JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

PEDAGOGICKÁ FAKULTA

KATEDRA FYZIKY

Graphical User Interface v programu MATLAB

Diplomová práce

Vedoucí práce: RNDr. Petr Bartoš, Ph.D. **Autor:** Bürger Tomáš

Anotace

Tato práce obsahuje celkem 92 stran, na kterých je ukázána tvorba GUI. Práce začíná kapitolou o MATLABu. Po této kapitole se věnujeme tvorbě uživatelské aplikace, dále následuje Graphical User Interface. V další části diplomové práce se věnujeme tvorbě GUI, ve které si ukazujeme vzorový příklad pro vykreslení sinu a cosinu. Na závěr práce nalezneme věnování Dopplerovu jevu a jeho následný GUI.

Abstract

This work contains a total of 92 pages on which is shown creating GUI. The work begins with a chapter on MATLAB. After this chapter we deal with the creation of custom applications, followed by a Graphical User Interface. In another part of the disertation we consider the creation of GUI, which show a model example for plotting sine and cosine. Finally, please visit dedication Doppler phenomenon and its subsequent GUI.

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 diplomové práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG, v souladu s § 47b zákona č. 111*1998 Sb. v platném znění.

V Českých Budějovicích Podpis studenta

Poděkování

Na tomto místě bych rád poděkoval RNDr. Petr Bartoš, Ph.D., za pečlivé přečtení textu, cenné připomínky a v neposlední řadě i za pomoc pří vývoji jednotlivých aplikací.

Obsah

Úvod

Jako téma mé diplomové práce, jsem si vybral Graphical User Interface v programu MATLAB. Toto téma jsem si zvolil, protože mě zaujal jeho název a později po získání bližších informací, od pana Mgr. Petra Bartoše PhD., se mi toto téma jevilo velmi zajímavě.

Práci jsem si rozdělil do pěti kapitol. V první kapitole, se zabývám historií programu MATLAB a tím, co nám program nabízí. Dále, zde v první kapitole popisuji prostředí MATLABu.

Do druhé kapitoly, jsem zařadil uživatelskou aplikaci, u které jsem popsal model, předvedl m-file editor a figure a dále předvedl způsob vytvoření aplikace.

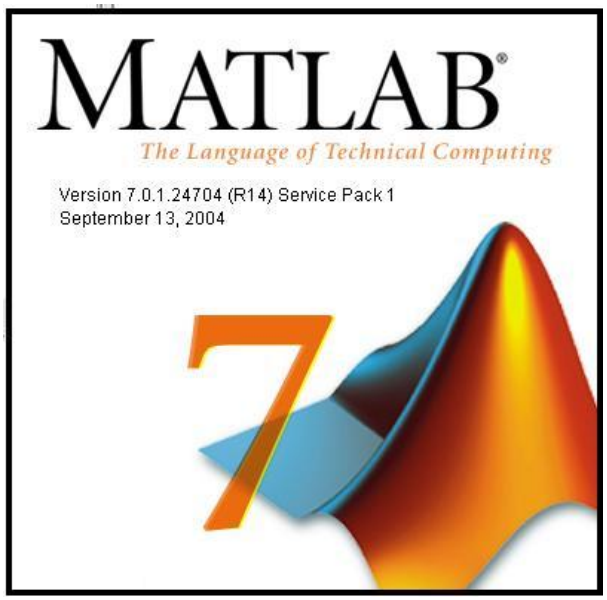
Ve třetí kapitole popisuji GUI, kde ukazuji princip tvorby aplikací, GUIDE a vytváření grafické aplikace.

Do čtvrté kapitoly jsem zahrnul postup při tvorbě GUI. Tento postup, se snažím dělat velmi podrobně a srozumitelně, což je ukázáno i na jednoduchém příkladu jak postupovat. Kroky jsem se snažil pečlivě a srozumitelně popsat.

V poslední kapitole používám Dopplerův jev, ke kterému vytvářím aplikaci. Čtvrtá a pátá kapitola je vytvářena za pomocí programu MATLAB, se kterým jsem se naučil pracovat a ovládat jeho funkce, které mi postačili k vypracování této práce.

1. MATLAB©

MATLAB® [1] je vysoce výkonný jazyk, pro technické výpočty. Integruje výpočty, vizualizaci a programování, do jednoduše ovladatelného prostředí, kde problémy a řešení jsou vyjádřeny pomocí dobře známých matematických vztahů. Aplikace MATLAB® je dnes považována za jedno z nejvýkonnějších interaktivních prostředí, pro vědecké a inženýrské výpočty a jejich grafickou prezentaci. Výhodou zpracování 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 širokou vědeckou veřejnost, bez nutnosti hlubokých znalostí programovacích jazyků.



Obr. 1: Logo MATLAB®

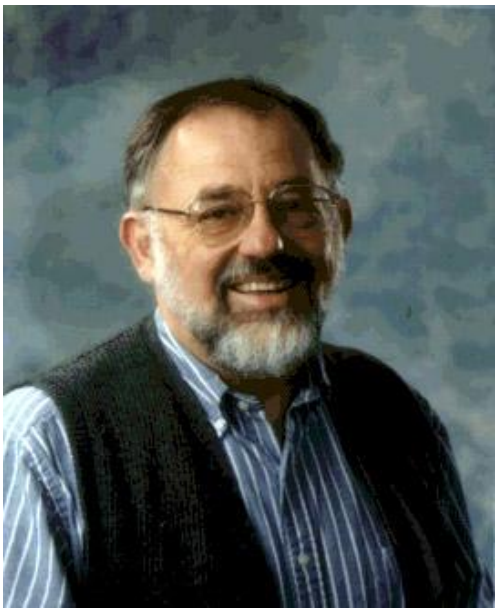
1.1. Historie a vývoj aplikace MATLAB

Název MATLAB je zkratka ze slov „matrix laboratory“. MATLAB byl vyvinut před lety s podněty od mnoha uživatelů. Výpočetní systém MATLAB se během uplynulých let stal celosvětovým standardem, v oblasti technických výpočtů a simulací ve sféře vědy, výzkumu, průmyslu i v oblasti vzdělávání.

MATLAB poskytuje svým uživatelům nejen mocné grafické a výpočetní nástroje, ale i rozsáhlé specializované knihovny funkcí, spolu s výkonným programovacím jazykem čtvrté generace. Knihovny jsou svým rozsahem využitelné prakticky ve všech oblastech lidské činnosti.

MATLAB byl vyvinutý v projektech LINPACK a EISPACK, který měl za úkol poskytovat jednoduchý přístup k matematickým knihovnám. Primárně byl určen pro operační systémy Unix, odkud získal řadu dobrých vlastností. Ty si přenesl i do prostředí MS Windows®, které se projevuje velmi jednoduchým komunikačním rozhraním, příkazovým řádkem.

Historie MATLABu se mapuje od poloviny 70. let, kdy ve Spojených státech započal zmíněný vývoj matematických knihoven LINPACK a EISPACK, určených pro programovací jazyk FORTRAN®. Tento vývoj byl podporován Národní Vědeckou Nadací (*National Science Foundation*) a byl veden týmem odborníků v čele s Clevem Molerem (obr. 2).



Obr. 2: Cleve Moler

Ve FORTRANu® ,se kolekce knihoven zahrnující postupy při řešení lineárních rovnic nazývá LINPACK, který ve spojení s balíčky knihoven EISPACK dopomáhá k jejich numerickému řešení. Pro vývoj softwarových architektur se tyto dva druhy knihoven staly základním stavebním kamenem.

Cleve Moler v době, kdy se stal vedoucím katedry počítačového výzkumu na univerzitě v Novém Mexiku (konec 70. let), dostal myšlenku využít tyto dvě knihovny k výuce posluchačů lineární algebry. Z důvodu složitosti programovacího jazyku FORTRAN®, začíná Moler vyvíjet jednodušší program, který umožní přístup k funkcím obsažených jak v LINPACKu tak i ESPACKu. Výsledkem jeho snažení, je program nazvaný MATLAB® . Tento název vznikl jako zkratka ze slov MATrix LABoratory. V následujících letech Moler podniká jednotlivé stáže po počítačových univerzitách, na kterých svůj MATLAB® představuje jak studentům, tak i profesorům. Zároveň rozšiřuje a vylepšuje jeho aplikace. Program se díky své jednoduchosti brzy začíná šířit mezi studenty a profesory jak na univerzitách v USA, tak i v Evropě.

Obr. 3: John Little

Při návštěvě Molera, na Standforské Univerzitě a jeho přednesení programu MATLAB, se Moler setkává s inženýrem John Littlem (obr. 3.), kterému se program na tolik zamlouvá, že ho chce využít ve vývoji inženýrských aplikací. Proto se v roce 1983 rodí za přispění Cleva Molera a Steva Bangerta nový tým pro vývoj druhé generace, která již bude profesionální formou MATLABu®. 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.

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

MS Windows®

MS Dos®

LINUX®

Solarius®

IRIX®, IRIX® 64

AIX

HP-UX

První verze (pro PC XT), vznikla kolem roku 1985. Jako většina programů té doby, se potýkala s nedostatkem paměti, což omezovalo hlavně největší možnou velikost matic, se kterou bylo možno provádět výpočty. Další verze, byla určena speciálně pro počítač PC AT, kde byla velikost matic omezena 16 MB (maximální velikost paměti). Tato verze se již stala více atraktivní, ale s jedním velikým neduhem a to ve formě vysoké ceny pamětí. V této době, nebylo zvykem počítače osazovat dostatečně velikou pamětí, potřebnou pro správný chod této verze, a virtuální paměť MATLAB ještě nevyužíval. Pro upřesnění, práce s virtuální pamětí 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, což je dáno rychlostí 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). Virtuální paměť začal MATLAB používat s nástupem počítačů s procesorem 80386 a verze MATLAB386® se stala jednou z nejoblíbenějších verzí pod OS MS Windows®, výpočty ale byly stále velmi pomalé. Pro ilustraci, výpočet inverze matice 1000x1000 trval několik minut. Poslední verze MATLAB386 se využívá na počítačích pracujících na systému MS DOS.

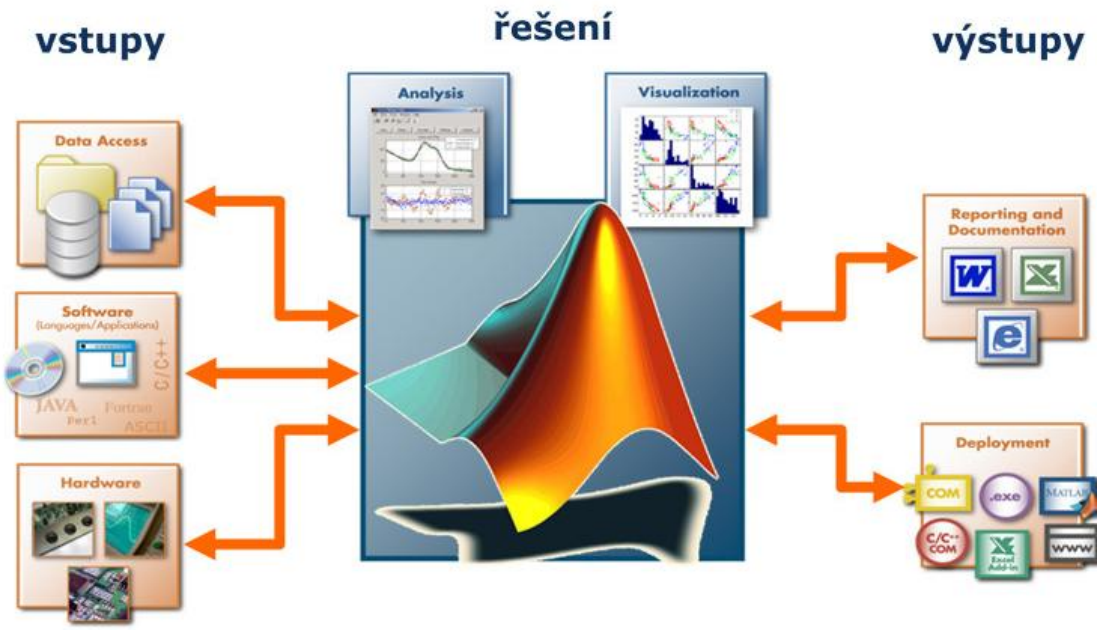
V roce 1994 byla na trh uvedena verze pro Windows, což s sebou přineslo velkou výhodu v podobě větších možností grafiky. Nevýhodou bylo zpomalení výpočtů. Čas procesoru byl totiž z velké části věnován potřebám Windows. Někdy se i stávalo, že rychlost výpočtů byla pomalejší než u předchozí verze.

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 koncem 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. Přehled verzí [1]:

- rozmezí let 1984-1990: MATLAB 1.0, MATLAB 2, MATLAB 3, MATLAB 3.5
- rozmezí let 1991-1997: MATLAB 4, MATLAB 4.2c, MATLAB 5.0, MATLAB 5.1, MATLAB 5.1.1
- rozmezí let 1998-2005: MATLAB 5.2, MATLAB 5.2, MATLAB 5.3, MATLAB 5.3.1, MATLAB 6.0, MATLAB 6.1, MATLAB 6.5, MATLAB 6.5.1, MATLAB 6.5.2, MATLAB 7, MATLAB 7.0.1, MATLAB 7.0.4, MATLAB 7.1
- rozmezí let 2006-2009: MATLAB 7.2, MATLAB 7.3, MATLAB 7.4, MATLAB 7.5, MATLAB 7.6, MATLAB 7.7, MATLAB 7.8, MATLAB 7.9.

1.2. Co nám MATLAB nabízí

Jak už jsme si řekli, MATLAB je integrované prostředí pro vědeckotechnické výpočty. Nazývá se též programovacím jazykem 4 generace, který obsahuje více jak 1000 funkcí. Jeho tradiční využití spadá do oblastí letectví, kosmonautiky, automobilového průmyslu, automatizace a strojírenství, komunikace a dále finance a ekonomika, energetika a přírodní vědy.



Obr. 4: Nabídka MATLAB

MATLAB nám nabízí [2]:

- rychlé výpočetní jádro
- podpora paralelních výpočtů
- akcelerace výpočtů
- otevřený a rozšiřitelný systém
- působivá 2D a 3D grafika
- konfigurovatelné uživatelské rozhraní Matlab Desktop
- velké množství aplikačních knihoven
- programovací jazyk 4. Generace
- objektové programování
- integrace s jazykem Java
- podpora vícerozměrných polí a uživatelsky definovaných datových struktur
- interaktivní nástroje, pro tvorbu grafického uživatelského rozhraní

- podpora řídkých matic
- interaktivní průvodce importem dat
- zvukový vstup a výstup, animace
- komunikace s externím přístrojovým vybavením (sériová linka, GPIB, VISA)
- výpočetní jádro, pro programy psané ve Fortranu a jazyce C
- distribuce nezávislých uživatelských aplikací: překlad do jazyka C, runtime modul, WWW technologie
- rozsáhlá dokumentace v pdf nebo v on-line hypertextové formě a další

MATLAB se skládá z těchto základních částí:

- výpočetní jádro
- grafický subsystém
- otevřená architektura
- pracovní nástroje
- toolboxy.

1.2.1. Výpočetní jádro

Mezi nejpodstatnější části numerického jádra MATLABu se řádí algoritmy pro operace s maticemi reálných a komplexních čísel. Program MATLAB nám samozřejmě umožňuje provádět všechny běžné operace jako je násobení, dělení, sčítání, odčítání, inverze, atd. Pokud program MATLAB použijeme v nejjednodušší podobě, lze ho použít jako maticový kalkulátor, protože se nám tyto operace zapisují téměř stejně tak, jako bychom je zapisovali na papír.

