STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 1: Matematika a statistika

CSE-Lab

Kamil Mudruňka
Pardubický kraj

Pardubice 2018

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 1: Matematika a statistika

CSE-Lab

CSE-Lab

Autoři: Kamil Mudruňka
Škola: Gymnázium, Dašická 1830, Pardubice, Dašická 1083, Pardubice
Kraj: Pardubický kraj
Konzultant: -

Pardubice 2017

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Pardubice dne 16.3.2018 ……

Poděkování

Za podporu v práci bych rád poděkoval všem svým učitelům, především pak Mgr. Soně Křišťanové, která koordinuje SOČ a řadu dalších vědecko-technických soutěží na naší škole. V neposlední řadě děkuji také svým rodičům a přátelům, kteří mě po celou dobu v práci velmi motivovali a byli oporou.

Anotace

Cílem práce je vytvořit softwarový systém, který v sobě bude efektivním způsobem kombinovat nástroj pro měření a sběr dat na počítači s matematickým softwarem (Computer Algebra System), který bude možné použít pro jejich přímé zpracování či samostatné výpočty a řešení úloh. Software disponuje přehledným grafickým rozhraním, jehož ovládání zvládne i běžný středoškolák (bez nutnosti např. ovládat programovací jazyk). Jeho funkce v současnosti pokrývají celou středoškolskou matematiku a vybrané obory vyšší matematiky.

Klíčová slova

Počítačový algebraický systém; měření; symbolické výpočty; numerické výpočty; zpracování dat

Annotation

The goal of this project is to create a software system which combines a measurement tool for taking measurements and collecting data from sensors with a computer algebra system that can be used for direct processing of the data and/or individual calculations and problem solving. The software system offers a user friendly GUI simple enought so that even ordinary high school students can work with it (without for example having to learn a programming language). At the moment its functions cover al topics of high school math a some chosen university level math topics.

Keywords

Computer algebra system; measurement; symbolic computation; numerical computation; data processing

Obsah

ÚVOD

CSE-Lab je software napsaný v C# zaměřený na studenty, učitele a celkově všechny zájemce o matematiku, fyziku a blízké obory. Software v jednom celku kombinuje nástroj pro měření (vzorkování signálu z analogových senzorů a jeho zpracování) a matematický software, který může být použit pro okamžité vyhodnocení naměřených dat a jejich názornému srovnání s teorií, či samostatné výpočty, simulace a řešení úloh. Součástí práce je i tvorba měřicího hardwaru s USB připojením, který zajišťuje vzorkování signálu a komunikaci s PC. CSE-Lab představuje řádově levnější alternativu ke komerčním systémům stejného zaměření.

1 MĚŘICÍ SYSTÉM

Samotný měřicí systém se sestává z grafického rozhraní přímo spojeného s matematickým softwarem a hardwarového modulu po připojení senzorů s USB rozhraním.

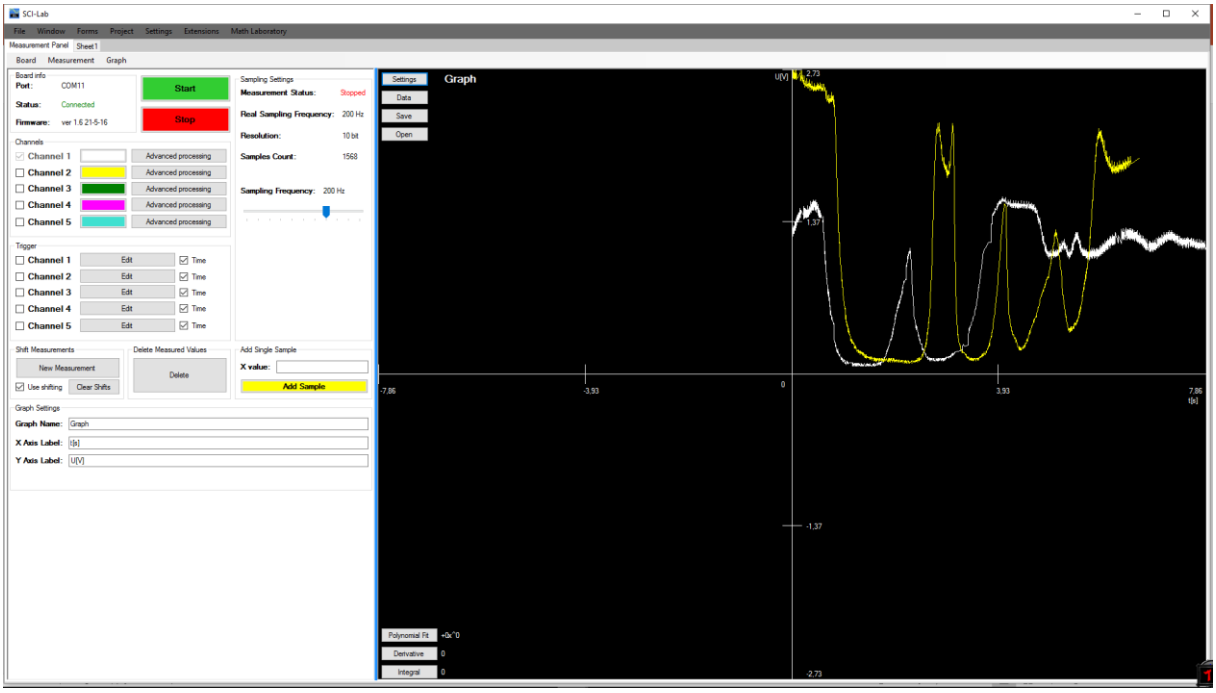
1.1 Hardware

Hardware je založený na mikrokontroléru ATMEGA328P-PU naprogramovaném v Arduino IDE a umožňuje připojení 1-2 senzorů při vzorkovací frekvenci až 2 kHz nebo 1-5 senzorů při 400 Hz. Rozlišení převodníku je 10 bitů. Je možné připojit libovolné analogové senzory s výstupem v rozsahu 0-5 V. Kromě toho je systém osazen indikačními LED diodami, USB portem a bluetooth modulem pro bezdrátovou komunikaci. Napájení je zajištěno přes USB, v případě bezdrátové operace přes napájecí konektor. V současnosti je hotový i první prototyp vylepšené verze hardwarové desky, který navíc disponuje i dotykovou obrazovkou, přes kterou je možné měření řídit a měnit jeho parametry bez neustálého odebíhání k počítači.

Obr. 1 - Měřicí hardware

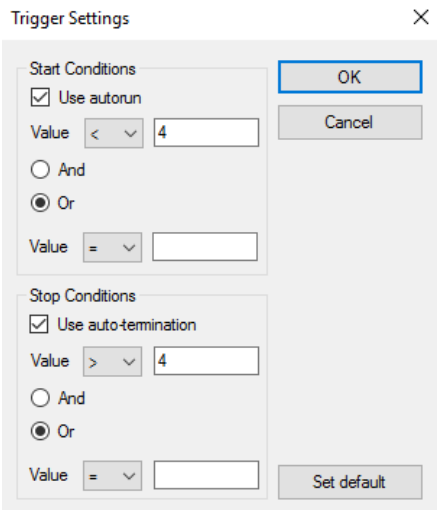
1.2 Softwarové rozhraní

Softwarové rozhraní slouží k ovládání měřicího hardwaru a nastavení parametrů měření. Zároveň zobrazuje v reálném čase naměřená data do grafu, ze kterého je možné např. přímo odečíst integrál přes zvolenou oblast, nebo data proložit polynomem.



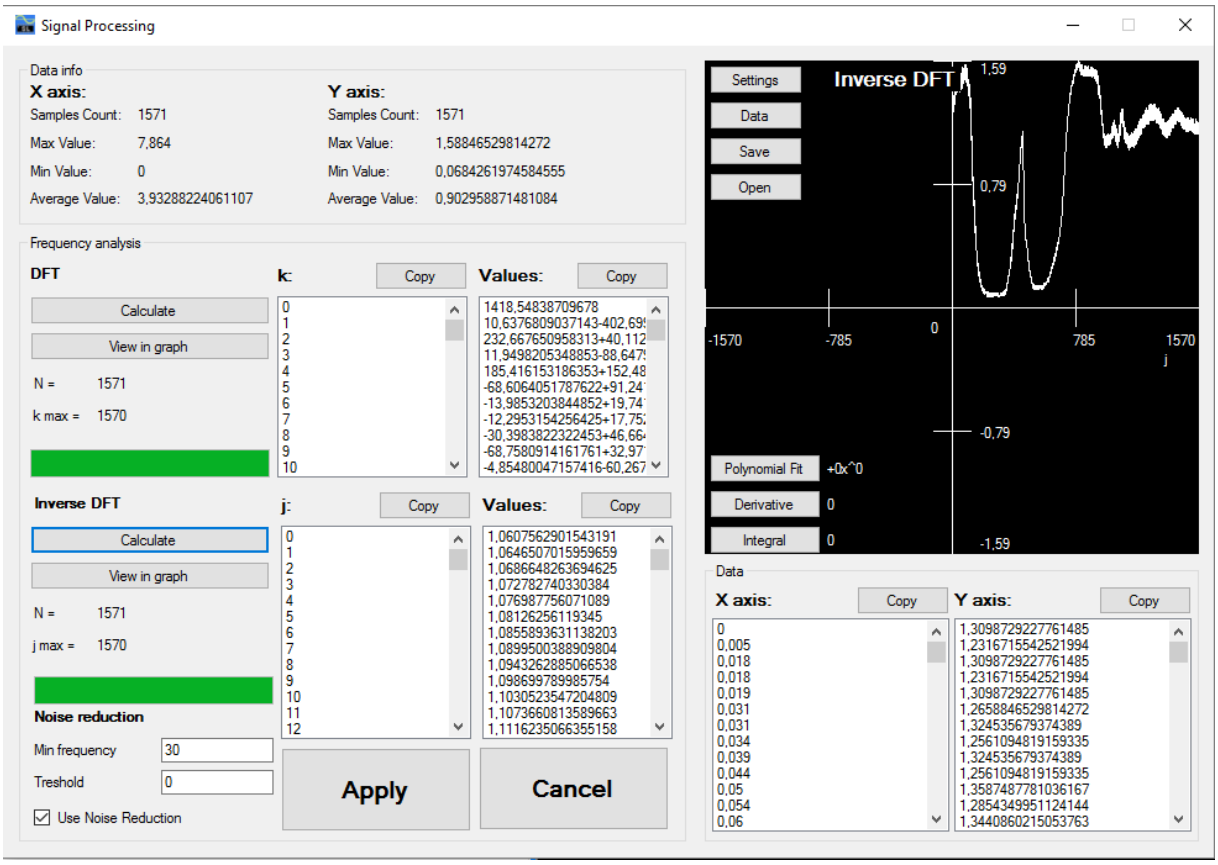
Obr. 2 - Grafické rozhraní měřicího systému

Systém umožňuje nastavit pro každý senzor podmínky automatického spuštění a pozastavení měření, umožňující uživateli automaticky filtrovat vstupní hodnoty a uchovat pouze ty, které zadané podmínky splňují (použitelné např. pro zjištění, za jak dlouho napětí poklesne na určitou úroveň nebo jak dlouho byla zakryta optická závora).



Obr. 3 – Podmínky automatického spuštění/ukončení měření

Pro odstranění šumu je k dispozici zpracování pomocí diskrétní Fourierovy transformace, při kterém se ze signálu odstraní uživatelem vybrané složky o vysoké frekvenci. Před konečnou aplikací zpětné transformace je vždy možné si účinky zvoleného nastavení prohlédnout v náhledovém grafu.



