JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

PEDAGOGICKÁ FAKULTA

KATEDRA FYZIKY

Bakalářská práce

Autor: Jiří Uraj

Vedoucí práce: Mgr. Petr Bartoš

2006

Využití Grafical User Interface programového balíku MATLAB při výuce fyziky

Prohlášení:

Prohlašuji, že jsem tuto práci vypracoval samostatně a že jsem všechny použité zdroje uvedl v Seznamu použité literatury na konci této práce. Zároveň povoluji Katedře fyziky PF JU v Č. Budějovicích libovolné využití této práce.

Poděkování

Na tomto místě bych rád poděkoval Mgr. Petru Bartošovi za pečlivé přečtení textu, cenné připomínky a v neposlední řadě i za pomoc pří vývoji jednotlivých aplikací.

Obsah

Úvod

Tato práce se zabývá potenciálem, který se skrývá v aplikaci MATLAB® a možnostmi využití grafického uživatelského rozhraní při použití ve výuce fyziky. Aplikace MATLAB® mě zaujala svými neomezeným možnostmi realizace fyzikálních jevů a mnoha způsoby zpracování a výstupu matematických dat. Tyto důvody mě vedly k výběru bakalářské práce s tématem zabývajícím se právě programováním v MATLABu®.

V první části své práce se věnuji samotné aplikaci MATLAB® a její historii vývoje, popisu pracovního prostředí, kde chci klást důraz na neutuchající možnost expanze celého programu pomocí jednotlivých rozšiřujících balíčku tzv. toolboxů.

Po základním seznámení s programem a jeho funkcí, se chci v následujících kapitolách zaměřit na funkci a možnosti vývoje aplikací za pomoci grafického uživatelského rozhraní. Definovat zásady správného návrhu programu, který by posloužil k výuce fyziky. Problémy, kterým se vyvarovat při psaní samotné aplikace a naopak zdůraznit poznatky které pomohou při didaktice fyzikálních jevů.

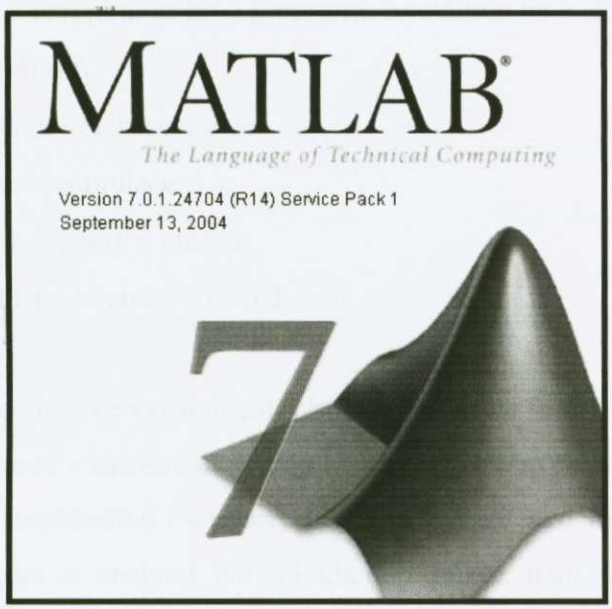
Definované zásady pro návrh výukového softwaru využiji ve třetí části své práce, kde jako praktický příklad uvedu program zabývající se modelováním jevů z mnou vybrané oblasti fyziky. Čtenář si postupně projde všemi stádii při psaní tvorbě programu od matematické přípravy přes grafický návrh programu, psaní zdrojového textu až po jeho samotné odladění v testovacích verzích.

Na závěr práce si ponechám pojednání o možnostech využití takto vytvořených aplikací nejen při výuce, ale pro použití v prezentacích. Tím se dostanu ke srovnání MATLABu® s modelovacím programem pro fyziku FAMULUS®, kde chci porovnat jednotlivá pro a proti obou programů.

Práci jsem rozvrhl tak, aby byla přehledně dělená na hlavní kapitoly, které jsou doplňovány dalšími podkapitolami až s dvouúrovňovým číslováním. Veškeré podkapitoly spojuje společné téma dané hlavní kapitolou.

1. MATLAB©

Aplikace MATLAB® (obr. 1.1) je dnes považována za jedno z nejvýkonnějších interaktivních prostředí pro vědecké a inženýrské výpočty a jejich grafickou prezentaci. MATLAB® v sobě integruje matematické výpočty, numerickou a statistickou analýzu, maticové výpočty, zpracování signálu a grafiku do uživatelsky příjemného, tzv. user-friendly prostředí. Výhodou zpracování těchto dat a úprav je forma zápisu stejná jako v matematické podobě. To nám tím pádem umožňuje při lehčích výpočtech vynechat tradiční programování, což přináší vysoký potenciál přístupnosti MATLABu pro širší vědeckou veřejnost bez nutnosti hlubokých znalostí programovacích jazyků, které se tak většinou stávají těžko zdolatelnou překážkou.



Obr. 1.1: Logo MATLAB®

1. 1. Základní charakteristika aplikace MATLAB

MATLAB® je interaktivní systém, jehož základním datovým prvkem je matice (odtud pochází i pojmenování samotné aplikace z anglický slov *Matrix Laboratory – Maticová Laboratoř*). Za výhodu můžeme považovat, že u této matice se nezadává její rozměr. To umožňuje řešení numerických problémů rychleji, než při užití klasických programovacích jazyků (Basic, Delphi, C), kde se rozměry matic musí na začátku definovat a dále pak naplnit daty většinou za využití programovacích příkazů (cyklů).

Základním datovým typem je tedy, jak jsem již naznačil, dvourozměrné pole (bez nutnosti deklarace jeho rozměrů). Tato charakteristika spolu s dalšími integrovanými funkcemi umožňuje relativně snadné řešení technických problémů, zvláště takových, které využívají vektorovou nebo maticovou formulaci problému. Tím nám MATLAB® umožňuje již jednou zmíněné zrychlení řešení výpočtů.

Typické oblasti použití toho programu rozdělíme na 6 základních skupin:

- Vývoj algoritmů
- Analýza dat a jejich vizualice
- Modelování a simulace
- Inženýrské výpočty a grafika
- Vývoj aplikací včetně GUI rozhraní

Využití nachází ve vysokoškolském prostřední ve formě doprovodného výukového programu v matematice, dále ve specializovaných technických oborech a dokonce se s ním setkáváme i v průmyslovém sektoru, jako velice efektivní nástroj pro vývoj, výzkum a analýzu dat. Neklamný důkaz nám mohou podat mnohé publikace, které se zabývají výukou i řešením daných problémů za pomocí MATLABu nebo jeho doprovodných funkcí a balíčků.

Přídavné balíčky (dále jen Toolboxy) pro MATLAB® se staly "elixírem mládí" celého nástroje. Toolbox je soubor m-file skriptů obsahující funkce psané a tykající se různých vědních oborů a tím dává neutuchající možnosti základní verzi programu se neustále rozvíjet a pro vědecké účely tak nezestárnout, jak se tomu děje u jiných aplikací.

Poslední nezanedbatelnou součástí celého vývojového prostředí je přítomnost samostatné nadstavby MATLABu – SIMULINK, sloužící pro řešení soustavy nelineárních diferenciálních rovnic s grafickým zadáváním řešené soustavy připomínající zapojení na analogovém počítači. Podrobněji tuto nadstavbu rozvedu v jedné následující kapitole.

Ještě bych se rád zastavil u problému, kdy se najde uplatnění MATLABu® a kdy jej tedy použít. Jak jsem předestřel, MATLAB® má mnoho směrů použití. Mezi ty nejzajímavější a nejběžnější mohu uvést:

- Zápis a výpočty symbolické matematiky s definovanou přesností
- Použití MATLABu jako WEB serveru pro zajištění výpočtů v rámci WWW stránky
- Kompilace hotové aplikace v MATLABu (včetně grafického uživatelského rozhraní – GUI) do spustitelného tvaru bez jakékoliv další přítomnosti jádra MATLABu
- V novějších verzích překlad algoritmů do funkce v jazyce "C"
- Propojení s aplikacemi MS Excel® a MS Word® umožňující dále zpracování protokolu o vývoji či měření

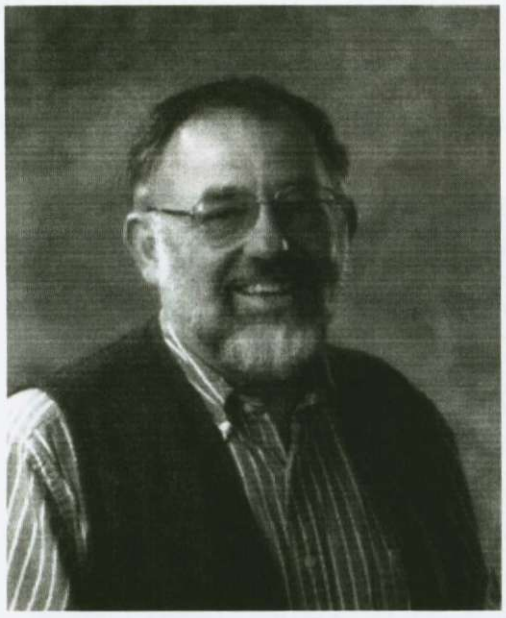
Odpověď na otázku správného načasování využití MATLABu tedy není jednoduchá. Ze všech vlastností MATLABu vyplývá, že se nejlépe hodí pro masivní výpočty a zpracovávání rozsáhlých datových souborů, a operace s vektory či maticemi. Lze jej tedy využít všude tam, kde nezáleží na rychlosti výpočtu stále stejného objemu dat. Nehodí se však pro dynamický vývoj algoritmu a jeho úpravu pro momentální použití.

1. 2. Historie a vývoj aplikace MATLAB

MATLAB® byl napsán, aby poskytoval jednoduchý přístup k matematickým knihovnám vyvinutých v projektech LINPACK a EISPACK. Jeho primární určení bylo pro operační systém UNIX a tato vlastnost jej doprovází až dodnes i v prostředí MS Windows®, které se projevuje velmi jednoduchým komunikačním rozhraním – příkazovým řádkem.

Historie MATLABu začíná v polovině 70. let ve Spojených Státech a to právě zmíněným vývojem matematických knihoven LINPACK a EISPACK pro programovací jazyk FORTRAN®. Vývoj byl veden týmem v čele s Clevem Molerem (obr. 1.2) za podpory Národní Vědecké Nadace (*National Science Foundation*). LINPACK je kolekce knihoven pro FORTRAN® zahrnující postupy řešení lineárních rovnic a ve spojení s balíčkem knihoven EISPACK dopomáhá k jejich numerickým řešením. Tyto dva soubory knihoven se staly dohromady základním stavebním kamenem pro vývoj softwarových architektur maticových výpočtů.

Na konci 70. let Cleve Moler, který se stal vedoucím katedry počítačového



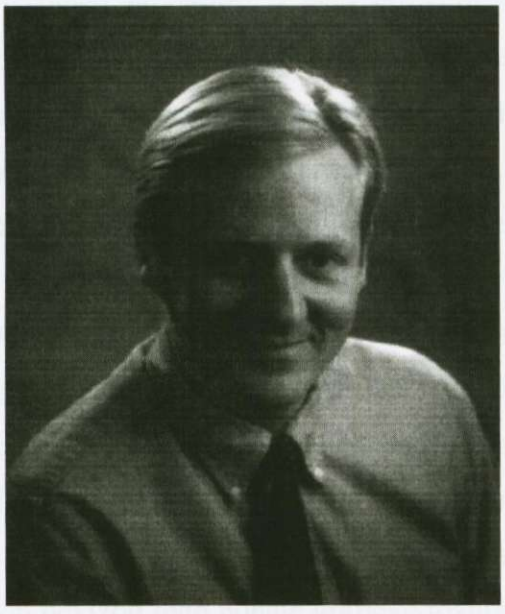
Obr. 1.2: Cleve Moler

výzkumu (*Computer Science Department*) na Univerzitě v Novém Mexiku (*University of New Mexico*), dostal myšlenku využít knihoven LINPACK a EISPACK k výuce posluchačů kurzu lineární algebry. Základním problémem se mu

stal samotný programovací jazyk FORTRAN®. Ten se svou složitostí nehodil do osnov výuky předmětu, ale zároveň pro využití knihoven byl nepostradatelným.

Proto Cleve Moler ve svém volném čase začal vyvíjet program, který by umožnil studentům jednoduchý přístup k funkcím obsaženým v LINPACK a EISPACK knihovnách a pojmenoval jej MATLAB® ze slov MATrix LABoratory. V následujícím několika letech Cleve Moler na svých stážích na různých univerzitách v USA rozšiřoval a vylepšoval svoji aplikaci na univerzitních počítačích. Během dalšího roku se začal MATLAB® šířit mezi studenty a profesory všech univerzit v USA, ale také i v Evropě. Díky jeho jednoduchosti v přístupu k jednotlivým funkcím a zápisu se z nástroje pomalu stává fenomén.

Z kraje roku 1983 byl inženýr John Little (Obr. 1.3) seznámen s aplikací MATLAB® při návštěvě Cleva Molera na Standfordské Univerzitě a brzo pochopil, že se rodí nový jazyk pro vývoj inženýrských aplikací. John Little během téhož roku dal dohromady nový tým a za přispění Cleva Molera a Steva Bangerta vyvinuli druhou generaci, nyní již profesionální podobu aplikace MATLAB®. Celá aplikace byla napsána v jazyce C a již měla integrovanou grafickou formu výstupu. O rok později, v roce 1984 tito tři kolegové (Cleve Moler, John Little, Steve Bangert) založily firmu Mathworks a vstoupili tak na softwarový trh se svou aplikací MATLAB® v jeho novějších a rychle se rozvíjejících verzích.



Obr. 1.3: John Little

Původní předurčení aplikace MATLAB® pro OS Unix se brzy změnilo a s příchodem nových operačních systémů se rozšířil o dalších 8 platforem.

- MS Windows®
- MS Dos®
- LINUX®
- Solarius®
- IRIX®, IRIX® 64
- AIX
- HP-UX

První verze pro PC XT se objevila v roce 1985. Prvotním problémem byl nedostatek pamětí a z toho plynuly problémy s omezením velikostí matic. Přirozeným vývojem se zanedlouho objevil PC AT a ten se již stal více atraktivním pro MATLAB®. Zde byla velikost matice omezena celkovou fyzickou pamětí (pro PC AT max. 16 MB). Vzhledem k tehdejším cenám pamětí (tento neduh je provází až dodnes díky vysokým výrobním nákladům) nebylo zvykem osazovat počítače větší pamětí. Jednou z nejoblíbenějších verzí před OS MS Windows® se stala verze MATLAB386®, která byla vyvinuta primárně pro počítače PC s procesorem 80386. Využívala jednu z vlastností procesoru – jeho virtuální paměť. To znamená., že program pracuje s virtuální pamětí, která může dosáhnout větší velikosti, než osazená fyzická operační paměť v PC. Pracuje tak, že dochází k průběžnému ukládání dat na pevný disk a k jejich zpětnému čtení. Tento proces je vykoupen zpomalením výpočtů zásluhou rychlosti disků, ale zároveň je možné operovat s velkými maticemi (např. u PC 386/40 s 4 MB fyzické paměti bylo možné provést výpočet inverzní matice o velikosti 1000x1000. Samotná matice však potřebuje pro uložení 8 milionů bytů tj. skoro 8 MB). Poslední verze MATLAB386® g z října roku 1995 se využívá v některých případech na počítačích pracujících pod operačním systém MS DOS® dodnes. Je to díky relativně rychlému zpracování výpočtů dat pod tímto systémem.

Během roku 1994 byla na trh zavedena nově vyvinutá verze MATLAB for Windows®, jejíž hlavní devízou bylo bohaté grafické rozhraní. To vše bylo však opět zaplaceno na úkor rychlosti výpočtů (dokonce v mnoha případech byly výpočty

pomalejší než u verze MATLAB386® na totožných počítačích). Vývoj MATLAB for Windows® byl ukončen verzí 4.2c z října roku 1996.

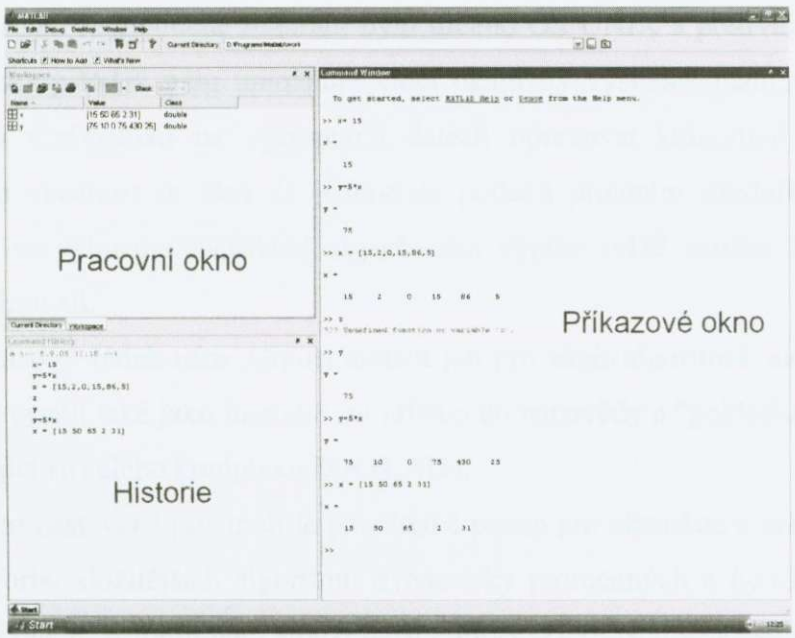
Plnou 32 bitovou podporu získal MATLAB®, v rámci platformy Intel ve své verzi 5, kdy jeho funkčnost byla podřízena OS MS Windows® 95 a MS Windows® NT. Poslední release verzí se stala z konce roku 1999 verze 5.3. Tato verze je již plně 32 bitová a integruje v sobě novou možnost objektově orientovaného programování – tvorba nových datových struktur, nové vizuální funkce, silná podpora toolboxů v základní verzi a rychlejší grafika.

Po verzi 5 následovaly další, a to v podobě MATLAB® 6 a MATLAB® 6.5 a jejich release verzi až do současné poslední verzi MATLAB 7.01 se Service Packem 3.

1. 3. Uživatelské rozhraní v MATLABu

Uživatelské rozhraní v průběhu let procházelo změnami, které byly většinou vyžádané z potřeb při psaní algoritmů pro zvýšení efektivity a rychlosti. V této kapitole se zaměřím na prostředí známé od verze MATLAB 6.0.1. Tedy plně 32 bitové verze pro osobní počítače s operační systémem MS Windows 2000 a vyšší.

