funkciu do *mModelTransformation*. Ak je meno `FT_TRANSFORM_X` alebo `FT_TRANSFORM_Y` funkcia bude uložená do vektora *mTransformation.*

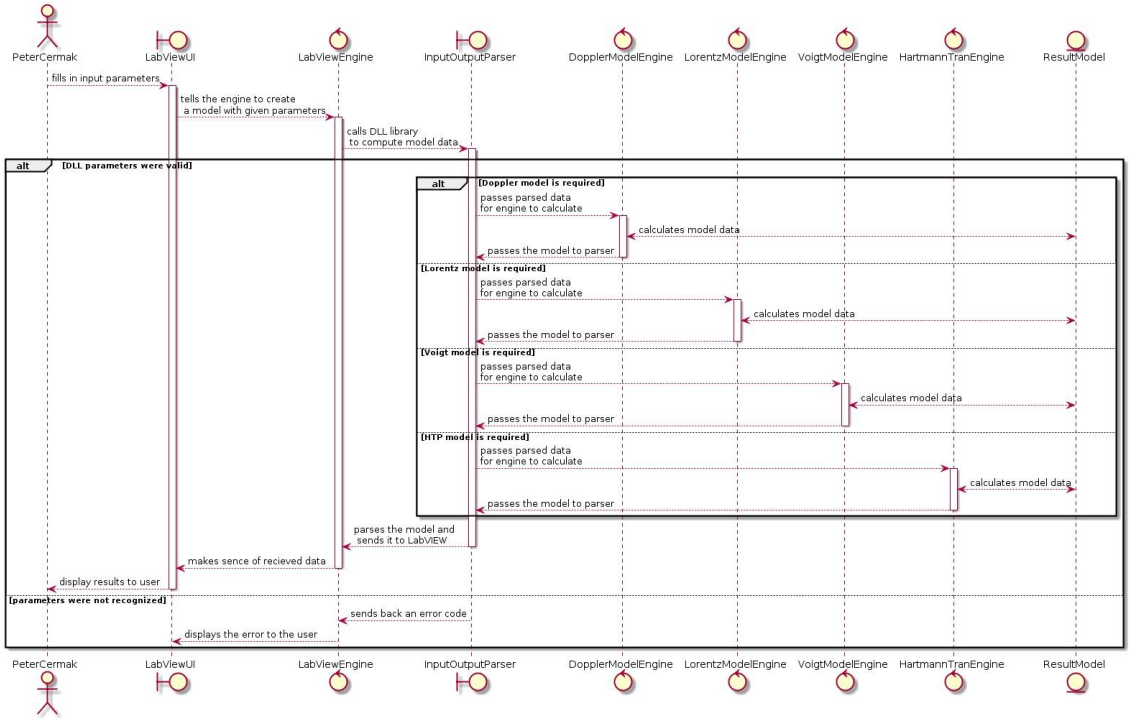
CalculateTransformation vstupný parameter je `Buffer<MeasurementSample> &output.` Tento parameter je smerník na výstupný buffer, z ktorého LabView číta dáta a vykresľuje podľa nich graf. Táto metóda naplní *output* dátami. Každú transformáciu uloženú vo vektore *mTransformations* a pre každý bod, ktorý má uložený v členskej premennej *mInputSamples* zavolá funkciu Apply na prepočet týchto hodnôt. Transformácia je v tomto zmysle objekt, ktorý bol inicializovaný a má definovanú vlastnú funkciu Apply, podľa typu transformácie.

CalculateModel vstupný parameter je `Buffer<MeasurementSample> &output.` Tento parameter je smerník na výstupný buffer, z ktorého LabView číta dáta a vykresľuje podľa nich graf. Táto metóda naplní *output* dátami. Každý model uložený vo vektore *mModelTransformation* a pre každý bod uložený v členskej premennej mInputSamples zavolá funkciu Apply na prepočet hodnôt zo vstupu. Model je v tomto zmysle objekt nejakej IFunkction presnejšie definovanej (je to napríklad BL_FUNCTION, PK_FUNCTION, ...). Teda má definovanú vlastnú metódu Apply, podľa ktorej majú byť dáta prepočítané.