Obr. 4 – Okno pro zpracování dat pomocí DFT

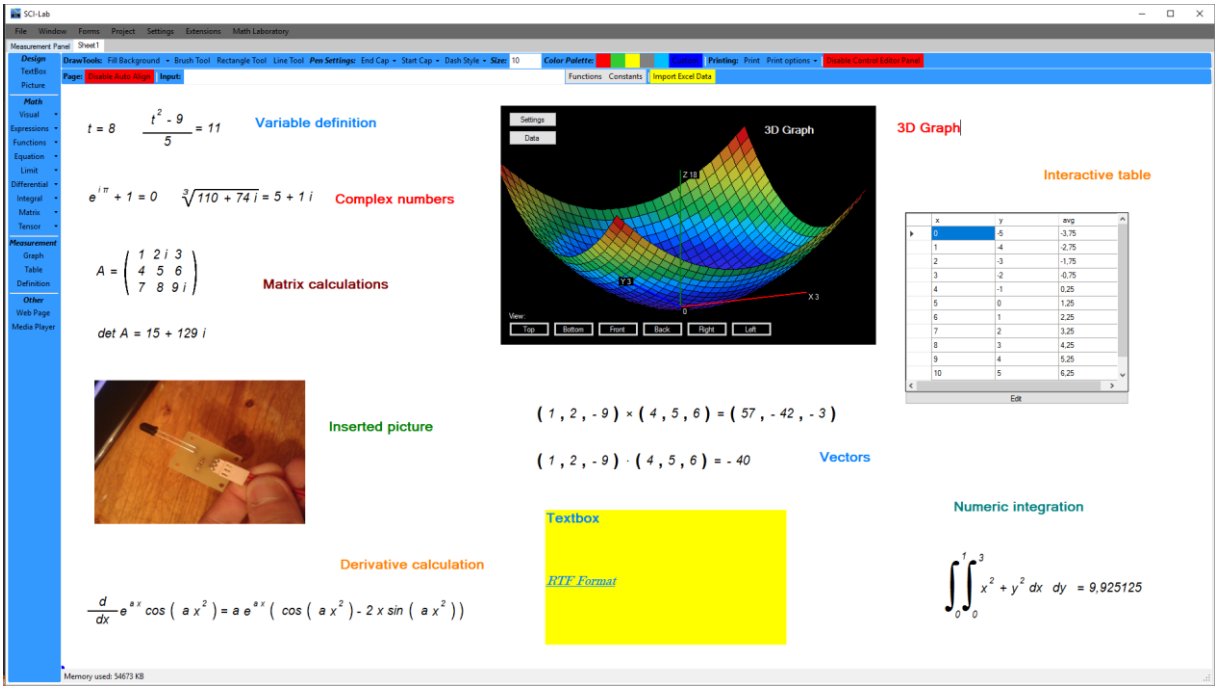
Systém představuje při zachování všech funkcí, které běžný student využije, řádově levnější alternativu ke komerčním školním měřicím systémům (cena hardwaru cca. 250 Kč). Oproti ostatním systémům nabízí mnohem jednodušší a uživatelsky přívětivé rozhraní a přímé propojení s matematickým softwarem pro zpracování a uložení naměřených dat.

2 MATEMATICKÝ SOFTWARE

Matematický software je založený na grafických pracovních listech, na které je možné kromě rovnic, matematických výrazů, grafů a dalších prvků přímo souvisejících s výpočty vkládat i obrázky, textová pole, video nebo dokonce plně funkční okna webového prohlížeče. Díky tomu je možné software používat i pro výuku a tvorbu názorných interaktivních prezentací, které může učitel přímo obohatit i o experiment a práci s reálnými daty.

CSE-Lab disponuje velmi širokou výbavou funkcí pokrývající celou středoškolskou matematiku a vybraná témata vyšší (komplexní čísla, matice, integrály a derivace, diferenciální rovnice, tenzory a další). Všechny funkce jsou dostupné bez potřeby učit se jakýkoliv programovací jazyk, díky čemuž je software dostupný opravdu pro každého.

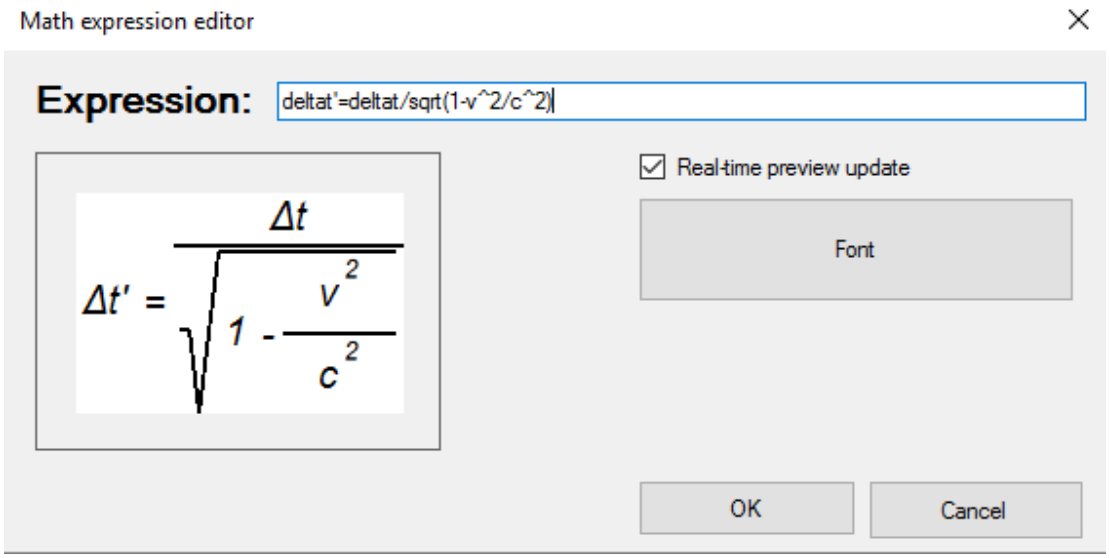
V současnosti jsou již hotové základy vestavěného programovacího jazyka, který by zkušenějším uživatelům zjednodušil práci a umožnil plně využít potenciál softwaru, jde však o nadstavbovou funkci, všechno bude nadále přístupné i bez ní.



Obr. 5 – Pracovní list s ukázkovými komponentami

2.1 Zobrazování výrazů

Veškeré výrazy jsou zobrazovány v grafické formě (jako kdyby byly psány na papír). Toto se děje zcela automaticky, uživatel pouze zapisuje výrazy v textové formě, nemusí se učit žádný jazyk pro tvorbu jejich grafických reprezentací.



Obr. 6 – Grafické zobrazení zadávaného výrazu

2.2 Proměnné

Pro zjednodušení výpočtů nabízí program možnost definovat neomezené množství vlastních proměnných a funkcí. Proměnné mohou být definovány konstantní hodnotou, vybranými daty z měření nebo pomocí vstupního pole, které dynamicky mění obsah proměnné podle uživatelského vstupu. Proměnných využívá i systém zadávání matic a tenzorů.

$$z = 25\,\pi^2$$

$$k = \frac{\sqrt{z}}{2}$$

$$\sin(k) = 1$$

Obr. 7 – Definice proměnných a jejich použití ve výpočtu

2.3 Funkce

Uživatel může snadno definovat a používat vlastní funkce jedné i více proměnných. Argumenty funkcí mohou být libovolné matematické objekty podporované programem (čísla, vyčíslitelné výrazy, matice, vektory, tenzory). V případě více argumentů se zadávají jako vektor (oddělují se pomocí „;“).

$$f(x) = \log_2\left(x^2 + 7\right)$$

$$f(5) = 5$$

$$g(h) = \sqrt{x1\ x2}$$

$$g\left(\frac{1}{8}, 2\right) = 0{,}5$$

Obr. 8 – Definice a použití vlastních funkcí

2.4 Vyhodnocování výrazů

Všechny výrazy jsou vyhodnocovány automaticky v oboru komplexních čísel, vybrané funkce však používají jinou variantu výpočetního algoritmu pro ně optimalizovanou, která používá pouze reálná čísla. Program přímo nabízí předdefinované základní konstanty (e, pí, i) a sadu přibližně 50 funkcí plně pokrývající běžné výpočty, včetně např. aproximací Gama funkce a Riemannovy zeta funkce.

$$\sqrt[3]{-1090 + 54\,i} = 5 + 9\,i$$

$$e^{i\pi} + 1 = 0$$

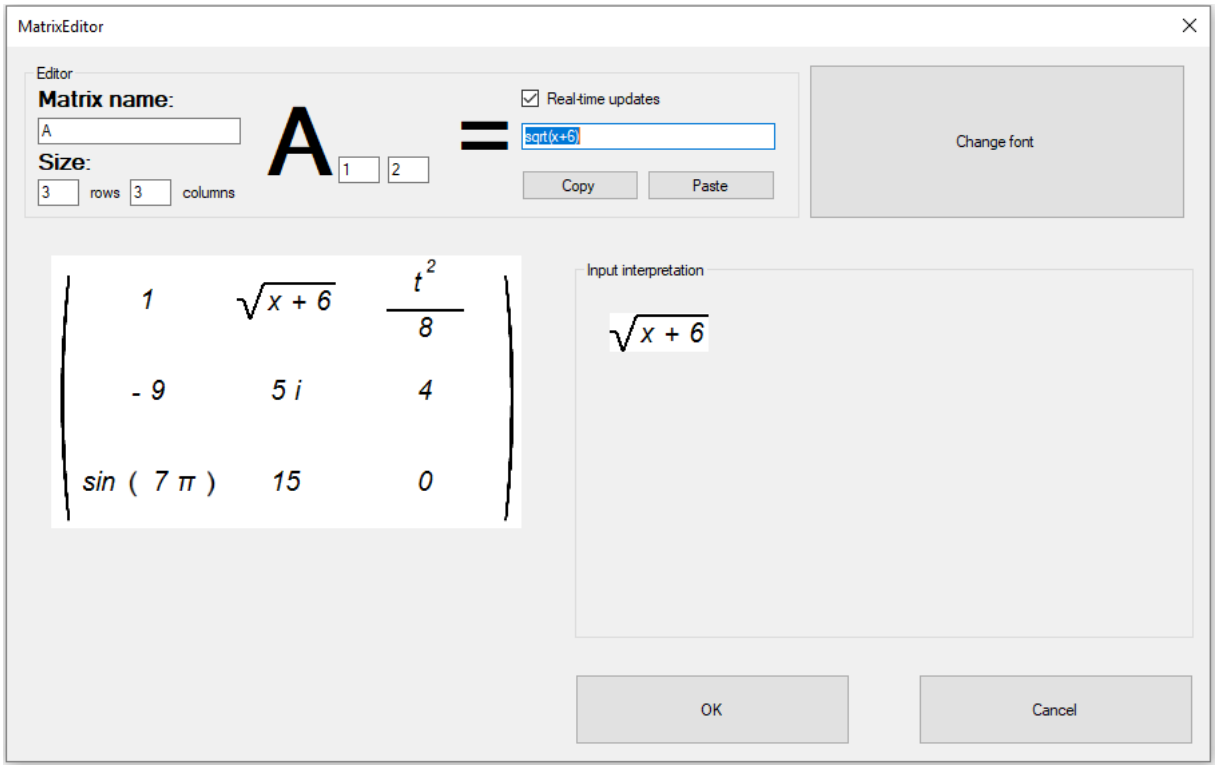
$$\Gamma(6) = 120$$

$$\ln\left(\sin^2(12) + e^7 + \cos^2(12) - 1\right) = 7$$

Obr. 9 – Ukázka vyhodnocování výrazů

2.5 Matice

Matice se zadávají zkrze názorný dialog, který uživatele provede celým procesem. Během zadávání program vždy po stisknutí klávesy enter uloží aktuální vstup na dané souřadnice a sám se posune do vedlejšího pole. V případě potřeby je možné se vrátit nebo rychle přesunout na libovolné pole v matici jednoduchým zadáním jeho souřadnic. Uživatel může vytvářet matice libovolných rozměrů, jediným omezením je výkon počítače a velikost obrazovky.



Obr. 10 – Okno pro zadávání matic

Prvky matice mohou být libovolné výrazy, které je možné vyhodnotit v oboru komplexních čísel. Závisí pouze na výsledku, výrazy samy mohou obsahovat vektory, tenzory i další matice. S maticemi lze po jejich definování pracovat stejně jako s ostatními výrazy. Matice je možné sčítat, odčítat a násobit, kromě toho je k dispozici řada funkcí např. pro spočítání determinantu (libovolného řádu), inverzní matice nebo transpozici. K jednotlivým prvkům matice je možné přistupovat zapsáním jejich souřadnic do hranatých závorek za jméno matice.

$$A = \begin{pmatrix} i & 5\,i & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 0 & 1 \\ 2 & 3 & 45 & -9 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix}$$

$$\det A = 584 - 10806\,i$$

$$A^H = \begin{pmatrix} -1\,i & 4 & 8 & 2 \\ -5\,i & 5 & 9 & 3 \\ 2 & 6 & 0 & 45 \\ 3 & 7 & 1 & -9 \end{pmatrix} \qquad A\,B = \begin{pmatrix} 31 + 16\,i & 36 + 22\,i \\ 98 & 120 \\ 42 & 60 \\ 173 & 214 \end{pmatrix}$$

Obr. 11 – Ukázka maticových výpočtů

K výpočtu determinantu jsou k dispozici funkce používající Gaussovu eliminační metodu (rychlejší, vzhledem k použití dělení však může způsobit zaokrouhlovací chyby v případě, že matice obsahuje velmi velká nebo malá čísla) nebo rozvoj podle řádku/sloupce a výpočet pomocí subdeterminantů (pomalejší, ale vzhledem k použití pouze násobení a sčítání vždy přesný).