Po spuštění aplikace se nám zobrazí úvodní logo MATLAB, během kterého probíhá v pozadí procesu načítání knihoven. Po této akci se zobrazí základní pracovní prostředí (obr 1.4). V tuto chvíli jsme již schopni začít tvořit algoritmy popř. aritmetické výpočty pomocí výpisů příkazů na příkazový řádek.



Obr. 1.4: Základní pracovní prostředí MATLABu

Základní prostředí je složeno z pracovního okna, historie a z příkazového řádku, které jsou po instalaci standardně ukotveny a vždy spouštěny. Dále zde můžeme najít volbu umístění pracovního adresáře, základní menu a nabídku Start umožňující např. spouštění a vkládání toolboxů, přístup do nápovědy apod. Jelikož je MATLAB velice user-friendly, umožňuje všechny tyto okna libovolně přesouvat, ukotvovat, měnit velikost, zavírat nebo otevírat jiná pro další akce.

Tato základní okna jsou určena pro přehlednost zápisu a rychlou dostupnost informací ve skriptu. Pracovní okno (*Workspace*) slouží jako seznam použitých a

deklarovaných proměnných či konstant s jejich základními vlastnosti jako jsou jméno, aktuální hodnota, velikost a datový typ. Takto můžeme jednoduše editovat jednotlivé proměnné a vytvářet jejich grafy několika typů. Pracovní okno může také zobrazovat při přepnutí záložky aktuální pracovní adresář, odkud lze spouštět jednotlivé M-file soubory či MAT soubory bez nutnosti spouštění průzkumníka.

Okno Historie (*Command History*) zaznamenává všechny vypsané příkazy v příkazovém okně. Tím se stává dobrým pomocníkem při hledání chyb a navazování při přerušení práce díky vlastnosti záznamu příkazů od prvního spuštění MATLABu. Navazování funguje, pokud si uživatel sám historii nevymaže, např. po dokončení projektu. Z okna historie můžeme jednotlivé příkazy kopírovat a dále používat v příkazovém řádku.

Základem celého ovládání a práce s aplikací již od její první verze se stal příkazový řádek. Jednoduché rozhraní bylo určeno OS UNIX a přetrvalo do dnešní doby. Příkazový řádek nám umožňuje vidět okamžitý výsledek námi vytvářeného algoritmu a v závislosti na výstupních datech upravovat jednotlivé koeficienty zápisu. Tato vlastnost se však dá jednoduše potlačit přidáním středníku za každý výraz. Tím se vyhneme například zbytečnému výpisu velké matice, kterou jsme právě nadefinovali.

Příkazový řádek nám nemusí sloužit jen pro zápis algoritmů, ale máme zde možnost jej využít také jako interaktivní přístup do nápovědy a "pokládat" jednotlivé dotazy týkající se celého komplexu MATLABu.

Takto nastavené prostředí je použitelné pouze pro okamžité a krátké výpočty a ne pro tvorbu složitějších algoritmů dynamicky proměnných a použitelných pro běžného uživatele. K tomu nám slouží další editory, které jsou součástí MATLABu. Tyto součásti si v následujících podkapitolách rozebereme.

1. 3. 1. M-file editor

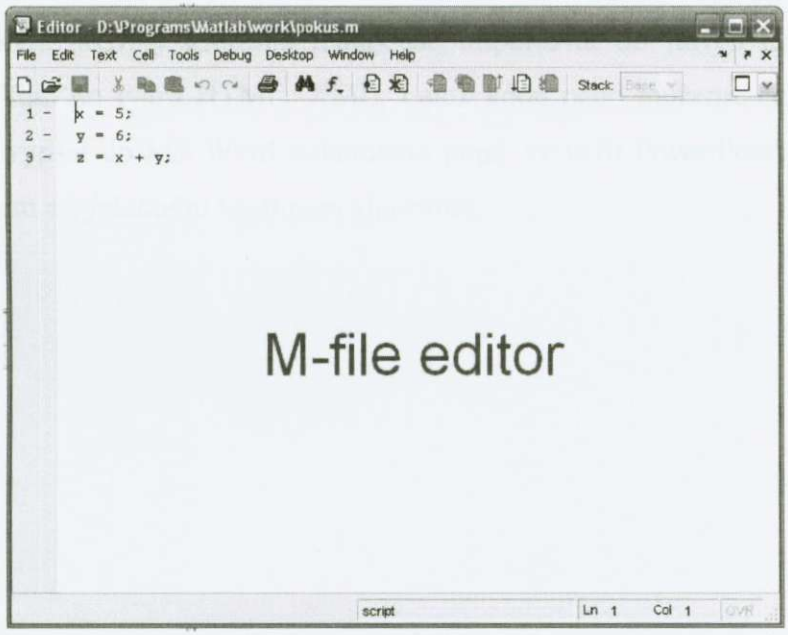
Než začneme rozebírat prostředí a funkci M-file editoru, osvětlím vlastní podstatu M-file souborů. Ve své podstatě se jedná o běžný textový soubor, ve kterém jsou na jednotlivých řádcích zapsány příkazy námi psaného algoritmu se všemi proměnnými, konstantami, funkcemi, procedurami, které může skript obsahovat. Soubor je v počítači uložen s příponou *.m* a je upravitelný jakýmkoliv textovým editorem. Spustit však samotný skript lze pouze za pomoci M-file editoru nebo z příkazového řádku zadáním příkazu v níže uvedeném příkladu (1).

run jméno *souboru* (1)

Jednou z možností tvorby a editace skriptů resp. M-file souborů je použití integrovaného editoru v MATLABu. Z příkazového řádku jej spustíme jednoduše zadáním příkazu (2) či vybrání z menu **File -> New -> M-file**.

edit *pokus* (2)

tímto jsme vytvořili nový soubor v editoru s názvem **pokus.m.** V tuto chvíli se spustí jednoduchý editor (obr. 1.5) velice podobný poznámkovému bloku v MS Windows.



Obr. 1.5: M-file editor

Editor můžeme spustit, aniž bychom zadali jméno souboru, a poté otevřít již stávající M-file a ten editovat. K těmto úkonům nám slouží základní menu.

Celé rozhraní editoru je složeno z pracovní plochy vlastního editoru, menu, nástrojové lišty s nejpoužívanějšími ikonami a kontextové lišty.

Kromě základních úkonů známé uživateli z běžných textových editorů, od práce se souborem (např. **New File, Open File, Close File, Save as, Print, Preference, Copy, Paste, Find**) přes úpravu umístění jednotlivých oken, nabízí výbava M-file editoru také speciální nadstandardní funkce.

Jednotlivé napsané skripty lze okamžitě testovat, krokovat a spouštět. Všechny tyto pomůcky umožňují odladit psané algoritmy. Každou řádku lze označit červeným, resp. zeleným praporkem pro vynechání, resp. začlenění při spuštění skriptu. Lze také nastavit bod začátku/konce konce skriptu pro co nejvyšší komfort a rychlý pohyb v psaní M-file souborů.

Při psaní rozsáhlých M-file skriptů nám pro lepší orientaci v celém textovém souboru slouží lišta s nástrojem zaznamenávání nových funkcí procedur a událostí, které byly v souboru vytvořeny. Díky jednoduchému seznamu umožňuje rychlý pohyb po celém souboru. Uživatel tak neztrácí přehled v celém souboru a skriptu, pokud je jeho soubor rozsáhlejší.

Vytvořené skripty můžeme jednoduše importovat do jiných jazyků nebo editorů, například do tvaru HTML, XML, Latex kódu nebo můžeme celý zapsaný skript nechat vypsat do MS Word dokumentu popř. vytvořit PowerPoint prezentaci s celým zápisem a výsledným výstupem algoritmu.

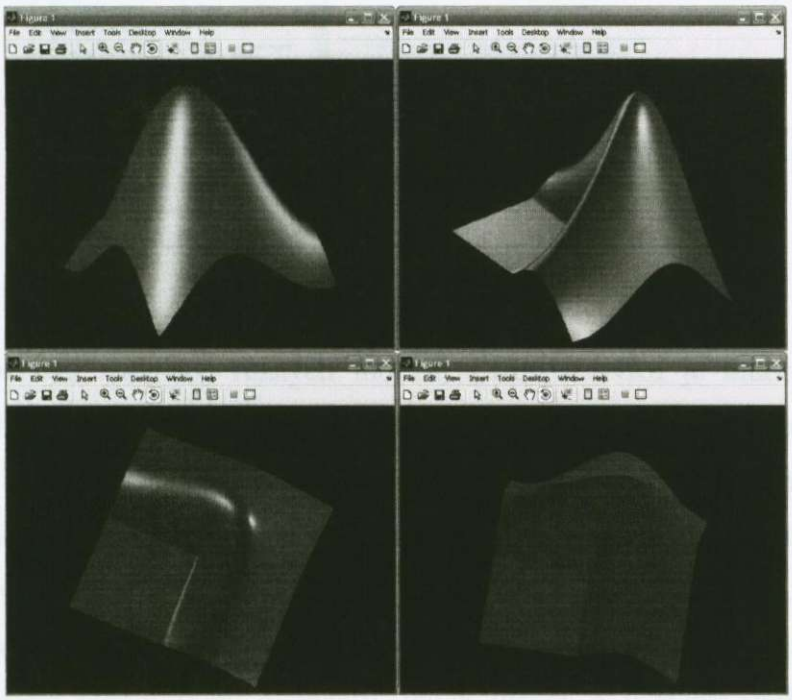
1. 3. 2. Figure

Figure je dalším prvkem tvořící celý matematický komplex MATLAB® určený k vytvoření grafických objektů. Ty nadále slouží jako výstup u algoritmů pracujících s grafickým výsledkem.

Objekt Figure vytvoří aplikace po zadání položky z menu **File – > New – > Figure** na ploše nové okno připravené k zobrazení grafických výstupů. Velikost, vlastnosti okna, grafu, síla čar, osy jsou uživatelsky přednastavené a jsou popsané základními parametry a dají se tak upravovat dle potřeby.

Figure je vybaven sadou nástrojů uzpůsobených k práci s grafikou. Jednotlivé modely lze tak upravovat za pomocí nástrojové lišty či menu. Princip je tedy velmi podobný jako u M-file editoru a komunikace je tudíž jednoduchá. Výsledné výstupy lze ukládat do souboru popřípadě je importovat do formátu souboru kompatibilních v jiných aplikacích a využít pro další zpracování (např. import do JPG/BMP formátu obrázku a následného užití v MS Word).

Pokud však chceme zpracovávat grafické objekty ve Figure, musíme do něj importovat data. Nejjednodušší způsob je načtení již existujícího modelu, přičemž model musel být již dříve vytvořen. Proto je Figure úzce navázáno na funkce MATLABu, které zajišťují vykreslování objektů (Obr 1.6).

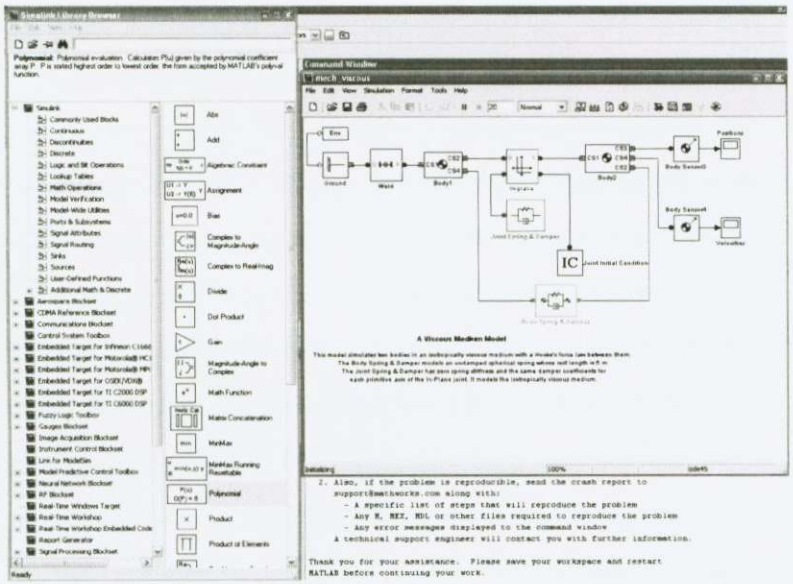


Obr. 1.6: Figure

1. 3. 3. Model

Další z nových objektů, které lze vytvářet v MATLABu stojí tak trochu stranou. Zatímco se všechny nové projekty týkaly striktně MATLABu samotného, objekt Model je nástrojem části zvané SIMULINK. Jeho přesnou funkci a logiku popisuji v další kapitole. Zde si řekneme o samotném návrhu a tvorbě jednotlivých modelů.

Klasicky po vybrání položky z menu **File – > New – > Model** se před námi otevře nové "plátno", kde můžeme začít s návrhem. Než však tuto funkci přiblížím, zkráceně osvětlím o jaké modely se jedná. SIMULINK dokáže vytvářet diskrétní a lineární systémy, které jsou reprezentovány např. simulací automobilových brzd, převodovky, pohyby planet, stavba a chování zesilovačů, D/A a A/D převodníků.



Obr. 1.7: Knihovna prvků

Možností simulace různých modelů v SIMULINKu je mnoho, základem jejich návrhu je stavba z jednotlivých diskrétní nebo lineárních elementárních prvků. Jednotlivé prvky vkládáme z knihovny (Obr 1.7), její otevření musí uvést uživatel sám – ikonou na plovoucí liště, kde máme jednotlivé elementy rozděleny do skupin, dle svých vlastnosti. V 37 skupinách se tak skrývá cca na 800 prvků. Umisťování probíhá přetahováním myší z knihovny na plátno. Po umístění je nutné nastavit propojení mezi jednotlivými elementy, jejich vstupy a výstupy společně s jejich nastavením vlastností. Tyto akce opět probíhají za jednoduché použití myši

(kliknutím a tažením pro propojení prvků; dvojklikem pro nastavení vlastností prvků). Celý návrh se tak podobá programování a konfiguraci analogových počítačů.

Po dokončení designu modelu můžeme provést jeho spuštění/simulaci a za pomoci ladících nástrojů provést odstranění chyb popř. nastavení jednotlivých hodnot prvků a po-té jej uložit ve formátu souboru s příponou *. *MDL*. Takovýto soubor můžeme zakomponovat do vyvíjených skriptů či aplikací

1. 3. 4. Variable

Z anglického názvu vyplývá, že se tato položka nového projektu bude zabývat proměnnými.

V hlavním prostředí MATLABu máme možnost po jeho spuštění založit nový soubor, který v sobě bude shromaždovat nadefinované proměnné. Za pomoci menu **File – > New – > Variable** nás MATLAB přesune do okna Workspace a vytvoří první nepojmenovanou proměnnou. Workspace slouží jako pracovní prostor, kde si uživatel sám nadefinuje všechny potřebné proměnné a veličiny s příslušnými vlastnostmi. To znamená., že můžeme definovat hodnotu proměnné, zda se jedná o matici a pohodlně ji naplnit vestaveným editorem, který se podobá tabulkovému procesoru. Jednotlivá data můžeme načítat ze souborů s použití ikon na plovoucí liště, načíst lze soubory v ASCII kódu a MATLAB je schopný je podle inteligentního algoritmu sám rozdělit na budoucí proměnné. Poslední slovo a volbu má uživatel. Celý projekt pak lze dále uložit ve formátu MAT souboru a kdykoliv jej načíst do paměti pracovního prostoru aplikace za současného vymazání stávajících proměnných.

Vystává tak otázka, na co si ukládat proměnné do souboru a později je vyvolávat. Jednu nespornou výhodu to však má. Představme si rozsáhlý projekt v MATLABu s velkým počtem proměnných. Širokou část zdrojového textu bychom zaplnili jen definováním proměnných a konstant a jednoduchý zdrojový kód se tak snadno stane nepřehledným. Dalším argumentem je snadná editace MAT souboru a "plnění" jednotlivých proměnných rozličnými daty. Na příklad nám počítač může ukládat V-A charakteristiku do souboru *VA_data.txt*. Ten není problém za pomoci Workspace načíst popřípadě poupravit a nechat dále zpracovat vybraným algoritmem.

Stejně jako většina šablon projektů lze vyvolat z příkazového řádku, dává MATLAB možnost načítání proměnných také tímto způsobem a to za pomocí příkazů: *load* (pro načtení proměnných ze souboru) a *save* (pro uložení do formátu MAT)

1. 3. 5. GUI

Jako poslední šablonu pro vytvoření nového projektu v rozhraní MATLAB uvedu GUI (Grafical User Interface). V této kapitole nastíním pouze stručný popis funkce a využití. Podrobnější vysvětlení funkce a popisu vývojového prostředí věnuji následující kapitolu.

Možnost vytvářet GUI aplikace nebyla vždy součástí MATLABu. Teprve s příchodem operačního systému MS Windows a první verze MATLABu pracující pod tímto systémem, se objevila možnost navrhovat pomocí grafického enginu aplikaci. V této době se jedná o nově rozvíjející se trend objektově orientovaného programování. Firmy zabývající se vývojem programovacích jazyků rychle využívají této možnosti a své programovací jazyky nabízejí v nových verzích s možnosti tvorby GUI aplikací.

Grafický návrh aplikace dává uživateli (programátorovi) možnost jednoduššího a rychlejšího vývoje aplikací. Za pomoci jednotlivých utilit starajících se o vlastnosti jednotlivých objektů, se návrh aplikace stává přehlednějším a efektivnějším z hlediska designu. Rozměry jednotlivých oken, tlačítek, popř. dialogů si jednoduše nastavíme tažením myší za okamžitého viditelného výsledku. GUI programování tak přináší komfort do programování.

Vývoj GUI aplikací se však nesestává pouze z designu komponent. Stále zde zůstává nutnost ručního psaní jednotlivých funkcí, rutin či událostí. V této oblasti již ale došlo také ke zlepšení a to ve formě již vytvořených událostí (funkce vytvořené pro určený děj v počítači – kliknutí myší, přetažení okna).

Jak jsem již zmínil, návrh GUI aplikací nebyl pouze trendem firmy Mathworks®, ale s tímto druhem objektově orientovaného programování přišli i jiní softwaroví giganti (Microsoft®, Borland® Enterprise). Teprve po nich přichází MATLAB ve verzi 5 s rozšířenou možnosti vývoje GUI.