9 Diagramy

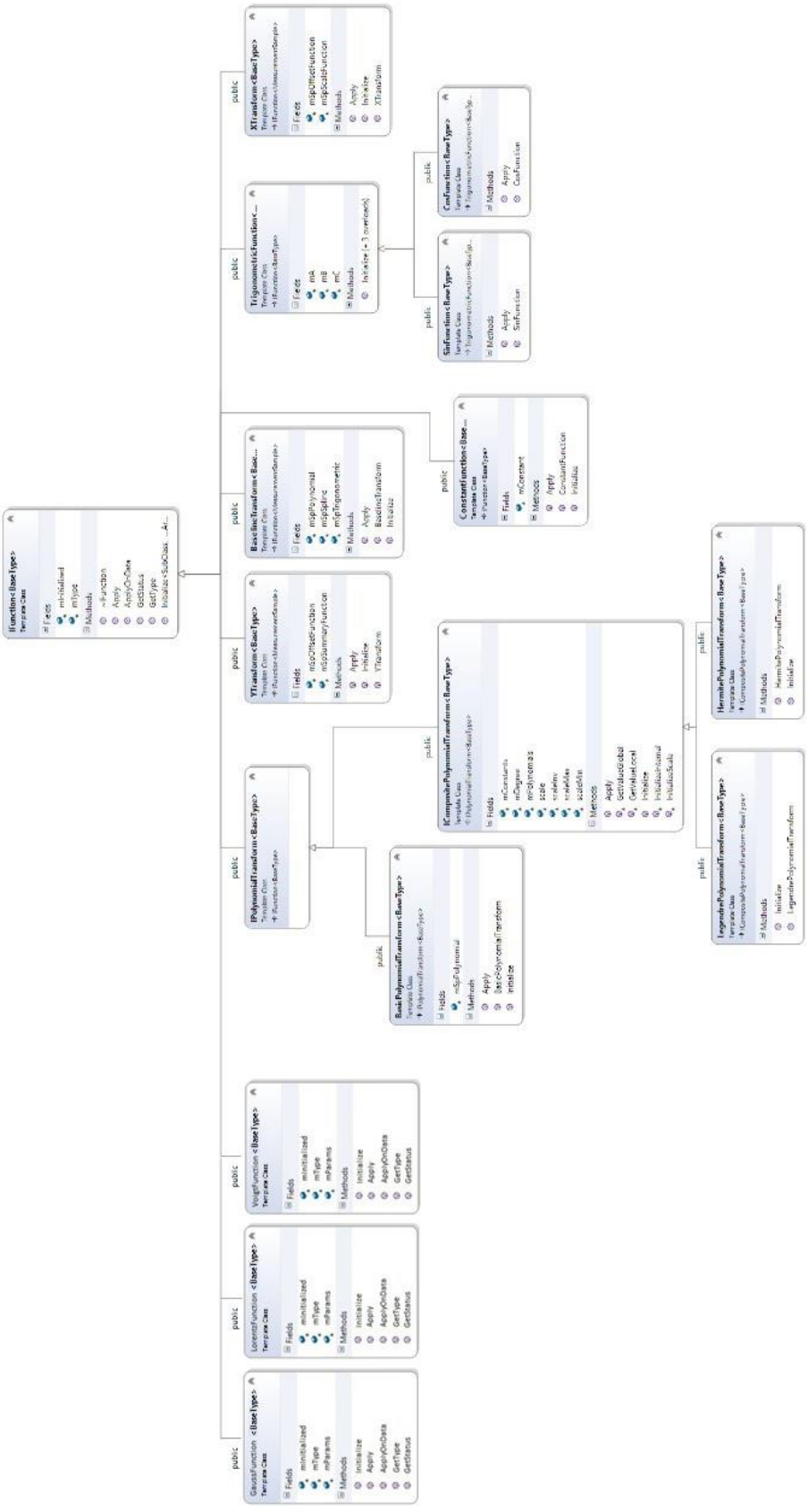
9.1 Sekvenčný diagram

Tento sekvenčný diagram detailnejšie popisuje fungovanie vyššie uvedeného použitia. Zameriava sa na interakciu medzi LabVIEW programom a DLL knižnicou, ktorá obsahuje moduly na spracovanie vstupu, transformáciu funkciami a vyprodukovanie modelov.



Obr. 6 Sekvenčný diagram

9.2 Triedny diagram



Obr. 7 triedny diagram

IFunction je základná trieda od ktorej sú odvodené všetky funkcie, ktoré sa v tejto knižnicu počítajú. Je definovaná v súbore *TransformationsLibPrivate.h*. Členské premenné sú:

mInitialized – je typu boolean. Je True ak prebehla inicializácia

mType – je typu string. Predstavuje meno funkcie. Napr. FT_TRANSFORM_X, PK_FUNKCTION, BL_FUNKCTION, ...

Obsahuje metódy:

`GetStatus()`, vráti hodnotu členskej premennej typu *boolean* podľa toho, či bola funkcia inicializovaná. Ak áno vráti True, inak False.

`GetType()`, ktorá vráti názov funkcie. Napr. (`FT_TRANSFORM_X, FT_TRANSFORM_Y`)

```
ApplyOnData( __in const size_t count, __in_ecount( count ) const BaseType
*pIn, __out_ecount( count ) BaseType *pOut )
```
, ktorá na vstupe dostáva počet bodov, ktoré má prepočítať a smerník na vstupné dáta a smerník na miesto v pamäti, kam sa majú prepočítané dáta uložiť. Pre každý objekt v pIn, sa prepočítajú hodnoty a ich výsledok sa uloží do premennej pOut. Tento prepočet sa vykoná pomocou abstraktnej triedy *Apply*.

Ďalšie metódy sú abstraktné a pre každú transformáciu alebo matematickú funkciu je ich telo rozdielne. Tieto metódy sú:

`Initialize( ArgTypes& ... args )` – vstupné parametre si uloží do členských premenných. Tieto premenné má každá podtrieda vlastné.

`Apply ( __in const BaseType &in, __out BaseType &out )` – vstupnými parametrami sú smerník na vstupné dáta a smerník na výstupné dáta. V rámci tejto metódy sa majú vstupné dáta z *in* prepočítať a uložiť do *out*.

GaussFunction je trieda, ktorá detí z triedy IFunction. Navyše má implementované metódy Initialize() a Apply(), ktoré sú v IFunction triede virtuálne.

- Metóda Initialize() dostane ako vstupný parameter smerník na parametre pre túto funkciu. Nastaví členskú premennú mInitialize na True.
- Metóda Apply() má vstupné parametre: *in* smerník na vstupné dáta a *out* smerník na výstupné dáta, kam sa ukladajú transformované dáta. Táto metóda podľa vzorca (popísaného v kapitole 8) prepočíta vstupné dáta a uloží ich do premennej *out*.

LorentzFunction je trieda, ktorá detí z triedy IFunction. Navyše má implementované metódy Initialize() a Apply(), ktoré sú v IFunction triede virtuálne.

- Metóda Initialize() dostane ako vstupný parameter smerník na parametre pre túto funkciu. Nastaví členskú premennú mInitialize na True.
- Metóda Apply() má vstupné parametre: *in* smerník na vstupné dáta a *out* smerník na výstupné dáta, kam sa ukladajú transformované dáta. Táto metóda podľa vzorca (popísaného v kapitole 8) prepočíta vstupné dáta a uloží ich do premennej *out*.