2.6 Vektory

Z libovolných výrazů je možné sestavit vektor napsáním do závorek „()“ a oddělením „;“, např. (1;a;z+5). Rozměry jednotlivých vektorů nejsou nijak omezeny. S vektory je možné pracovat stejně jako s ostatními prvky, navíc jsou podporovány operátory · (skalární součin, Ctrl+*) a × (vektorový součin, Alt+*). K jednotlivým prvkům vektoru lze přistupovat zapsáním příslušného indexu do hranatých závorek za vektor.

$$h = (1, 2, 3)$$

$$k = (4, 5, 6)$$

$$h \cdot k = 32$$

$$h \times k = (-3, 6, -3)$$

$$h \times (2, 4, 6) = (0, 0, 0)$$

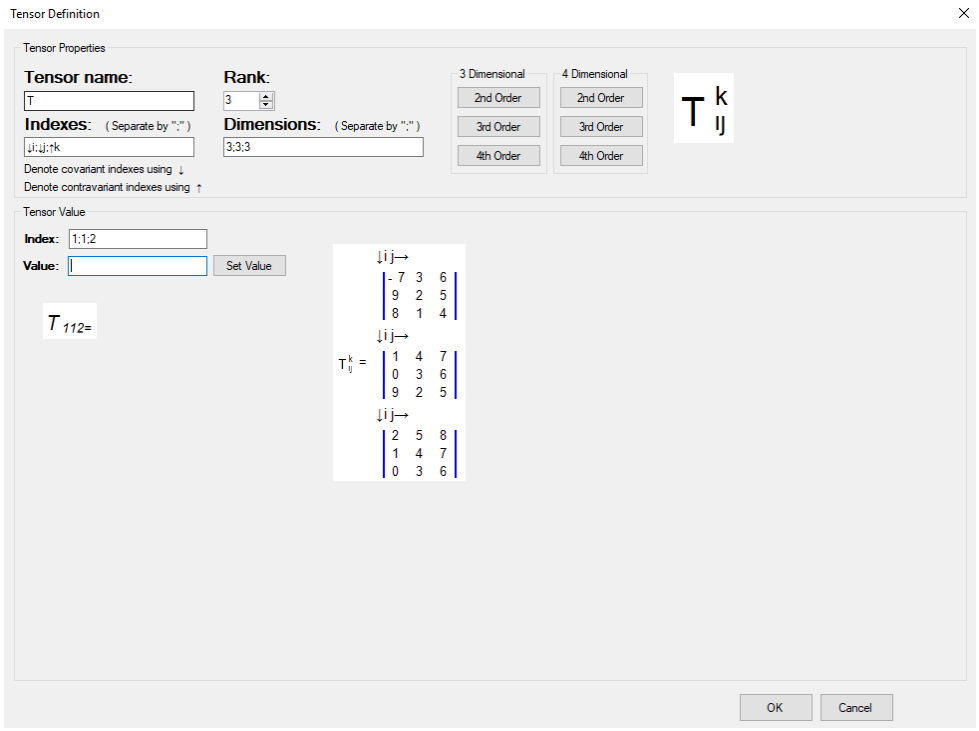
Obr. 12 – Definice a výpočty s vektory

2.7 Tenzory

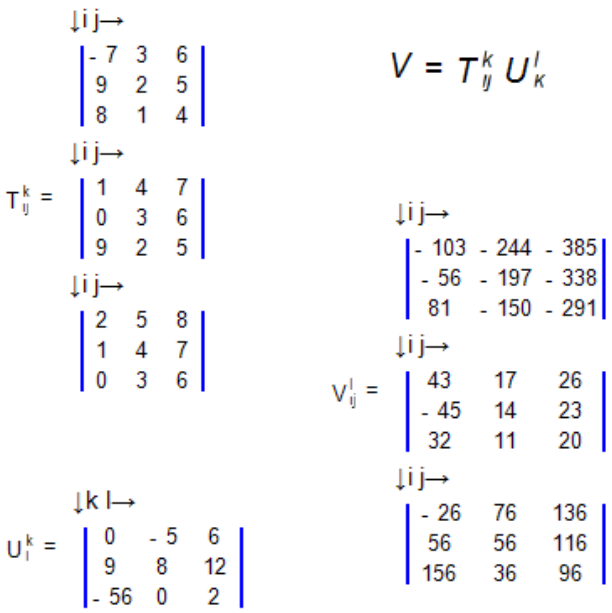
Tenzory libovolných řádů a rozměrů se stejně jako matice zadávají přes názorný grafický formulář. Defaultně nejsou složky tenzorů na pracovních listech kvůli jejich velkému počtu zobrazovány, v případě potřeby je však možné celý tenzor zobrazit. Jednotlivé prvky tenzorů opět nejsou nijak obsahově omezeny, dokud jsou vyhodnotitelné v reálných číslech.

K odlišení kovariantních a kontravariantních indexů jsou použity znaky „↑“ a „↓“ zadávané přímo pomocí šipek na klávesnici. K přechodu mezi kovariantními a kontravariantními složkami tenzorů je použit metrický tenzor, který může uživatel nastavit pro každý projekt na libovolnou hodnotu (implicitně je nastaven 3 rozměrný euklidovský prostor, k dispozici je několik předem vytvořených metrik, např. povrch koule, plochý časoprostor, Schwarzschildova metrika, uživatel si však může definovat zcela vlastní). S tenzory je možné

provádět všechny běžné operace, ve výpočtech je použita Einsteinova sumační konvence. K jednotlivým prvkům tenzorů se přistupuje stejným způsobem jako u matic a vektorů.



Obr. 13 – Dialogové okno pro zadávání tenzorů



Obr. 14 – Násobení tenzorů[1]

[1] Tenzory vyšších řádů jsou graficky zobrazovány pomocí vnořených tabulek zobrazující hodnoty pro různé kombinace indexů, např. tabulky u tenzoru T odpovídají hodnotám indexu k 1, 2 a 3 (shora dolů)

2.8 Jednotky

Uživatel může ve výpočtech přímo použít jednotky. K dispozici jsou všechny základní jednotky SI a množství vedlejších s vlastními jmény, v případě potřeby je možné jednotky mezi sebou libovolně kombinovat (násobit a dělit) a definovat pomocí známých nové. Aby nedošlo ke kolizi s názvy proměnných, začíná název jednotky vždy znakem „%“. Program akceptuje i přímo násobky jednotek, zadávají se klasicky pomocí ustanovených předpon např. 5*%mA program interpretuje jako 5 miliampérů. Výsledek výpočtu je vždy automaticky vyobrazen ve vhodných jednotkách a jejich násobku.

$$\frac{C}{5\ kV} = 200\ uF \qquad a = 5\ m\ s^{-2}$$

$$\frac{N\ s}{m^2} = 1\ Pa\ s \qquad v(x) = \int_0^x a\ dx$$

$$10^{19}\ eV = 1{,}60217653\ J \qquad v(0{,}5\ s) = 2{,}5\ m\ s^{-1}$$

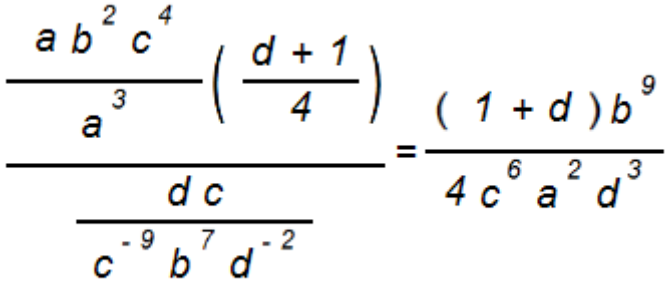
$$\frac{\sqrt{\Omega^2 + (Hz\ H)^2}}{\sqrt{2}} = 1\ \Omega \qquad s(x) = \int_0^x v(x)\ dx$$

$$s(30\ s) = 4{,}5\ km$$

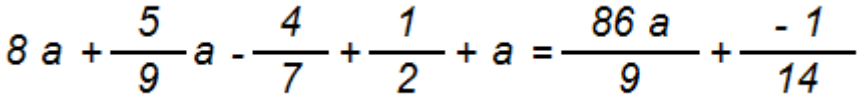
Obr. 15 – Výpočty s jednotkami

2.9 Zjednodušování výrazů

Součástí programu je funkce pro zjednodušování výrazů. K tomu je na výraz aplikováno množství různých pravidel, program např. posčítá všechny číselné hodnoty v součtu a výsledek ve výrazu zapíše jako celé číslo nebo zlomek v základním tvaru, vykrátí výrazy ve zlomcích, kde je to vhodné vytkne část výrazu před závorku, přepíše opakované členy v součtech a součinech na jejich násobky, resp. mocniny a další… Zjednodušování probíhá v čistě symbolické formě, výrazy proto mohou obsahovat i nedefinované členy.



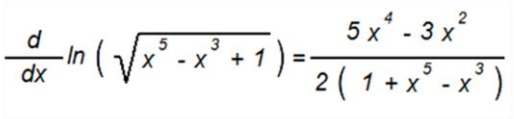
Obr. 16 – Zjednodušení zlomku

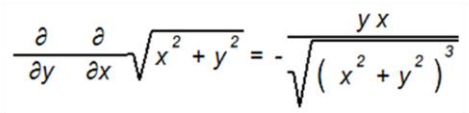


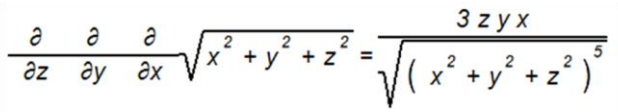
Obr. 17 – Úprava výrazu

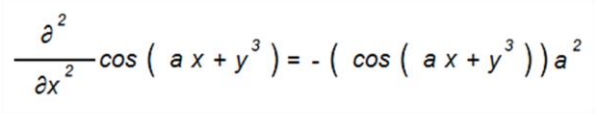
2.10 Symbolické derivování

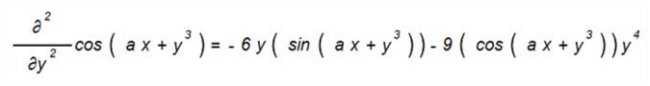
CSE-Lab umožňuje uživateli spočítat derivaci libovolného řádu jakékoliv diferencovatelné funkce v symbolické formě. Všechny výsledky jsou automaticky zjednodušovány.











Obr. 18 – Symbolické derivování

2.11 Symbolické integrování

Stejně jako derivovat může uživatel i integrovat libovolné funkce. Program se vždy pokusí integrál najít pomocí základních metod (tabulkové integrály, substituce, racionální funkce, per partes), pokud je integrace úspěšná, nabídne program k výsledku i postup, kterým jej získal.

$$\int \frac{\ln(x)}{x} + a\,x\,dx = \frac{a\,x^2 + \ln^2(x)}{2} + C$$

$$\int \frac{\sin(x)}{\sqrt[3]{\cos^2(x)}}\,dx = -3\sqrt[3]{\cos(x)} + C$$

$$\int \frac{\ln^n(a\,x)}{b\,x}\,dx = \frac{\ln^{1+n}(a\,x)}{b\,(1+n)} + C$$

$$\int \frac{5\,x}{(x^2 + a)^3}\,dx = -\frac{5}{4\,(x^2 + a)^2} + C$$

Obr. 19 – Symbolické integrování

2.12 Numerické integrování

Integrály ve výrazech jsou standardně vyhodnocovány numericky s buďto pevně danou velikostí nebo počtem kroků (uživatelsky nastavitelné). Takto je možné počítat i vícerozměrné integrály.