1.4. Toolboxy – rozšiřující balíčky aplikace MATLAB

Toolboxy jsou základní filosofií celého systému MATLAB. Již v kapitole o historii programu jsem se zmínil, že prvotní verze aplikace fungovala na základě přístupu do matematických knihoven LINSPACK a EISPACK. Přístup se prováděl přes výpis příkazů na příkazovém řádku. Tento způsob a systém v MATLABu funguje i dnes. Toolbox je rozšiřující balíček zejména m-file souborů obsahující funkce a procedury pro práci s daty. Jelikož funkce MATLABu je velice univerzální nebylo problémem postupně přidávat nové a nové knihovny funkcí. Tak se základní prostředí MATLABu podařilo rozšířit a zvýšit uplatnění ve více odvětvích, která se zabývají výpočty či zpracováním dat a dopomoct tak k naučení a aplikování v praxi různé technologie. Z původního řešení lineárních rovnic postupně přibyly knihovny funkcí zabývající se problémy mnoha jiných odvětví jako jsou statistika, finanční matematika a zpracování databází.

Hlavní výhodou je možnost vývoje vlastních toolboxů a pozdější integrace do celého systému MATLABu. Každý uživatel má tak možnost vytvářet balíčky funkcí daného oboru a pak je následovně využívat při psaní aplikací.

Během let vývoje MATLABu vzniklo mnoho toolboxů, které se staly součástí již základní instalace prostředí MATLABu. Současně vznikaly taky další knihovny, které byly vyvinuty instituty nebo jedinci pro řešení speciálních problémů. Většina takovýchto toolboxů je volně ke stažení na internetu, kde se během let vytvořily rozsáhlé databáze. Další možností je zakoupení těchto knihoven (záleží na distribuci a využití autora toolboxu).

Omezíme-li se na standardně dodávané toolboxy k MATLABu najdeme funkce zabývající těmito obory:

- **Bionformatics Toolbox** – sada nástrojů zabývající se genetickým inženýrstvím, biologickým a chemickým výzkumem z oblasti organické chemie
- **Communications Toolbox** – balíček rozšiřující MATLAB o funkce projektování, návrhu a simulace komunikační sítí a systémů. Dopomáhá k tvorbě algoritmů pro bezdrátové/drátové technologie
- **Control System Toolbox** – kolekce m-file algoritmů se stará o návrh řízení, analýzu a modelování systému. S pomocí GUI (grafického uživatelského prostředí) usnadňuje návrh a modelování

- **Curve Fitting Toolbox** – sada GUI aplikací a m-file skriptů, která umožňuje předzpracování dat a (bez)parametrické dělení dat. Výběr a úprava dat se provádí několika způsoby, např. polynomicky, exponenciálně, racionálně, Gaussovým dělením
- **Data Acquisition Toolbox** – toolbox MEX a m-file souborů zajišťující práci a zpracování získaných dat z měřících přístrojů připojených k PC
- **Database Toolbox** – jeden z rozšiřujících balíčků, který dává možnost importu/exportu dat z/do nejpoužívanějších databázových programů. Pohodlně se dají zpracovat data v MATLABu a poté je možné je exportovat do formátu uživatelem zvolené databáze.
- **Datafeed Toolbox** – sada funkcí úzce propojená na výše zmíněnou databázovou sadu umožňuje přemístění velkého množství dat (např. z finanční oblasti), importovat ho do prostředí MATLABu a aplikovat další balíčky.
- **Filter Design Toolbox** – Set nástrojů poskytující pokročilé techniky designu, simulace a analýzy digitální filtrů. Výrazně tak rozšiřuje možnosti Signal Processing Toolboxu
- **Filter Design HDL Coder** – balíček urychlující vývoj a design aplikací, jejich ladění a modelování, aplikací určených pro integrované obvody a polem řízená logická hradla generování HDL (Hardware Description Language)
- **Financial Toolbox** – toolbox poskytující plně integrované prostředí do MATLABu zabývající se ekonomickou analýzou a inženýrstvím, včetně statistických a matematických aplikací, grafických výstupů apod.
- **Financial Derivates Toolbox** – balíček rozšiřující možnosti Financial Toolbox o přidané nástroje obsahující funkce úrokových sazeb a majetkových práv.
- **Financial Time Series Toolbox** – kolekce nástrojů umožňující práci a analýzu dat proměnných v čase zabývající se peněžním obchodem.
- **Fixed-Income Toolbox** – sada funkcí určených pro výpočty a odhady návratnosti hypotéčních a dlužních úvěrů, manipulaci s cennými papíry a měnou.
- **Fixed-Point Toolbox** – matematický set funkcí a procedur definující algoritmy pracující s pevnou řádovou čárkou, umožňující vytvářet fixed-point číselné objekty, jejich rozměry vlastnosti apod.
- **Fuzzy Logic Toolbox** – toolbox pracující tzv. fuzzy logikou (matematická metoda práce s umělou inteligencí => používá se tam, kde se dá problém popsat

pouze obecně, ale konkrétní řešení není možné z důvodu velkého rozptylu hodnot), umožňující vytvářet systémy a simulace použitelných nadále v SIMULINKu

- **GARCH Toolbox** – optimalizační a statistický balíček nástrojů tvořící prostředí pro modelování nestálosti univalentních ekonomických časových úseků. Balíček v sobě kombinuje možnosti předpovědí, simulace, parametrické chování vybraného a modelovaného ekonomického systému.
- **Genetic Algorithm And Direct Search Toolbox** – sada funkcí rozšiřující základní matematické funkce za použití dvou algoritmů (Genetic Algorithm – vývojový; Direct Search – přímý)
- **Image Acquisition Toolbox** – funkce nabízející široké možnosti získávání obrazových dat z profesionálních zařízení až po běžné uživatelské nástroje (USB webkamery), živé sledování videa či export obrazových dat do pracovního prostředí MATLABu
- **Image Processing Toolbox** – tento balíček zahrnuje rozšiřující metody práce s obrazovým materiálem (daty) a jejich následovné úpravy, jako je filtrování obrazu, transformace, analýza, ostření fotek, redukce barev apod.
- **Instrument Control Toolbox** – toolbox rozšiřující MATLAB o vývoj struktur komunikační protokolů (TCP/IP UDP), práci s *plug&play* funkcemi, funkce pro přenosy dat mezi vlastními zařízeními nebo programování událostí.
- **Link For Code Composer Studio™** – set funkcí určený pro komunikaci s Code Composer Studio™ , je zde možnost obdržená data ukládat do paměti
- **Link For ModelSim®** – je rozhraní integrující simulační a designové prostředí pro logická hradla a integrované obvody
- **Mapping Toolbox** – poskytuje komplexní sadu funkcí a GUI aplikací pro vývoj prezentace zeměpisných map, analýzu geostatických dat podle volených parametrů
- **Model Predictive Control Toolbox** – toolbox v sobě zahrnuje software, reprezentovaný ve formě GUI aplikací, napomáhající s designem, analýzou a implementací složitější automatizační algoritmů
- **Model-Based Calibration Toolbox** – balíček nástrojů, který v sobě obsahuje dvě základní interaktivní prostředí pro experimentální statistické modelování a

ladění složitých systémů. Model Browser zahrnuje experimentální modelování a CAGE Browser zajišťuje analytickou kalibraci systémů

- **Neural Network Toolbox** – aplikační balíček pracující s matematickými modely neuronových sítí, jejich konstrukcí, řízením a linearizací regulátorů
- **OPC Toolbox** – implementuje do MATLABu objektově orientovaný přístup ke komunikaci s OPC servery za použití OPC Data Access Standard (komunikační protokol OPC serverů) v oblasti automatizace
- **Optimalization Toolbox** – kolekce nástrojů a rutin sloužící k optimalizaci včetně neomezené nelineární minimalizace, lineární a kvadratické programování, nelineární řešení rovnic
- **Partial Differential Equation Toolbox** – matematická sada nástrojů poskytující komplexní řešení diferenciálních rovnic včetně jejich grafické reprezentace.
- **RF Toolbox** – toolbox funkcí, který slouží k tvorbě a kombinování RF (Radio Frequency) okruhů pro jejich simulaci. Za pomoci jednotlivých návrhů získáváme možnost návrhu a odzkoušení bezdrátových širokopásmových sítí TV/Rádio/LAN/Celulárních sítí
- **Robust Control Toolbox** – RCT je sadou nástrojů umožňující návrh a sledování MIMO (Multi-Input-Multi-Output – systémy s velkým datovým tokem) systémů
- **Signal Processing Toolbox** – toolbox vyvinutý na matematickém jádru MATLABu, který zahrnuje široké odvětví signálových a tvarových generátorů pro spektrální analýzu
- **Statistics Toolbox** – balík nástrojů obsahující velké množství matematických funkcí ze statistického oboru. Tím umožňuje velké možnosti zpracování a vyhodnocování měřených dat
- **Symbolic Math Toolbox** – kolekce s více jak 100 matematickými funkcemi umožňující tvar syntaxí užitých v jádru MAPLE za použití základního jazyka MATLABu
- **System Identification Toolbox** – sada nástrojů a rutin umožňující vývoj modelů systémů postavených na základech vstupních a výstupních informací
- **Virtual Reality Toolbox** – toolbox, který nabízí základní řešení interaktivních modelů virtuální reality a jejich simulaci za pomoci SIMULINKu

- **Wavelet Toolbox** – set nástrojů, který umožňuje analýzu a syntézu různorodých signálů a obrazů (vibrace, seismické otřesy, lidská řeč, lékařské snímky) s možností statistického vyhodnocení dat

1. 5. Tvorba zdrojového kódu (skriptů a funkcí)

Práci s MATLABem z hlediska tvorby zdrojových kódů můžeme rozdělit do dvou částí. Pro jednoduché či pokusné výpočty nejčastěji využijeme interaktivní režim programu tj. příkazové řádky, kdy každý příkaz po jeho dokončení je ihned proveden (dokončením chápejme stisk klávesy enter za napsaným příkazem). Velmi často se však setkáme s problémem, kdy potřebujeme vytvořit posloupnost příkazů a tu nadále opakovat (z rozličných důvodů např. změna hodnot proměnných nebo pozdější prezentace). Neustálý opakovaný výpis příkazů a plnění proměnných novými hodnotami je časově velice náročné. V této situaci je nezbytně nutné využít psaní skriptů nebo funkcí. Ty se vytvářejí v textové podobě za pomoci libovolného textového editoru. Ve Windows můžeme využít Poznámkový blok (standardní součást OS MS Windows). MATLAB® od verze 5 má integrovaný editor/debbuger (*M-file editor*).

1. 5. 1. Skript

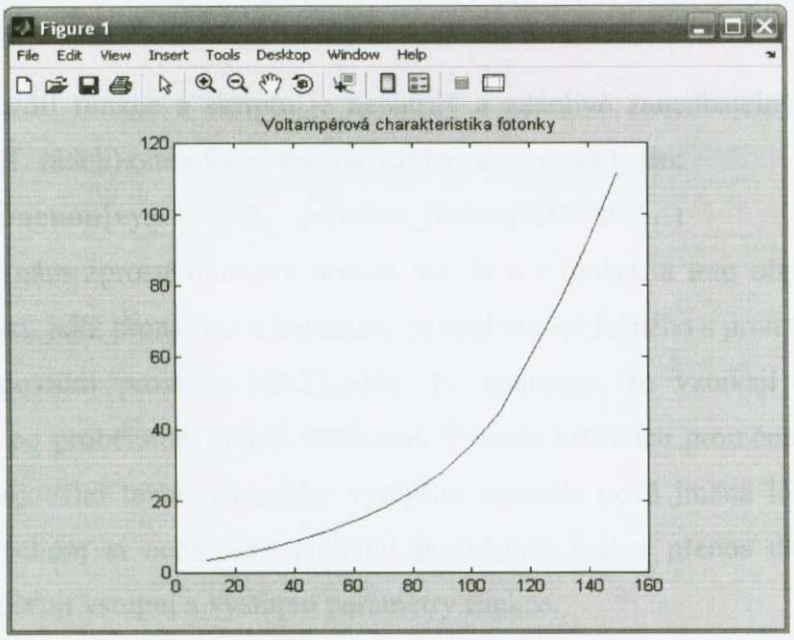
Skript je posloupnost příkazů a proměnných, které provádí činnost se stále stejnými vstupními daty. Ve spojení s MATLABem je skript posloupností příkazů uloženou v textovém souboru končící s příponou *.m*. Takto uložený soubor můžeme kdykoliv vyvolat pomocí příkazového řádku MATLABu tak, že napíšeme jméno souboru (bez přípony). Vyvoláním skriptu proběhnou všechny zapsané příkazy a všechny proměnné a konstanty po skončení běhu skriptu zůstanou zachovány a mohou být nadále měněny bez jakéhokoliv vlivu na skript.

Existence skriptů může najít využití např. ve zpracování souboru naměřených dat. Pro příklad si uvedeme jednoduché zpracování dat na výstup ve formě grafu. Měření se zabývá voltampérovou charakteristikou fotonky za konstantního osvětlení. Jako vstup použijeme soubor *data.dat*, do kterého jsme předtím do dvou sloupců vložili získané hodnoty proudu I a napětí U a na výstupu získáme graf V-A charakteristiky.

Spustíme v MATLABu M-file editor (příkaz **edit**, popř. můžeme využít jiného textového editoru) a vložíme do něj následující řádky zdrojového kódu:

```
load data.dat;              %nacteni dat ze souboru
y=data(:,1);                %ulozeni 1. sloupce do matice
x=data(:,2);                %ulozeni 2. sloupce do matice
plot(x,y)                   %vykresleni grafu
title('Voltampérová charakteristika fotonky');
axis([0,160,0,120]);
axis square;
save new_dat.dat -ASCII; %ulozeni novych dat do souboru
```

Poté soubor uložíme a z příkazového řádku nebo stiskem klávesy F5 v M-file editoru spustíme skript jehož výsledkem je graf (obr. 1.8).



Obr. 1.8: Voltampérová charakteristika fotonky

Tímto způsobem můžeme efektivně generovat grafy k naměřeným hodnotám a dále je zpracovávat.

1. 5. 2. Funkce

Efektivnějším nástrojem k řešení úloh a problémů s pomocí MATLABu je psaní funkcí. Mnoho toolboxů (rozšiřovací balíčky pro MATLAB) je řešeno ve formě m-funkcí.

Hlavní rozdíl funkce a skriptu je nepatrný a zdánlivě zanedbatelný. Je to hlavička funkce (1. řádek) ohraničená rezervovaným slovem ve tvaru:

function[vyst1,vyst2,...]=*jmeno_funkce*(vst1,vst2,...)

Hlavička funkce nám zprostředkovává přenos dat do a z funkce a toto ohraničení nám vytváří funkci, jejíž proměnné a konstanty nemají nic společného s proměnnými v globálním pracovním prostoru MATLABu. To znamená, že vznikají lokální proměnné, které po proběhnutí funkce zaniknou. Výhoda lokálních proměnných je, že při psaní zdrojového textu nemusíme vymýšlet neustále nová jména lokálních proměnných, abychom si nepřepsali globální proměnné. Jediný přenos dat do a z funkce nám zajišťují vstupní a výstupní parametry funkce.

Hlavička začíná klíčovým slovem function, dále následuje název vstupních proměnných a rovnítko. Za ním se vypíše jméno funkce, které musí být shodné se jménem souboru, ve kterém je funkce uložená a zároveň nesmí nabýt názvu rezervovaných slov MATLABu. Po jménu funkce je v kulatých závorkách uzavřen seznam vstupních parametrů.

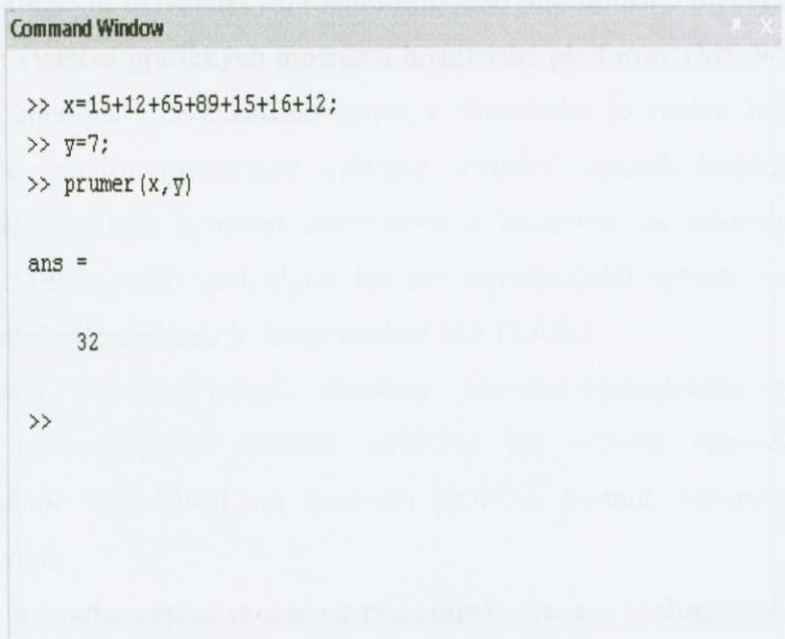
Nyní (po nadefinování hlavičky) se již píší příkazy funkce, které tvoří tělo funkce. Uvnitř funkce používáme vstupní a pomocné proměnné sloužící pro výpočty. Výsledný výpočet konané procedury se ukládá do výstupního parametru a jeho hodnota se přenese do proměnné, která byla použita na místě příslušného parametru funkce v programu.

Pro pochopení rozdílu mezi funkcí a skriptem uvedu příklad se zdrojovým kódem pro porovnání:

```
function prum=prumer(soucet,pocet)  %hlavicka funkce
x=soucet/pocet;                     %vlastni vypocet, lokalni promenna
                                     x, vstupni promenne soucet, pocet
prum=x;                             %ulozeni do vystupni promenne
```

```
x=15+12+65+89+15+16+12;
y=7;
prumer(x,y)
```

V první části je napsaná funkce pro vypočet průměrné hodnoty, ta je uložena na disku pod jménem *prumer.m*. Druhá část ukazuje vyvolání funkce v příkazovém okně MATLABu. Po spuštění dostaneme tento výstup (obr. 1.9).