VoigtFunction je trieda, ktorá detí z triedy IFunction. Navyše má implementované metódy Initialize() a Apply(), ktoré sú v IFunction triede virtuálne.

- Metóda Initialize() dostane ako vstupný parameter smerník na parametre pre túto funkciu. Nastaví členskú premennú mInitialize na True.
- Metóda Apply() má vstupné parametre: *in* smerník na vstupné dáta a *out* smerník na výstupné dáta, kam sa ukladajú transformované dáta. Táto metóda podľa vzorca (popísaného v kapitole 8) prepočíta vstupné dáta a uloží ich do premennej *out.*

YTransform je trieda, ktorá detí z triedy IFunction. Navyše má implementované metódy Initialize() a Apply(), ktoré sú v IFunction triede virtuálne.

- Metóda Initialize() má vstupné parametre spOffsetFunction a spSummaryFunction. Tieto Parametre majú typ IFunction. V inicializácii sú uložené do členských premenných.
- Metóda Apply() má vstupné parametre: *in* smerník na vstupné dáta a *out* smerník na výstupné dáta, kam sa ukladajú transformované dáta. Táto metóda podľa vzorca prepočíta vstupné dáta a uloží ich do premennej *out.*

XTransform je trieda, ktorá detí z triedy IFunction. Navyše má implementované metódy Initialize() a Apply(), ktoré sú v IFunction triede virtuálne.

- Metóda Initialize() má vstupné parametre spOffsetFunction a spScaleFunction. Tieto Parametre majú typ IFunction. V inicializácii sú uložené do členských premenných.
- Metóda Apply() má vstupné parametre: *in* smerník na vstupné dáta a *out* smerník na výstupné dáta, kam sa ukladajú transformované dáta. Táto metóda podľa vzorca prepočíta vstupné dáta a uloží ich do premennej *out.*

BaseLineTransform je trieda, ktorá detí z triedy IFunction. Navyše má implementované metódy Initialize() a Apply(), ktoré sú v IFunction triede virtuálne.

- Metóda Initialize() má vstupné parametre spPoly, spTrig, spSpline.
  Tieto Parametre majú typ IFunction. V inicializácii sú uložené do členských premenných. Nastaví členskú premennú mInitialize na True.
- Metóda Apply() má vstupné parametre: *in* smerník na vstupné dáta a *out* smerník na výstupné dáta, kam sa ukladajú transformované dáta. Táto metóda podľa vzorca prepočíta vstupné dáta a uloží ich do premennej *out.*

ConstantFunction je trieda, ktorá detí z triedy IFunction. Navyše má implementované metódy Initialize() a Apply(), ktoré sú v IFunction triede virtuálne.

- Metóda Initialize() má vstupný parameter cnst typu double. Túto hodnotu uloží do členskej premennej mConstant. Nastaví členskú premennú mInitialize na True.
- Metóda Apply() má vstupné parametre: *in* smerník na vstupné dáta a *out* smerník na výstupné dáta, kam sa ukladajú transformované dáta. Táto metóda keďže je konštantná, tak na výstup uloží na každé miesto v *out* hodnotu uloženú v premennej mConstant.

TrigonometricFunction je trieda, ktorá detí z triedy IFunction. Navyše má implementované metódy Initialize ktorá je v IFunction triede virtuálna.

- Členské premenné tejto triedy sú mA, mB, mC. Použijú sa do výpočtu vo vzorci tejto funkcie.
- Metóda Initialize() má preťažený konštruktor. Teda môže byť inicializovaná so vstupnými parametrami:
    - double a, double b, double c; hodnoty týchto premenných uloží do vlastných členských premenných
    - double *pParams; smerník na parametre pre trigonometrickú funkciu. Zároveň môžu byť iba tri parametre. Budú uložené do členských premenných
    - buffer parametrov obsahujúci tri parametre, ktoré budú uložené do členských premenných
    - Vektor parametrov, ktorý uloží prvé tri hodnoty do členských premenných.