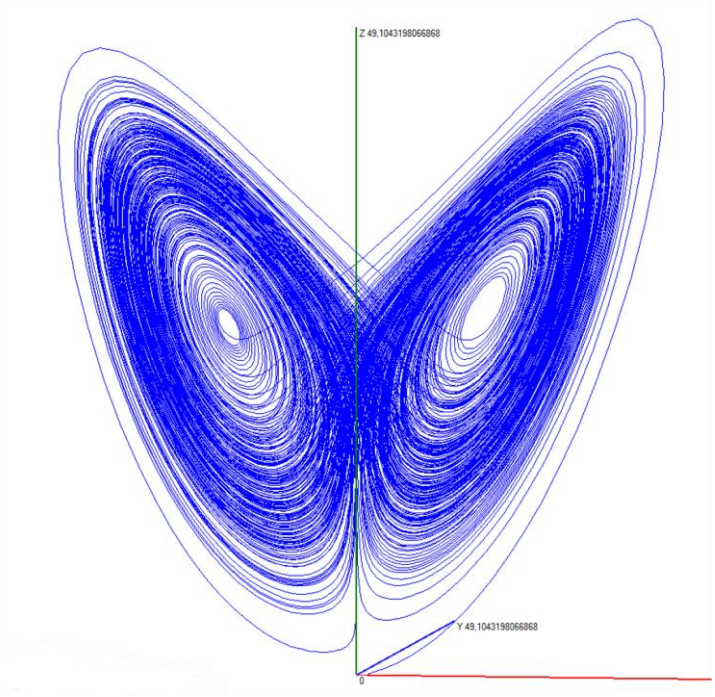
$$\int_{1}^{3}\int_{0}^{4} x^2 + y^3 \, dx \; dy = 121{,}828$$

Obr. 20 – Numerické integrování

2.13 Numerické řešení diferenciálních rovnic

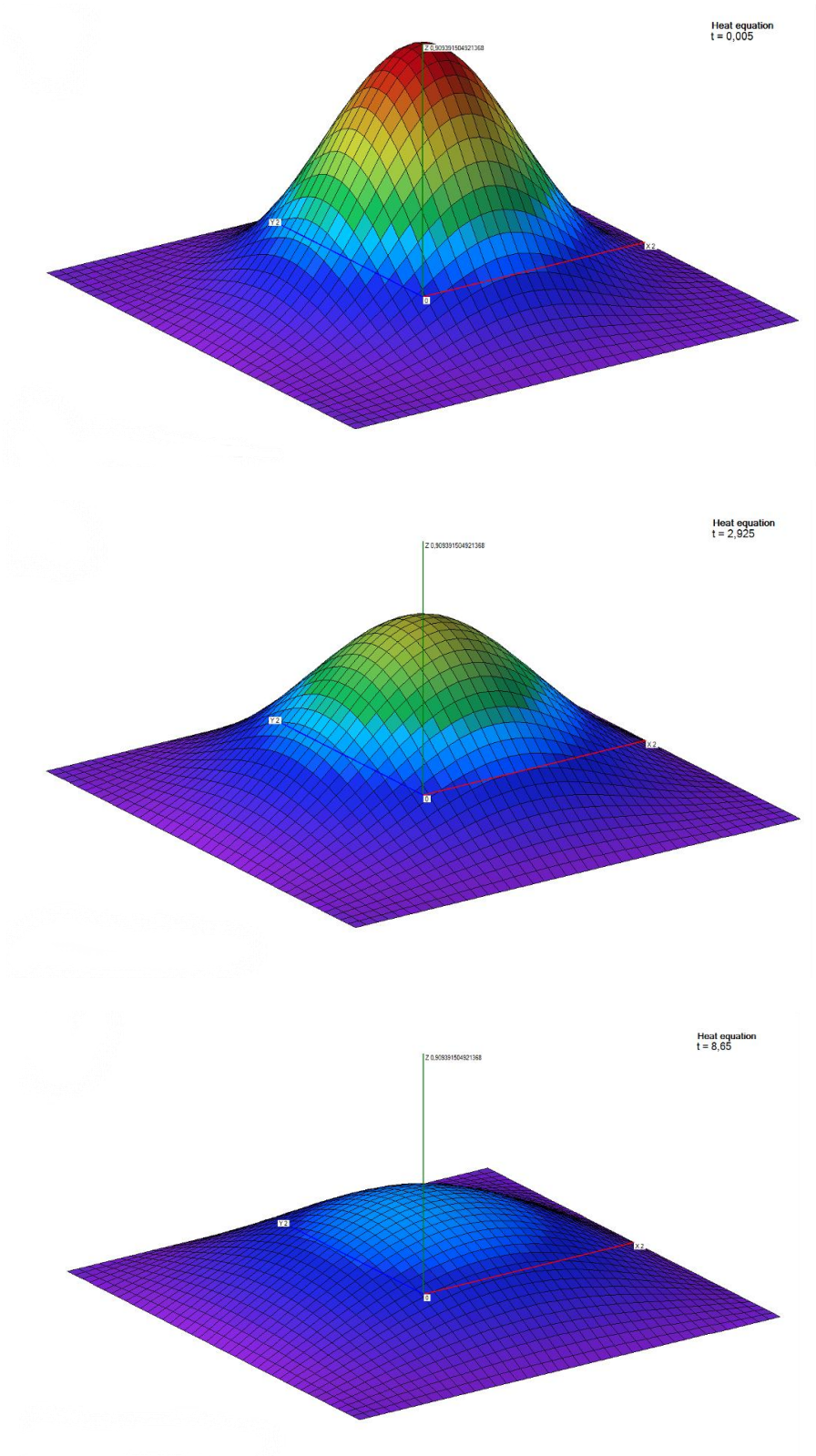
Program nabízí řešení jak obyčejných diferenciálních rovnic a jejich soustav, tak i parciálních diferenciálních rovnic (1 a 2 rozměrných).

Pro obyčejné diferenciální rovnice a soustavy jsou implementovány metody Runge-Kutta až 4. řádu. Řešení je zobrazeno vždy graficky, v případě soustavy rovnic si může uživatel zvolit mezi zobrazením jednotlivých funkcí nebo vývoje celé soustavy (pro 2 a 3 rovnice).



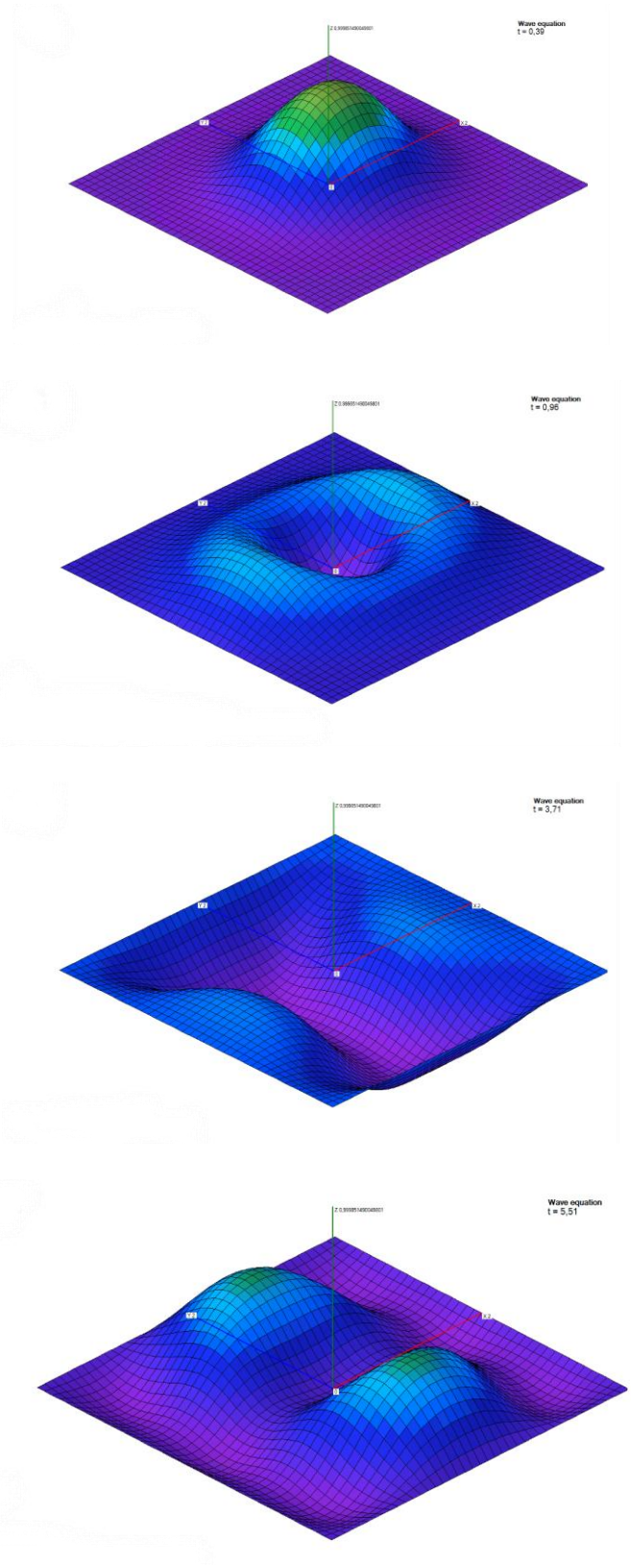
Obr. 21 – Lorenzův atraktor

U parciálních diferenciálních rovnic je řešení zobrazeno jako animovaný graf, který si uživatel může libovolně pozastavit a přehrávat, dokonce je možné nechat vygenerovat sekvenci obrázků do zvolené složky. K řešení se používá metoda konečných diferencí, k dispozici je jak explicitní, tak i implicitní schéma.