Obr. 1.9: Command Window po inicializaci funkce prumer

1. 6. Simulink

SIMULINK® (Obr 1.10) postupně přerostl z knihovny funkcí určené k simulaci jednoduchých lineárních spojitých a diskrétních systémů v samostatný subsystém s dokonalým uživatelským rozhraním. Jeho přítomnost v MATLABu je až od verze 4, kde využívá grafických možností hostitelské platformy (MS Windows®). Způsob zápisu modelu i celý způsob práce v Simulinku je tvořen bloky, které reprezentují elementární dynamické systémy. Odlišný způsob komunikace od MATLABu tak svádí jej používat samostatně a nezávisle na znalostech práce s MATLABem. Toto využití však slouží jen pro nejjednodušší systémy a modely a pro řešení složitějších problému je nutná znalost MATLABu.

Simulink je určen pro řešení – simulaci – chování dynamického systému při znalosti jeho matematického modelu. Můžeme tak určovat časové průběhy výstupních hodnot v závislosti na časovém průběhu hodnot vstupních s jejich počátečním stavem.

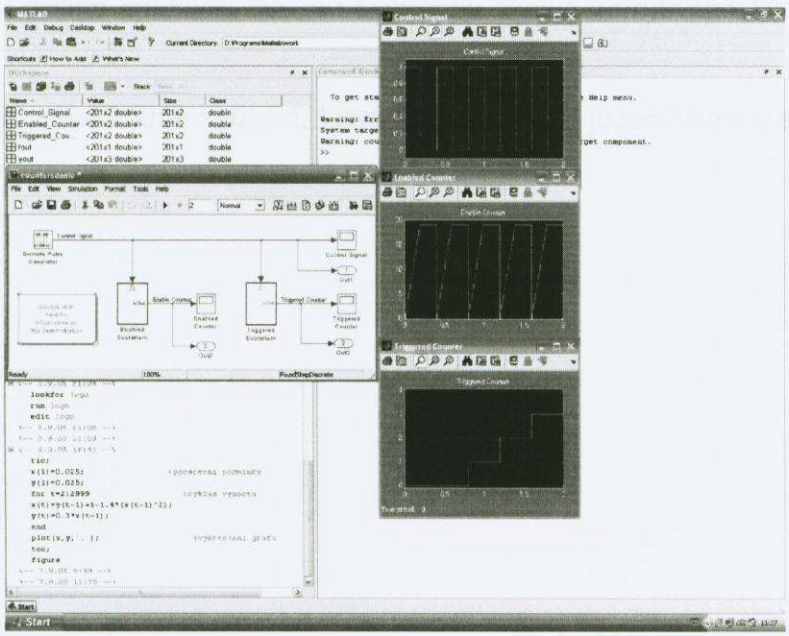
Přístup k návrhu zápisu problému připomíná zapojení analogových počítačů. Celý zápis probíhá graficky tvorbou tzv. modelu. Z nabídek knihoven se myší umisťují jednotlivé bloky a následovně se pospojují ve vstupy a výstupy.

Propojením signálových vstupů a výstupů těchto bloků vznikají modely složitějších systémů. Libovolnou skupinu bloků lze uzavřít do subsystému a určit externí vstupy a výstupy této skupiny. Dále lze pracovat s takovouto skupinou jako se základními blokem. Je li potřeba zastínit proměnné parametry bloků uvnitř skupiny, lze uzavřenou skupinu zamaskovat a doplnit informacemi, které vytvoří při modifikaci parametrů bloku dotazový dialog a postarají se o přepočítání a přenesení zadaných parametrů dovnitř do zamaskované skupiny. Pro zamaskovanou skupinu lze také vytvořit grafickou reprezentaci skupiny, která také může být závislá na nastavených parametrech. K výpočtům parametrů lze užít všech forem výrazů a volání funkcí, které MATLAB® umožňuje.

Simulace chování dynamických systémů nesestává jen z řešení soustav nelineárních diferenciálních rovnic (ač patři mezi jedny z nejdůležitějších částí). U každého modelu řešíme problémy od výběru vhodné metody řešení a nastavení počátečních stavy přes synchronizaci výpočtu na zadaný čas až po řešení nespojitých nelinearit.

Většinu systémů, které mohou být řešeny v diskrétním čase (lze je diskretizovat), je možno přes RTW převést přímo do zdrojového kódu v jazyku C, který lze po doplnění zdrojovými texty pro uživatelem definované speciální funkce přímo kompilovat do strojových jazyků počítačů určených k řízení. Pak vytvoření i vcelku složitého systému může vypadat tak, že návrhář sestaví model navrhovaného požadovaného systému z předdefinovaných bloků SIMULINKu a po stisknutí jediného tlačítka (je-li již správně zvolena konfigurace) dojde ke kompletnímu překladu a stažení výsledného kódu do řídící jednotky.

SIMULINK® vedle své výbavy standardních knihoven a rozšíření o toolboxy, je výhodný v tom, že jsou v tomto prostředí podporovány funkce a příkazy dostupné z prostředí MATLABu.



Obr. 1.10: Ukázka SIMULINK aplikace

2. GUI – Grafical User Interface

Ačkoliv se návrh a tvorba GUI aplikaci objevily teprve nedávno a může se tak zdát, že je zatím nedostatečně prozkoumána nebo propracovaná, opak je pravdou. Vývoj a i samotné užívání GUI v sobě kombinuje 3 základní vlastnosti: **Jednoduchost**, **Spojitost**, **Známost**. Pod jednotlivými vlastnostmi si můžeme představit jednotnost, spojitost, přímost, integritu, harmonii, eleganci, pohodlí, přátelský přístup.

Tyto vlastnosti nám však MATLAB při vývoji GUI aplikací pouze propůjčuje, a jak jsou využity, závisí ve velké míře na samotném autorovi programu. Pro atraktivně vyhlížející a precizně fungující aplikaci je nutné držet se pravidel ať již pro správný design aplikace, tak pro samotnou stavbu zdrojového kódů a jeho ladění.

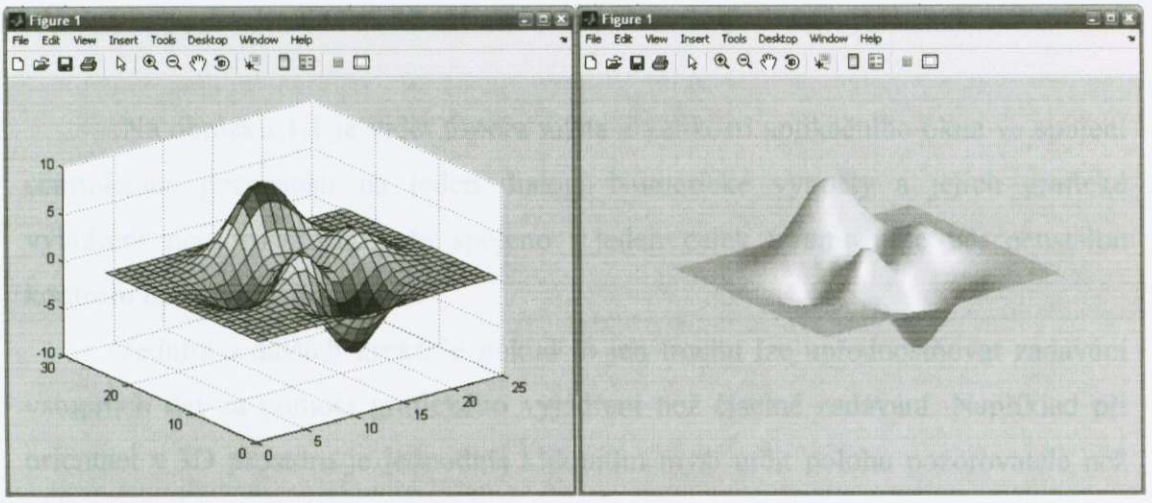
Pak teprve získáváme aplikaci, která v sobě spojuje jednoduchost, efektivitu, eleganci.

2. 1. Princip designu aplikací

Před vývojem aplikací založených na grafickém kontextu je nutné si stanovit pravidla designu takového programu, abychom se vyhnuli nezájmu takového programu, uživatel nesmi získat pocit, že aplikace není určena jemu nebo že je pro něj zbytečně složitá. Proto jsou stanovena všeobecná základní pravidla designu aplikací, která platí ve vše objektově orientovaných programovacích jazycích.

Návrh programu je jednoduchý, není problém přidávat jednotlivé funkce a dialogy, ale měli bychom se vyhnout nepřehledné změti tlačítek na obrazovce. Jen jednoduchá, přímá a uspořádaná aplikace je přístupná každému uživateli. Příkladem můžou být nepřehledné grafy, málo může někdy znamenat víc (viz obr.1.1)

U grafického výstupu si musíme při vývoji programu ujasnit na jakých



Obr. 2.2: Přehlednost aplikace

informacích nám záleží, pokud jde o tvar grafu, není nutné zatěžovat uživatele nadbytečnými numerickými hodnotami, postačí holá prezentace grafu a naopak, znázornění grafu může být jednoduší, ale doplněné o osy se stupnicí.

Dalším pravidlem u návrhu aplikací, kterých bychom se měli držet je soustředění pozornosti pouze na jedno místo – pracovní plochu. Dělení na více aktivních zároveň pracujících oken, které nám poskytují důležité informace zapříčiňuje neefektivnost projektu. Uživatel se soustředí nejlépe na jedno okno aplikace, avšak při neustálém dělení pozornosti na více objektů mu unikají důležité informace (viz obr.)

Obr. 2.3: Minimalizace prostorových nároků

Na obrázku 1.1 je vidět úspora místa a velikosti aplikačního okna ve spojení centralizace pozornosti na jeden dialog. Numerické výpočty a jejich grafické vyjádření není rozděleno, ale spojeno v jeden celek a umožňuje tak neustálou kontrolu nad během programu.

Jedním z dalších kroků je pokud to jen trochu lze upřednostňovat zadávání vstupních dat za pomoci grafického vyjádření než číselné zadávání. Například při orientaci v 3D prostoru je jednoduší kliknutím myší určit polohu pozorovatele než vkládat číselné souřadnice polohy.

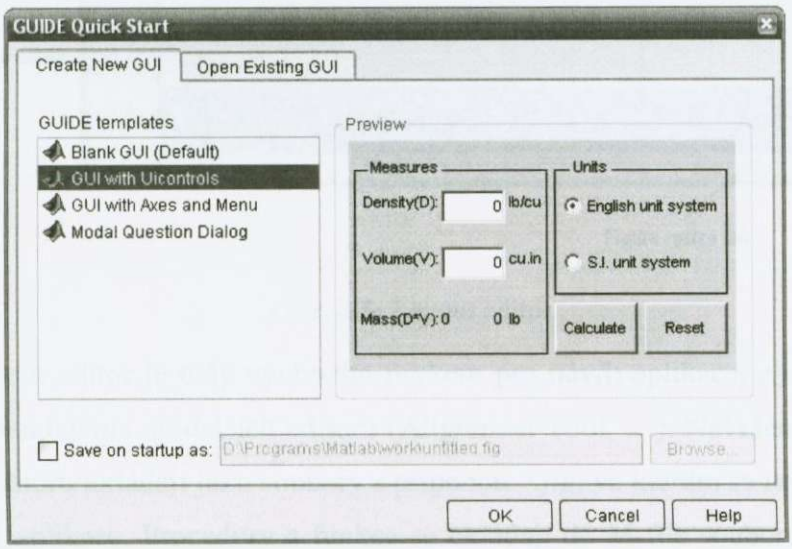
Při tvorbě aplikace je důležité neuchýlit se k přílišné odbornosti (např. ve smyslu nastavení vstupních parametrů), aby se uživatel nepřestal orientovat v hlavní funkci programu tzn. udržet tak spojitost celého projektu. Aplikace by měla tvořena ve všeobecně známém konceptu, design a prezentace funkcí, tak jak je známe v reálném životě je mnohem efektivnější než vytvářet vlastní pojetí zobrazení (viz obr. mince)

2. 2. Vývojové prostředí GUIDE

Aplikace GUI se v MATLABu dají psát dvěma způsoby. První, těžší, je vývoj celé aplikace pouze za pomocí M-file editoru. To ale není v rozsáhlosti jednotlivých deklarací objektů a jejich vlastnosti dost dobře možné. Druhou možnosti tedy je využití editoru GUIDE (Obr. 2.4) v kombinaci M-file editoru.

GUIDE je tedy matlabovské vývojové prostředí pro GUI programy. Kombinuje v sobě možnosti vytvářet a upravovat uživatelský interface a to prostřednictvím list boxů, pull-down menu, push buttonů, radio butonů, checkboxů, sliders a dalších jak bývá zvykem a standardem řady objektově orientovaných vývojových prostředí. GUIDE prostředí se skládá z:

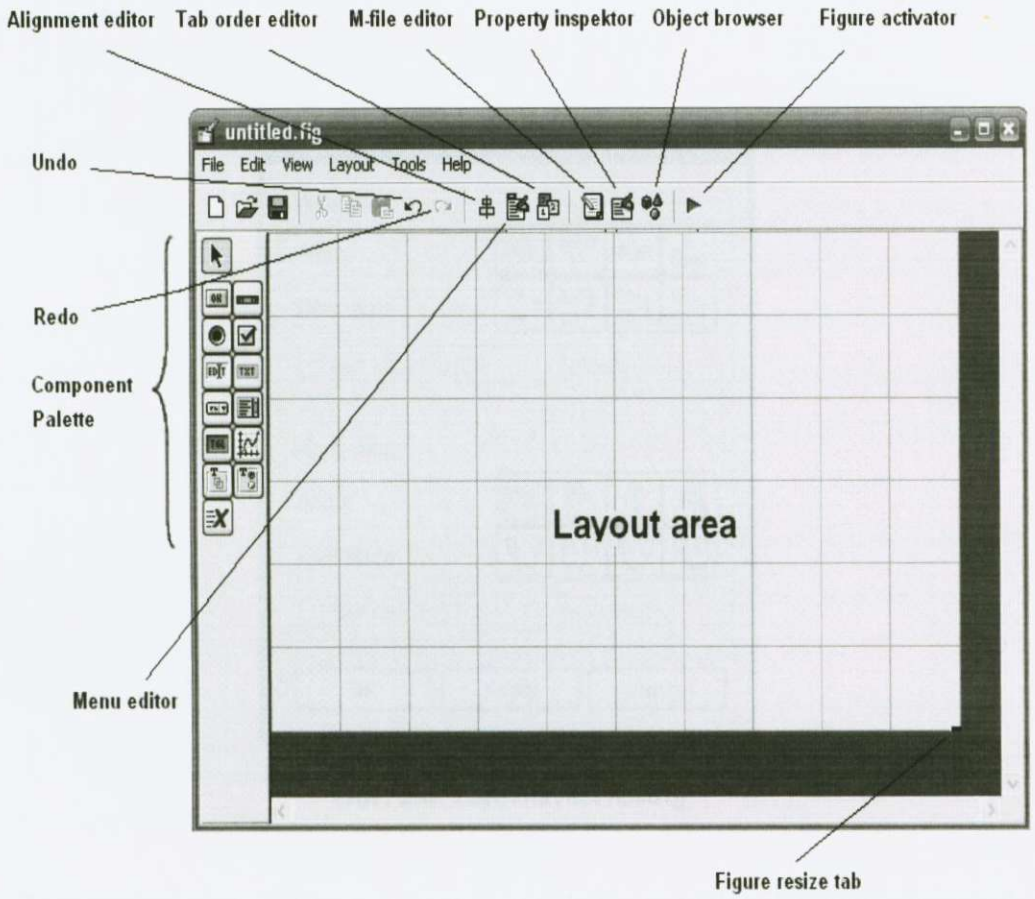
- **Layout editor** – pracovní plocha pro přidávání a uspořádání objektů v aplikaci
- **Aligment tool** – nástroj pro zarovnávání a konečně doladění pozice jednotlivých komponent
- **Property inspektor** – utilita umožňují jednoduché editování vlastností jednotlivých komponent
- **Object browser** – nástroj pro sledování hierarchické struktury Handle Graphics objektů
- **Menu editor** – editor pro tvorbu menu



Obr. 2.4: GUIDE průvodce

2. 2. 1. Layout editor

Layout editor (Obr. 2.5) je ovládací panel pro editor GUIDE, spustit se dá za pomoci příkazu *guide* nebo z menu **File – > New – > GUI**. Než se však dostaneme do samotného Layout editoru, předchází mu dialogové okno jednoduché průvodce z možnosti nabídky otevření již existujícího projektu nebo možnost výběru jednoho ze 4 základní konceptu pracovní plochy od prázdné po dialogová okna a grafické výstupy. Z každé této volby po potvrzení výběru se dostaneme do Layout editoru. Ten nám umožňuje vybírání komponent z palety a jejich následné uspořádání.



Obr. 2.5: Layout editor

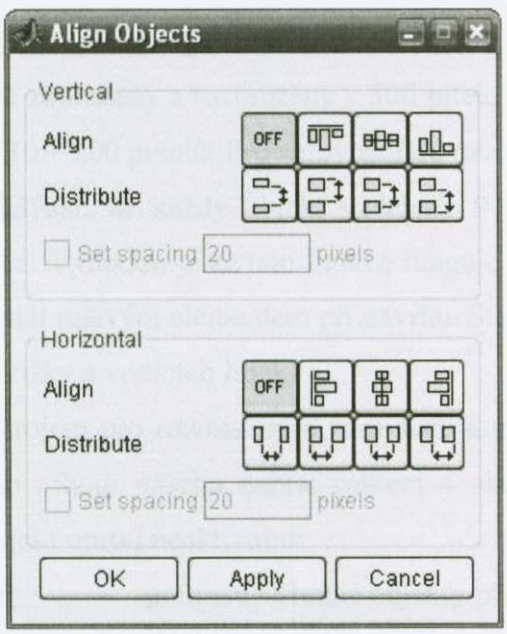
Layout editor je tedy výchozím prvkem pro návrh aplikace, z jeho prostředí snadno expandujeme do dalších editorů (Alignment Tool, ...). Navržené aplikace se v Layout editoru ukládají jako soubory s příponou **.fig*, ve kterém se nachází binární obsah GUI aplikace. Procedury a funkce se ukládají do M-file souboru. Ty lze pak spustit za pomoci *Figure Activator.*

2. 2. 2. Alignment tool

Alignment tool (Obr. 2.6) slouží ke konečnému design a umístění jednotlivých komponent. V Layout editoru jednoduše „naházíme" komponenty a za pomoci nástrojů Alignment tool dokončíme jejich přesné zarovnání.