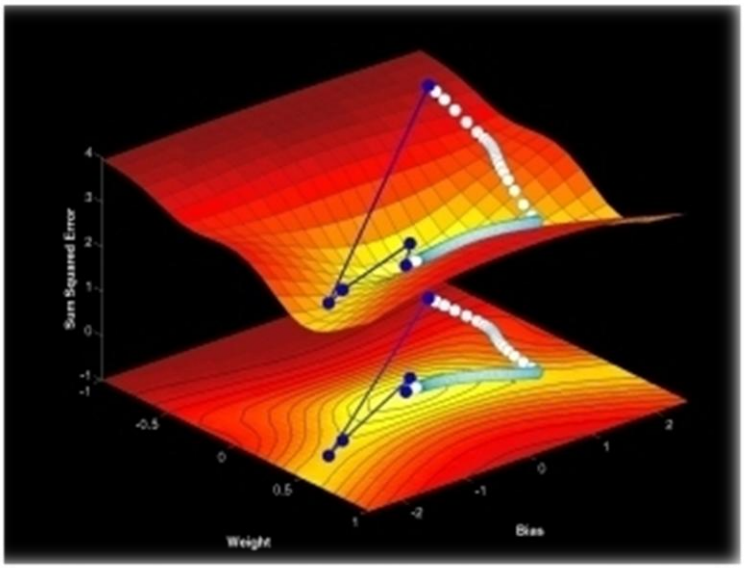
Kromě datových typů, jednodušších než tradiční matice, podporuje MATLAB také typy složitější, jako jsou např. vícerozměrná pole reálných nebo komplexních čísel. Dalším datovým typem jsou tzv. pole buněk, tedy struktury podobné maticím, ve kterých ovšem každý prvek může být jiného typu. Kde nejsou prvky rozlišeny souřadnicemi nýbrž jménem, lze podobně tvořit datové struktury, které připomínají struktury známé s běžných programovacích jazyků. Skládáme-li tyto datové typy, je pak možné vytvořit libovolně složité datové struktury. Podle přání uživatele můžeme zvolit formu ukládaní čísel, buď jako double, která je přednastavena, nebo podle uživatele nějakou úspornější formu.

Vektor reálných čísel může v MATLABu představovat i polynom a operace s polynomy, jsou v programu rovněž obsaženy. Vektory mohou také reprezentovat časové řady nebo signály a MATLAB obsahuje funkce pro jejich analýzu - výpočet střední hodnoty, hledání extrémů, výpočet směrodatné odchylky, korelačních koeficientů, rychlé Fourierovy transformace. Další vlastností jazyka MATLABu, je možnost práce s objekty. Ty uživateli umožňují rozšířit výpočetní prostředí o nové datové typy, na kterých je možno definovat libovolné funkce a operátory.

Výpočetní jádro, je implementováno s využitím základních matematických knihoven s podporou více jader. K využití této vlastnosti stačí pouze základní MATLAB, žádný dodatečný toolbox ani psaní paralelních algoritmů není potřeba. Není ani nutné nijak upravovat starší programy, zrychlení se projeví automaticky.

1.2.2. Grafický subsystém

Grafika v MATLABu umožňuje snadné zobrazení a prezentaci výsledků získaných výpočtem. Je možné vykreslit různé druhy grafů, ať už dvourozměrný (funkce jedné proměnné), třírozměrné (funkce dvou proměnných), dále histogramy, koláčové grafy a další. U všech grafických objektů je možno měnit vzhled, jak při jejich vytváření, tak i po jejich nakreslení. Dále je možno stínovat třírozměrné grafy s určením zdroje dopadajícího světla, animovat grafy a další. Tyto efekty lze ve většině případů dosáhnout jedním nebo několika málo příkazy a pro jejich rychlé vykreslení se používá algoritmus Z-buffer nebo technologie OpenGL, pokud ji použitý počítač podporuje.



Obr. 5: Grafické porovnání výsledků učení neuronové sítě

Obrázky v grafických oknech MATLABu navíc nejsou statické, každý již nakreslený objekt má přiřazen identifikátor, jehož prostřednictvím je možné měnit vlastnosti objektu a tím i jeho vzhled. Vzhled grafických objektů je také možno měnit interaktivně, pomocí lišty nástrojů umístěné pod záhlavím obrázku.

1.2.3. Otevřená architektura

Otevřená architektura je vlastnost, která nejvíce přispěla k rozšíření MATLABu. MATLAB je úplný programovací jazyk, ve kterém uživatelé můžou vytvářet funkce přímo na míru jejich aplikaci. Tyto funkce se způsobem volání nijak neliší od vestavěných funkcí a jsou uloženy v souborech v čitelné formě. Dokonce většina funkcí s MATLABem dodávaných je takto vytvořena a opravdu vestavěné jsou jen funkce základní. To má dvě velké výhody: jazyk MATLABu je téměř neomezeně rozšiřitelný a kromě toho se uživatel může při psaní vlastních funkcí poučit z dodaných algoritmů.

1.2.4. Pracovní nástroje

MATLAB je koncipován tak, že umožňuje programování aplikací. Programovací jazyk obsahuje všechny nezbytné příkazy pro psaní programů. Tento jazyk se nazývá úplným programovacím jazykem 4. generace a to z důvodu, že má snadnou zvladatelnost a je jednoduchý, i přes to lze v něm možno vyvíjet složité aplikace. Základním nástrojem výpočetního systému je uživatelské rozhraní MATLAB Desktop. Pracovní nástroje jako prohlížeč adresářů a souborů, prohlížeč pracovního prostoru, okno historie příkazů, interaktivní spouštěč aplikací, editor, debugger, profiler, hypertextová nápověda a příkazové okno jsou do prostředí plně integrovány.

Uživatel nalezne vše potřebné k programování a přizpůsobení zdrojových kódů. Systém navíc nabízí integrovanou tvorbu grafických prvků (tlačítek, menu atd.) a podporu, která usnadňuje načítání dat z jiných zdrojů. Dále si můžeme nastavit prvky pracovního prostředí podle svého, lze si tak vytvořit prostředí pro začátečníka i profesionála.

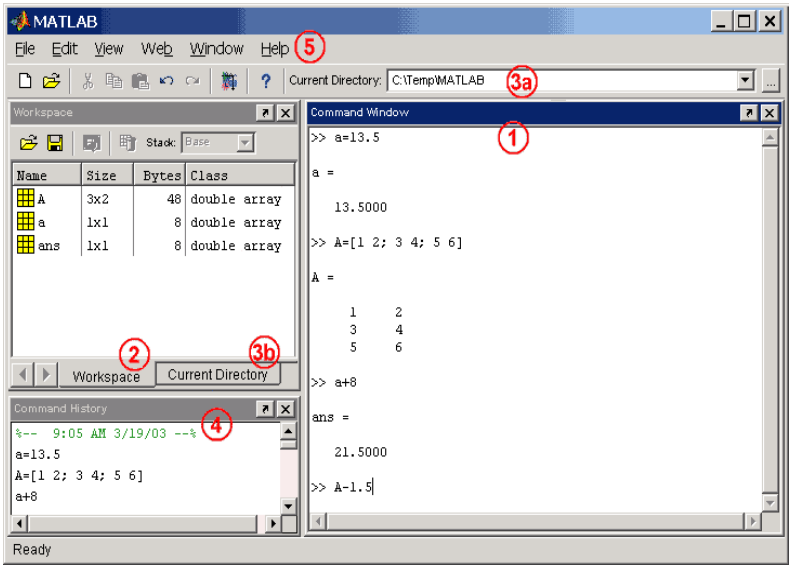
Další významnou přednosti MATLABu je jeho velmi těsná integrace s programovacím jazykem Java, kde můžou být objekty Java použité přímo programem MATLAB. Kromě toho je možné k MATLABu připojovat také moduly napsané v jazyce C a ve Fortranu.

1.2.5. Toolboxy

Otevřená architektura MATLABu vedla ke vzniku knihoven funkcí, nazývaných toolboxy, které rozšiřují použití programu v příslušných vědních a technických oborech. Tyto knihovny nabízejí předzpracované specializované funkce, které je možno rozšiřovat, modifikovat, anebo jen čerpat informace z přehledně dokumentovaných algoritmů. Toolboxy, byly většinou vyvinuty na univerzitě týmem okolo významného odborníka z dané oblasti. Součástí každého toolboxu je rozsáhlá dokumentace, která obsahuje i odkazy na vědecké zdroje a podrobně popisuje použité vědecké algoritmy. Toolboxy si může uživatel zakoupit jako přídavné moduly, např. pro návrh filmů, práci s neuronovými sítěmi atd. Toolboxy se budeme zabývat ještě dále v textu.

1.3. Popis prostředí MATLABu

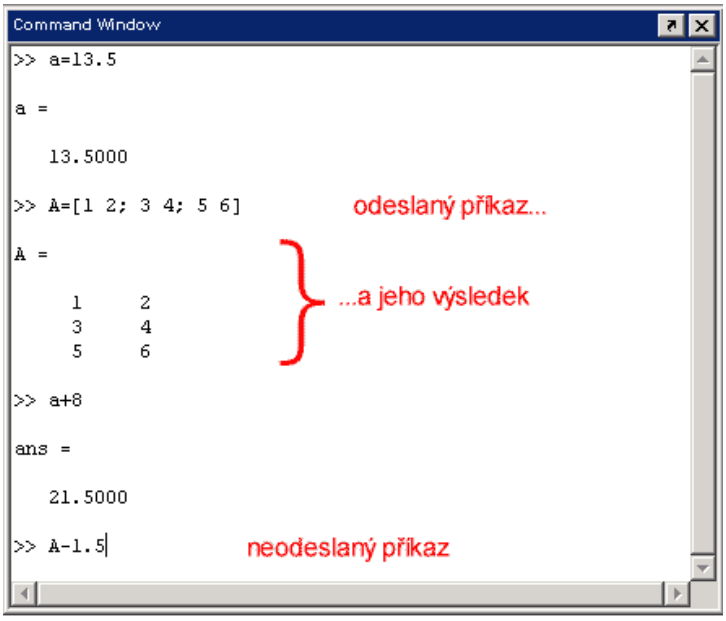
Po spuštění systému se objeví okno složené z několika částí (viz obr. 6). Nejdůležitější z nich je (pod)okno Command Window. Uspořádání (pod)oken můžeme změnit, resp. můžeme některá (pod)okna zavřít. Obnovení původního nastavení, provádíme pomocí menu View (viz bod 5).



Obr. 6: Okno MATLABu

Bod 1: Okno **Command Window(obr. 7)** - práce v dialogovém režimu

V tomto místě, lze program MATLAB používat jako kalkulačku. Pracujeme zde na principu, že po napsání příkazu pomocí tlačítka enter odešleme příkaz, který se nám ihned provede, nebo vyhodnotí (např. `2+3` ENTER).



Obr. 7: Okno Command Window

K editaci lze použít několik příkazů:

klávesa	význam
ENTER	odešle řádek ke zpracování
ESC	smaže celý řádek
DEL	smaže jeden znak (za kurzorem)
BACKSPACE	smaže jeden znak (před kurzorem)
HOME	přesun kurzoru na začátek řádku
END	přesun kurzoru na konec řádku
→	posun kurzoru o jeden znak vpravo
←	posun kurzoru o jeden znak vlevo
CTRL+ →	posun kurzoru na začátek dalšího slova
CTRL+←	posun kurzoru na začátek předchozího slova
↑a ↓	listování dříve napsanými příkazy (více viz poznámka u bodu 4)

Sami si můžeme určit způsob zobrazování výsledku, a to pomocí příkazu *format*, kde můžeme například zadat:

- >> *format compact* (kolem výsledků nebudou prázdné řádky)
- >> *format long* (čísla se nebudou vypisovat se čtyřmi, ale se čtrnácti desetinnými místy)

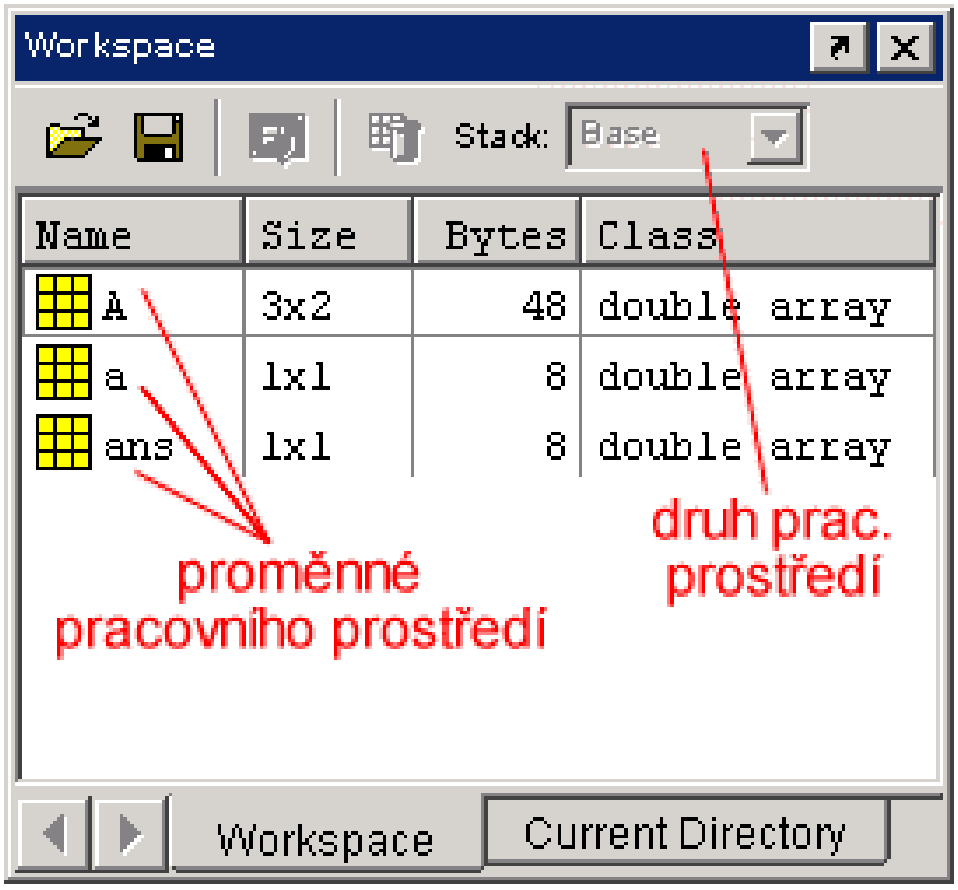
Původní nastavení obnovíme příkazem >> *format.* Více o těchto příkazech se můžeme dočíst v helpu, který vyvoláme pomocí příkazu >> *help format.* Výsledky, které jsou nepojmenované, se ukládají v proměnné *ans. Proměnná ans* je vytvářena automaticky, pokud některý z příkazů potřebuje vypsat hodnotu, kterou jsme nepřiřadili do žádné proměnné. Proměnná *ans* , obsahuje poslední zobrazenou nepojmenovanou hodnotu.

Příklad: pokud proměnná `ans` neexistovala, vytvoří se po vyhodnocení příkazu `>> 2*3-5`. Pokud proměnná `ans` již existovala, tak se změní její hodnota.

Bod 2: Okno Workspace (obr. 8) - práce s proměnnými

Okno Workspace zobrazuje všechny dostupné proměnné pracovního prostředí. Pracovní prostředí je základní (Base) nebo lokální (pro funkce). Umožňuje práci s proměnnými, například:

- smazání proměnné: pomocí klávesy DEL
- zobrazení hodnoty proměnné: dvojklikem
- uložení všech proměnných do souboru pomocí ikonky s disketou.