Obr. 22 – Řešení PDE – rovnice vedení tepla

$$\frac{\partial}{\partial t} u = 0{,}05 \nabla^2 u$$

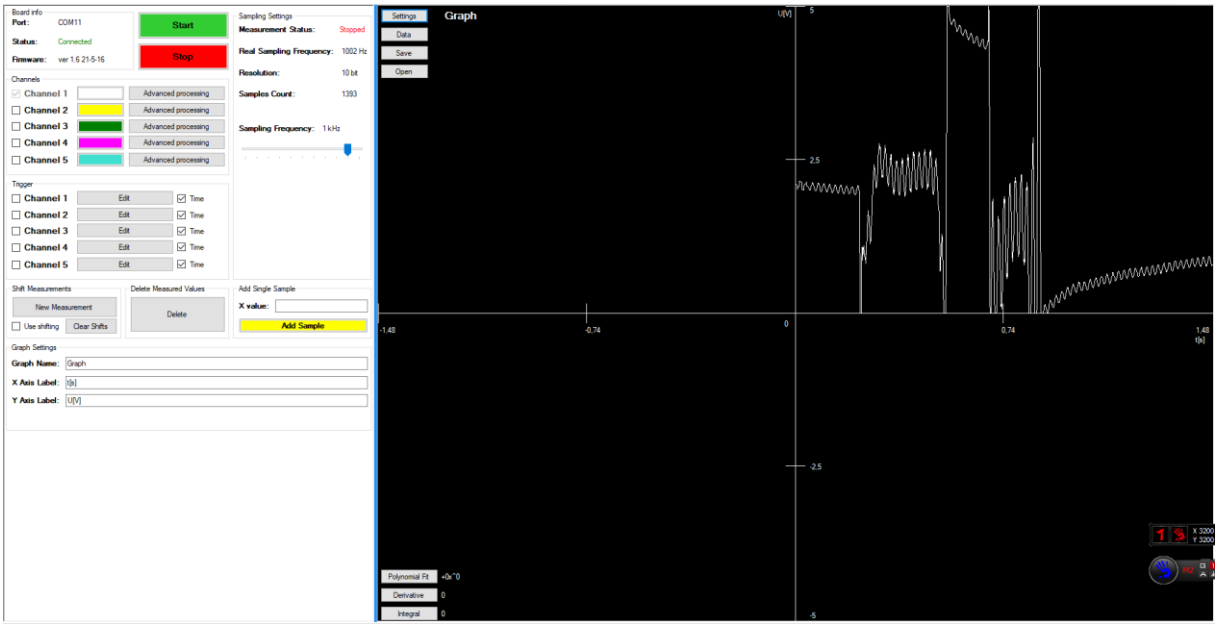


Obr. 23 – Řešení PDE – Vlnová rovnice

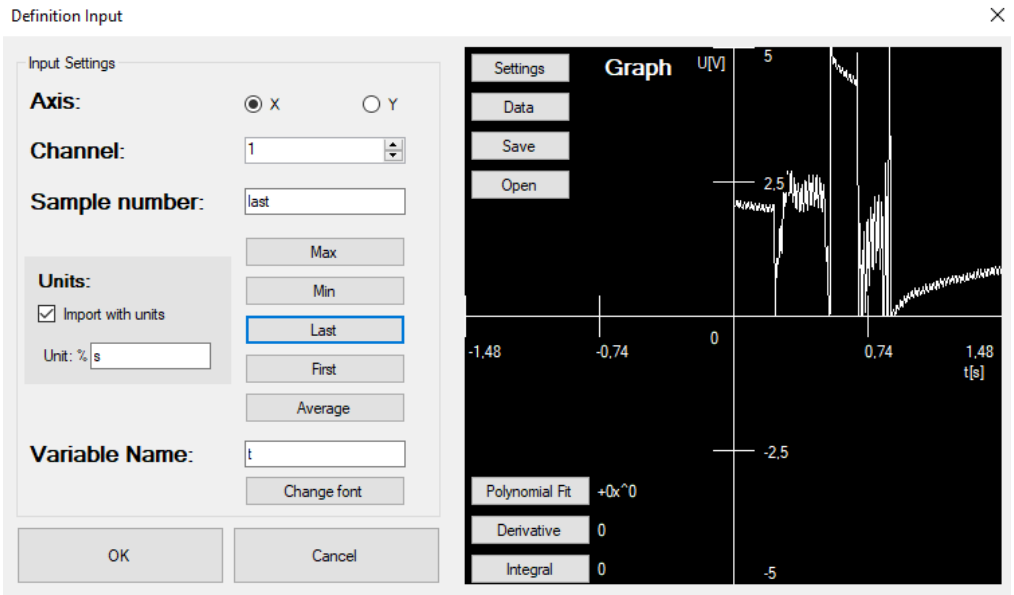
$$\frac{\partial^2}{\partial t^2} u = \frac{\partial^2}{\partial x^2} u + 0{,}5 \frac{\partial^2}{\partial y^2} u$$

2.14 Provázání s naměřenými daty

Přes definice proměnných je možné do pracovních listů přímo vložit naměřená data. Kromě importu konkrétních vzorků jsou k dispozici funkce jako maximum, poslední hodnota, průměr. Zároveň je v programu vestavěná i přímo matematická funkce, která vrátí hodnotu odpovídající zadanému času (užitečná např. při použití naměřených dat jako okrajové podmínky pro diferenciální rovnice). Při opětovném provedení měření se všechny hodnoty automaticky aktualizují a přepočítají.



Obr. 24 – Měřicí rozhraní s naměřenými daty



Obr. 25 – Dialogové okno pro import hodnot

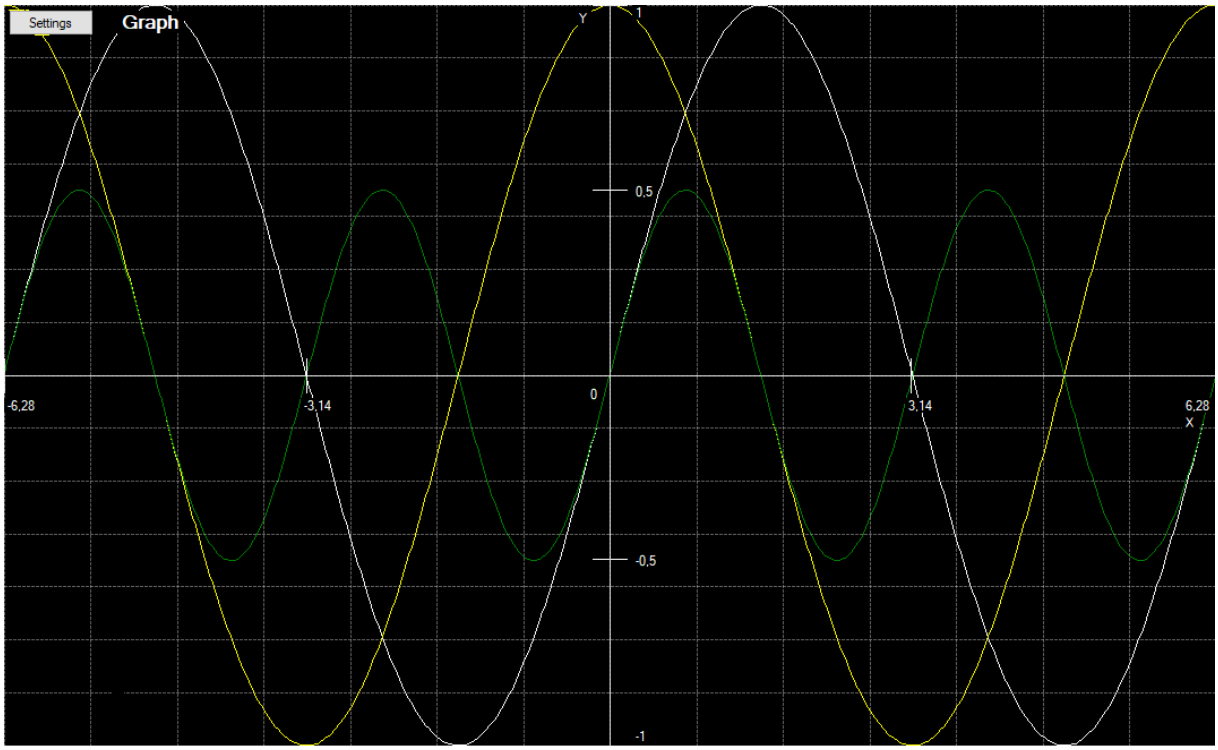
$$t = 1{,}47799999999995 \ \ s$$

Obr. 26 – Importovaná hodnota

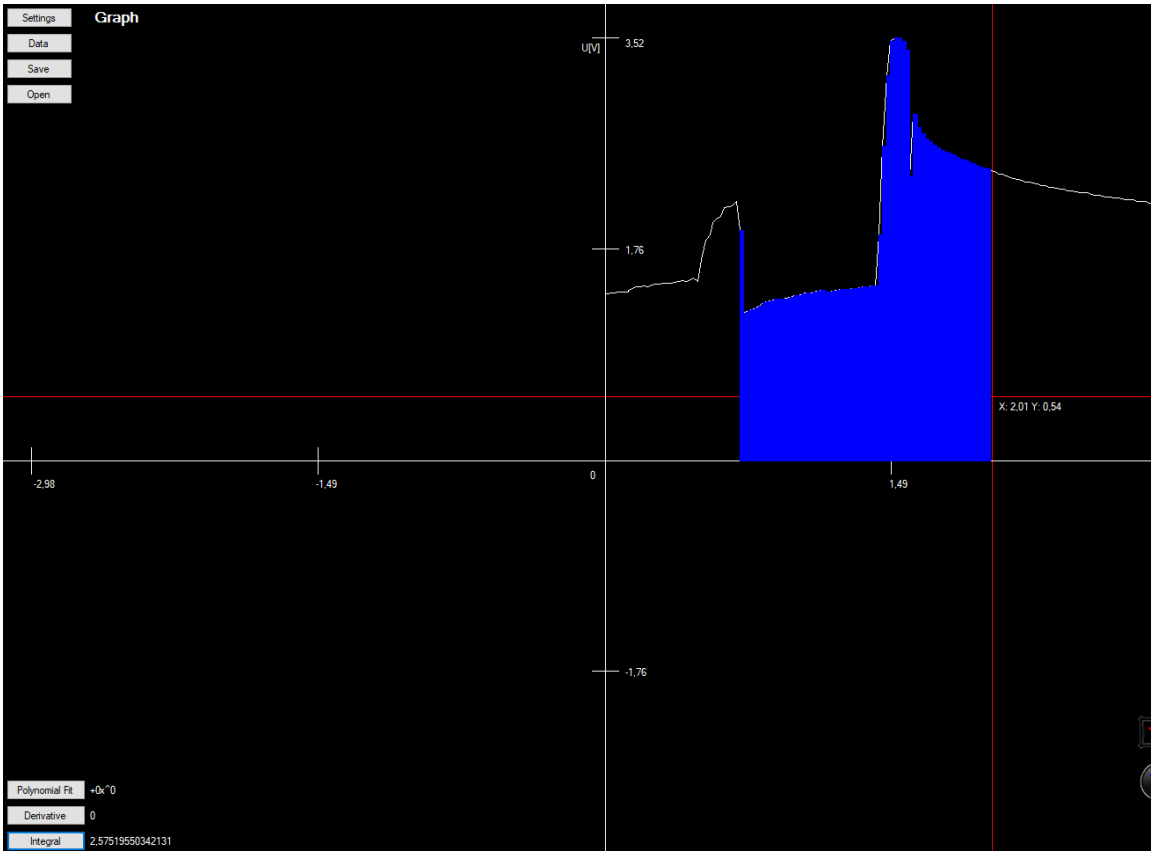
2.15 Grafy

2.15.1 2D Grafy

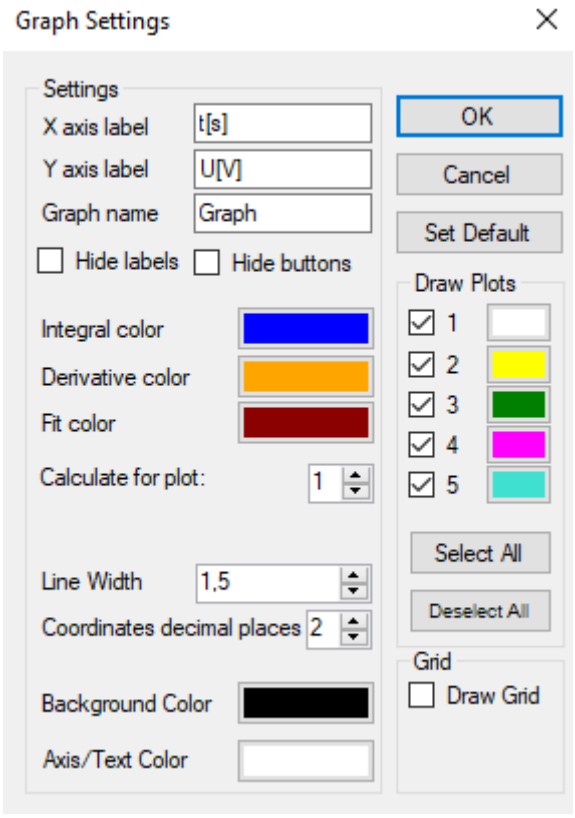
Dvojrozměrné grafy umožňují zobrazení až pěti řad do jednoho grafu. Graf vždy automaticky přizpůsobí své měřítko nejvyšší absolutní hodnotě, která se v něm vyskytuje, graf však může uživatel libovolně dále posouvat a zoomovat. Přímo v grafu je možné odečíst derivaci v určitém bodě a určitý integrál přes zvolenou oblast (obojí numericky), stejně jako provést regresi polynomem až 7. řádu. Do grafu je možné zadat nebo editovat přímo jednotlivé hodnoty, automaticky vytvořit graf zadané funkce na libovolném intervalu nebo nechat vstup postupně naplnit čísly ze zadaného intervalu.