K tomuto zarovnání nám dopomáhají:

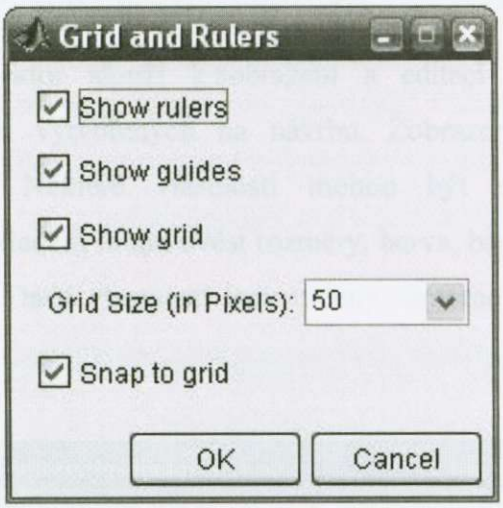
- **Vodící linky** – umožňují přichycení komponenty k vodícím linkám v jakémkoliv místě
- **Mřížka a pravítko** – přichycení komponenty k mřížce dle zvolených kroků
- **Přenést před/za na popředí/pozadí** – uspořádání komponent ze předu do zadu



Obr. 2.6: Zarovnávací nástroj

Zarovnávací nástroj dává možnost pravidelného rozdělení komponent v vertikální a horizontální poloze. V obou směrech rozlišujeme dva úkony:

- **Zarovnání (Align)** – zarovnání komponent (tlačítek, textboxů) k referenční čáře
- **Rozmístit (Distribuce)** – rozmístění komponent rovnoměrně mezi sebe o zvolené rozestupy v pixelech



Obr. 1.1: Mřížka a pravítko

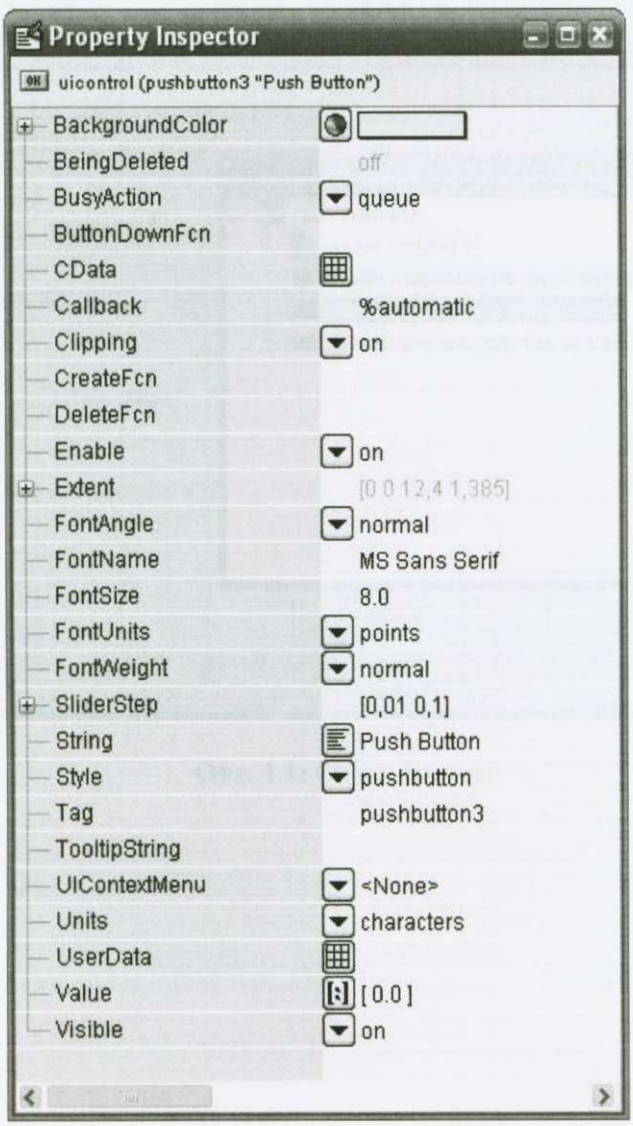
Mřížka a pravítko při designu aplikace velice usnadňují zarovnání. Čáry mřížky jsou standardně zobrazeny a rozmístěny v 50ti pitelových intervalech, rozpětí intervalu lze měnit od 10 – 200 pixelů. Pro zarovnání do roviny je zde možnost kroku dle mřížky, který zajišťuje, že každý objekt vzdálený 9 pixelů od čáry mřížky, přiskočení k této mřížce. Výhodou je že tato funkce funguje i bez zobrazené mřížky, kdy by se čáry mohly stát rušivým elementem při návrhu. Stejně tak můžeme nastavit viditelnost pravítka, mřížky a vodících linek.

Posledním nástrojem pro rozmístění a zarovnání komponent je přenášení ze předu do zadu. Režim tohoto návrhu nabízí celkem 4 akce, které řídí jednotlivá přenášení aktivních objekt oproti neaktivním:

- **Bring to Front** – přemístí vybrané objekty před nevybraný objekt
- **Send to Back** – přemístí vybrané objekty za nevybraný objekt
- **Bring to Forward** – přemístí vybrané objekty dopředu o jednu hladinu
- **Send to Backward** – přemístí vybrané objekty dozadu o jednu hladinu

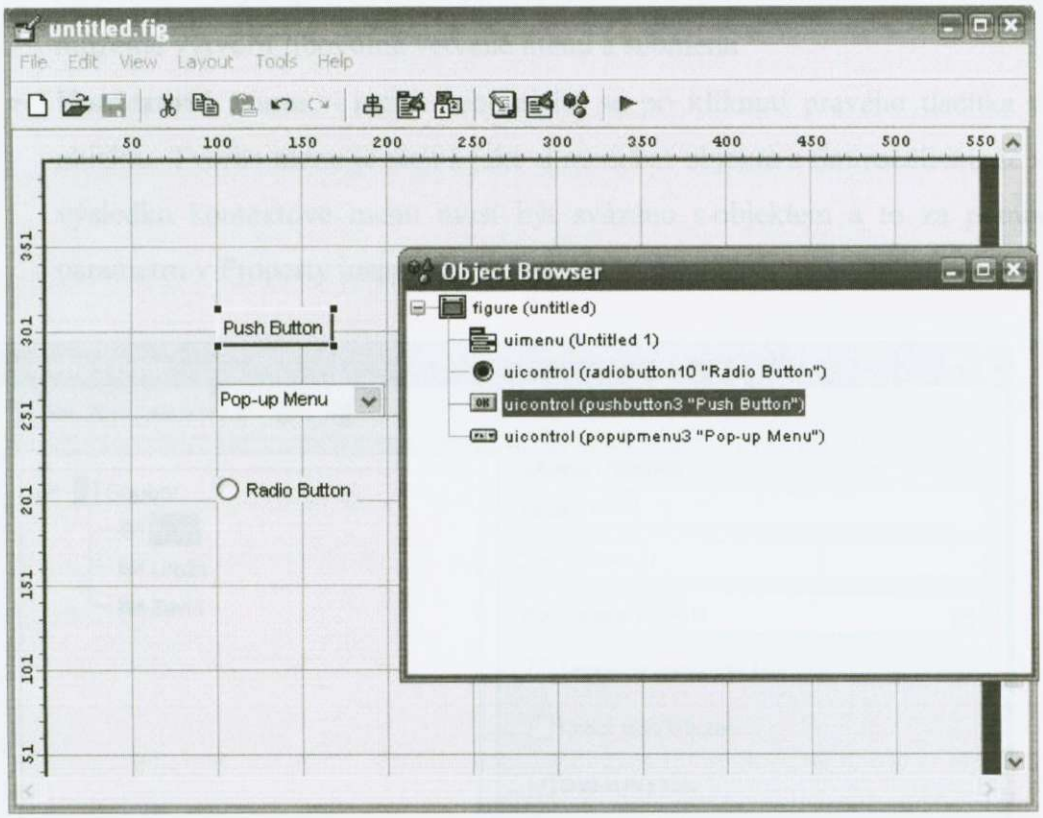
2. 2. 3. Property inspektor a Object browser

Property inspektor slouží k zobrazení a editaci jednotlivých vlastnosti komponent a objektu vytvořených na návrhu. Zobrazené vlastnosti záleží na přiřazeném objektu. Některé vlastnosti mohou být společné pro všechny komponenty, jako příkladem mohu uvést rozměry, barva, barva pozadí, identifikátor, viditelnost, umístění. Další vlastnosti jsou určeny samotnou komponentou a jejím použitím.



Obr. 1.1: Property inspektor

Object browser zobrazuje hierarchickou strukturu objektů v pořadí jak byly vytvářeny a jejich dědičnost apod. Na obrázku je vidět objekt *figure* a jeho potomky (*radiobutton, pop-up menu, ...*) jak byli postupně vytvářeny. Nástroj nám tak poslouží pro jednoduchou orientaci mezi objekty a k snadnému přístupu k jejich vlastnostem přes Property inspektor pouhým poklepáním myši na názvu objektu.

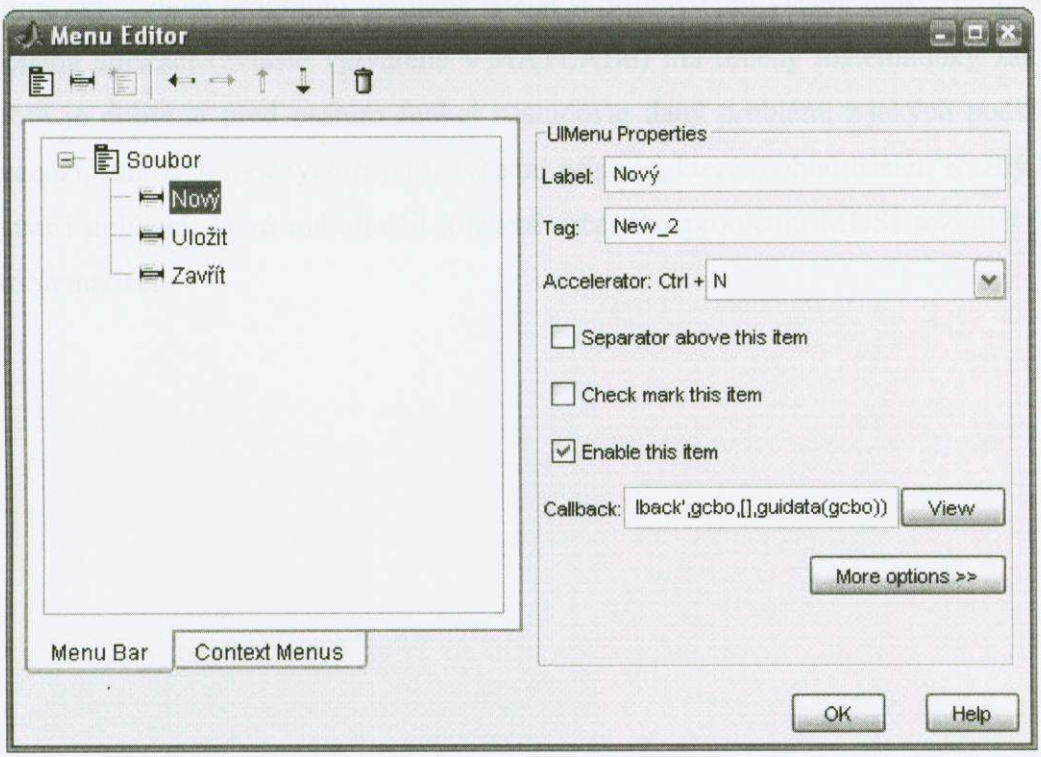


Obr. 1.1: Object browser

2. 2. 4. Menu editor

Většina aplikací se neobejde bez menu, které zajišťuje přehledný a jednoduchý přístup k jednotlivým funkcím programu. GUIDE prostředí nabízí editor, který je uzpůsoben pro návrh takovýchto menu. V základě máme možnost navrhovat dva druhy nabídek:

- **Menubar objekty** – menu zobrazované v horní liště okna, ve kterém můžeme vytvořit libovolně větvené menu a submenu
- **Kontextová menu** – menu zobrazující se po kliknutí pravého tlačítka na objektu. Tvorba menu je stejná jako u menubar objektů s tím rozdílem, že ve výsledku kontextové menu musí být svázáno s objektem a to za pomoci parametru v Property inspektoru *IContextMenu*.



Obr. 1.1: Menu editor – Menubar objekty

Takto vytvořená menu, jsou připravená pro aktivaci, samotné položky zatím nefungují a jejich procedury a funkce se teprve musí napsat v M-file editoru.

2. 3. Postup při tvorbě GUI aplikace

Po osvětlení pracovního prostředí GUIDE s jeho komponenty a samotným designem aplikací, můžeme přistoupit k vlastnímu postupu při tvorbě GUI aplikace. Její návrh je výhodné si rozdělit do dvou ucelených částí – grafický návrh, tvorba funkcí.

Grafickému návrhu by mělo předcházet dobré rozmyšlení jak by celá aplikace měla vypadat. Její budoucí funkce a které komponenty bude zapotřebí použít. Nejlépe se taková příprava dá udělat za pomocí skiců na papír. Takovou aplikaci, kterou jsme si předem rozmysleli, začneme postupně skládat dohromady.

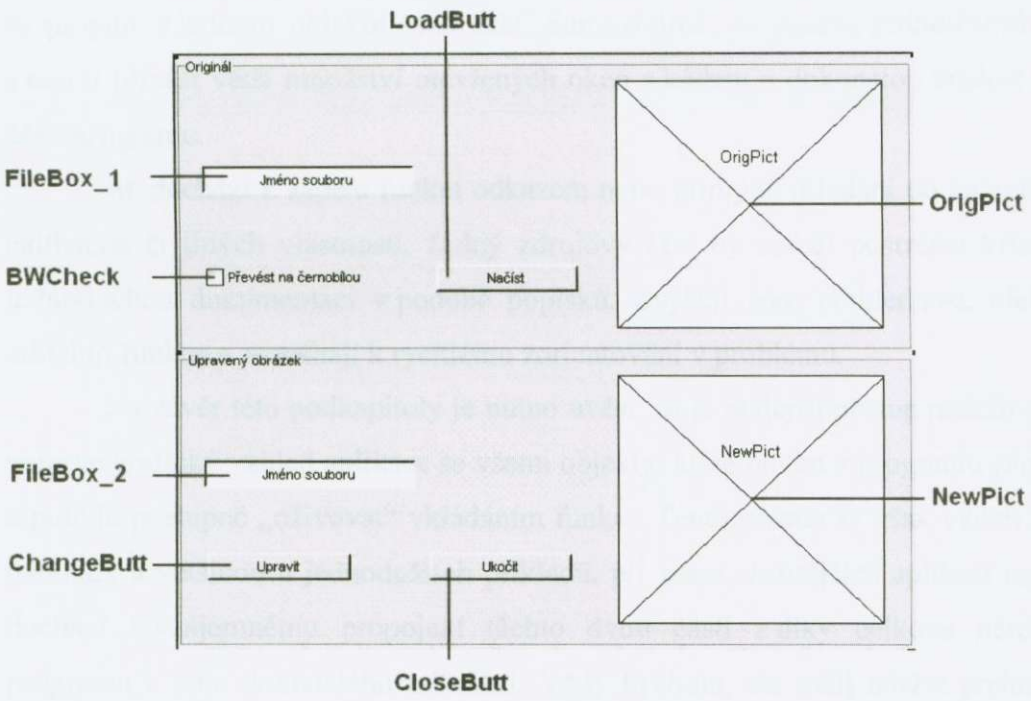
Po jejím designu, který není obvykle dlouhý, následuje vlastní psaní zdrojového textu – funkcí. Stejně jako u grafického návrhu předcházela příprava ve formě skic celého programu. Je nutné udělat si přípravu pro vlastní funkci programu. Většina aplikací (zvláště vytvářená v MATLABu) má určený matematický základ. Proto je dobré si před psaním funkcí nastudovat daný problém, z jakých podkladů vychází, množina možných řešení, zvláštní případy, které mohou nastat. Každý kdo vytváří aplikaci, která má sloužit k řešení určeného problémů MUSÍ rozumět jeho problematice.

2. 3. 1. Grafický návrh

Prvním krokem je tedy, že víme, jaké komponenty potřebujeme v aplikaci resp. její pracovní ploše (figure). Jednotlivé vstupy/výstupy, ovládací komponenty. Po-té všechny komponenty vybereme a umístíme je na pracovní plochu. Designem se prozatím zabývat nemusíme do nijak velké hloubky a ponecháme jej na finální úpravy programu.

Dalším krokem je po umístění všech komponent přiřadit jim jména (identifikátory), které dobře vystihnou jejich úlohu v aplikaci. To se bude následovně hodit pro snazší orientaci mezi jednotlivými funkcemi, callbacky apod. Dále mohou následovat úpravy nezbytné prvotní vlastnosti jednotlivých komponent za moci *Property inspektoru*. Například nastavení rozpětí jezdců nebo vytvoření položek pro pop-up menu popřípadě listboxů.

Po dokončení těchto úprav uložíme první část vytvořené aplikace a to právě její vzhled jako **.fig* soubor, který bude úzce navázán na následovně vytvořený **.m* soubor.



Obr. 1.1: Příklad grafického návrhu aplikace

2. 3. 2. Zdrojový text

Vytvořený grafický návrh je nutno „oživit", sám o sobě je již funkční, ale nenese v sobě žádnou akci. To vše probíhá doplňováním již generovaného zdrojového kódu a vytvářením úplně nových částí.

Každý objekt, který vložíme na pracovní plochu ve vývojovém prostředí je automaticky nadefinován a zapsán do základního zdrojového kódu, nese s sebou několik vlastnosti ovlivňující jeho velikost, umístění, barvu, název, popisek, přiřazení atd. Mezi nejzajímavější části toho kódu z hlediska vytvořeného příkladu pro tuto práci, ale i pro mnoho dalších příkladů se stala vlastnost/funkce *Callback.* Tato vlastnost/funkce v sobě nese definici akce (odpovědi), která nastane po výběru objektu uživatelem (např. kliknutí na tlačítko, položku v menu).

Při jednodušších akcích zakomponujeme požadovanou akci přímo do této části. Přehlednost kódu neutrpí a snadno se modifikuje při doladění celého programu.

Pro složitější akce (např. modelování fyzikálního jevu) je vhodné přiřadit v této části pouze odkaz na funkci modelovaného jevu, jež je dobré definovat mimo hlavní tok programu. Z důvodu přehlednosti, jednoduché ladění, snadné náhrady funkce. Modelování je většinou obsahově náročné a hrozí nepřehlednost kódu, pokud se propojí v jednom objektu více akcí. Samozřejmě, že úprava jednotlivých částí s sebou přináší větší množství otevřených oken s kódem a dokonalou znalost všech částí programu.