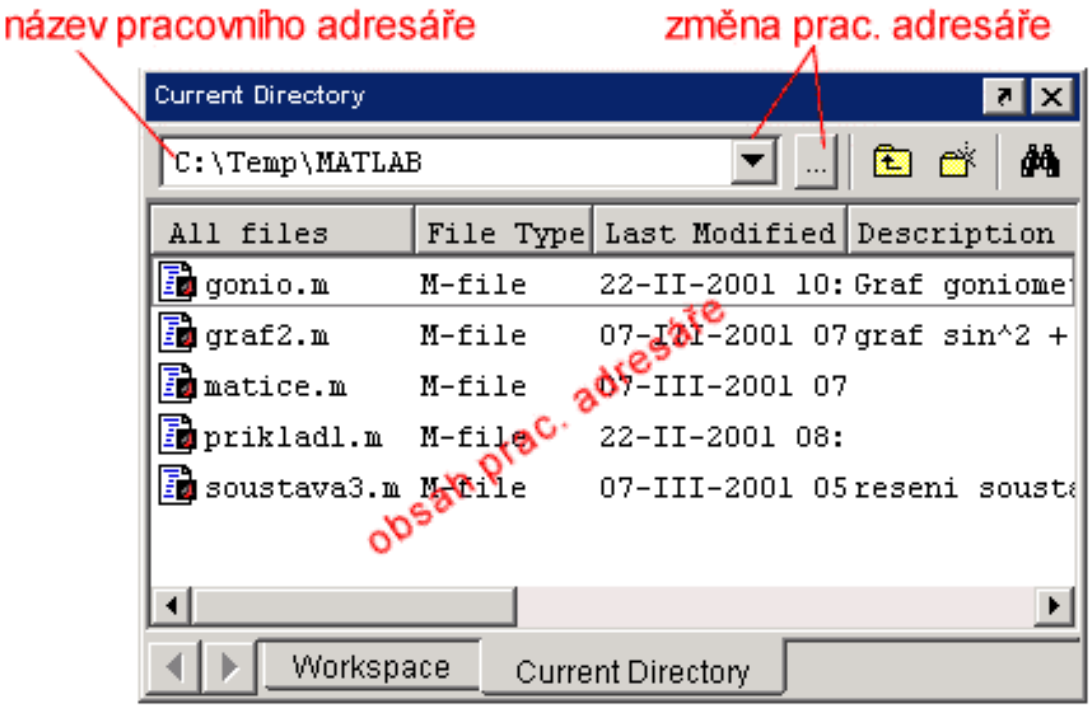


Obr. 8: Okno Workspace

Ačkoliv nám okno Workspace usnadní práci, nemusíme ho nezbytně používat. Veškerou práci s proměnnými zvládneme pomocí příkazů (Command Window). Přehled proměnných lze získat příkazem `>> whos` (nebo `>> who` - jen názvy proměnných), výpis proměnné uvidíme po odeslání jejího názvu.

Bod 3: Okno **Current Directory** (obr. 9)- pracovní adresář

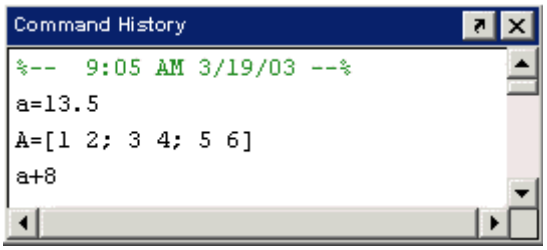
Toto okno zobrazuje obsah aktuálního adresáře, čehož lze také docílit pomocí příkazu >> *dir* v Command Window. Dále nám toto okno umožňuje změnit pracovní adresář, např. pomocí ikony se třemi tečkami (na obr. je tato část zobrazena **bodem 3a** a to z důvodu rychlejšího přístupu). Změnu pracovního adresáře, můžeme také docílit pomocí příkazu >> *cd celý_název_adresáře* (>> *cd D:\temp\matlab*).



Obr. 9: Okno Current Directory

Kontextové menu souboru, umožňuje standardní práci se soubory pracovního adresáře (přejmenování, kopie, přesun, smazání) a navíc také např. otevření m- souboru. Kontextové menu se otevře ihned po stisku tlačítka myši, na název daného souboru. Tyto soubory lze řadit několika způsoby ať už podle názvu (All files), podle typu (File Type), podle data (Last Modified), nebo podle jejich popisu (Description). Ale můžeme si také nechat zobrazit pouze soubory určitého typu, kdy si vybereme způsob řazení a poté zaškrtnutí určité volby si vybereme typ zobrazovaných souborů.

Bod 4: Okno **Command History** (obr. 10)- přehled použitých příkazů



Obr. 10: Okno Command History

Toto okno je klasická historie, která obsahuje všechny použité příkazy. Jsou označovány podle spuštění a to tak, že každé spuštění MATLABu je označeno datem a časem (obr. 10 – zelený text). Pomocí historie můžeme opětovně spouštět dříve zadané příkazy, a to pomocí dvojkliku na vybraný příkaz. Pokud přetáhneme příkaz pomocí myši do Command Window můžeme upravovat, nebo opravovat použité příkazy. Ale okno Command History nemusíme používat, protože v Command Window lze **listovat použitými příkazy** za použití šipek (Up, Down). Pokud před stiskem šipky napíšeme začátek hledaného použitého příkazu (alespoň jeden znak), listuje se jen v názvech těch příkazů, které začínají napsaným textem.

Bod 5: Menu (nabídky)

V tomto bodě si ukážeme, co každá část nabídky znamená:

- **File** nám umožňuje např. vytvořit nebo otevřít M-soubor, ukončit systém, atd.
- **Edit** obsahuje např. položky pro kopírování textu přes schránku Windows, ale také umožňuje smazat Command Window, smazat Command History (přijdeme o historii příkazů) a smazat Workspace (přijdeme o všechny proměnné!).
- **View** umožňuje nastavit vzhled pracovního prostředí, např. ubrat/přidat výše popsaná okna.
- **Web** umožňuje získat podrobnější informace o výrobci, technické podpoře atd.
- **Window** je aktivní při práci v grafickém režimu MATLABu.
- **Help** otevírá rozsáhlou nápovědu k systému MATLAB a jeho součástem.

1.3.1. Režim práce

Režim práce si rozdělíme do tří režimů a to:

- Dialogový režim

Tento režim je přístupný v okně Command Window. V dialogovém režimu je možno MATLAB používat skoro jako kalkulačku s funkcemi. Námi napsaná příkazy se ihned po stisknutí ENTRU odesílají a vyhodnocují. Můžeme zde pracovat s proměnnými z pracovního prostředí (Workspace).

- Programový režim

Tento režim slouží pro editaci m-souborů, skriptů nebo uživatelských funkcí. Programový režim je spjat s oknem m-editorů, což jsou editory m-souboru, u nichž je barevně vyznačena syntaxe příkazů. Další důležitou věcí, kterou obsahuje je *debugger*, který umožňuje hledat chyby ve funkcích nebo skriptech.

- Grafický režim

V tomto režimu si zobrazujeme grafy, jinými slovy slouží tento režim pro vizualizaci výsledků, které otevírá v samostatných oknech *Figure.*

1.3.2. Proměnná

Proměnnou můžeme nazývat objekt, který má svůj název, typ a obsah, nebo hodnotu.

- Název proměnné

V názvu proměnné jsou povoleny jen znaky anglické abecedy spolu s číslicemi a podtržítkem, přičemž název nesmí začínat číslicí a může být dlouhý maximálně 31znaků. V názvu jsou rozlišována malá a velká písmena. Příklad špatně zapsaných názvů:

- 4abs - nesmí začínat číslicí
- Ab ab – nesmí obsahovat mezeru jen _, nesmí ani obsahovat tečku, pomlčku nebo závorky.

Správně zapsaná proměnná může vypadat takto: abc, d, ad99a, aa_DD, atd.

- Typ a hodnota proměnné

Základní pravidlo zní: Každá proměnná je matice. MATLAB tedy nerozlišuje různé typy proměnných. Každá matice může obsahovat jakákoliv komplexní čísla. Občas se

setkáme s maticemi složenými ze znaků. Z hlediska rozměrů matic rozlišujeme proměnné na:

- matice (mxn, kde m>1, n>1)
- vektory (mx1 nebo 1xn)
- skaláry (1x1, tedy jen jedno číslo)

- Vytvoření proměnné

Pro vytvoření proměnné, můžeme použít přiřazovací příkaz >> *název_proměnné= výraz.* Ukážeme si tedy, jak vytvořit skalár a vektor nebo matici.

Vytvoření skaláru

>> a=8

>> pom=-2.145…desetinná čísla zadáváme pomocí tečky ne čárky, nebo zlomku.

>> skalar8=15e-2 … nebo můžeme čísla zadávat ve vědeckém formátu.

Vytvořeni vektoru nebo matice

Matice vytváříme pomocí hranatých závorek, ve kterých jednotlivé řádky oddělujeme středníkem, přičemž jednotlivé sloupce oddělujeme mezerou nebo čárkou. Důležité pro vytváření je, že se počet prvků v každém řádku musí shodovat. Nesmíme si také plést při psaní desetinných čísel to, že jako čárka se zde používá tečka. Ukážeme si, zde vytvoření:

- Řádkového vektoru:

>> v1=[2 0.1 -3.7 4/5 0.14]

- Sloupcového vektoru

>> vek2=[5; 0.1; -4.1; 4/5; 0.14]

- Matice 3x2:

>> maticeA=[1 2; 0.1 -3; .7 1/4]

Zobrazení hodnoty proměnné

Zobrazit hodnotu proměnné můžeme dvěma způsoby a to buď pomocí okna Workspace, nebo v dialogovém režimu napíšeme název proměnné >> *název_proměnné*.

Smazání proměnné

Jako u zobrazení proměnné, máme dva stejné způsoby. Příkaz pro smazání je *clear*.

>> *clear název_proměnné* … smaže vybranou proměnou

>> *clear název1 název2* ... smaže vybrané proměnné, oddělujeme je od sebe mezerou

>> *clear* … smaže všechny proměnné z Workspace

S proměnnými můžeme provádět řadu operací, které zde nebudeme vypisovat podrobně. Informace lze nalézt například v Helpu.

1.3.3. Výraz

Výrazem můžeme nazývat posloupnost konstant, názvů proměnných, operátorů. Pokud je výraz pro MATLAB smysluplný, tedy může ho program vyhodnotit, tak po jeho napsání a stisku klávesy ENTER, je výraz ihned vyhodnocen. Po vyhodnocení výrazu vzniká vždy hodnota. Tuto výslednou hodnotu výrazu můžeme odebrat, uložit do nějaké jiné proměnné, jinak je uložena do proměnné *ans* a zobrazena. Pokud nepotřebujeme její hodnotu vidět, můžeme toto potlačit napsáním středníku na konec řádku. Samozřejmě že tato hodnota i přes potlačení výpisu, vznikne ve Workspace.

1.3.4. Příkaz

Příkaz se v MATLABu bere jako zapsaný výraz, který je odeslán ke zpracování. Ve své podstatě je brán jako příkaz k nějaké činnosti. Kromě odeslání výrazů, existuje spousta tzv. pravých příkazů, jako je například příkaz přiřazovací a další.

Ukázka:

- přiřazovací příkaz (=)

  >> *název_proměnné= výrazu*
- podmíněný příkaz, neboli větvení (*if*)
- přepínač (*svitch*)
- cyklus neboli smyčka (*for, while*)

Kromě těchto základních příkazu MATLAB obsahuje i příkazy pro:

- práci s adresářem (např. *cd, dir*)
- práci s proměnnými (např. *save,load,clear*)
- a další

Zapsání jednoho příkazu na více řádků

V případech, kdy potřebujeme v MATLABu zapsat jeden příkaz na více řádků, použijeme tři tečky. Tyto tři tečky se zvýrazní modře a pro MATLAB to znamená, že s provedením příkazu má počkat, protože ještě není celý napsaný. Další řádek nezačíná >>, ale jen v případě čekání MATLABu na dopsání příkazu. Malá ukázka čekání:

Příklad:

>> A = [1.5 -3 4.1; 2…

7 8 0.3; 7 15.1 8/7 0];

nebo

>>A = [1.5 -7…

8; 2 4 8; -8…

2 4.1];

Potlačení vypsání výsledku

V MATLABu některé příkazy ihned vypisují své výsledky. Což vede ke znatelnému zpomalení výpočtu. Pokud nechceme, nebo nepotřebujeme vidět tyto výsledky, lze potlačit výpis výsledku a to tak, že nakonec příkazu napíšeme středník (např. >> a = 24;). Tento postup umožňuje zápis dlouhých příkazů a zpřehledňuje zdrojový kód.

Násilné ukončení příkazu

Pokud potřebujeme z jakéhokoliv důvodu ukončit příkaz, který se právě provádí, je možné právě prováděnou operaci ukončit stiskem **CTRL+C.**

Více příkazů na jednom řádku

Doposud jsme příkazy ukončovali vždy pomocí klávesy ENTER, která zároveň odeslala příkaz ke zpracování. Mnohdy ale chceme odeslat ke zpracování více příkazů najednou. Tehdy můžeme použít buď m-soubory, anebo zápis více příkazů na jeden řádek, přičemž k oddělení jednotlivých příkazů se používá čárka nebo středník [3].

Příklad:

*>> m=8, n=3; vysledek=3*m+0.5*n*

1.3.5. Komentáře

Komentáře slouží většinou k vysvětlení významu jednotlivých příkazů, skriptů nebo funkcí a je velmi vhodné je používat. Komentáře u uživatelských funkcí navíc slouží jako nápověda, kterou umí MATLAB zobrazovat např. příkazem >>*help název*. Komentář začíná znakem procento a končí spolu s koncem řádku. Text komentáře bývá označen zeleně.

1.4. Toolboxy – ukázka toolboxů

Program MATLAB je tvořen jádrem, které obsahuje základní funkce a prostředky. Toolboxy je rozšiřující balíček m-file souborů, který obsahuje funkce a procedury pro práci s daty. Postupně se přidávali nové a nové knihovny funkcí. Díky tomu, se základní prostředí MATLABu rozvinulo a uplatnilo se jeho použití v širším spektru odvětví, která se zabývají výpočty nebo zpracování dat. Toolboxy, pracující na principu systému Handle Graphics, dovolují využívat značně sofistikované matematické postupy. Uživatelům slabším v matematice, umožňují jednoduše zadat naše data, posléze vybrat metodu a nakonec spustit výpočet. Přitom tyto funkce komunikují s uživatelem v pojmech běžných v této oblasti a nenutí uživatele cokoliv vědět o matematických metodách vedoucí k získání výsledku.

Toolboxy jsou určené pro použití v určitých oblastech (firma MathWorks):

- *Obecné použití*

NAG Foundation Tbx, Symbolic Math Tbx, Partial Differential Equation Tbx, Statistic Tbx, Spline Tbx atd.

- *Zpracováni obrazu a signálu*

Wavelet Tbx, System Identification Tbx, Signal processing Tbx, High-Order Spectral Analysis Tbx atd.

- *Návrh řízení*

Control Systém Tbx, Robust Control Tbx, Model Predictice Control Tbx , Nonlinear Control Design Tbx, LMI Control Tbx atd.

Kromě výše uvedených toolbooxů, které pocházejí z dílny firmy MathWorks existují i další od jiných firem. Ty mohou být buď součástí základního balíčku, nebo jsou tvořeny na univerzitní půdě pro řešení specializovaných 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 se, těmito obory [4]:

- **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.