Obr. 27 – Graf zobrazující funkce sin(x), cos(x) a jejich součin



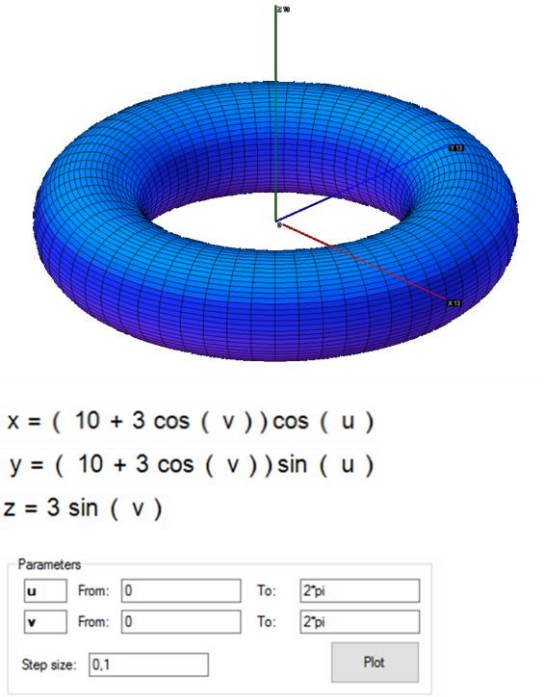
Obr. 28 – Odečtení obsahu plochy přímo v grafu



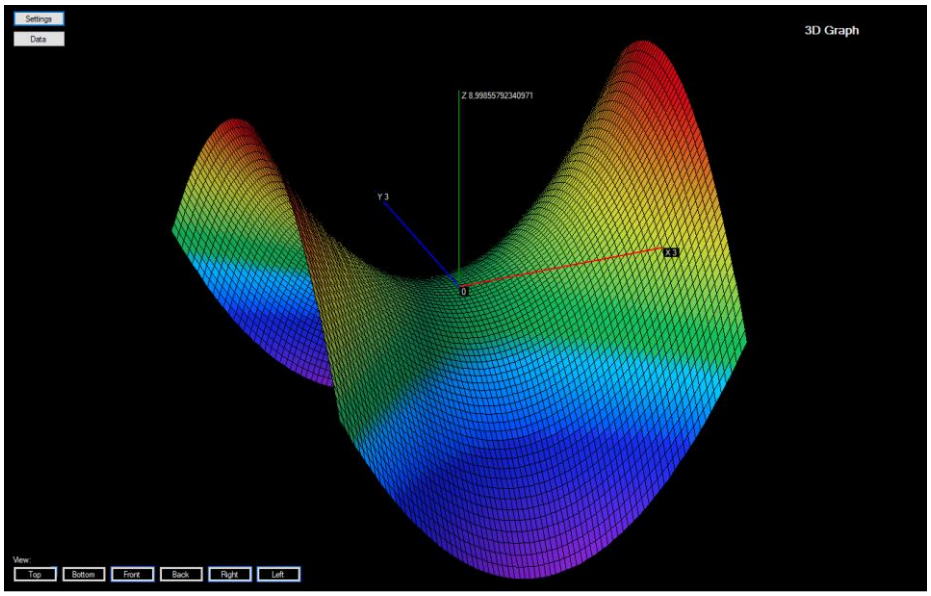
Obr. 29 – Vizuální nastavení grafu

2.15.2 3D Grafy

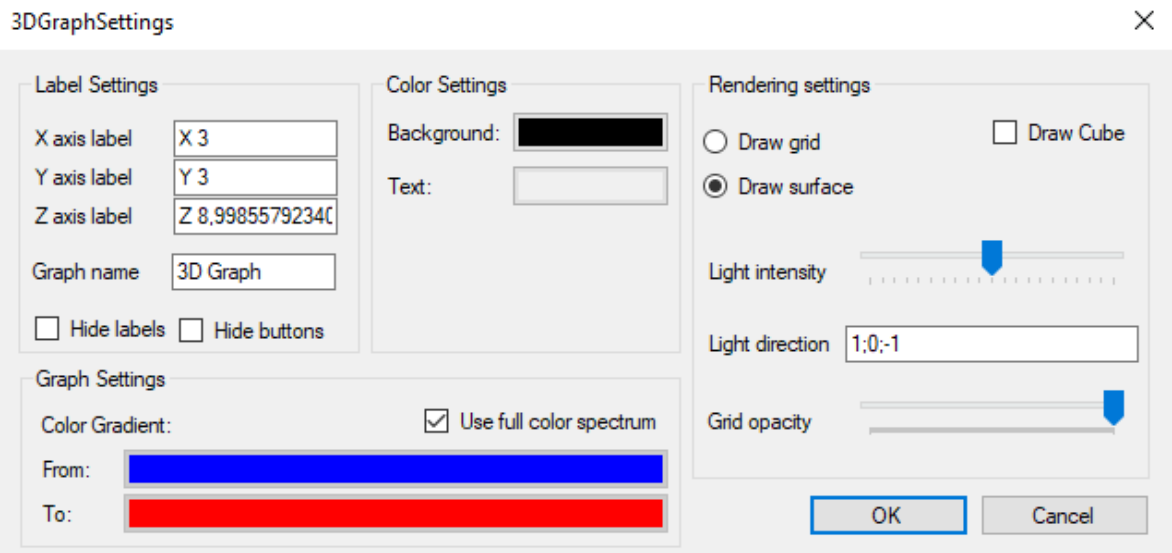
3D grafy přichází s řadou vizuálních nastavení, jako intenzita a směr osvětlení nebo barevná škála, ve které je graf zobrazen. Uživatel si může zvolit, zda chce zobrazit graf pouze jako síť bodů nebo jako spojitý pevný povrch. S grafy je možné libovolně posouvat, zoomovat a otáčet. Uživatel může přímo editovat jednotlivé hodnoty, nechat zobrazit graf libovolné funkce 2 nezávisle proměnných nebo vykreslit plochu zadanou parametricky.



Obr. 30 – Torus zadaný parametrickými rovnicemi



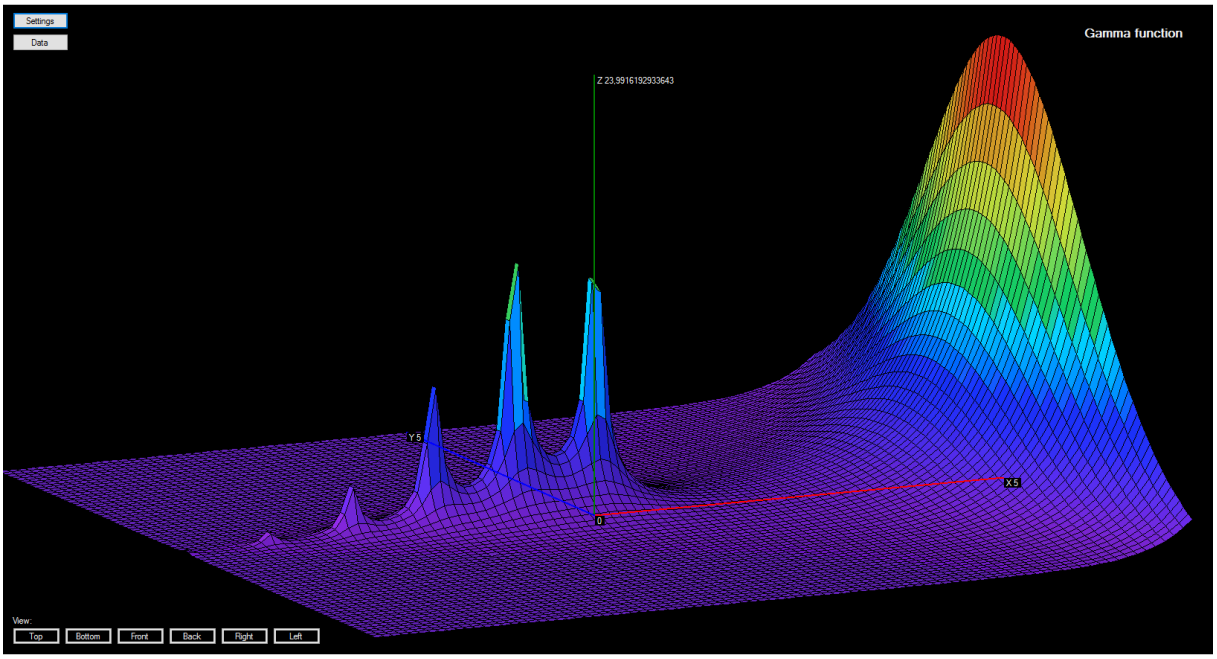
Obr. 31 – Graf funkce 2 nezávisle proměnných, hyperbolický paraboloid



Obr. 32 – Vizuální nastavení 3D grafu

2.15.3 Komplexní grafy

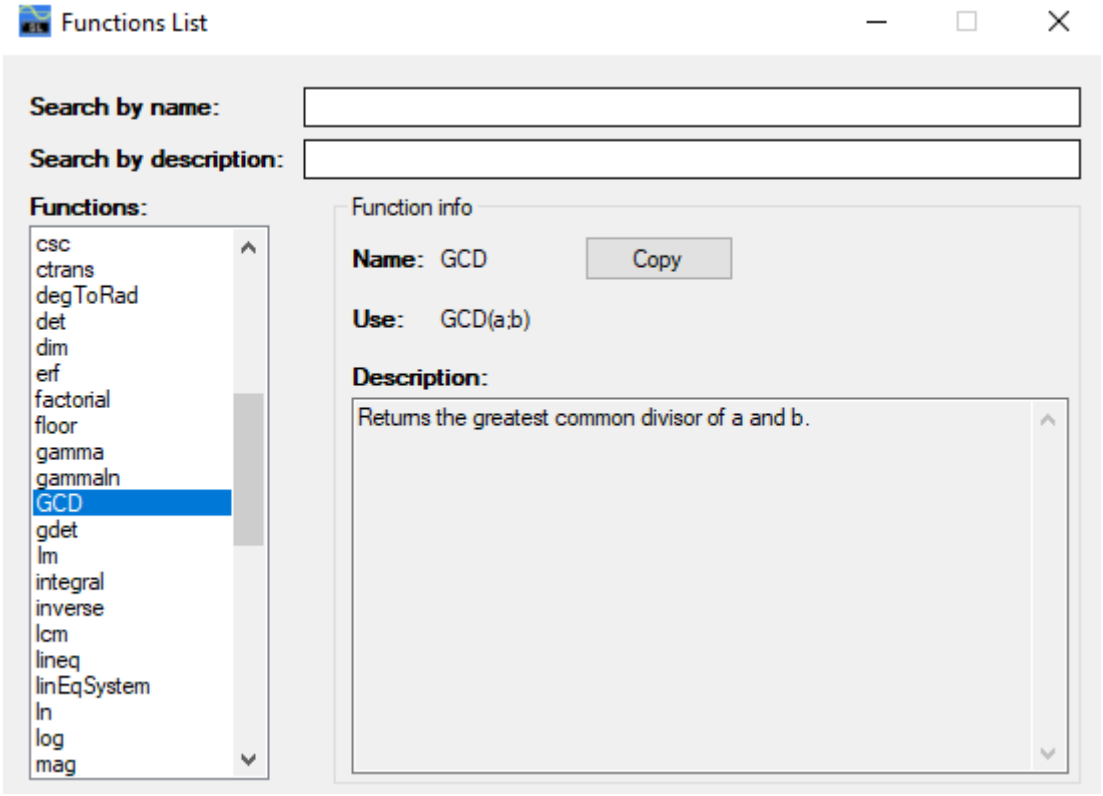
3D grafy jsou použity i k zobrazení grafů komplexních funkcí. Pro každou funkci jsou vytvořeny tři oddělené grafy zobrazují reálnou složku, imaginární složku a absolutní hodnotu závislé proměnné.