Ať dochází k zápisu funkcí odkazem nebo přímým vkládání do jednotlivých callbacků či jiných vlastností, žádný zdrojový kód by neměl postrádat krátkou a jednoduchou dokumentaci v podobě popisků. Zvyšují nám přehlednost, efektivně oddělují funkce a pomáhají k rychlému zorientování v problému.

Na závěr této podkapitoly je nutno uvést, že je nejlepší postup nadefinovat si nejprve grafický vzhled aplikace se všemi objekty, které budou v programu zapotřebí a poté je postupné „oživovat" vkládáním funkcí. Tento postup se však vydaří pouze málokdy a většinou u jednodušších příkladů, při psaní složitějších aplikací nakonec dochází k vzájemnému propojení těchto dvou částí z díky celkové náročnosti programu a jeho dokonalého odladění. Vždy bychom, ale měli udržet přehlednost celého programu.

3. Postup při tvorbě GUI aplikace pro výuku fyziky

V následujících několika kapitolách se pokusím nastínit, co by mělo předcházet samotné tvorbě aplikace určené pro výuku daného problému ve fyzice.

Pro názornost jsem zpracoval příklad a v jednotlivých kapitolách se zabývám jednotlivým časovým vývojem celé aplikace.

Návrh programové aplikace rozdělím na dvě částí: teoretickou a praktickou část. Aby při programování nenastal problém vyplívající z neznalosti teorie problému je nutné si daný jev nastudovat nejdřív z hlediska fyziky, jeho vývoji a principu fungování. Následně se musí daná problematika vyjádřit matematicky do úrovně koncových funkcí/rovnic, které budou využity při samotném praktickém zpracování.

Pokud učiníme všechny základní kroky a uděláme teoretickou přípravu, pak přistoupíme k samotnému návrhu aplikace, ať již grafickému nebo přímo k funkční části programu. V praktické části bychom si měli dobře promyslet přehlednost celé aplikace a její jednoduchost na obsluhu. Mějme na paměti, že jde o aplikaci určenou pro výuku fyzikálního jevu a ne pro výuku obsluhy počítače. Jedná se hlavně o součásti, se kterými přijde uživatel do styku (interface aplikace). Technické fungování a řešení vlastního programu je na autorovi aplikace, ale i zde by se měla zachovávat jistá přehlednost a jednoduchost zápisu, pokud tak lze učinit.

3. 1. Teorie kmitání a optiky

Pro prezentaci využití GUI aplikací při výuce fyziky jsem zvolil odvětví fyziky z teorie kmitání a optiky. Tato část fyziky v sobě kombinuje přiměřenou obtížnost matematiky a zajímavé grafické vyjádření jednotlivých matematických vzorců. Fyzika jako vědní obor je velice rozsáhlá a každá její část je neméně významná a zajímavá. Proto muselo dojít k vhodnému výběru. Snažil jsem se zvolit části, které jsou úplnými základy teorie kmitání (harmonické kmitání, tlumené kmity) a zároveň graficky efektní součásti (Lissajousovi obrazce).

Naprogramování aplikace předcházelo nastudování daného problému, po absolvování semestru fyziky – kmitání a optiky jsem vybral vhodná témata a vytvořil matematické podklady pro aplikaci.

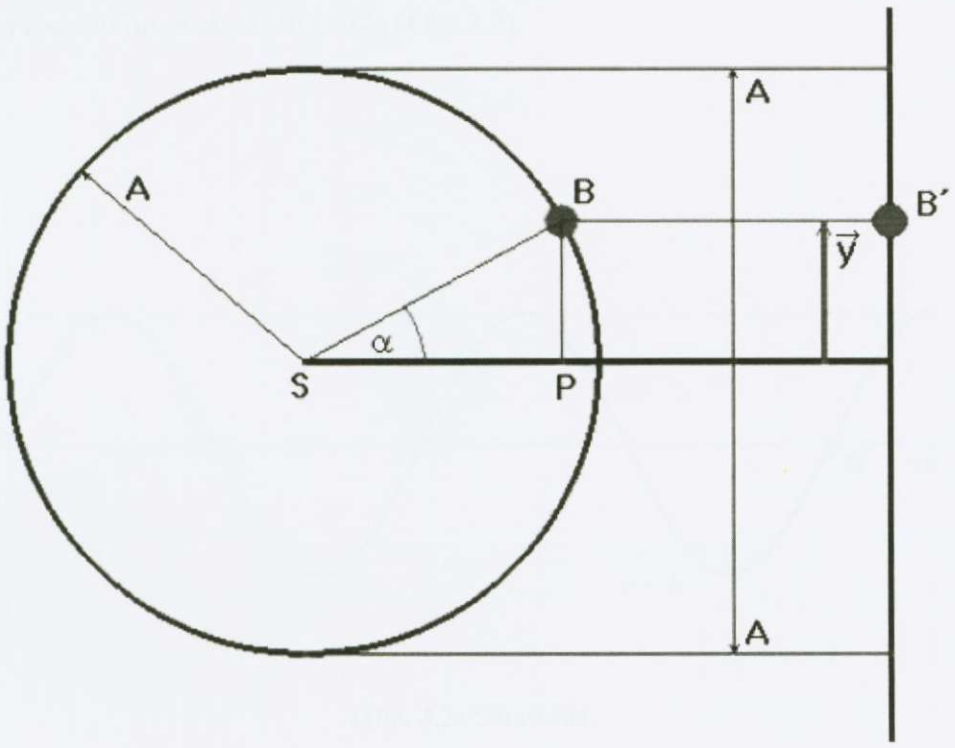
S dovolením vedoucího diplomové práce Mgr. Petra Bartoše, jsem použil části jeho vypracované přípravy pro přednášky Fyziky – Teorie kmitání a optiky, jež mi posloužila jako velmi dobrý základ pro nastínění daných jevů a jejich následné matematické vyjádření.

3. 1. 1. Rovnice kmitavého pohybu hmotného bodu

Rovnici kmitavého pohybu hmotného bodu lze odvodit více způsoby. Jedním z nich je porovnání kmitavého pohybu a pohybu hmotného bodu po kružnici.

Představme si situaci vyobrazenou na obrázku 3.1. Hmotný bod B, vykonává rovnoměrný harmonický pohyb po kružnici o poloměru A s úhlovou rychlostí ω.

V počátečním okamžiku (v čase t_0) svírala spojnice středu S a bodu B s vodorovnou osou úhel φ_0. Za čas t urazí hmotný bod při úhlové rychlosti ω úhel ωt. S vodorovným směrem tedy spojnice SB svírá úhel $\alpha = \omega \cdot t + \varphi_0$ (při uvážení hodnoty úhlu do 360°, jinak je potřeba převést do intervalu od 0 do 360 ° odečtením vhodného poctu násobku 360°).



Obr. 3.1: Znázornění harmonického pohybu

Promítneme bod B do bodu B' (bod B' leží na přímce rovnoběžné s osou y a zároveň na rovnoběžce s osou x vedenou bodem B). Pokud se bod B pohybuje po kružnici, tak nám bod B' vykonává harmonický netlumený kmitavý pohyb. Okamžitá výchylka tohoto bodu z rovnovážné polohy je rovna velikosti úsečky PB, jejíž velikost můžeme určit z pravoúhlého trojúhelníka SPB takto:

$$\frac{|BP|}{|SB|} = \sin\alpha$$

$$\frac{\mathbf{y}}{A} = \sin\alpha \Rightarrow \mathbf{y} = A \cdot \sin\alpha \tag{1}$$

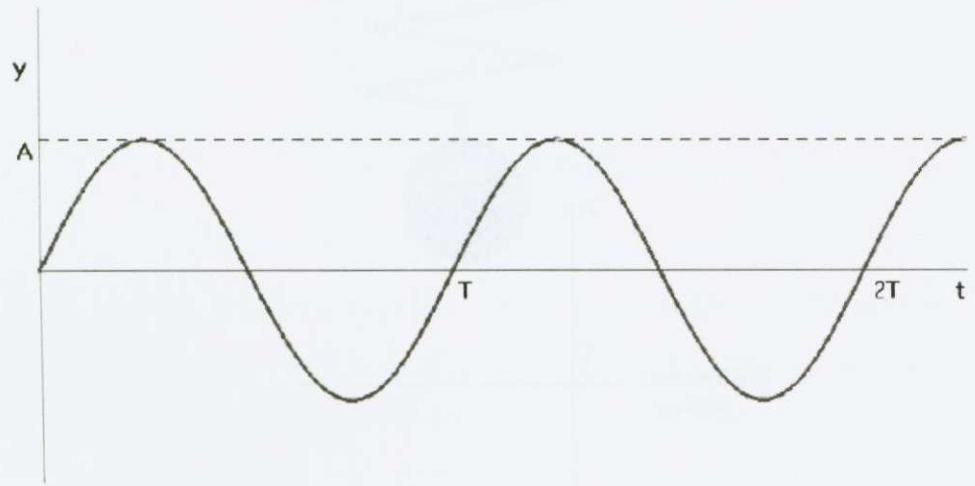
Jelikož jsme odvodili, že hodnota α je $\omega t + \varphi_0$, dostáváme výsledný vztah popisující harmonický netlumený kmitavý pohyb

$$\mathbf{y} = A \cdot \sin(\boldsymbol{\omega} t + \varphi_0) \tag{2}$$

Veličinu ω nazýváme úhlová frekvence, můžeme ji vypočítat ze vztahu:

$$\boldsymbol{\omega} = 2\pi f$$

veličinu φ_0 j nazýváme počáteční fáze a výraz $\omega t + \varphi_0$ v závorce fáze kmitavého pohybu. Z výše uvedeného plyne i grafické znázornění závislosti výchylky hmotného bodu na čase. Jejím známá sinusoida (Obr 3.2).

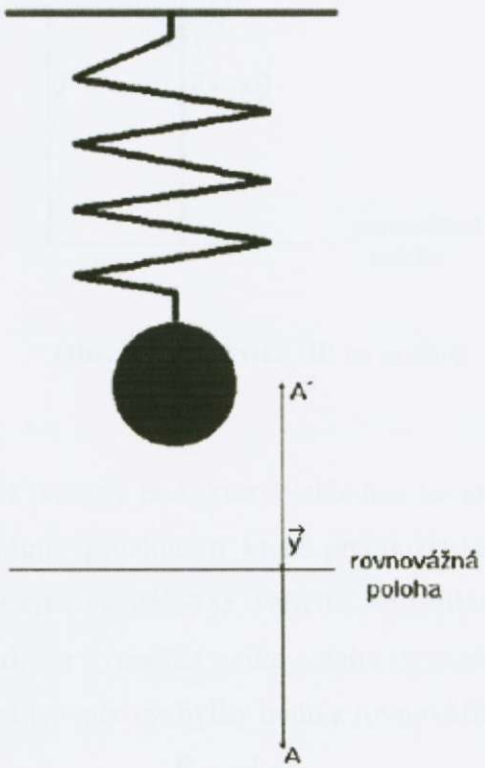


Obr. 3.2: Sinusoida

Připevněme na pružinu hmotný bod o hmotnosti m (Obr. 3.3). Pokud je tento hmotný bod v klidu, nachází se v rovnovážné poloze. Z toho vyplyne, že velikost síly, působící na bod je rovna velikosti síly gravitační. Při vychýlení pružiny z rovnovážného stavu do bodu A a následovním uvolněním. Bod začne rytmicky pohybovat okolo své rovnovážné polohy – vykonává kmitavý pohyb.

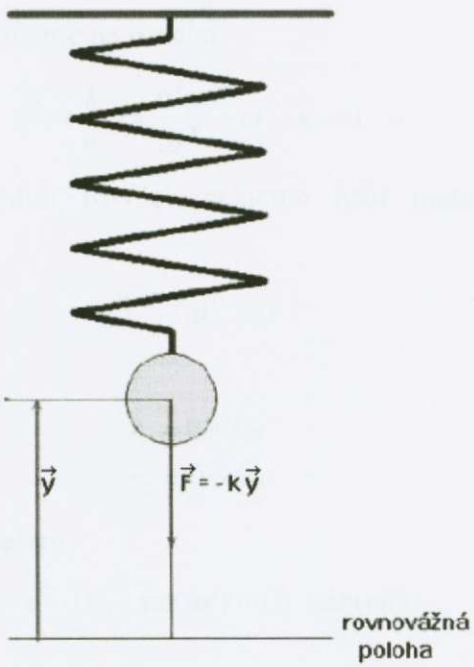
Vzdálenost hmotného bodu od rovnovážné polohy, kterou v daném okamžiku naměříme, nazveme výchylkou y.

Největší hodnotu, kterou výchylka nabývá, nazýváme amplituda s označením A.



Obr. 3.3: Hmotný bod na pružině

Pro odvození rovnice pro kmitavý pohyb pružiny budeme postupovat následovně, pro názornou představu použijeme následující obrázek 3.4, pomocí kterého odvodíme závislosti.



Obr. 3.4: Výchylka HB na pružině

Na obrázku je vidět hmotný bod, který vzhledem ke své rovnovážné poloze výchylku y. Dále zde vidíme pružinu, o které předpokládáme, že má tuhost k (charakteristika pružiny). Experimentálně je ověřeno, že velikost síly, kterou působí pružina na náš hmotný bod, lze vypočítat podle vztahu (znaménko mínus vyjadřuje, že síla má opačný směr, než je směr výchylky bodu z rovnovážné polohy):

$$\mathbf{F} = -k \cdot \mathbf{y} \tag{3}$$

Podle Newtonova zákona lze sílu počítat také ze vztahu

$$\mathbf{F} = m \cdot \mathbf{a} = m \cdot \frac{\mathbf{d}^2\mathbf{y}}{dt^2} \tag{4}$$

Nyní oba dva vzorce porovnáme a získáme následující vztah resp. vztahy:

$$m \cdot \frac{\mathbf{d}^2\mathbf{y}}{dt^2} = -k \cdot \mathbf{y} \Rightarrow m \cdot \frac{\mathbf{d}^2\mathbf{y}}{dt^2} + k \cdot \mathbf{y} = 0 \tag{5}$$

Dostali jsme tak diferenciální rovnici, kterou vyřešíme následujícím postupem. Nejprve rovnici vydělíme hmotností, čímž získáme:

$$\frac{\mathbf{d}^2\mathbf{y}}{dt^2} + \frac{k}{m} \cdot \mathbf{y} = 0 \tag{6}$$

zavedeme substituci a aplikujeme na rovnici,

$$\omega^2 = \frac{k}{m} \Rightarrow \frac{\mathbf{d}^2\mathbf{y}}{dt^2} + \omega^2 \cdot \mathbf{y} = 0 \tag{7}$$

takto upravenou diferenciální rovnici začneme řešit metodou charakteristické rovnice,

$$r^2 + \omega^2 = 0 \tag{8}$$

získáme výsledek

$$\begin{aligned} r_1 &= 0 + i\omega \\ r_2 &= 0 - i\omega \end{aligned} \tag{9}$$

a vypočítáme partikulární řešení

$$\mathbf{y} = e^0 \cdot (C_1 \cdot \sin(\boldsymbol{\omega} t) + C_2 \cdot \sin(\boldsymbol{\omega} t)) \tag{10}$$

jelikož

$$e^0 = 1 \Rightarrow \mathbf{y} = C_1 \cdot \sin(\boldsymbol{\omega} t) + C_2 \cdot \sin(\boldsymbol{\omega} t) \tag{11}$$

S využitím počátečních podmínek ($t = 0$) pro polohu a rychlost za pomocí derivace získáme následující vztahy:

$$\begin{aligned} y(0) &= A \cdot \sin\alpha \quad \text{- poloha} \\ \mathbf{v}(0) &= A\omega\cos\varphi \quad \text{- rychlost} \end{aligned} \tag{12}$$

dosazením do partikulárního řešení získáme jednotlivé konstanty C_1 a C_2

$$\mathbf{y}(0) = C_1 \cdot \sin\boldsymbol{\omega} \cdot 0 + C_2 \cdot \sin\boldsymbol{\omega} \cdot 0 \Rightarrow C_2 = A \cdot \sin\varphi \tag{13}$$

a

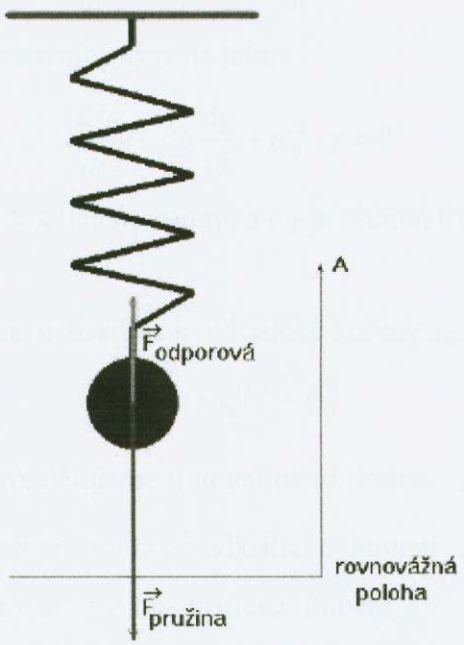
$$\begin{aligned} \mathbf{v} &= \boldsymbol{\omega}(C_1 \cos\boldsymbol{\omega} t - C_2 \sin\boldsymbol{\omega} t) \\ \mathbf{v}(0) &= A \cdot \boldsymbol{\omega} \cdot \cos\varphi = \boldsymbol{\omega}(C_1 \cos\boldsymbol{\omega} \cdot 0 - C_2 \sin\boldsymbol{\omega} \cdot 0) \Rightarrow C_1 = A \cdot \cos\varphi \end{aligned} \tag{14}$$

Spočítané konstanty dosadíme opět do partikulární rovnice a dostaneme následující tvar po jehož úpravě dostaneme konečnou podobu rovnice pro netlumené harmonické kmitání:

$$\mathbf{y} = A \cdot \sin(\boldsymbol{\omega} t + \varphi_0) \tag{15}$$

3. 1. 2. Rovnice tlumeného kmitavého pohybu hmotného bodu

Tlumený mechanický oscilátor odevzdává část své energie svému okolí a snižuje tak svou amplitudu při kmitaní. Situaci nám znázorňuje následující obrázek 3.5.