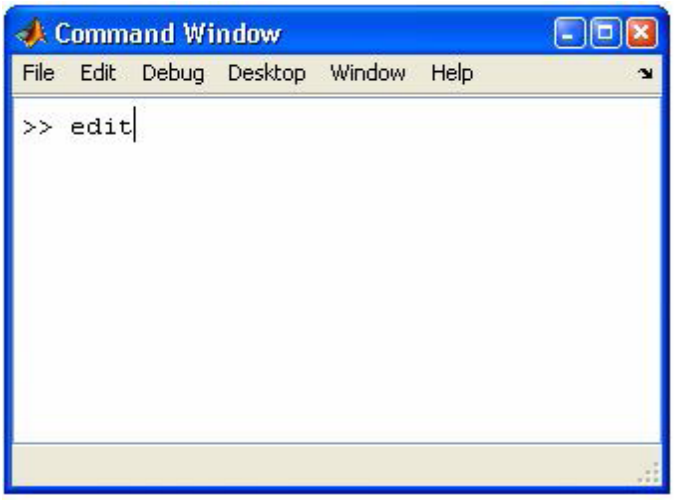
- **Financial Time Series Toolbox –** kolekce nástrojů umožňující práci a analýzu dat proměnných v čase zabývající se peněžním obchodem.
- **Fixed-Income Toolbox –** sada funkcí určených pro výpočty a odhady návratnosti hypotéčních a dlužních úvěrů, manipulaci s cennými papíry a měnou.
- **Fixed-Point 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 následné 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.

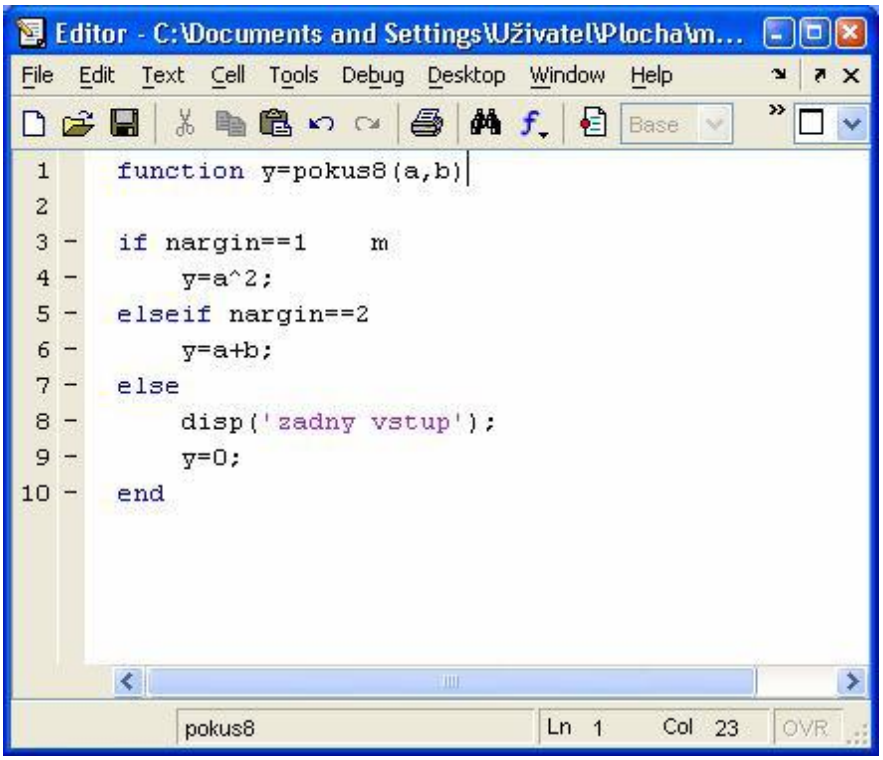
2. Uživatelské aplikace

Při práci se systémem MATLAB může postupovat několika způsoby. Záleží na tom, jaký problém potřebujeme řešit. Pokud chceme spočítat něco narychlo, využijeme zadaní příkazů pomocí COMMAND WINDOW (obr. 11). Do příkazového okna zapíšeme výraz, potvrdíme klávesou ENTER a na obrazovce se zobrazí výsledek, či odezva systému, jako je např. chybové hlášení.



Obr. 11: Vyvolání vestavěného editoru z příkazového okna.

V případech, kdy řešíme složitější úlohy, využijeme způsob, který připomíná práci ve vývojové aplikaci. Data zapisujeme do předem spuštěného editoru a poté uložíme na disk. Zápisem a následným potvrzením v okně příkazů MATLABu dojde k vykonání a spuštění uloženého textu, který je kódem či zdrojovým textem. Zdrojový text můžeme optimalizovat, ladit, spouštět a dokonce provazovat se zdrojovými texty uloženými v jiných souborech. Editor, do kterého můžeme zapisovat příkazy atd., je součástí MATLABu. Můžeme použít i jiný editor, v praxi se však tento způsob práce příliš často nepoužívá [5].



Obr. 12: Příklad zápisu v editoru MATLABu.

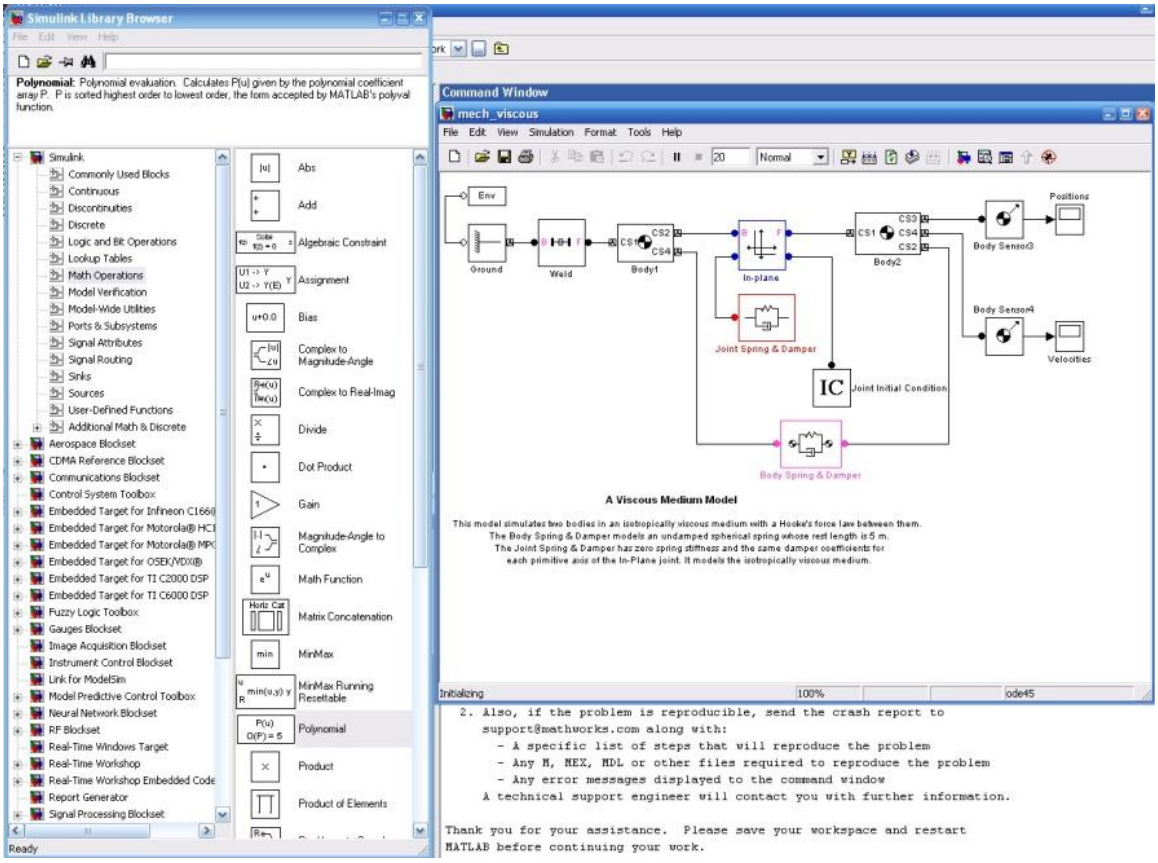
Výsledek naší práce může být soubor, nebo systém souborů, které mohou obsahovat rozsáhlé zdrojové kódy. MATLAB umožňuje programovat aplikace s podporou grafiky. Z této informace vyplývá, že se v MATLABu objevují kromě grafů i jiné grafické prvky jako jsou např.: tlačítka, různá menu, posuvní a další. Díky těmto grafickým prvkům, může naše práce dostat velmi profesionální vzhled.

Grafické prvky můžeme tvořit zápisem příkazů s mnoha parametry v editoru textů nebo můžeme využít automatického generátoru, jenž je do systému MATLAB vestavěn.

2.1. Simulink

Simulink, jako jeden z mnoha objektů, které lze vytvářet v MATLABu stojí tak trochu stranou. Zatímco všechny nové objekty, které si ukážeme, se týkají striktně MATLABu samotného, objekt Model je nástroj části zvané SIMULINK.

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 [6].



Obr. 13: Knihovna prvků

Možností jak vytvářet modely je v SIMULINKU mnoho. Základem je stavba z jednotlivých diskrétních nebo lineárních elementárních prvků. Z knihovny, kterou si uživatel musí uvést sám pomocí ikony na plovoucí liště, vkládáme jednotlivé prvky. Na plovoucí liště máme jednotlivé elementy rozděleny do skupin dle vlastností. Umisťování je jednouché, a probíhá přetahování myší z knihovny na plátno. Jakmile přetaháme potřebné elementy, musíme nastavit propojení mezi jednotlivými elementy a nastavit jejich vstupy a výstupy. Tyto akce jsou rovněž jednoduché a probíhají za

použití myši (kliknutím a tažením pro propojení prvků; dvojklikem pro nastavení vlastností prvků).

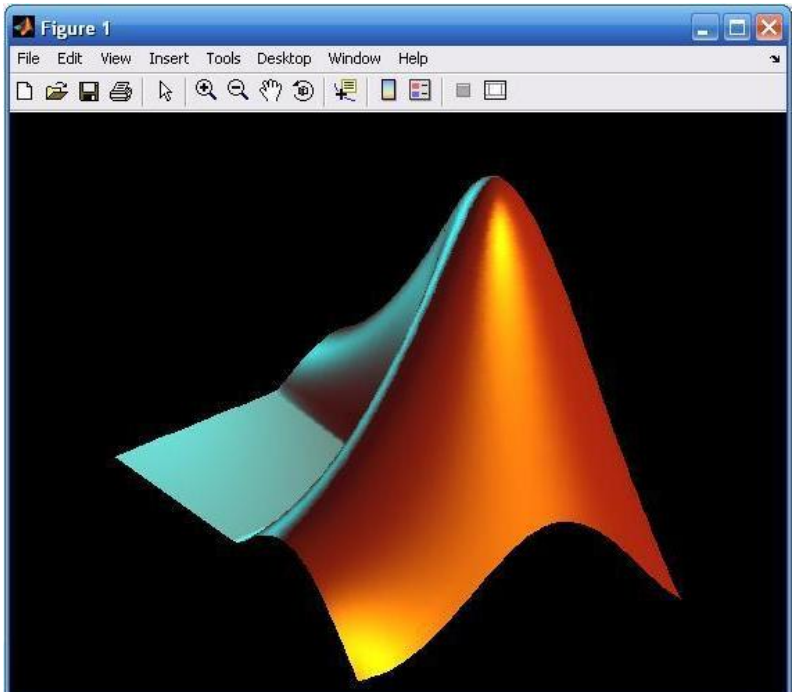
Poté co dokončíme model, přichází na řadu jeho simulace, nebo spuštění. Pomocí ladicích nástrojů odstraňujeme chyby, nastavujeme jednotlivé hodnoty prvků a poté je uložíme ve formátu souboru s příponou*.mdl. Takto uložený soubor se pak může zakomponovat do vyvíjených aplikací.

2.2. Figure

Figure je prvek používající se k vytvoření grafických objektů. Spolu s dalšími prvky tvoří celý komplex MATLAB. Pokud pracujeme z objektem Figure, vytvoříme si na ploše nové okno pro zobrazení grafických výstupů. Toto okno vytvoříme pomocí položky z menu následovně: File, dále New, dále Figure, poté se zobrazí nové okno. Vlastnosti a velikost tohoto okna jsou přednastavené, stejně je tomu tak i u grafiky, tloušťky čar a osy. Většinu objektů ve figure lze dále upravovat podle potřeby.

Protože se jedná o práci s grafikou, je Figure vybaven sadou nástrojů pro práci s grafikou. Jednotlivé modely tudíž můžeme upravovat za pomoci nástrojové lišty nebo menu. Figure se v principu podobá m-file editoru. Celková komunikace je velmi jednoduchá. Výsledek, nebo spíše výsledný výstup lze ukládat do souboru, popřípadě je importovat do formátu kompatibilních v jiných aplikacích a následně využít pro další zpracovaní. Jako příklad můžeme uvést import do JPG formátu a jeho následné využití v MS Wordu, nebo exportovat do souboru *eps* pro využití v Catexu.

Základem pro zpracování grafických objektů ve Figure je čerpání určitých dat, které musíme do něj importovat. Nejlehčí cesta je načtení již existujícího modelu, který již byl dříve vytvořen. Z toho důvodu Figure je hodně úzce navázáno na funkce MATLABu, které nám zajišťují vykreslování obrázků (obr. 14).



Obr. 14: Figure

2.3. M-file editor

Jak již bylo zmíněno, je hlavní prostředkem při práci v MATLABu m-file editor. Jsou to vlastně běžné textové soubory, ve kterých jsou na jednotlivých řádcích zapsány příkazy námi vytvářeného algoritmu se všemi proměnnými a konstantami, funkcemi, 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 [5]

run jméno *souboru*

2.3.1. Práce s m soubory

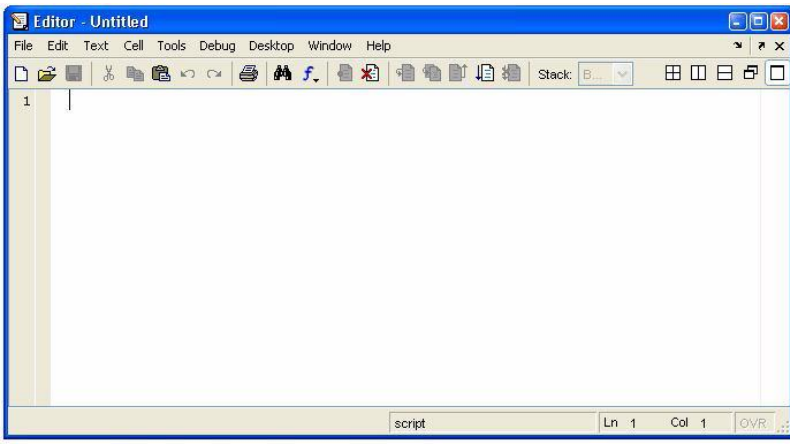
Jak už jsme si řekli soubory obsahující příkazy MATLABu, se nazývají m-soubory nebo mají příponu*.m M-soubory se můžou odkazovat na jiné m-soubory, a obsahují posloupnost příkazu MATLAB. M-soubor může volat sám sebe, dále ho také můžeme vytvořit v libovolném textovém editoru (nejčastěji vestavěným).

Rozlišujeme 2 typy m-souborů a skripty a funkce. Soubory skládající se s dlouhé posloupnosti příkazů se nazývají Skripty, nebo také skriptové soubory (viz dále). Kdežto soubory, které poskytují MATLABu rozšiřitelnost se nazývají funkce, nebo funkční soubory. Funkce nám umožňují přidávat nové funkce k těm, co už existují.

2.3.2. Vytvoření m souboru

M-soubor obsahuje příkazy a povely a je ve většině případů uložen na pevném disku. Jak jsem už poukázal, nejjednodušší způsob jak si vytvořit m-soubor, je použít vestavěný editor. Tento editor vyvoláme v okně příkazů povelem edit a následným potvrzením pomocí klávesy enter (obr. 6).