Obr. 33 – Graf Gamma funkce – absolutní hodnota

2.16 Databáze funkcí

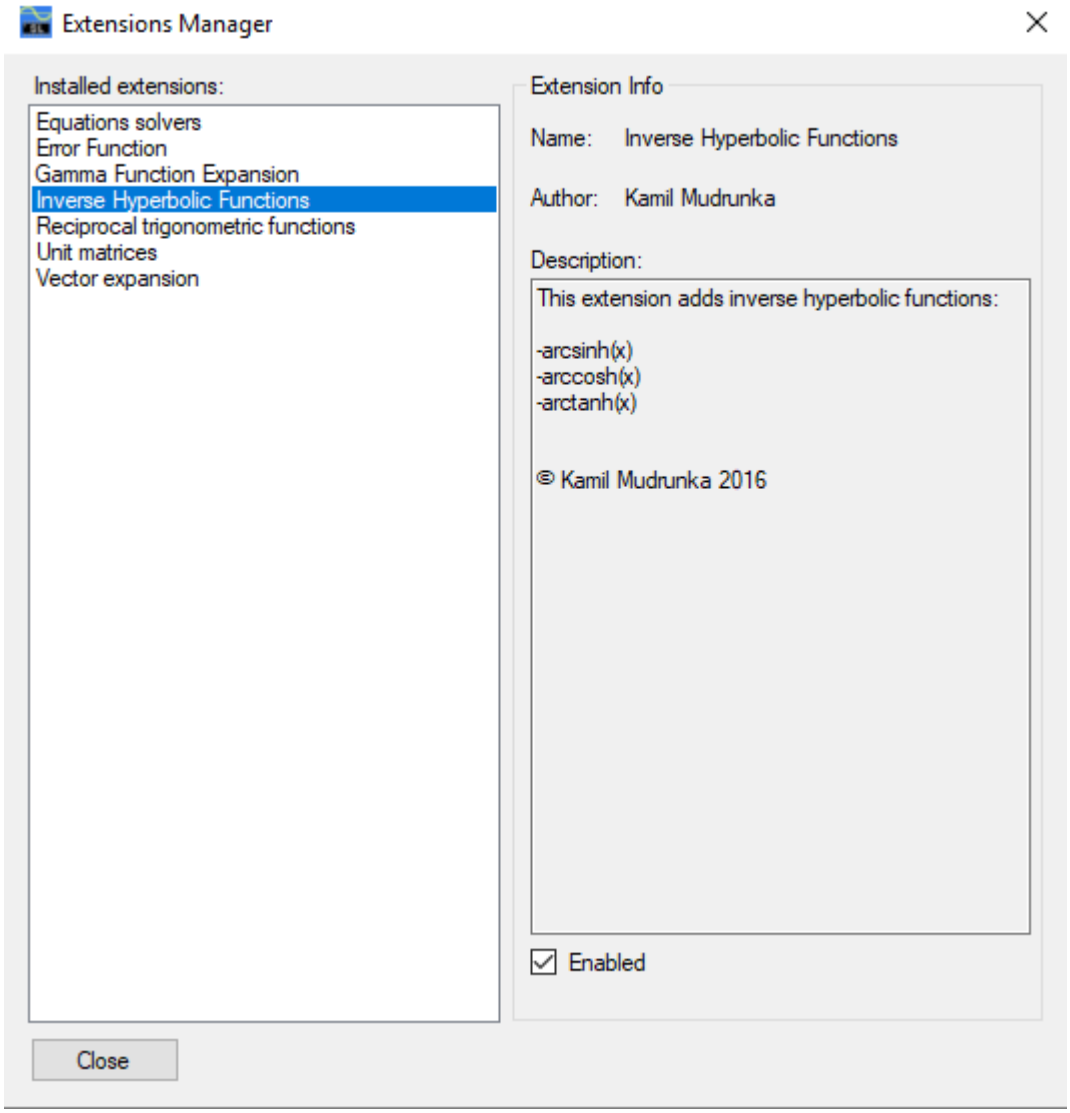
Vzhledem k množství vestavěných matematických funkcí obsahuje program databázi funkcí, ve které je možné vyhledávat podle jména nebo popisu funkce. U každé položky je zobrazeno jméno, ukázkový způsob použití a popis. Stejné databáze existuje i pro konstanty.



Obr. 34 – Databáze funkcí

2.17 Uživatelská rozšíření

Uživatelé mohou z vlastních definic funkcí a proměnných (včetně matic, vektorů a tenzorů) snadno vytvářet vlastní balíčky rozšíření, které mohou sdílet s ostatními. Program nové rozšíření sám nainstaluje a vloží nové funkce a konstanty do databáze. Podle potřeby je pak možné jednotlivá rozšíření spravovat, zapínat nebo vypínat.



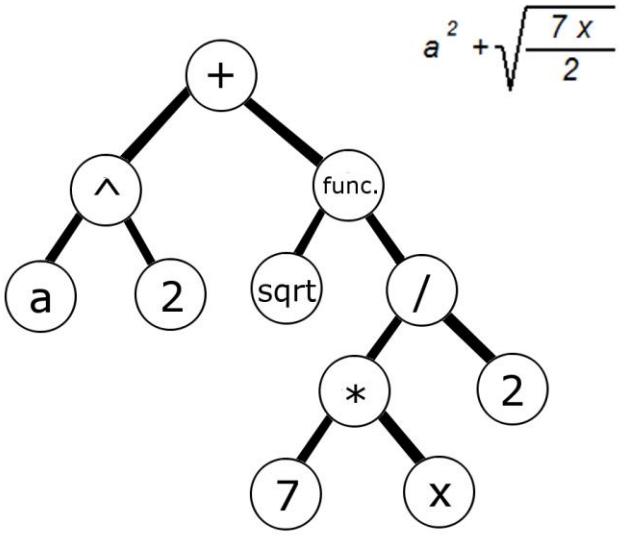
Obr. 35 – Správce rozšíření

3 ALGORITMY A IMPLEMENTACE

CSE-Lab je celý napsaný v jazyce C# a kromě samotné aplikace se sestává z velkého množství dynamických knihoven a dalších souborů, proto bylo nakonec nutné vytvořit vlastní instalační balíček. Veškerý kód je vlastní dílo autora, program nepoužívá žádné knihovny třetích stran. V této části práce Budou detailněji popsány vybrané části programu, jejich návrh a tvorba.

3.1 Zpracování výrazů

Většina výpočetních modulů, zpracování výrazů a jejich grafické vykreslování vychází z rozkladu výrazu do binárního stromu, jehož uzly představují jednotlivé operátory a větve operandy (popř. údaje o funkci, mezích integrace, …). Stromy jsou pak zpracovávány rekurzivně.



Obr. 36 – Rozložení výrazu do binární stromu

Při tvorbě bylo primárně cíleno na univerzálnost systému, tj. schopnost zpracovat opravdu libovolný výraz a nabídnout uživateli co nejširší výčet matematických objektů a funkcí pro práci, mnohdy za cenu nižšího výkonu oproti specializovaným softwarům. Pro určité úlohy, kde hraje výpočetní výkon podstatnou roli, nabízí software uživateli kromě obecné varianty i variantu výpočetního algoritmu optimalizovanou pro daný problém, avšak ne tak obecnou (např. když si budu chtít vykreslit graf jednoduché funkce, nepotřebuji matice nebo tenzory, tudíž použiji rychlejší variantu, která matice ani tenzory neřeší, stejně tak při řešení diferenciálních rovnic až na konkrétní případy většinou nejsou potřeba komplexní čísla a uživatel si tak může vybrat i rychlejší variantu pracující pouze s reálnými čísly).

3.2 Matematické objekty

Pro všechny matematické objekty (komplexní čísla, matice, vektory, tenzory) byly napsány samostatné dynamické knihovny, díky čemuž je možné je použít i v dalších aplikacích bez nutnosti vše programovat znovu. Při zpracování výrazů ve formě stromu vrací vždy program objekt obecné třídy jazyka C# „object“, od které jsou všechny ostatní datové typy odvozeny a teprve poté se zkontroluje typ vráceného objektu a provede se potřebná operace nebo vypsání na obrazovku. Tento mechanismus mi umožňuje snadno a efektivně program rozšiřovat o nové objekty při zachování jednoduchosti a přehlednosti kódu, který se zůstává stejný, pouze se rozšíří o zpracování operací s přidaným objektem (např. kontrola, zda má daná operace vůbec smysl, nelze třeba sčítat vektory a skaláry, násobit vektor skalárem naopak jde).

3.3 Tenzory

Pro podrobnější popis a ukázku implementace konkrétního problému jsem zvolil třídu představující tenzor obecného řádu, která zároveň implementuje i základní operace sčítání, odčítání a násobení včetně Einsteinovy sumační konvence.

Na začátek je třeba poznamenat, že program z praktických důvodů rozlišuje mezi vektory a tenzory prvního řádu. Zatímco tenzory pro plnou funkčnost vyžadují od uživatele definování metrického tenzoru, který umožňuje převádět mezi různými polohami indexů a zároveň ale omezuje rozměry tenzorů v daném projektu na pouze jednu hodnotu (s 3D metrickým tenzorem musím používat trojrozměrné tenzory, jiné rozměry nebudou fungovat), vektory jsou osvobozeny od této komplikované definice a pracují s euklidovským prostorem, rozměr nijak není omezený, zadávají se prostým oddělením složek středníky a uzavřením do závorek. Tento krok byl proveden pro zpřístupnění a zesnadnění použití vektorů pro středoškoláky, kteří tenzory neznají a pro zjednodušení výpočtů, které tenzory nepoužívají.

Prvním problémem je samotné uložení tenzoru do paměti. Bylo potřeba vymyslet mechanismus, který by byl schopen popsat tenzor libovolného řádu a rozměrů a zároveň by umožňoval snadnou implementaci základních operací. Současným řešením je třída nazvaná „varArray“ (pole proměnných), ve které jsou uloženy všechny prvky tenzoru do dvou jednorozměrných polí (jedno obsahuje numerické hodnoty prvků, druhé jejich symbolickou formu) spolu se systémem adres a metod jednoznačně přiřazující každé kombinaci indexů právě jeden odpovídající prvek. Systém adres je navržen nejobecnějším možným způsobem, tenzor může mít dokonce různé rozměry v různých indexech.

Nejjednodušší je vysvětlit jej na názorném příkladu: Ukládám čtyřrozměrný tenzor třetího řádu. Celkem tedy bude mít pole 64 prvků. Indexy budou nabývat hodnot 0 až 3. Pro každou hodnotu prvního indexu dostávám 16 různých možností, jak zvolit další dva indexy. Poté, co zafixuji i druhý index mi zbývají další 4 hodnoty. Prvky tedy můžu do pole uložit tak, aby jejich adresy vracely odpovídající první index jako adresu dělenou 16 (se zbytkem), hodnotu druhého indexu jako zbytek po prvním dělení dělený 4, atd. Takto je možné uložit jednoznačně tenzor libovolného řádu. V případě různých rozměrů pro různé indexy se budou pouze měnit dělitelé v jednotlivých krocích, kteří jsou vždy součinem počtu rozměrů všech následujících indexů. Část kódu vytvářející strukturu adres je k nahlédnutí spolu s vysvětlením v Příloze 1.

Kromě samotných polí a adresové soustavy obsahuje třída tenzoru ještě pole určující rozměry v jednotlivých indexech, defaultní indexy, které budou použity, pokud uživatel při výpočtu žádné nezadá a pole určující, které indexy jsou kovariantní a které kontravariantní.

Na takto uložených strukturách je snadno implementovatelné sčítání a odčítání. Stačí pouze zkontrolovat, zda mají tenzory shodné rozměry a řády a poté projet jednorozměrná pole prvků obou tenzorů a sečíst příslušné prvky na stejných adresách, neboť ty jsou pro stejné rozměry a řád vždy shodné.

Násobení je opět jednoduché, vytvoří se nový tenzor, jehož řád je součtem řádů násobených tenzorů, zkopírují se jejich indexy a polohy a projedou všechny jejich kombinace, pro každou se akorát pronásobí odpovídající prvky a uloží se výsledek (symbolicky, numerické vyhodnocení pak provede samostatný blok kódu, který je součástí jádra programu a stará se i o numerické vyhodnocování matic a vektorů).

Pokud se při násobení objeví dva stejné indexy v opačných polohách, je aplikována Einsteinova sumační konvence. I zde je implementace poměrně prostá, nejprve se najdou všechny dvojice stejných indexů a pak se vždy po jedné odstraňují a provádí se součet přes všechny jejich hodnoty pro všechny kombinace ostatních indexů až do odstranění všech dvojic. Kód provádějící na tenzoru automatickou sumaci je k nahlédnutí spolu s vysvětlením v Příloze 2.

3.4 Symbolické derivování a integrování

Řešení symbolických výpočtů ukážu na derivacích a integrálech. Výraz je nejprve, jak je popsáno výše, rozložen do binárního stromu.

Při derivování jsou rekurzivně zpracovány jednotlivé podstromy, na které jsou podle operátorů v uzlech aplikována příslušná pravidla pro derivování. Výsledek je poté odeslán na zjednodušení.

Při integrování je postup složitější, protože neexistuje žádná obecná metoda. Existuje Rischův algoritmus [8] schopný zintegrovat libovolnou funkci, jejíž integrál je možné vyjádřit pomocí elementárních funkcí a základních operací, ten je ovšem nesmírně složitý a jeho implementace daleko převyšuje rozsah této práce. Proto byly kromě základních tabulkových integrálů implementovány běžně vyučované metody substituce, per partes a parciálních zlomků.

Nejprve jsou z integrálu vyjmuty všechny konstanty. Následuje přepsání všech odmocnin na racionální mocniny a dělení na vynásobení výrazem umocněným na -1. Poté se provede test vůči tabulkovým integrálům. Pokud není daný výraz v tabulce, postupuje se dále a výraz je testován, zda není racionální funkcí.