  Nastaví členskú premennú mInitialize na True.

SinFunction je trieda, ktorá detí z triedy TrigonometricFunction. Má dodefinovanú funkciu Apply(). má vstupné parametre: *in* smerník na vstupné dáta a *out* smerník na výstupné dáta, kam sa ukladajú transformované dáta. Táto metóda podľa vzorca na výpočet sínusu prepočíta vstupné dáta a uloží ich do premennej *out.*

CosFunction je trieda, ktorá detí z triedy TrigonometricFunction. Má dodefinovanú funkciu Apply(). má vstupné parametre: *in* smerník na vstupné dáta a *out* smerník na výstupné dáta, kam sa ukladajú transformované dáta. Táto metóda podľa vzorca na výpočet cosínusu prepočíta vstupné dáta a uloží ich do premennej *out.*

10 Testovací scenár

- Vstup:

    1. Vyplnenie vstupných parametrov v grafickom prostredí LabView
    2. Zvolí sa funkcia, ktorou majú byť vstupné dáta spracované
    3. Spustíme proces vypočítania dát z prostredia LabView stlačením tlačidla

- Výstup:

    1. Kontrola výstupných parametrov naplnených v tabuľke data_out_fast
    2. Kontrola požadovaného grafického modelu
    3. Kontrola grafického modelu po zvolení údajov do transformačných funkcií

11 Používateľská príručka

11.1 Ovládanie aplikácie

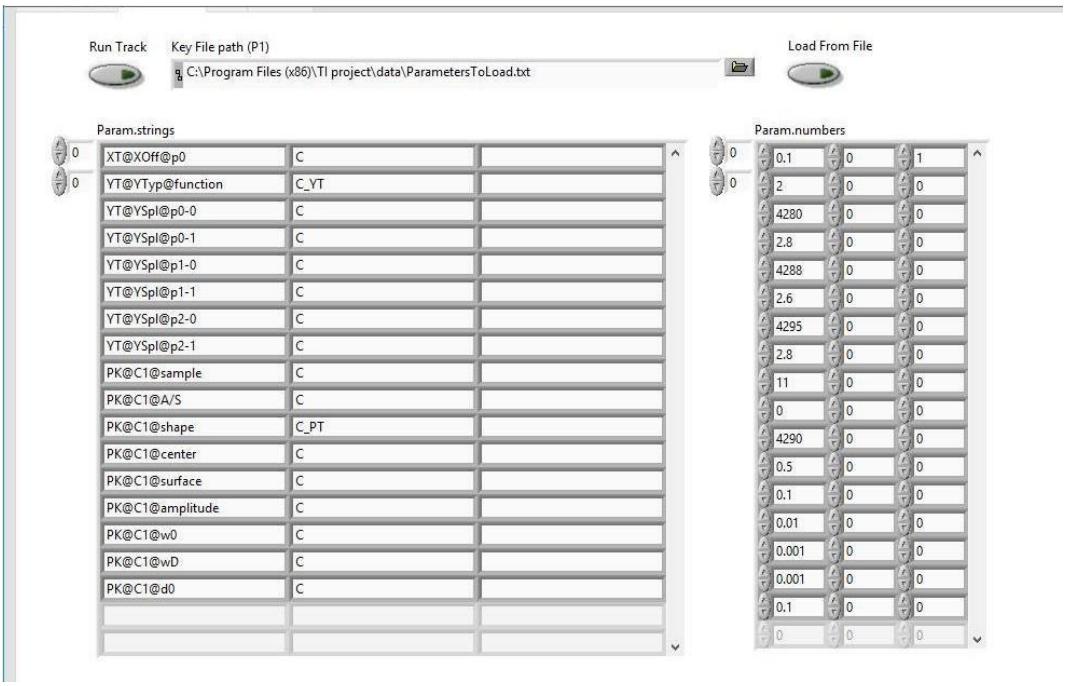
Naša aplikácia je DLL knižnica, ktorá poskytuje výpočtovú funkcionalitu. Je úzko spätá so softvérom LabVIEW, ktorý poskytuje aj užívateľské rozhranie na používanie tejto knižnice. Samozrejme, ako každú inú knižnicu, aj túto knižnicu je možné použiť mimo LabVIEW. Takéto využitie nie je očakávané a vyžadovalo by vytvorenie špecifických vstupných parametrov, ktoré LabVIEW generuje z užívateľského vstupu. Preto je z takéhoto hľadiska pre používateľa nepraktické.