Odezvou na tento povel je otevření námi požadovaného vestavěného editoru m-souborů (obr. 15).



Obr. 15: Prázdné okno vestavěného editoru m-souborů.

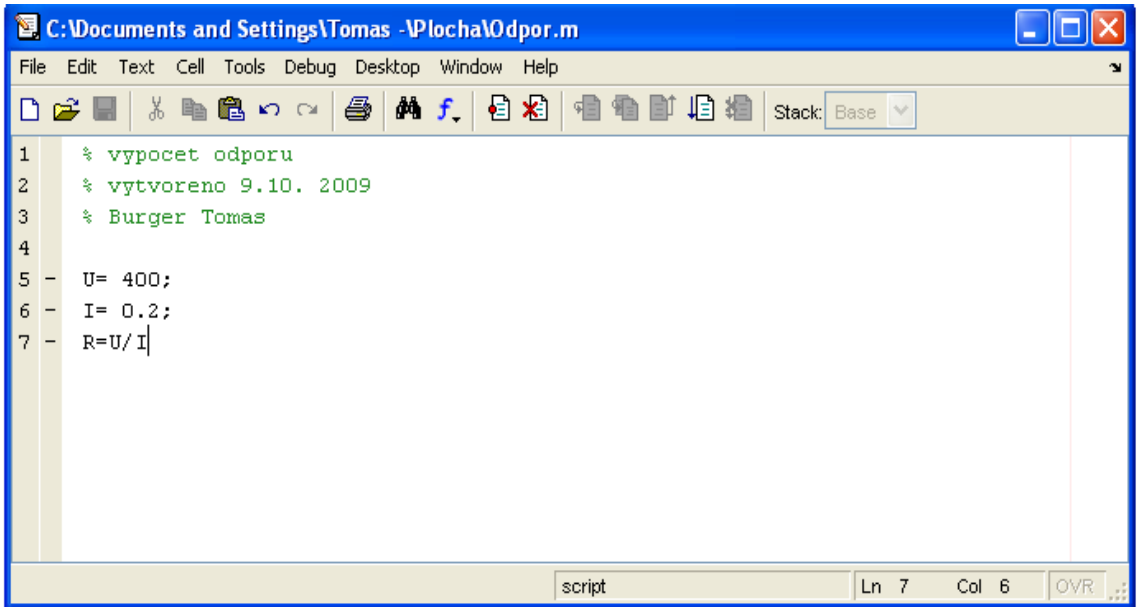
M-soubory jsou obyčejné textové soubory, a proto je lze psát i v libovolném textovém editoru. Z důvodu vyššího komfortu (zvýraznění syntaxe, možnost krokování) je však součástí MATLABu také M-editor/Debugger, který se otevírá v samostatném okně po otevření nebo vytvoření M-souboru (skriptu či funkce).

2.4. Tvorba zdrojového kódu

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*) [7].

2.4.1. Skripty

Za nejjednodušší m-soubor označujeme skripty. A to proto, že obsahují pouze posloupnost příkazů, které se používají v příkazovém okně MATLABu. Ukážeme si skript, který se jmenuje *odpor (*obr. 16).



Obr. 16: Skript odpor

První řádky začínající % jsou poznámky k programu, tzv. komentář. Tyto komentáře nám pomáhají k udržení přehledností v m-souborech, a také nám pomáhají s nápovědou. Lze je vyvolat zadáním: help odpor a stiskem klávesy enter (obr. 17).

Následně se nám vypíšou řádky začínající % u skriptu s názvem odpor. Takto si můžeme prohlížet hlavičky libovolných m-souborů, bez toho aniž bychom museli každý soubor editovat.

```
>> help Odpor
  vypocet odporu
  vytvoreno 9.10. 2009
  Burger Tomas


>>
```

Obr. 17: Výpis hlavičky m-souboru v okně příkazů.

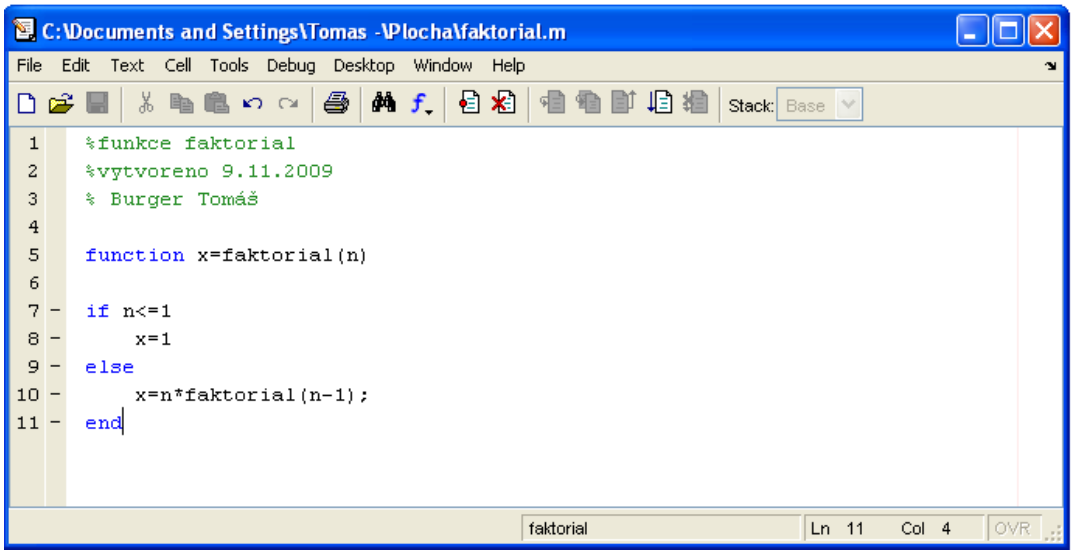
Pokud chceme vytvořený skript z obr. 17 spustit, zadáme v okně MATLABu jméno nového skriptu bez přípony a potvrdíme klávesou enter (obr. 18).

```
>> help Odpor
  vypocet odporu
  vytvoreno 9.10. 2009
  Burger Tomas


>> Odpor

R =

        2000

>>
```

Obr. 18: Spuštění skriptu z prostředí MATLABu.

2.4.2. Funkce

Vedle skriptů využívá MATLAB tzv. funkce. Funkce nám nabízejí mnohem větší škálu možností než skripty. Hlavním rozdílem, který umožňuje tvořit bohatší aplikace, je v tom, že funkce mohou mít jeden čí více vstupních a výstupních parametrů. Všechny hodnoty uvnitř funkce zůstávají skryté, pokud nejsou označeny jako globální.

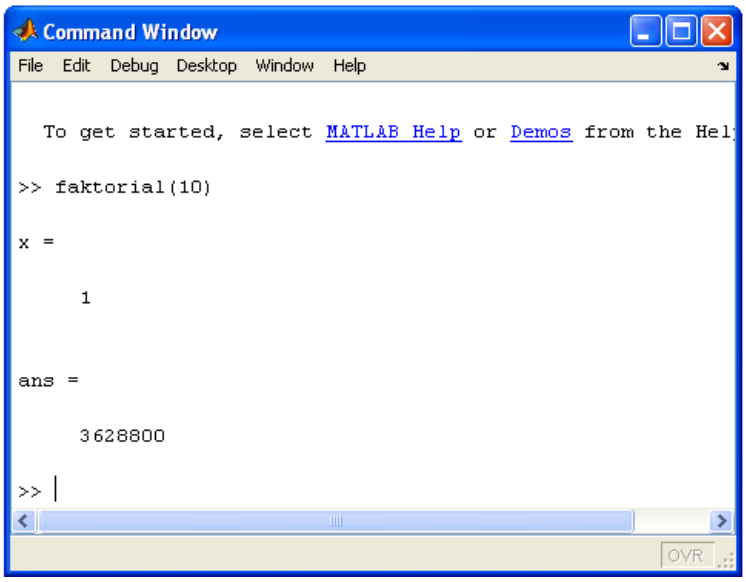


Obr. 19: Zdrojový text funkce faktorial.m.

Příkaz pro vytvoření funkce či skriptu z příkazového okna MATLABu je stejný. Opět používáme příkaz edit a následně potvrdíme klávesou enter. Vyvolá se vestavěný editor MATLABu.

Hlavička zůstává stejná jako u skriptů. Hlavní rozdíl je v použití klíčového slova *function*. Toto slovo se vždy musí uvést ve vestavěném editoru a je označeno modrou barvou. Poté následuje jméno výstupní funkce, v našem případě to je x. Za ni pak název funkce (faktoriál) a v závorce za názvem funkce následuje jen jedna proměnná *n*, označující vstupní parametr. Pří více vstupních nebo výstupních proměnných je oddělujeme čárkou. Název proměnné si volíme sami, avšak nesmíme zapomenout na uložení zdrojového kódu, nejlépe pod stejným názvem jako je název funkce ve zdrojovém textu.

V kulatých závorkách jsou uvedeny vstupní parametry, v našem případě jen jeden a to *n*. Tato funkce umožňuje výpočet faktoriálu libovolného čísla (obr. 20).



Obr. 20: Spuštění a odezva funkce faktoriál.m.

Všeobecně rozlišujeme různé druhy funkcí [4]:

- Funkce s jedním vstupním parametrem,
- Volání funkce uvnitř skriptu,
- Volání funkce uvnitř jiné funkce,
- Funkce bez vstupního parametru,
- Funkce volající sama sebe,
- Funkce se dvěma či více vstupními parametry,
- Funkce s proměnným počtem vstupních parametrů,
- Funkce s jedním výstupním parametrem,
- Funkce se dvěma či více vstupními parametry,
- Funkce s proměnným počtem výstupních parametrů,
- Funkce se vstupními a výstupními parametry.

3. GUI – Graphical User Interface

GUI je grafické uživatelské rozhraní sestavené z grafických komponentů, jako jsou tlačítka, textová pole apod. Uživatel si může vytvářet sám grafické objekty přímo v editoru zdrojových textů, a to podle jeho potřeb. Hlavním prostředkem pro vytváření grafických rozhraní aplikací vytvořených v MATLABu je nástroj zvaný GUIDE (Graphical User Interface Development Environment). GUIDE je přehledný nástroj, který výrazně usnadní práci při vytváření grafického rozhraní. Základem práce je umísťování tlačítek a vytváření vazeb mezi nimi.

U správně navrženého GUI by mělo být každému uživateli jasné, jak GUI pracuje a jak fungují jeho jednotlivé části. Velkou výhodou u GUI je, že umožňuje uživateli obsluhovat aplikaci bez znalosti příkazů potřebných při práci s příkazovým řádkem v MATLABu. Této výhody lze využít jak pro praktická měření, pro simulaci, tak i pro výuku, kdy studenti se mohou seznámit detailně s probíranou látkou. V průběhu běhu aplikace můžeme měnit hodnoty různých parametrů algoritmu a sledovat tak projevy těchto změn na výstupu (resp. výstupech) algoritmu [5].

3.1. Princip tvorby aplikací

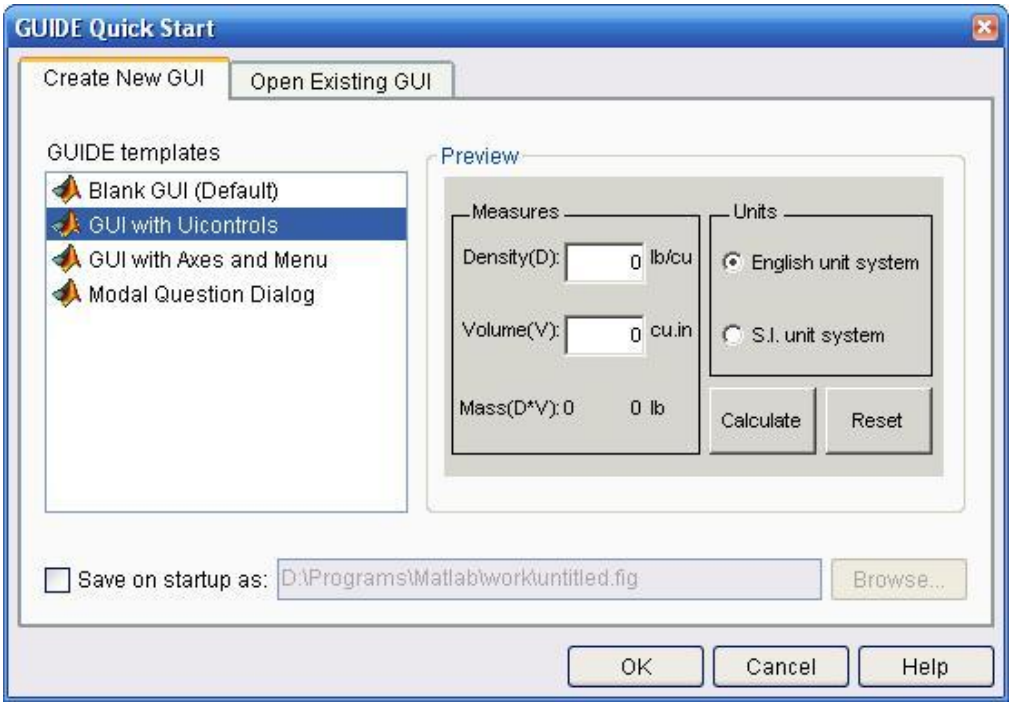
Než začneme vyvíjet aplikaci, je nutné si stanovit pravidla designu takového programu, abychom se vyhnuli vytvoření nepřehledné aplikace. Z tohoto důvodu jsou určena všeobecná základní pravidla designu aplikaci, která platí i v ostatních objektově orientovaných programovacích jazycích:

- **JEDNODUCHOST –** Návrh programu by měl být jednoduchý, měli bychom se vyhnout nepřehledné změti tlačítek. Dalo by se říci, že „platí méně je více“. Pohyb v aplikaci by měl být rychlý. Kliknutí na tlačítko či přepínač je vždy snadné a rychlé.
- **PROVÁZANOST** – Musíme počítat s tím, že program bude používat i méně znalý člověk. Musíme tedy program ošetřit tak, aby v průběhu programu nedošlo k situaci, že se uživatel ocitne ve slepé uličce, odkud nebude mít možnost návratu.
- **KOMPLEXNOST –** Dalo by se říci, že musíme ošetřit veškeré možné eventuality a možnosti.

Při tvorbě aplikace je důležité neuchýlit se k přílišné odbornosti (např. ve smyslu nastavení vstupních parametrů), aby se uživatel dokázal orientovat v hlavní funkci programu, tzn. udržet tak spojitost celého projektu. Aplikace by měla být tvořena ve všeobecně známém konceptu, design a prezentace funkcí, tak jak je známe v reálném životě.

3.2. GUIDE

Aplikace GUI se dají psát dvěma způsoby. První, těžší způsob, je takový, že celá aplikace je vytvořena jen za použití m-file editoru. Tento způsob je bez větších znalostí nepoužitelný. Druhým, pro nás lepší způsob, je využití m-file editoru v kombinace s GUIDE (obr. 21).