Pokud je výraz racionální funkce, je provedeno polynomiální dělení a výraz je rozdělen na podíl a zbytek. Podíl je zintegrován pomocí pravidel pro polynomy. U zbytku se hledají kořeny jeho jmenovatele. K tomu slouží vzorce pro jmenovatele řádů 2 a 3, nebo extrakce racionálních kořenů či numerické metody a následné dělení pro vyšší řády. U numerických metod program po nalezení neceločíselného kořenu vždy testuje také nejbližší celočíselnou hodnotu, numerické algoritmy nezaručují vždy úplně přesné řešení. Poté jsou sestaveny všechny jmenovatele parciálních zlomků a příslušná soustava lineárních rovnic pro určení jejich čitatelů. Ta je poté vyřešena pomocí inverzní matice. Zbývá už jen podle tabulek zintegrovat jednotlivé parciální zlomky.

Pokud výraz není racionální funkce nebo integrování parciálními zlomky z nějakého důvodu selže, přichází na řadu substituce. Postupně jsou zkontrolovány všechny podstromy daného výrazu (kontrola je optimalizována proti zbytečným iteracím jako je kontrola podstromu lineární funkce, neboť tu má pro efektivitu smysl vždy substituovat celou). Následně se všechny potenciální substituce vyzkouší a vyberou se ty proveditelné. Ty se následovně ohodnotí podle míry, do jaké výraz zjednoduší (počet uzlů ve stromu) a v získaném pořadí

pošlou znovu na začátek integrační procedury. Pokud je některá z integrací úspěšná, je substituce vrácena pro získání výsledku.

Jako předposlední vyzkouší program metodu per partes, pokud je to možné. Funkce je rozdělena na součin dvou. Opět se vyzkouší všechny kombinace a vyberou se ty proveditelné. Pro každou se pak určí, která z funkcí se bude integrovat a která derivovat. Pro urychlení výpočtu toto program odhadne na základě tabulek hodnotících jednotlivé funkce. Pokud vybraný postup nevede k řešení, je vyzkoušeno ještě prohození obou funkcí. Jestliže první aplikace per partes nevede přímo k výsledku, ale pouze k novému integrálu, je k němu při odesílání na novou integraci přiložena informace o získání z per partes a původním integrálu a při opakovaném použití per partes se kontroluje, jestli se výpočet nevrátil k původnímu integrálu, ten je pak následně snadno spočítán.

Teprve až po vyčerpání všech metod je aplikována linearita a integrál je v případě součtu či rozdílu rozdělen na dva. Použití na začátku by mohlo znemožnit některé substituce a nevedlo by tak k výsledku. Pokud je integrál nalezen, je odeslán na zjednodušení. V průběhu integrování program také přehledně zapisuje jednotlivé kroky, které si poté uživatel může zobrazit.

3.5 Měřicí hardware

Měřicí hardware vychází z open-source platformy Arduino, konkrétně desky Arduino UNO. Pro snížení výrobní ceny však byly vypuštěny všechny komponenty, které pro daný účel nejsou potřebné. Mikrokontrolér ATMEGA328P-PU tvořící jádro celého systému byl naprogramován pomocí Arduino IDE a celý obvod byl postaven na prototypové pájecí desce (perfboard) a následně zasazen do ochranné krabičky. Díky použitému designu je možné po nahrání řídícího kódu místo měřicího hardware použít přímo samotné desky Arduino, které mají mnohé školy ve své výbavě.

ZÁVĚR

Cílem práce bylo vytvořit softwarový systém, který by efektivním způsobem kombinoval matematický a výpočetní software s nástrojem pro sběr dat ze senzorů a byl svým ovládáním dostupný i běžným středoškolákům. Cíle se podařilo dosáhnout jak s měřicí částí programu, která při zachování všech podstatných funkcí a rozumného rozlišení představuje řádově levnější alternativu ke komerčním systémům, tak i v oblasti matematického softwaru, který již v současnosti disponuje bohatým výčtem funkcí pokrývajících celou středoškolskou matematiku a množství témat z vyšší matematiky.

Software je neustále dále rozvíjen, obohacován o nové funkce a optimalizován z hlediska výkonu a uživatelského rozhraní. Díky své všeobecnosti je použitelný v prakticky libovolném oboru. Mezi hlavní cíle do budoucna patří dokončení vlastního vestavěného programovacího jazyka, jeho další vývoj a přenesení celé aplikace na další platformy, tj. tablety a chytré telefony, spojené s tvorbou webové formy celého softwaru.

4 POUŽITÁ LITERATURA

1. REKTORYS, Karel. Přehled užité matematiky. 7. vyd. Praha: Prometheus, 2000. Česká matice technická (Prometheus). ISBN 80-7196-180-9.

2. REKTORYS, Karel. Přehled užité matematiky. 7. vyd. Praha: Prometheus, 2000. Česká matice technická (Prometheus). ISBN 80-7196-181-7.

3. HAMHALTER, Jan a Jaroslav TIŠER. Diferenciální počet funkcí více proměnných. Vyd. 2. Praha: Česká technika - nakladatelství ČVUT, 1997. ISBN 80-01-03356-2.

4. HAMHALTER, Jan a Jaroslav TIŠER. Integrální počet funkcí více proměnných. Vyd. 2. Praha: Česká technika - nakladatelství ČVUT, 1997. ISBN 80-01-03357-0.

5. VIRIUS, Miroslav. C# 2010: hotová řešení. Brno: Computer Press, 2012. K okamžitému použití (Computer Press). ISBN 978-80-251-3730-7.

6. SHARP, John. Microsoft Visual C# 2010: krok za krokem. Brno: Computer Press, 2010. Krok za krokem (Computer Press). ISBN 978-80-251-3147-3.

7. PIRKL, Josef. Řešené příklady v C#, aneb, C# skutečně prakticky. České Budějovice: Kopp, 2005. ISBN 80-7232-265-6.

8. *Risch Algoritm* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-04-23]. Dostupné z: https://en.wikipedia.org/wiki/Risch_algorithm

5 SEZNAM OBRÁZKŮ

6 PŘÍLOHA 1: ZDROJOVÝ KÓD UKLÁDÁNÍ TENZORŮ

```
[Serializable]
  public class varArray
  {
      public varArray(int Rank, int[] Dims)
      {
          rank = Rank;
          dims = Dims;
          if (rank == dims.Length)
          {
              CreateInnerArray();
          }
          else
          {
              throw new ArgumentException("Rank and dims don't match!");
          }
      }

      public int rank;
      public int[] dims;
      public int[] adresses;

      public string[] vals;
      public double[] Dvals;

      private void CreateInnerArray()
      {
          int total = 1;
          foreach (int i in dims)
          {
              total *= i;
          }
          vals = new string[total];
          Dvals = new double[total];
          for (int i = 0; i < vals.Length; i++)
          {
              vals[i] = "";
          }
          adresses = new int[dims.Length];
          for (int i = adresses.Length - 1; i >= 0; i--)
          {
              int mult = 1;
              for(int j = i; j < adresses.Length-1; j++)
              {
                  mult *= dims[j + 1];
              }
              adresses[i] = mult;
          }
      }

      public int getLoc(int[] inputIndex)
      {
          int loc = 0;
          for (int i = 0; i < inputIndex.Length; i++)
          {
              loc += adresses[i] * inputIndex[i];
          }
          return loc;
      }
```

Vyobrazený kód je pouze úryvkem, třída dále pokračuje metodami pro přečtení či zápis na konkrétní hodnoty indexů. Členy rank a dims určují řád tenzoru, resp. rozměry v jednotlivých indexech. Pole adresses ukládá přístupové dělitele pro jednotlivé indexy. Pole vals a Dvals ukládají symbolickou (textovou) formu jednotlivých prvků a jejich numerickou hodnotu. Metoda CreateInnerArray volaná v konstruktoru třídy na základě daných rozměrů naplní pole adres a připraví pole vals a Dvals. Metoda getLoc pak slouží k získání jednorozměrné adresy odpovídající zadaným hodnotám indexů.

7 PŘÍLOHA 2: ZDROJOVÝ KÓD EINSTEINOVY SUMAČNÍ KONVENCE

```
public Tensor SumOverIndexes()
        {
            List<string> normalIndexes = new List<string>();
            normalIndexes.Add(this.Indexes[0]);
            List<string> dummyIndexes = new List<string>();
            for (int i = 1; i < this.Indexes.Length; i++)
            {
                bool normal = true;
                for (int j = 0; j < normalIndexes.Count; j++)
                {
                    if (normalIndexes[j] == this.Indexes[i] && covariant[j] != this.covariant[i])
                    {
                        normal = false;
                        dummyIndexes.Add(this.Indexes[i]);
                        normalIndexes.RemoveAt(j);
                        break;
                    }
                }
                if (normal)
                {
                    normalIndexes.Add(this.Indexes[i]);
                }
            }
            if (normalIndexes.Count == this.Indexes.Length)
            {
                return this;
            }
            else
            {
                Tensor res;
                res = (Tensor)this.Clone();
                while (dummyIndexes.Count > 0)
                {
                    string remIndex = dummyIndexes.Last();
                    dummyIndexes.RemoveAt(dummyIndexes.Count - 1);
                    List<int> newDims = new List<int>();
                    List<string> newIndex = new List<string>();
                    List<bool> newCov = new List<bool>();
                    int dummy1 = -1;
                    int dummy2 = -1;
                    for (int i = 0; i < res.Dims.Length; i++)
                    {
                        if (remIndex != res.Indexes[i])
                        {
                            newDims.Add(res.Dims[i]);
                            newIndex.Add(res.Indexes[i]);
                            newCov.Add(res.covariant[i]);
                        }
                        else
                        {
                            if (dummy1 == -1)
                            {
                                dummy1 = i;
                            }
                            else
                            {
                                dummy2 = i;
                            }
                        }
                    }
                    Tensor T = new Tensor(res.Rank-2, newDims.ToArray(), newIndex.ToArray(),
newCov.ToArray());
                    int[] idims1 = new int[T.Dims.Length];
                    for (int i = 0; i < idims1.Length; i++)
                    {
                        idims1[i] = T.Dims[i] - 1;
                    }
```

```
                for (int i = 0; i < T.values.vals.Length; i++)
                {
                    int[] finDim = new int[res.Dims.Length];
                    int dummyVal = 0;
                    string resStr = "";
                    while (dummyVal < res.Dims[dummy1])
                    {
                        int lastI = 0;
                        for (int j = 0; j < res.Dims.Length; j++)
                        {
                            if (j == dummy1)
                            {
                                finDim[j] = dummyVal;
                            }
                            else if (j == dummy2)
                            {
                                finDim[j] = dummyVal;
                            }
                            else
                            {
                                finDim[j] = idims1[lastI];
                                lastI++;
                            }
                        }
                        resStr += "(" + res.values.GetElement(finDim) + ")+";
                        dummyVal++;
                    }
                    resStr = resStr.Remove(resStr.Length - 1, 1);
                    T.values.SetElement(idims1, resStr);
                    for (int w = idims1.Length - 1; w >= 0; w--)
                    {
                        idims1[w]++;
                        if (idims1[w] < T.Dims[w])
                        {
                            break;
                        }
                        else if (w == 0)
                        {
                            for (int h = 0; h < idims1.Length; h++)
                            {
                                idims1[h] = 0;
                            }
                        }
                        else
                        {
                            idims1[w] = 0;
                        }
                    }
                }
                res = T;
            }
            return res;
        }
    }
```

Ze zdrojového kódu byly vynechány prázdné řádky, aby mohl být na dvě stránky rozumně rozdělen a zároveň zůstal čitelný. V prvním cyklu se postupně projdou všechny indexy a roztřídí se na neopakované (normální) a opakované (dummy). První index se bere jako normální, u každého dalšího se zkontroluje, jestli už se v normálních nenachází. Jestliže ano, je přesunut do opakovaných. Pokud jsou nalezeny opakované indexy, je nejprve vytvořena kopie původního tenzoru, na které se bude kontrakce provádět. Poté jsou zkopírována všechna pole vlastností původního tenzoru kromě těch odpovídajících poslednímu opakovanému indexu. Je vytvořen nový, o dva řády nižší tenzor. Poté se postupně projdou všechny

kombinace indexů nového tenzoru (k tomu slouží pole idims1) a na každou se uloží výsledek po posčítání přes všechny hodnoty vybraného opakovaného indexu (v textové, symbolické formě, o numerické vyhodnocení se následně postará jiná část kódu). Poslední cyklus posouvá hodnoty idims1 postupně tak, že se projdou všechny možné kombinace indexů. Celý cyklus se opakuje, dokud nejsou odstraněny všechny opakované indexy.