Obr. 3.5: Příklad tlumeného kmitání

Při pohybu je odporová síla velmi často úměrná rychlosti pohybu a působí v opačném směru než rychlost. Pro tento případ lze pro velikost odporové síly psát.

$$\mathbf{F}_{odporová} = -r \cdot \mathbf{v} \tag{16}$$

Dále platí jako v předešlé kapitole,

$$\mathbf{F}_{pružiny} = -k \cdot \mathbf{y} \tag{17}$$

Platí, že

$$\mathbf{F}_{celková} = \mathbf{F}_{pružiny} + \mathbf{F}_{odporová} \tag{18}$$

a z toho vyplývá rovnice

$$\mathbf{F}_{celková} = -k \cdot \mathbf{y} - r \cdot \mathbf{v} \tag{19}$$

Provedeme obdobné úpravy jako v předchozí kapitole a dostaneme tak diferenciální rovnici, kterou budeme opět řešit metodou charakteristické rovnice.

$$m \cdot \frac{\mathbf{d}^2\mathbf{y}}{dt^2} + r \cdot \frac{\mathbf{dy}}{dt} + k \cdot \mathbf{y} = 0$$
$$\frac{\mathbf{d}^2\mathbf{y}}{dt^2} + \frac{r}{m} \cdot \frac{\mathbf{dy}}{dt} + \frac{k}{m} \cdot \mathbf{y} = 0 \qquad (20)$$

Konečný diferenciální tvar rovnice vypadá takto:

$$\frac{\mathbf{d}^2\mathbf{y}}{dt^2} + 2b\frac{\mathbf{dy}}{dt} + \omega_0^2 \cdot \mathbf{y} = 0 \qquad (21)$$

Konstanta b je tzv. koeficient útlumu a ω_0 je úhlová frekvence netlumených kmitů.

Celou rovnici řešíme a získáme kvadratické kořeny $s_{1,2}$,

$$s_{1,2} = -b \pm \sqrt{b^2 - \omega_0^2} \qquad (22)$$

podle hodnoty $\sqrt{b^2 - \omega_0^2}$ rozdělujeme tlumení na tři druhy:

- $\mathbf{b^2 - \omega^2 > 0}$ – $b > \omega^2$ => tzv. nadkritické tlumení
- $\mathbf{b^2 - \omega^2 = 0}$ – $b = \omega^2$ => tzv. kritické tlumení
- $\mathbf{b^2 - \omega^2 < 0}$ – $b < \omega^2$ => tzv. podkritické tlumení

Tlumený pohyb za těchto podmínek vzniká pouze při podkritickém tlumení, kdy je tlumící síla dostatečně slabá. Pro výchylku lze pak s ohledem na obecné řešení diferenciálních rovnic psát:

$$\mathbf{y} = e^{-bt} \cdot (C_1 \cdot \sin \boldsymbol{\omega} t + C_2 \cdot \cos \boldsymbol{\omega} t) \qquad (23)$$

Stejně jako v případě netlumených kmitů i u této rovnice můžeme dojít k obecnému řešení využitím počátečných podmínek a získat tak konečný tvar rovnice:

$$\mathbf{y} = A \cdot e^{-bt} \cdot \sin(\boldsymbol{\omega} t + \varphi) \qquad (24)$$

3. 1. 3. Rovnice složených kmitů

Metod pro skládání dvou a více kmitavých pohybů jsou. Patří mezi ně metoda matematická (početní), vektorové skládání. Stejně tak můžeme skládat pohyby (kmity) stejného směru nebo např. směru kolmého (u takového skládání můžou být výsledným obrazcem Lissajousovi obrazce).

Pro využití matematického aparátu v MATLABu jsem zvolil vyjádření kmitů za pomoci matematické metody. V úvahu však může přijít i vektorová metoda, oba postupy vedou k totožným výsledkům.

Nejjednodušším případem je skládání pohybů po přímce, kdy směr kmitů je vždy stejný. Výsledný pohyb, však již není sinusovitý, ale je vždy harmonický.

Speciálně užíváme početní metodu v případech, kdy mají kmity podobně velké frekvence (například 400 Hz a 410 Hz) a kmity mají stejný směr, fázi a amplitudu. Nechť mají jednotlivé kmity rovnice:

$$\begin{aligned} \psi_1 &= \psi_0 \sin(\omega_1 t + \varphi) \\ \psi_2 &= \psi_0 \sin(\omega_2 t + \varphi) \end{aligned} \tag{25}$$

Výsledný kmit je dán dle zákona superpozice součtem okamžitých výchylek ψ_1 a ψ_2:

$$\psi = \psi_1 + \psi_2 = \psi_0 \sin(\omega_1 t + \varphi) + \psi_0 \sin(\omega_2 t + \varphi) \tag{26}$$

Nyní vytkneme hodnotu amplitudy:

$$\psi = \psi_0 \cdot [\ \sin(\omega_1 t + \varphi) + \sin(\omega_2 t + \varphi)\ ] \tag{27}$$

Užitím obecného matematického vzorce:

$$\sin\alpha + \sin\beta = 2 \cdot \sin\frac{a+b}{2} \cdot \cos\frac{a+b}{2} \tag{28}$$

Tím dostane výsledný vzorec pro skládání kmitů blízkých frekvencí:

$$\psi = 2 \cdot \psi_0 \cdot \cos\left(\frac{\omega_1 - \omega_2}{2} \cdot t\right) \cdot \sin\left(\frac{\omega_1 + \omega_2}{2} \cdot t + \varphi\right) \tag{29}$$

3. 1. 4. Lissajousovi obrazce

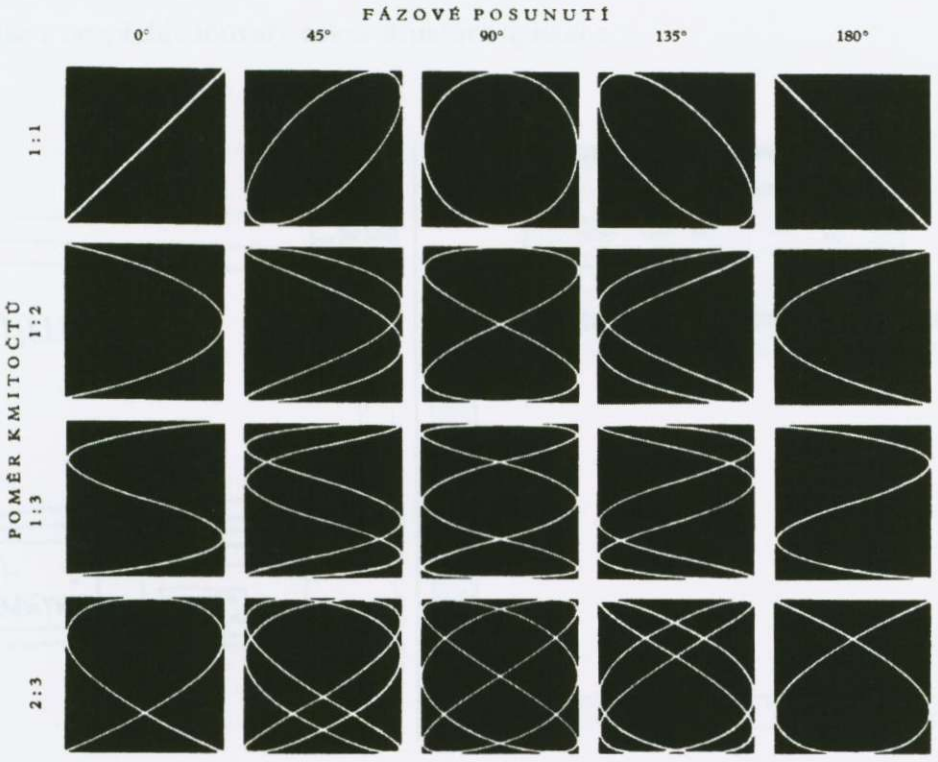
Početní metodu lze využít i k zvláštním případům složeného kmitání a to ke kmitům kolmých. Poměry dob kmitů musí být v poměru celých čísel a pak platí:

$$n_1 \cdot T_1 = n_2 \cdot T_2 \tag{30}$$

Počáteční fáze a amplitudy mohou být různé.

Výsledkem tohoto skládání je rovinný pohyb. Křivka je uzavřená a na vzniklé obrazce nazýváme Lissajousovi obrazce (Obr 3.6). Pro několik počátečních fází můžeme vidět na obrázku jednotlivé obrazce.

Pokud nebude poměr frekvencí z oboru celých čísel, výsledný obrazec nebude uzavřená křivka a výsledných kmit nebude harmonickým pohybem.



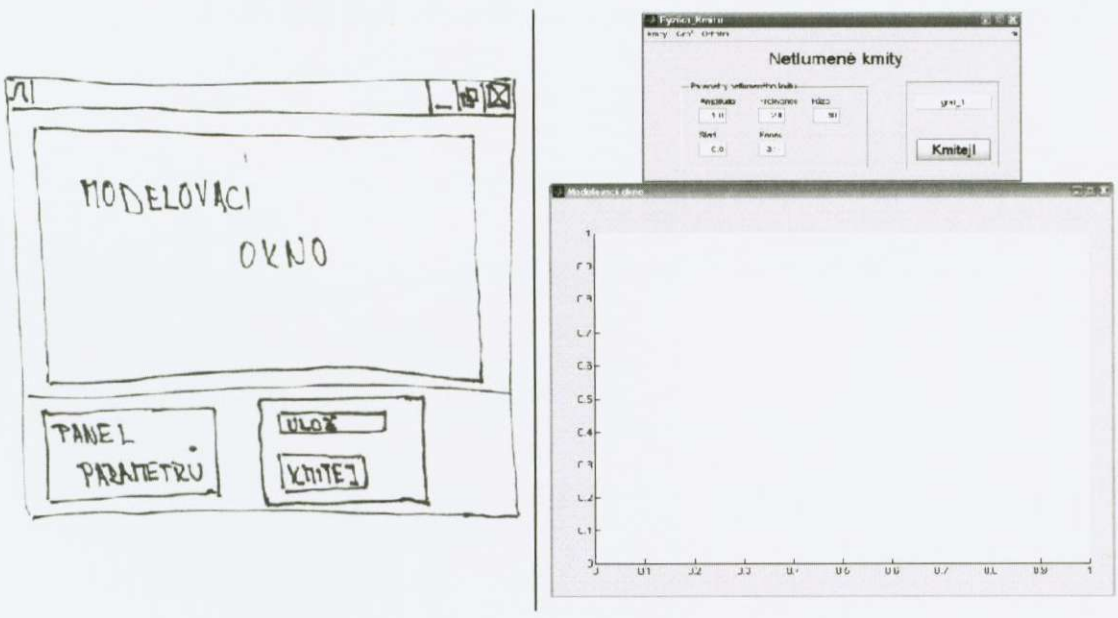
Obr. 3.6: Příklad Lissajousových obrazců

3. 2. Grafická úprava a design aplikace

Při grafickém návrhu a úpravě celé aplikace jsem se snažil řídit, zásadami popsaných v kapitole 2.1 Princip designu aplikací. Do jaké míry všechny podmínky byly splněny, závisí na uživateli (Obr. 3.6).

Schéma demonstrační programu jsem zvolil ucelené, všechny modelové problémy jsou přístupny z jednoho prostředí a to za pomocí menu. Každá položka zmiňovaného menu je přístupna přes klávesové zkratky, které jsou voleny, tak aby si je uživatel osvojil a zapamatoval, co nejrychleji.

Největší plochu, ale také zájem zaujímá samotné okno pro zobrazování grafů, pod ním je umístěn panel vztahující se k aktuálnímu fyzikálnímu jevu. Vše je popsáno „bublinkovou" nápovědou, pro nejnutnější pomoc. Celý program nadále pracuje na funkci jednoho tlačítka, v daném případě není nutné zavádět více prvků. Ty by spíše z nezpřehledňovaly celou strukturu aplikace.



Obr. 3.7: Původní a skutečný návrh designu aplikace

Celá aplikace včetně jejího designu, by si však zasloužila více času věnovaného pro další návrhy. Nepochybně by neměla chybět vysvětlená teorie přístupná každému uživateli počítače. Dále by měli být lépe ošetřeny chyby, které mohou při zadávání dat vzniknout. Uživatel by se měl dozvědět, že vznikla chyba a z jakého důvodu nastala, aby se jí nadále vyhnul. Tyto problémy budu nejlépe řešit dialogovými okny

s výstižným textem chyby a jeho návrhu opravy. Chci tak dosáhnout vysokého stupně interaktivity celého softwaru s uživatelem a dosáhnout tak jeho pohodlí a plného soustředí na výuku daného problému.

Do celé interaktivity by se hodilo zahrnout i komunikaci s tiskárnou, export modelovaných kmitu do obrázku nebo celý dynamický průběh zachytit do videosouboru, tak aby aplikace našla široké užití.

3. 3. Zpracování zdrojových kódů

Zpracování jednotlivých kódů probíhalo ve dvou rovinách, které jsem mohl nezávisle otestovat a poté je propojit.

Již od počátku jsem měl v úmyslu zpracovat jednotlivé m-file soubory obsahující funkce a modelování vybraných fyzikálních jevů. Vedly mě k tomu aspekty jako je získaní přehlednosti a jednoduché úpravy kódů. Při psaní jsem měl také velkou potřebu neustále testovat správnou funkčnost jednotlivých součástí.

V druhé rovině jsem se zaměřil na interface aplikace a integraci již vytvořených modelovacích funkcí, v jeden funkční celek. Samozřejmě, že při tomto zpracování došlo k částečné přeměně funkcí jevů, ale ne natolik velké, aby se změnil ráz programu

V následující části uvedu jednotlivé úseky aplikace, které z hlediska funkčnosti považuji za nosné elementy programu. Uvádět výpis celého zdrojového kódu by bylo zbytečně dlouhé a nic neříkající.

První za zmínku stojí jednotlivé funkce navržených kmitů a jejich funkční řešení v rámci MATLABu.

```
%Funkce pro netlumene kmity
%-----------------------------%
function kmit=nkmit(amplituda,frekvence,faze,tmin,tmax,stav)
% hlavicka funkce

%Definice konstant a promennych
%-----------------------------%
krok = 1000;                    % krok casoveho intervalu
dt = tmax/krok;
t = tmin:dt:tmax;               % casovy interval
omega = (2*pi*frekvence);       % uhlova frekvence

%Vypocet kmitu a vykresleni grafu
%-----------------------------%
y = amplituda.*sin(omega.*t + faze);  % rovnice kmitu

if stav == 1
  plot(t,y)                 % staticky vystup kmitu na graf
else
  comet(t,y)                % dynamicky vystup kmitu na graf
end
```

```
%Funkce pro tlumene kmity
%-------------------------%
function kmit=tkmit(amplituda,frekvence,faze,utlum,tmin,tmax,stav)
% hlavicka funkce

%Definice konstant a promennych
%-----------------------------%
krok = 1000;                    % krok casoveho intervalu
dt = tmax/krok;
t = tmin:dt:tmax;               % casovy interval
omega = (2*pi*frekvence);       % uhlova frekvence

%Vypocet kmitu a vykresleni grafu
%-------------------------------%
y = amplituda.*exp(-(utlum.*t)).*sin(omega.*t + faze);
% rovnice kmitu

% vystup kmitu na graf

if stav == 1
  plot(t,y)                     % staticky vystup kmitu na graf
else
  comet(t,y)                    % dynamicky vystup kmitu na graf
end
```

```
%Funkce pro skladani kolmych kmitu
%--------------------------------%
function
kmit=lobrazce(amplituda,frekvence_1,frekvence_2,faze,tmin,tmax,stav)
% hlavicka funkce

%Definice konstant a promennych
%-----------------------------%
krok = 1000;                    % krok casoveho intervalu
dt = tmax/krok;
t = tmin:dt:tmax;               % casovy interval
omega_1 = 2*pi*frekvence_1;     % uhlova frekvence 1. kmitu
omega_2 = 2*pi*frekvence_2;     % uhlova frekvence 2. kmitu

%Vypocet kmitu a vykresleni grafu
%-------------------------------%
x=amplituda.*cos(omega_1.*t);   % rovnice kmitu ve smeru osy X
y=amplituda.*sin(omega_2.*t + faze);  % rovnice kmitu ve smeru osy Y

if stav == 1
  plot(x,y)                      % staticky vystup kmitu na graf
else
  comet(x,y)                     % dynamicky vystup kmitu na graf
end
```

```
function varargout = Fyzika_Kmitu(varargin)
% Fyzika_Kmitu M-file for Fyzika_Kmitu.fig

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Fyzika_Kmitu_OpeningFcn, ...
                   'gui_OutputFcn',  @Fyzika_Kmitu_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);

if nargin == 0  % LAUNCH GUI
    modelokno;
    fig = openfig(mfilename,'reuse');
    % Use system color scheme for figure:
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

    % Generate a structure of handles to pass to callbacks, and store
it.
    handles = guihandles(fig);
    guidata(fig, handles);

    if nargout > 0
        varargout{1} = fig;
    end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

    try
        if (nargout)
            [varargout{1:nargout}] = feval(varargin{:}); % FEVAL
switchyard
        else
            feval(varargin{:}); % FEVAL switchyard
        end
    catch
        disp(lasterr);
    end

end

. . .

% --------------------------------------------------------------------
function varargout = desktop_CloseRequestFcn(h,eventdata, handles,
varargin)
delete(h);
delete(modelokno);
```

```
% --------------------------------------------------------------------
function varargout = NK_menu_Callback(h, eventdata, handles,varargin)

 set(handles.panel_1, 'Visible', 'on');
 set(handles.panel_2, 'Visible', 'off');
 set(handles.panel_3, 'Visible', 'off');
 set(handles.panel_4, 'Visible', 'off');

 set(handles.NK_menu, 'Checked', 'on');
 set(handles.TI_menu, 'Checked', 'off');
 set(handles.LO_menu, 'Checked', 'off');
 set(handles.SK_menu, 'Checked', 'off');

 set(handles.aktiv_kmit, 'String', '1');
 set(handles.nadpis, 'String', 'Netlumené kmity');
 close('Modelovací okno');

% --------------------------------------------------------------------
function varargout = ST_menu_Callback(h, eventdata, handles,
varargin)

 set(handles.ST_menu, 'Checked', 'on');
 set(handles.DY_menu, 'Checked', 'off');
 set(handles.IM_menu, 'Enable', 'on')

 set(handles.stav_grafu, 'String', '1');
% pomocny prepinac pro stav vykreslovani grafu {1-statik, 2-dynam}

% --------------------------------------------------------------------
function varargout = DY_menu_Callback(h, eventdata, handles,
varargin)

 set(handles.ST_menu, 'Checked', 'off');
 set(handles.DY_menu, 'Checked', 'on');
 set(handles.IM_menu, 'Enable', 'off');

 set(handles.stav_grafu, 'String', '2');
% pomocny prepinac pro stav vykreslovani grafu {1-statik, 2-dynam}

% --------------------------------------------------------------------
function varargout = TI_menu_Callback(h, eventdata, handles,
varargin)
printdlg(modelokno);

% --------------------------------------------------------------------
function varargout = ND_menu_Callback(h, eventdata, handles,
varargin)
printpreview(modelokno);

% --------------------------------------------------------------------
function varargout = IM_menu_Callback(h, eventdata, handles,
varargin)
saveas(modelokno, get(handles.file_name,'string'), 'bmp');
```

```
function varargout = kmitej_tlacitko_Callback(h, eventdata, handles,
                                              varargin)

ak_kmit      = str2double(get(handles.aktiv_kmit, 'string'));
st_grafu     = str2double(get(handles.stav_grafu, 'string'));
modelokno;

if ak_kmit == 1
  amplituda_hk = str2double(get(handles.amplituda_hk,'string'));
  frekvence_hk = str2double(get(handles.frekvence_hk,'string'));
  faze_hk      = str2double(get(handles.faze_hk,'string'));
  tmin_hk      = str2double(get(handles.start_hk,'string'));
  tmax_hk      = str2double(get(handles.konec_hk,'string'));

  if tmin_hk > tmax_hk
    tmin_hk = 0.0;
    tmax_hk = 1.0;
  end

  nkmit(amplituda_hk,frekvence_hk,faze_hk,tmin_hk,tmax_hk,st_grafu)

  title(['Netlumený kmit - Frekvence: ',
   get(handles.frekvence_hk,'string'), ' Hz, ', 'Fáze: ',
   get(handles.faze_hk,'string'), '°']);

  xlabel('cas - t [s]');
  ylabel('amplituda - A [m]');

else
  amplituda_sk  = str2double(get(handles.amplituda_sk,'string'));
  frekvence1_sk = str2double(get(handles.frekvence1_sk,'string'));
  frekvence2_sk = str2double(get(handles.frekvence2_sk,'string'));
  faze_sk       = str2double(get(handles.faze_sk,'string'));
  tmin_sk       = str2double(get(handles.start_sk,'string'));
  tmax_sk       = str2double(get(handles.konec_sk,'string'));

  if tmin_sk > tmax_sk
    tmin_sk = 0.0;
    tmax_sk = 1.0;
  end

  skmit(amplituda_sk,frekvence1_sk,frekvence2_sk,faze_sk,tmin_sk,
        tmax_sk,st_grafu)

  title(['Tlumený kmit - Frekvence 1: ',
   get(handles.frekvence1_sk,'string'), ' Hz, ', 'Frekvence 2: ',
   get(handles.frekvence2_sk,'string'),' Hz, ', 'Fáze: ',
   get(handles.faze_hk,'string'), '°']);

  xlabel('cas - t [s]');
  ylabel('amplituda - A [m]');
end
```