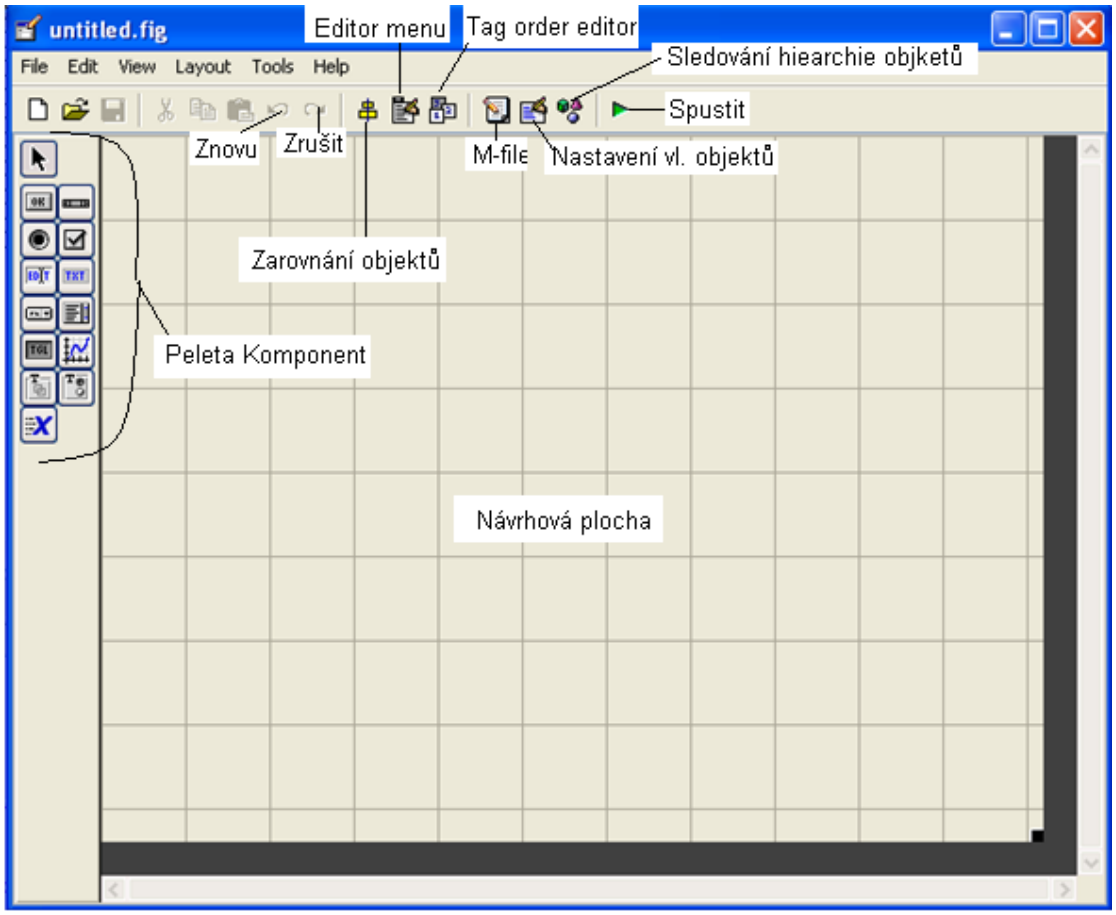
Obr. 21: Průvodce GUIdem

GUIDE je vývojové prostředí pro GUI programy. Pomocí něho můžeme vytvářet, nebo upravovat uživatelské rozhraní a to prostřednictvím push butonů, list boxů, radio butonů, Cheb boxů a dalších. Návrhové nástroje GUIDE se skládají z:

- **Layout Editor** - přidávání a uspořádání objektu v aplikaci
- **Aligment tool** - nástroj používaný pro konečné doladění pozice jednotlivých komponent, nebo pro zarovnání
- **Property inspektor** - nastavuje a dohlíží na vlastnosti objektu
- **Object browser** - sleduje seznam ukazatelů objektu v aktuální sekci MATLABu
- **Menu editor** - editor pro tvorbu menu
- **Tag order editor** - změní pořadí, ve kterém jsou prvky vybrané tabulátorem.

3.2.1. Layout editor

Lay editor nebo jinak též návrhový editor, je ovládací panel pro editor GUIDE. Pro spuštění návrhového editoru, použijeme příkaz guide, z příkazové řádky MATLABu nebo z menu File -> New -> GUI. Zobrazí se nám dialogové okno jednoduchého průvodce z možnosti nabídky otevření již existujícího projektu nebo možnost výběru jednoho ze čtyř základních konceptů pracovní plochy od prázdné po dialogová okna a grafické výstupy. Z každé této volby se dostaneme do Layout editoru. Návrhový editor umožňuje snadno a rychle vytvořit GUI přetažením součástí jako jsou tlačítka, vysouvací nabídky a jiná. Návrhový editor (obr. 22) je tedy výchozím prvkem pro návrh aplikace, z jeho prostředí snadno a expandujeme do dalších editorů.



Obr. 22: Návrhový editor

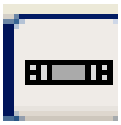
Pokud spustíme nebo uložíme GUI, automaticky se vytvoří dva soubory se stejným jménem ale lišící se příponou.

- **FIG- file –** nachází se v něm binární obsah GUI aplikace (kompletní popis grafické části GUI a jeho součástí)

- **M- file** – zde se ukládají procedury a funkce, které lze pak zpustit pomocí Figure Activator. Když poprvé spustíme, nebo uložíme GUI z návrhového editoru, GUIDE vygeneruje GUI m-file s prázdnými útržky kódu pro každou callback funkci. Ty musíme naprogramovat v m-file Editoru sami.

3.2.2. Popis prvků (paleta komponent)

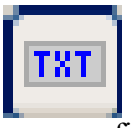
- Tento prvek nám slouží jako spouštěcí tlačítko. Které ve většině případů používáme k zahájení nějakého výpočtu, v našem případě k zahájení vykreslení grafu.



- Slider, jak už jeho název napovídá, slouží k posouvání v grafickém rozhraní do strany. Použití např. v případě, kdy navrhujeme více věcí, než by se nám vešlo na obrazovku.

- Další z prvků se nazývá Radio Button. Tento prvek slouží v případě, kdy máme na výběr z více možností. Např. si můžeme vybrat, jestli chceme mít výsledek na jedno desetinné místo, nebo na více podle možnosti výběru.

- Check box, jinak zaškrtávací políčko, má obdobný význam jako Radio Button. Dá se použít například tam, kde si z možností výběru pozaškrtáváme určité podmínky pro námi dělanou věc.



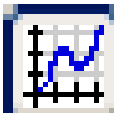
- Tento prvek nám slouží k vepsání libovolného textu. Ať už jako názvu grafu, nebo popisek hodnot, kterou máme zadávat. Pomáhá nám v orientaci v grafickém rozhraní a taky v značení co kde máme. Tomuto prvku říkáme Static text.

- Prvek jménem Edit Text slouží k zadávání hodnot. V tomto políčku si můžeme přednastavit hodnotu, kterou v průběhu běhu programu můžeme změnit. Jinými slovy pomocí tohoto prvku zadáváme vstupní hodnoty.

 - Pop-up Menu. Menu, které po rozbalení nám nabídne možnosti, které jsme si nastavili. Například si přepnou z kreslení cosinu na kreslení hyperboly a jiné námi předem udělané možnosti.

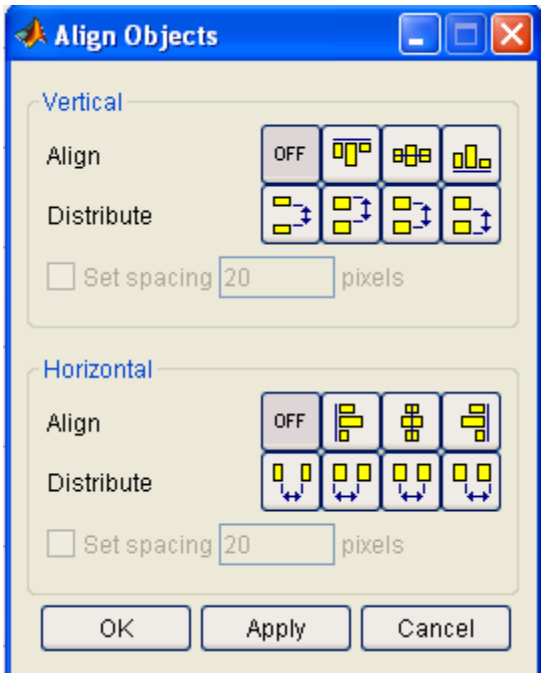
 - ListBox. Je to obdobný případ jako v Pop-up menu. Rozdíl je v tom, jak už název povídá, že v možnostech výběru jakoby listujeme. Je to jiná forma grafického znázornění než Pop-up menu.

- Togle button. Tlačítko sloužící k přepínání mezi například radiány a stupni. Opět záleží, k jaké práci si ho zvolíme, a upravíme.

 - Tato komponenta se nazývá axces. Pomocí této komponenty si do našeho návrhu naneseme místo, ve kterém se nám bude zobrazovat graf. Tento graf se bude vytvářet například pomocí funkce *plot*.

3.2.3. Alignment tool

Tento nástroj je určen pro úpravu vzhledu našich aplikací. V návrhovém jednoduše přetáhneme komponenty na plochu a za pomoci tohoto nástroje dokončíme jejich přesné rozmístění.



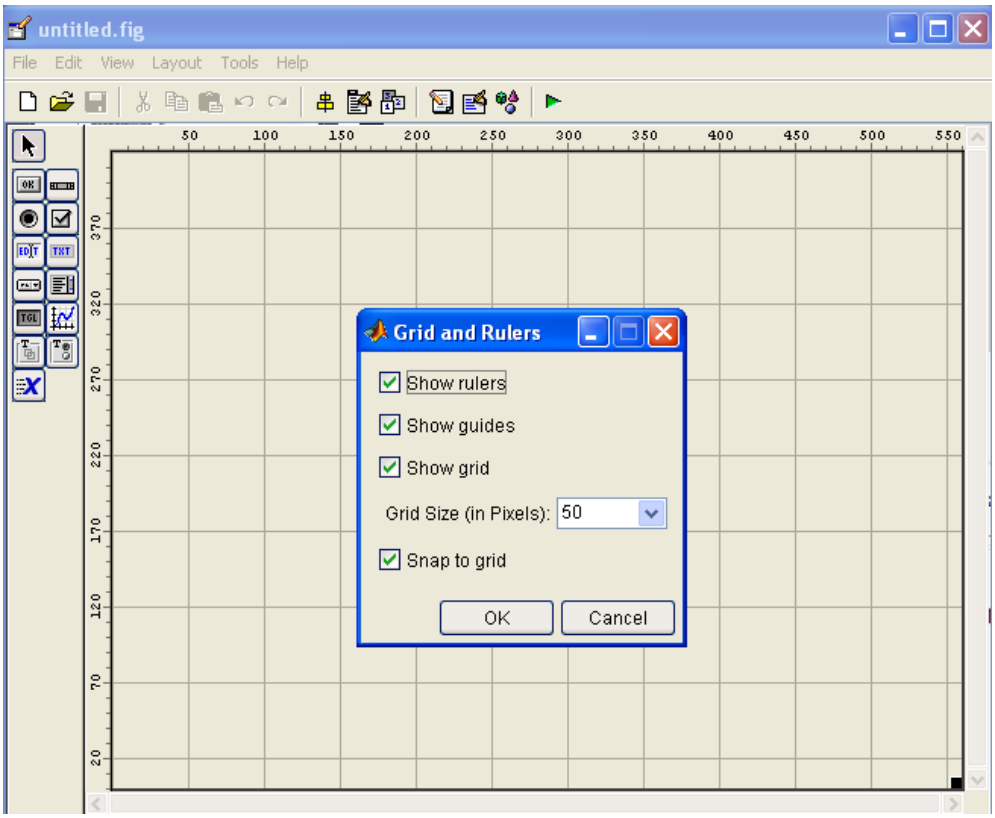
Obr. 23: Zarovnávací nástroj

Zarovnávací nástroj dává možnost pravidelného rozdělení komponent ve vertikální a horizontální poloze. Dále na zarovnání grafických objektů na části jako jsou horní nebo dolní okraj, na středy, nebo na spodní nebo pravý okraj. Dále můžeme mít rozložení, jako jsou pevné vzdálenosti mezi objekty, pevné vzdáleností mezi středy objektů atd.

K tomuto zarovnání nám pomáhají:

- **Vodící linky** (obr. 24) - umožňují přichycení komponenty k vodícím linkám v jakémkoliv místě. Získáme je vytažením z měřítka (Ruler). Najedeme-li myší na levý okraj Layout Area, změní se nám kurzor na opačnou dvojšipku. Podržíme-li nyní levé tlačítko a táhneme vpravo, vytáhneme si vodící linku. V místě, kde tlačítko pustíme, zůstane linka zafixována. Tento postup lze opakovat několikrát a vytahat si libovolné množství linek.

- **Mřížka a pravítko** (obr. 24) - přichycení komponenty k mřížce dle zvolených kroků. Mřížka a pravítko při designu aplikace velice usnadňují zarovnání. Čáry mřížky jsou standardně zobrazeny a rozmístěny v 50ti pixelových intervalech, rozpětí intervalu lze měnit od 10 – 200 pixelů. Výhodou je, že tato funkce funguje i bez zobrazené mřížky, kdyby se čáry mohly stát rušivým elementem při návrhu. Stejně tak můžeme nastavit viditelnost pravítka, mřížky a vodících linek.



Obr. 24: Mřížka a pravítko

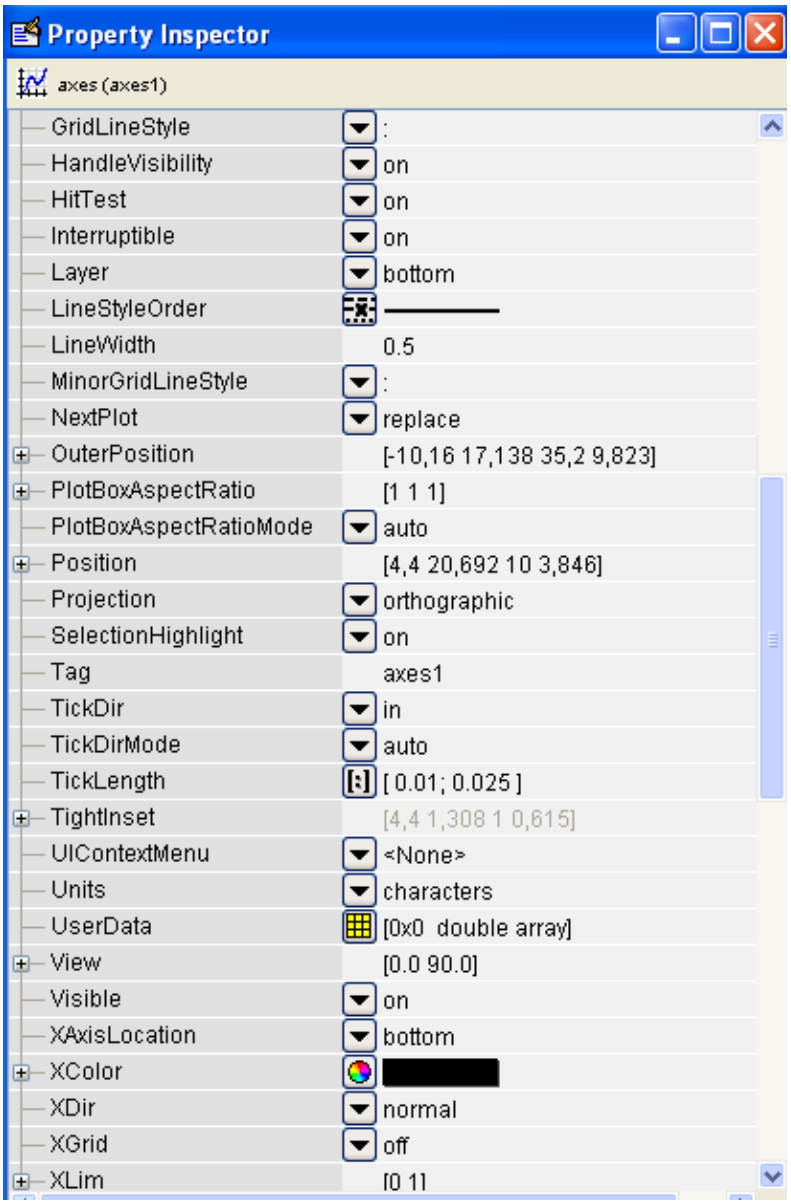
- **Přenést před/za na popředí/pozadí** – uspořádání komponent zepředu dozadu. Režim tohoto návrhu nabízí celkem 4 akce, které řídí jednotlivá přenášení aktivních objekt oproti neaktivním:

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

3.2.4. Property inspektor a Object browser

Property inspektor (obr. 25)

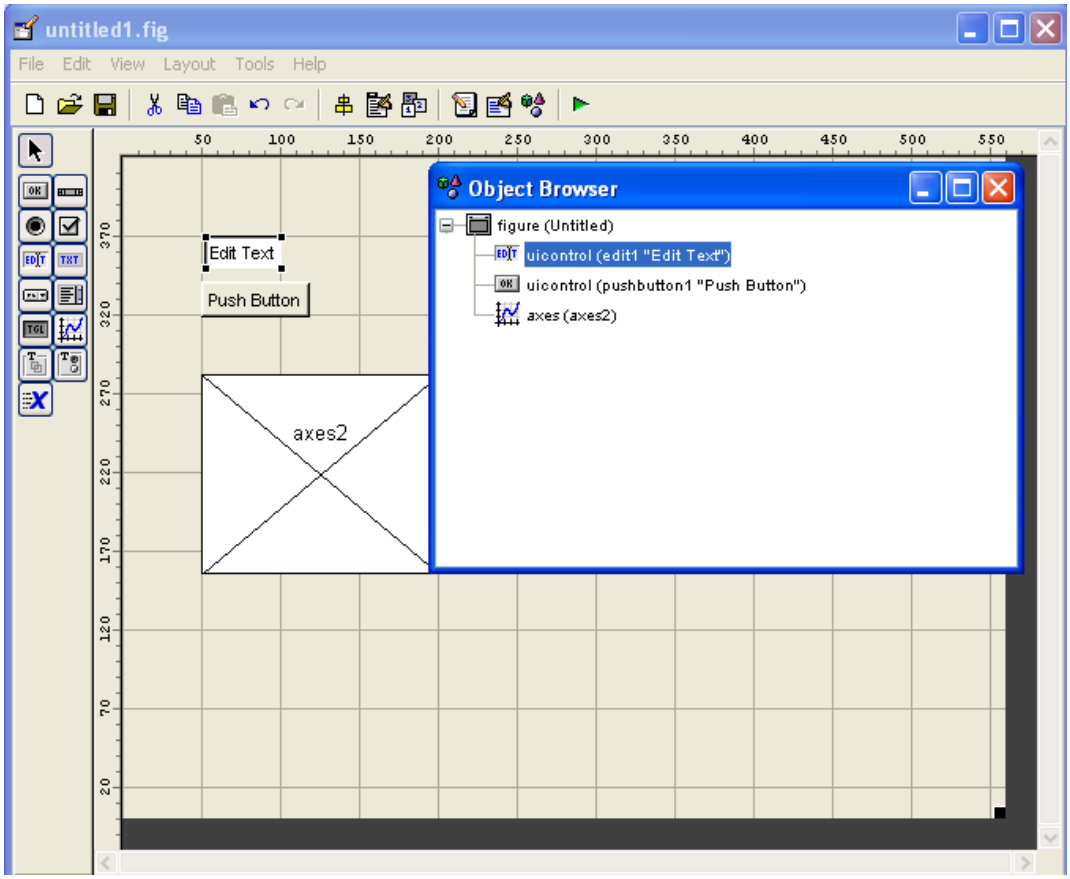
Slouží nám k zobrazení a editaci jednotlivých vlastností komponent vytvořených na návrhu. Každý objekt se liší počtem položek v Property inspektoru, avšak určitý počet položek je shodný pro všechny typy objektů (např. pásma, barva pozadí, identifikátor, viditelnost…). Další vlastnosti jsou určeny samotnou komponentou a tím na co se používá.



Obr. 25: Property inspektor.

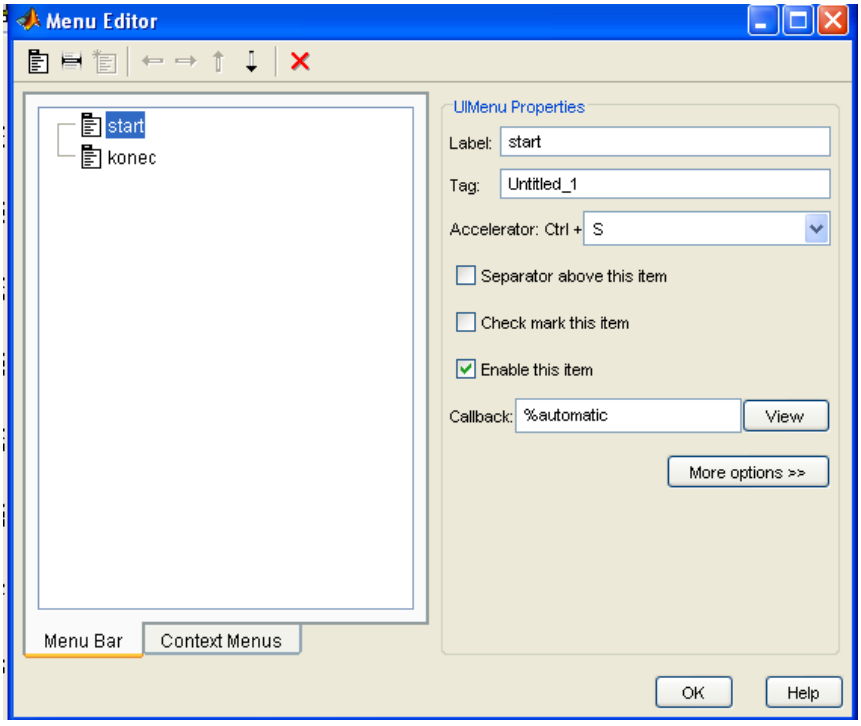
Object browser (obr. 26)

Object browser zobrazuje celý seznam objektů v pořadí, jak byly vytvářeny, jejich dědičnost atd. Nástroj může také posloužit k jednoduché orientaci mezi objekty. Na obrázku je vidět objekt figure a jeho potomky (Edit text, Push button, Axes) jak byly postupně vytvářeny.



Obr. 26: Object browser

3.2.5. Menu editor



Obr. 27: Menu Editor

Pro přehledný a jednoduchý přístup k funkcím programu se ve většině aplikací neobejdeme bez menu. V GUIDE už existuje menu editor, který byl již navržen pro návrh menu. Můžeme si vybrat ze dvou možných nabídek navrhování (obr. 27):

- **Menubar objekty**

Toto je menu zobrazované v horní liště okna, v němž můžeme tvořit libovolně větvené další menu.

- **Kontextové menu**

Menu, které se nám zobrazí po kliknutí tlačítka myši na objekt. Kontextové menu musí být svázáno s objektem a to za pomocí parametru v Property inspektoru (IContextMenu).

Takto udělaná menu jsou připravená na plnění své funkce, avšak samotné položky zatím nefungují. Jejich funkce a procedury si musíme vytvořit v m-file editor a tím je jakoby aktivovat.

3.3. Vytvoření grafické aplikace

Po osvětlení pracovního prostředí GUIDE s jeho komponenty a samotným designem aplikací, můžeme přistoupit k vlastnímu postupu při tvorbě GUI aplikace [4].

Při tvorbě aplikace je vhodné dodržet následující postup:

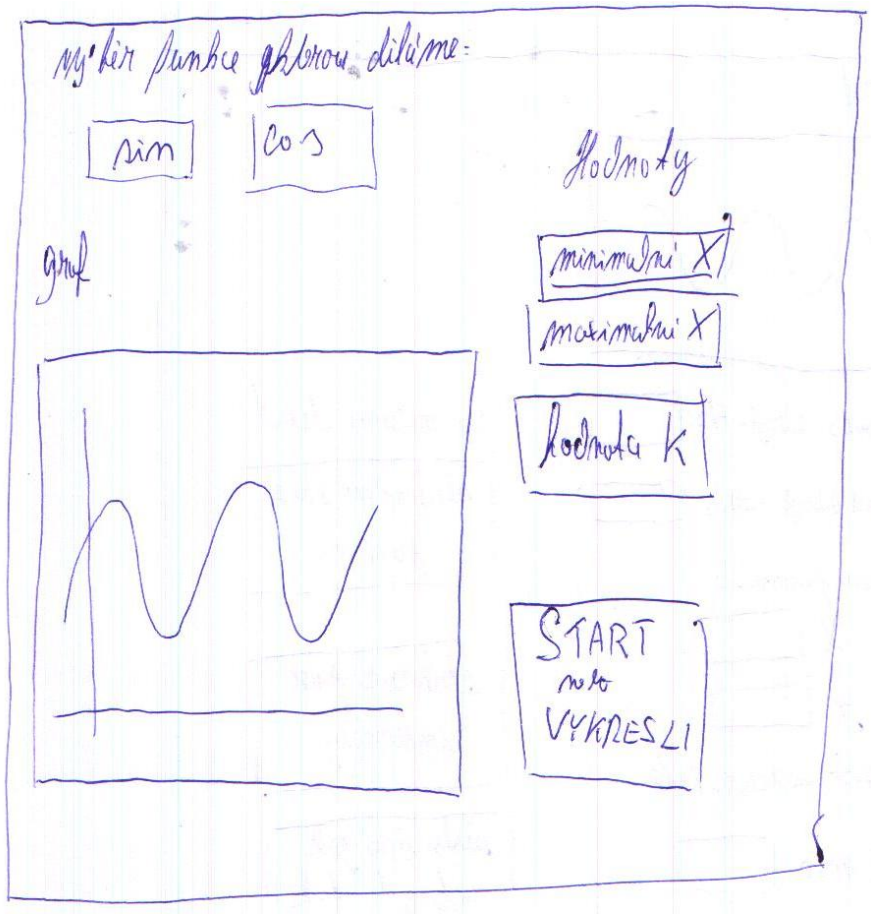
- Důkladné nastudování problematiky, udělat si všechna možná řešení (cíl porozumění problému).
- Pokusit se nakreslit na papír naši představu, jak by GUI mělo vypadat. Rozmístění prvků na návrhovou plochu, počet ovládacích prvků atp.
- Pokračujeme umísťování prvků na návrhovou plochu, jejich zarovnáváním atp.
- Samotné spuštění GUI, zkontrolování vzhledu, funkce
- Poslední věc je do vygenerovaného m-file (vygeneroval se o bod výše) musíme doplnit zpětné vazby tzv. callbacky, abychom dané objekty zprovoznili.

4. Postup při tvorbě GUI

V této části si ukážeme návrh GUI, na jednom jednoduchém a jednom složitějším příkladu. Budeme postupovat krok za krokem a podrobně vše vysvětlíme.

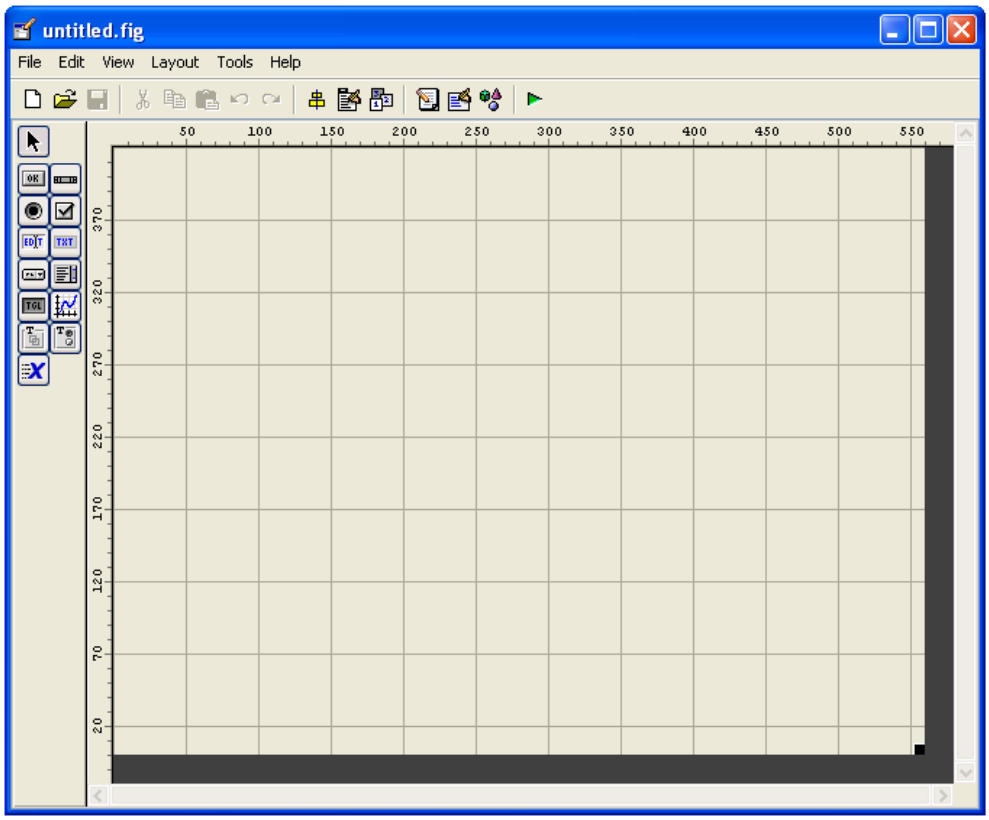
4.1. GUI pro vykreslení sinu a cosinu

Při vyváření programu musíme brát zřetel na to, aby byl přehledný a jednoduchý. Nejdříve je vhodné si vzhled načrtnout na papír a ujasnit si základní rozložení komponent (obr. 28).



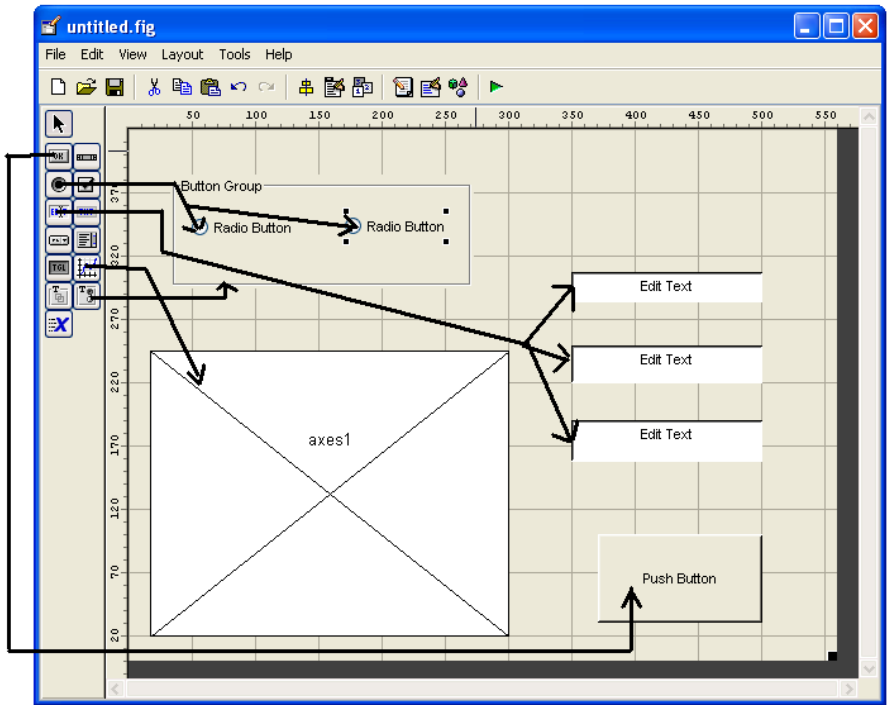
Obr. 28: Náčrt vzhledu

Ještě před tím, než vytvoříme vlastní aplikaci, je vhodné připravit si předem hlavní bloky strojového kódu, v našem případě se jedná např. o příkazy vykreslující grafy funkcí sinus a cosinus. Poté pokračujeme vytvořením vlastní aplikace. V MATLABu spustíme GUI rozhraní a to tak, že postupujeme v Menu přes FILE-NEW- GUI. Následně se nám ukáže obrazovka, kde si zvolíme vytvoření nového GUI a zobrazí se nám obrazovka pro tvorbu GUI (obr. 29).