11.2 Načítanie DLL knižnice

Závisí od konkrétnej konfigurácie LabVIEW aplikácie. V našom konkrétnom prípade sa musí DLL knižnica volať **Win32Project_AIstart** a musí byť umiestnená v adresári **data**.

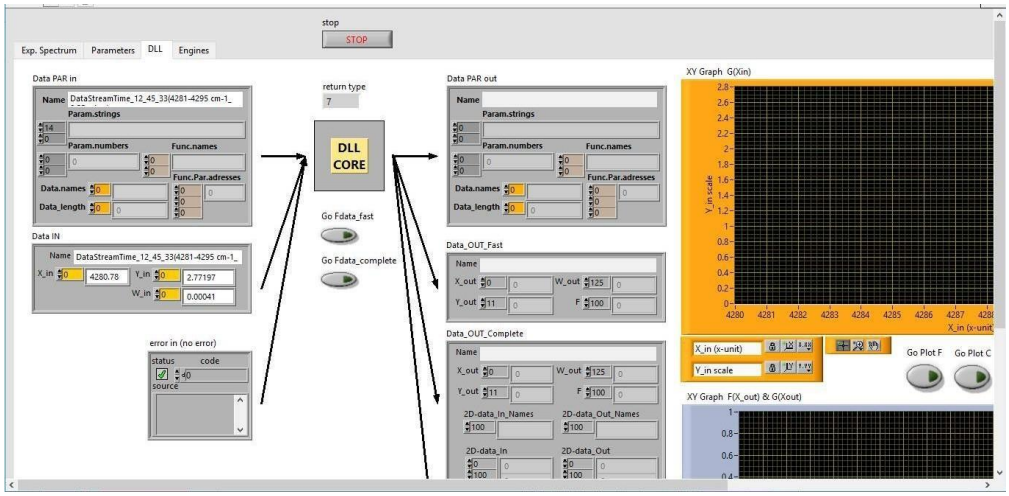
11.3 Zadanie vstupných parametrov

Formát vstupu taktiež závisí od konkrétnej aplikácie. Na ukážke nášho prípadu sú k dispozícii dva spôsoby a to vyplnenie formulára, alebo import zo súboru.



11.4 Analýza výstupu

Tlačidlami “Go_fdata_fast” a “Go_fdata_complete” sa púšťa výpočet so zadanými vstupnými parametrami. V poli “return” type sa zobrazuje výstupný kód a dáta, ktoré poskytne DLL knižnica sa použijú na vykreslenie grafu v pravej časti.



12 Záver

Knižnica DLL obsahuje ďalšie možnosti transformácie dát, podľa požiadaviek zadávateľa projektu. Umožňuje používateľovi zvoliť v prostredí LabView transformáciu vstupných dát pomocou funkcií: Gauss, Lorentz, Voigt. Používateľ môže zvoliť jednu z týchto funkcií. Knižnica podľa zvolenej funkcie dáta prepočíta a uloží ich na výstup. Prepočítané dáta sa zobrazia na grafe v LabView. Tieto zmenené dáta sú výstupom DLL knižnice, ktorá komunikuje s LabView.