3. 4. Odladění zdrojového textu

I přesto, že programovací jazyk MATLAB patří mezi ty jednodušší jazyky, což je zapříčiněné hlavně jeho syntaxí příkazů. Je nutné ve zdrojovém kódu hledat a opravovat chyby a pak i finálně zjednodušovat zápis.

Chyby rozdělíme na dva druhy. Jedním tím jednodušším typem, které zjistíme již při kompilaci, jsou chyby vzniklé špatnou syntaxí příkazů a funkcí. Kompilátor je nalezne a vypíše v pracovním prostoru řádek, ve kterém se chyba vyskytla. Druhým typem chyb jsou takové, které zapříčiní špatné chování programu. Mezi ně můžou patřit chyby způsobené též špatnou syntaxí. Může se však stát, že kompilátor neobjeví jemu logickou chybu a program spustí (např. nekonečné smyčky). Takové chyby se hledají obtížněji. A proto pro vyhledání chyb můžeme použít následující čtyři metody:

- **Odstranit středníky** – výpis aktuálních hodnot každé z hodnot přímo na příkazovou řádku
- **Funkční soubor změnit na skriptový soubor** – všechny mezivýsledky se vypíší a zaznamenají v hlavním pracovním prostoru
- **Příkaz Keyboard** – použitím příkazu v programu, dostaneme kontrolu nad aplikací s možností měnit proměnné popř. využít jiných příkazů z knihoven MATLABu
- **Využít dolaďovacích příkazů** – MATLAB je vybaven sadou příkazu sloužících pro ladění a řízením běhu programu

První tři metody vyžadují zásahy do zdrojového textu aplikace, metodu dolaďovacích příkazů popíšu dále.

Slovo ladění můžeme přeložit do angličtiny jako debug. Všechny příkazy sloužící k této akci zažínají db. Patří mezi ně:

- **dbclear**
- **dbcont**
- **dbdown**
- **dbstop**
- **dbstack**
- **dbstep**
- **dbtype**
- **dup**
- **dbquit**
- **dbstatus**

Většina těchto příkazů je přístupná z M-file editoru popř. za využití příkazové řádky. Pokud se tedy ve zdrojovém textu aplikace vyskytne chyba, je

výhodné využít ladících příkazů. Nejprve si nastavíme body přerušení, pomocí kterých lokalizujeme chybu. Poté spustíme funkční M-file soubor, když se program zastaví v bodě přerušení, zobrazí se v pracovním prostoru hlášení s řádkou bodu přerušení. Takto můžeme celý program krokovat nebo jej testovat po jeho funkčních částech.

Příkazy pro doladění programu fungují pouze s funkčními soubory a jsou navázány na zkompilované M-file soubory. Proto jakékoliv smazání těchto souborů nebo jejich úpravou se zruší např. všechny body přerušení.

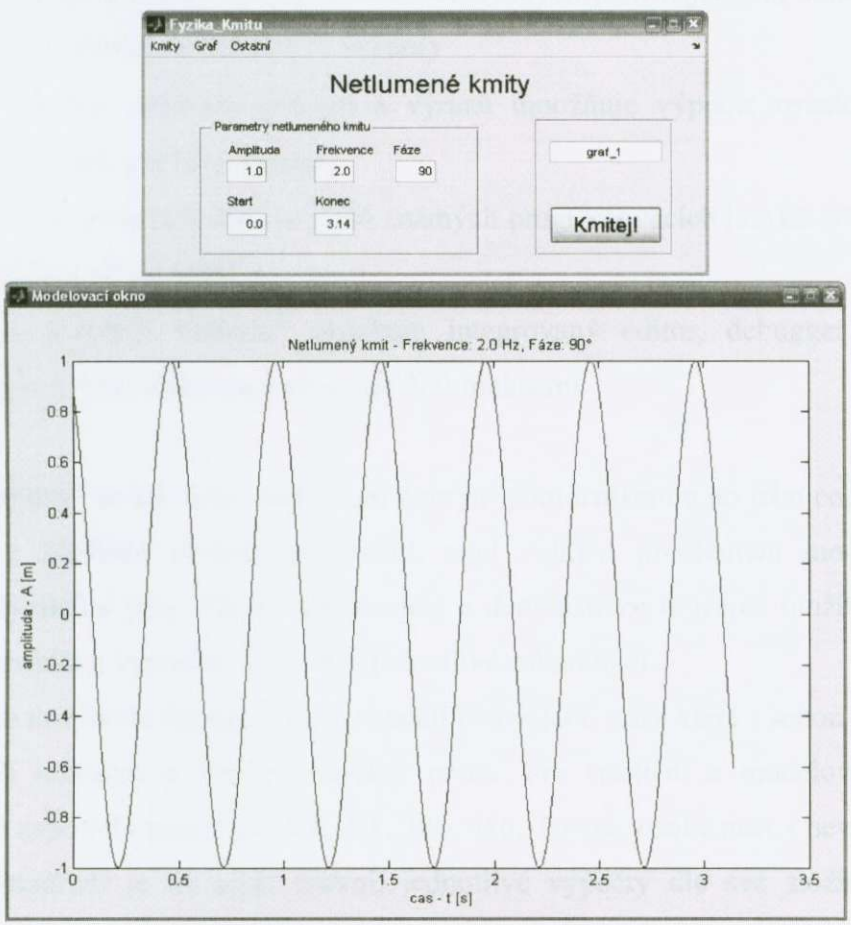
4. Využití GUI aplikací při výuce

Využití jednotlivých vytvořených aplikací je nasnadě. Podle zaměření aplikace ji můžeme použít pro názornou ukázku modelování jednotlivých fyzikálních problémů.

Podle míry interaktivity programu můžeme měnit jednotlivé hodnoty, které ovlivňují modelovanou situaci a sledovat tak její vývoj. Dynamické zobrazení fyzikálního problému je atraktivnější z hlediska upoutání pozornosti uživatele.

Tyto aspekty vedou k hlubšímu zkoumání daného problému a pochopení jeho principu funkce. S propojením vysvětlení teorie, která v rámci mezí bude dostatečně přístupná, dosáhneme efektivnější výuky z daného odvětví fyziky.

GUI aplikace s dobrým interaktivním základem může posloužit díky svým funkcím (výstup na tiskárnu, zachycení modelování do videosouboru) nejen k výuce, ale pro doplnění prezentací, výsledků v měření.



Obr. 1.1: Příklad učební aplikace

5. GUI aplikace vs. Famulus

Porovnání těchto dvou aplikací není zcela jednoduché a možné. Oba programy byly vyvíjeny pro jiný operační systém a je mezi nimi velký časový odstup. Vyberu proto prvky, kde má smysl porovnat kvalitu a vybavení jednotlivých aplikací.

Aplikace Famulus byla vyvinuta jako programovací systém sloužící k řešení, demonstraci a modelování fyzikální jevů a situací. Popřípadě ověřovat a ladit algoritmy. Vše se dá zvládnout na takřka libovolném počítači. Hardwarové nároky jsou zjednodušeně postaveny pro počítač spolehlivě pracující s OS MS-DOS®. Uplatní se tedy všude, kde je potřeba názorně předvést jevy s obtížností výpočtů nejen na úrovni střední školy, ale i pro složitější vysokoškolské problémy. Famulus® zvládá následující základní funkce:

- Výpočty s reálnými a komplexními čísly s přesností na 19 platných míst
- Umožňuje grafický výstup až 4 modelovaných jevů najednou, s možností tisku (výstupu) na jehličkové tiskárny
- Interaktivní zadávání příkazů a výrazů umožňuje výpočet rovnic nebo přímo jejich grafický výstup
- Možnost programování ve stylu známých programovacích jazyků (Pascal®, FORTRAN®, BASIC®)
- Celé prostředí Famulu® obsahuje integrovaný editor, debugger, help, podporu 16ti knihoven s přibližně 200 funkcemi.

Famulus® se tak může stát velmi dobrým pomocníkem a po jeho celkovém ovládnutí z hlediska obsluhy programu, není velkým problémem modelovat libovolné fyzikální jevy. Připravené modely a demonstrace můžeme uložit a pak nadále předvádět a vysvětlovat na nich jednotlivé zákonitosti.

Jako nevýhodu Famulu® bych označil pouze jeho stáří, které s sebou přináší fakt jistých omezení a zhoršení kvality práce. Pro spuštění a modelování ve Famulu® je zapotřebí pouze PC XT, AT, 386, 486. To vše v sobě nese i nevýhody. Grafické prostředí je na nižší úrovni, jednotlivé výpočty dle své složitosti si nárokují paměť a to vede ke zpomalení demonstrací. Tištěný výstup je omezen na staré LPT1 (CANON 21) rozhraní, předurčené pro jehličkové tiskárny popř. starší inkoustové.

Zápory aplikace Famulus® by však neměli zabránit jeho využití ve školách, kde i v současné době počítačové vybavení v některých případech nedosahuje závratných kvalit, z důvodu omezených financí. Může se tak stát relativně nenákladnou, ale názornou učební pomůckou při výuce fyziky.

Porovnat GUI aplikaci navrženou v MATLABu® s Famulem® je těžší. Kvalita této aplikace záleží na autorovi a tak „kus od kusu" může dosahovat nebo naopak někde v hloubi upadat za aplikací ve Famulu®. Jako kritérium k porovnání vezmu výhody/nevýhody, které nabízí MATLAB® jako editor GUI aplikace. Pomineme-li starší verze MATLABu® (MATLAB386®) přesouváme se v oblasti osobních počítačů do uživatelsky příjemnějšího prostředí a to do OS MS Windows®. Zvyšují se však i celkové nároky, které musíme zaplatit za pohodlí. GUI aplikace nám prostřednictvím MATLABu® může nabídnout tyto vlastnosti:

- Výpočty v oblasti reálných, komplexních čísel, vektorových polí a matic
- Libovolný grafický výstup omezený nároky na hardware, vystup na libovolnou tiskárnu podporovanou OS MS Windows®
- Interaktivní vkládání rovnic, příkazů s jejich výpočtem a grafickým výstupem
- Podporu moderních programovací jazyků (C®, C++®, Java®)
- Integrované prostředí se silnou podporou knihoven zabývajícími se jednotlivými obory fyziky, matematiky apod.
- Grafické uživatelské prostředí pro snazší a rychlejší práci

Příjemné prostředí a jednoduché ovládání GUI aplikací je vykoupeno nároky na obsluhující hardware, který musí umět pracovat s OS MS Windows® a mít dostatek paměti pro spouštěnou aplikaci. Její náročnost se odvíjí od množství použitých funkcí a grafickém zpracování. To je záležitost individuálního řešení autora projektu a daných podmínek. Nároky, které s sebou nesou GUI aplikace, jsou finančně náročnější, ale s postupujícím rozvojem počítačového vybavení a jeho klesající ceny, budou dostupné všude.

Závěr

Cíl mé práce byl vytvořit odborný text, který by se zabýval tvorbou učebních aplikací fyziky za pomocí vestavěného GUI prostředí v MATLABu®. V textu práce jsem naznačil a stanovil zásady designu aplikace, její funkčnosti a chyb, kterým by se člověk píšící takovéto aplikace, měl vyvarovat.

Během tvorby diplomové práce, jsem se potýkal s problémem prakticky neexistujícího uceleného odborného textu pro tvorbu aplikací GUI za pomoci MATLABu®. Problém jsem řešil hledáním odkazů na internetu a nápovědou vytvořenou pro MATLAB®. To mělo za následek naprogramování některých pasáží programu „neprofesionálně" a za pomoci „berliček". Způsob řešení některých problémů přikládám také krátké době, kdy se MATLABu® věnuji a také specializaci na jiné programovací jazyky, které se od sebe liší syntaxí příkazů a tak i následnému vzniku banálních chyb, které se špatně hledají.

V době psaní bakalářské práce nebyl dostatek času na vypilování jednotlivých detailů celého programu a čas musel byt spravedlivě dělen mezi doprovodný text a samotný ukázkový program. Dané téma práce mě však natolik zaujalo a především tvorba demonstrační aplikace se pro mne stala zábavou, které bych se chtěl i nadále věnovat a doplnit jej o další nápady a funkce, které by měli napomáhat didaktice jednotlivých odvětví fyziky. Ať se již bude jednat o komfortnější ovládání nebo o rozšíření výběru jednotlivých jevů.

V své práci, jsem dosáhl, poznatku, že i zdánlivě náročné jevy a problematika ve fyzice může být za pomocí MATLABu® a jeho graficky uživatelského rozhraní řešena elegantně, rychle a pohodlně a tím i přístupná široké skupině uživatelů PC.

Seznam použité literatury

[1] Dušek, F.: *MATLAB a SIMULINK - úvod do používání*, Univerzita Pardubice, Pardubice, 2002, ISBN 80-7194-273-1

[2] Görner, V.: *Programový systém MATLAB*, ČVUT, Praha, 1991, ISBN 80-01-00580-1

[3] MATLAB – jednotlivé manuály k jednotlivým toolboxům, a MATLAB – help

[4] Bartoš, P.: - *Učební texty Fyzika – Kmitání, Vlnění A Optika*

[5] Kozák, Š., Kajan, S.: *Matlab – Simulink I učebnice zaměřená na hlavní modul Matlab*, STU v Bratislavě, Bratislava, 1999, ISBN 80-227-1213-2

[6] Kozák Š.: *Matlab – Simulink II učebnice zaměřená na Control System Toolbox*, STU v Bratislavě, Bratislava, 1999, ISBN 80-227-1235-3

[7] MATLAB – sborník příspěvků z konference, všechny doposud vydané díly

[8] Stejskal, V a kol.: *Kmitání s MATLABem*, ČVUT, Praha, 2002, ISBN 80–01-02435–0

Seznam literatury dostupné na internetu

[9] MATLAB Tutorials, adresa: http://www.math.siu.edu/matlab/tutorials.html

[10] Mathworks – Web Pages, adresa: http://www.mathworks.com

[11] MATLAB GUI for EE, adresa: http://people.msoe.edu/~saadat/matlabgui.htm

[12] Humusoft – Web Pages, adresa: http://www.humusoft.cz

[13] Introduction to MATLAB and SIMULINK, adresa: http://people.msoe.edu/~saadat/Matlab.htm

[14] Control Tutorials for MATLAB, adresa: http://www.engin.umich.edu/group/ctm/

[15] ČVUT Fakulta Elektroenergetiky, adresa: http://k315.feld.cvut.cz/index.php?node=4&kod=cz&id=download

Anotace

Hlavní cíl práce se zaměřuje na vytvoření odborného textu, který by posloužil jako nástroj pro vývoj učebních aplikací Fyziky a fyzikálních jevů. Jako programovací jazyk byla zvolena z modelovacího a fyzikálního hlediska aplikace MATLAB® s nástavbovým prostředím pro tvorbu GUI aplikací. Práce se věnuje základnímu seznámení s programovým balíkem MATLAB® a jeho historii přes zaměření na popis a práci s GUI prostředím, které je v aplikaci integrováno. Velký důraz je kladen na definici pravidel pro správný návrh celkového designu výukové aplikace s její funkční a výpočetní částí, tak aby byly maximálně efektivní pro výuku a user-friendly pro uživatele. Snaha takto skloubit obě vlastnosti do jedné aplikace by měla přinést jako výsledek software s velkým didaktickým potenciálem.

Synopsis

This work deals with formation of a scientific text, which would do a good turn as a development teaching application device for physics and physical phenomena. From physical and modelling aspect the MATLAB® with penthouse background for GUI applications formation as a programming language was selected. The work attends to elementar introduce with programme box MATAB® and its history through description sight and work with GUI backgroung which is integrated in the application. The big emphasis on definition of the rules for regular design of teaching application, with its functional and calculation part is posed. It should be maximally impressive for user-friendly users teaching. Tendency joint these two properties into one application should give software with a big didactic potential as a result.