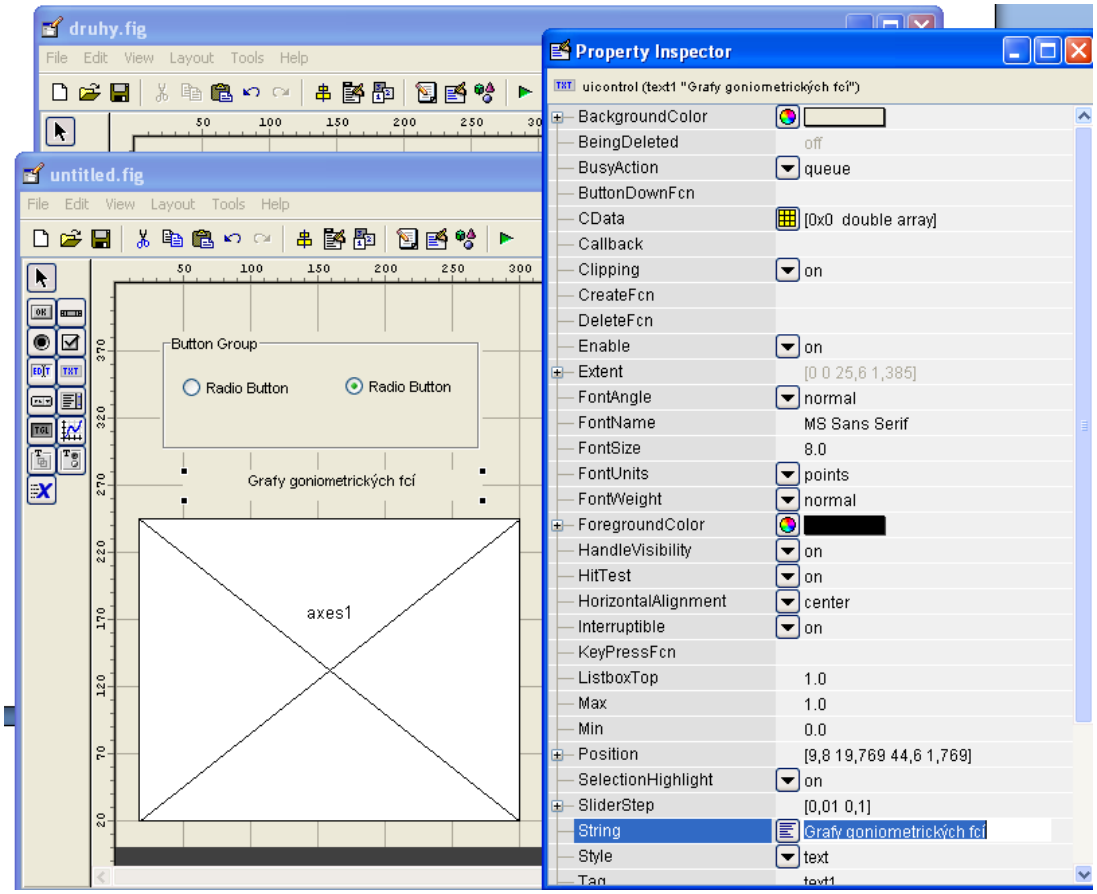
Obr. 29: Okno pro návrh GUI.

Pokračujeme přetahování zvolených komponent na pracovní plochu. Na obrázku 30 je pomocí šipek znázorněno vytvoření jednotlivých komponent.



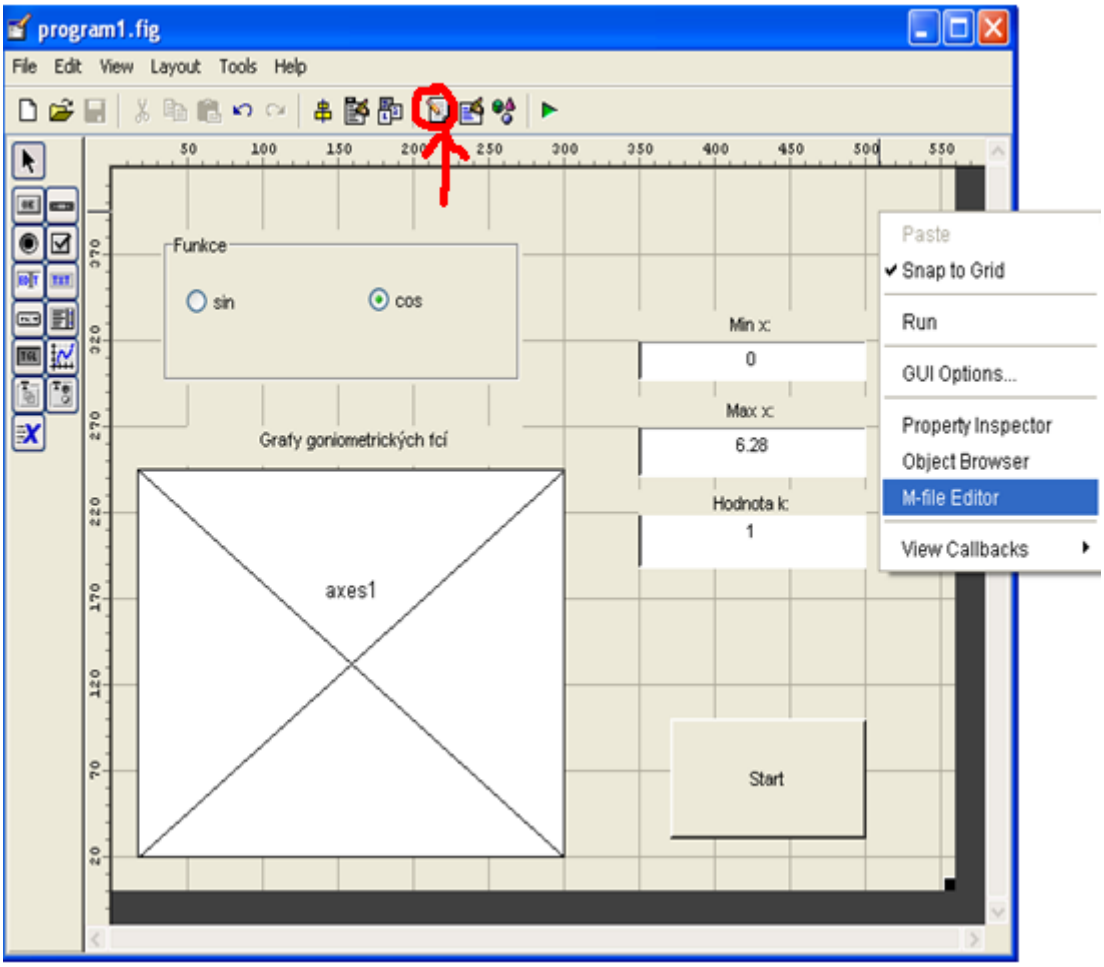
Obr. 30: Rozmístění komponent GUI

V dalším kroku si přejmenujeme jednotlivé komponenty, což nám umožní lepší orientaci v pracovním prostředí. Pomocí property inspektoru, který se nám zobrazí po dvojitém klepnutí na komponentu, můžeme nastavovat i další parametry. Jedná se například o minimální zobrazovanou hodnotu, o maximální zobrazovanou hodnotu, popisek tag, barvy, velikost písma atd.



Obr. 31: Popisky

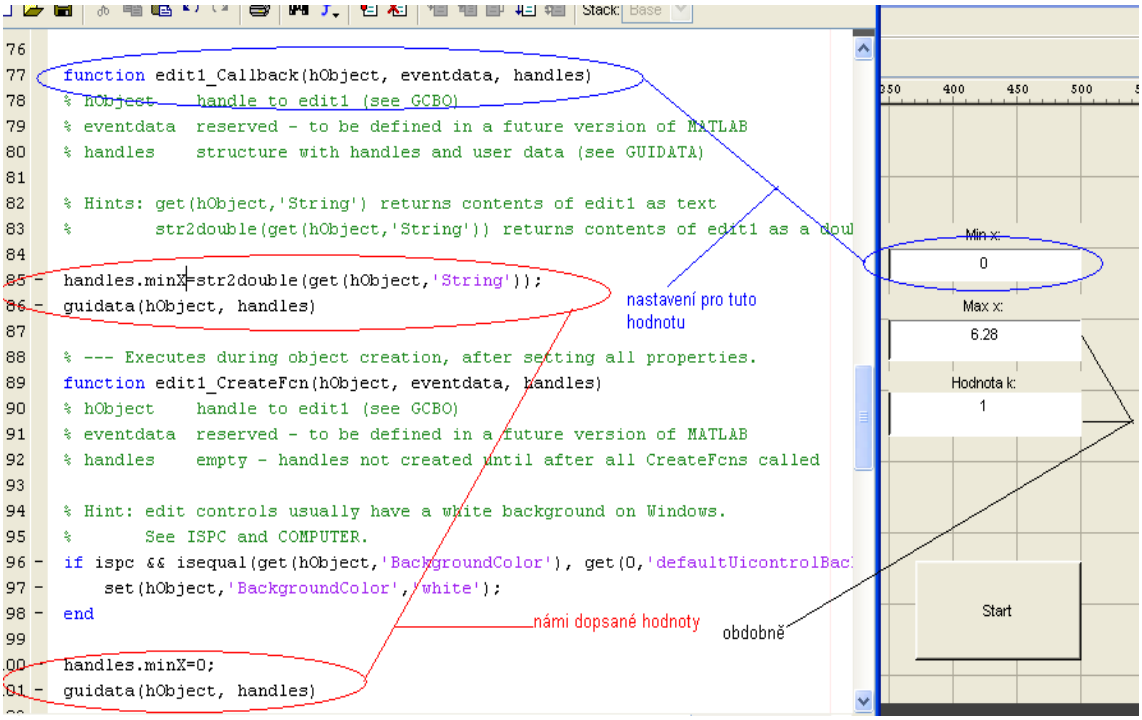
V našem případě jsme přejmenovali: Push button, neboli tlačítko pro povel vykreslit, jsme si označili nápisem Start, část Button Group jsme si přejmenovali na název Funkce (tomuto „rámečku“ se budeme věnovat až na závěr vytváření GUI aplikace). Všechny tyto úpravy vidíme na obr. 32. Poté jsme si naší prácí uložit pod námi vymyšleným jménem, v našem případě program1. Pracovní okno po přejmenování je znázorněno na obrázku 32.



Obr. 32: GUI s popiskami

V okamžiku, když jsme s rozložením grafických prvků spokojení, přistoupíme k vytváření zdrojového kódu aplikací. Uděláme to tak, že klikneme na tlačítko, které máme označeno na obrázku 32. Poté se nám zobrazí známý m-file editor s nagenerovaným zdrojovým kódem (obr. 33a).

Nagenerovaným zdrojový kód se skládá z jednotlivých funkcí, přičemž tyto funkce můžeme v zásadě rozdělit do dvojího druhu. Jednak na funkce typu *Callback*, což jsou funkce, které mají obsluhovat akci jednotlivých objektů a dále jsou to funkce typu *CreateFcn*, což jsou funkce, které daný objekt vytváří.



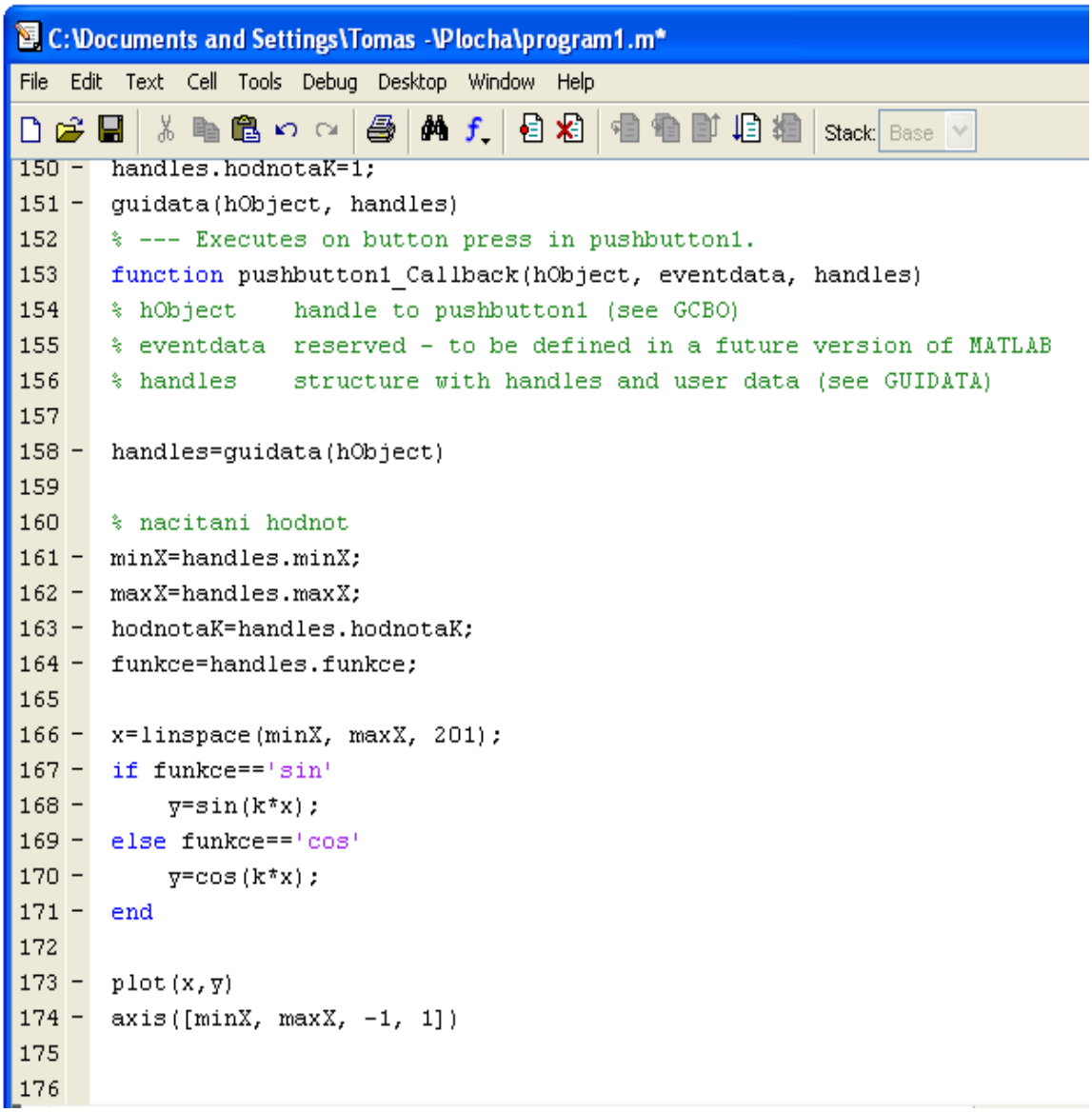
Obr. 33: Nagenerovaný zdrojový kód.

Do nagenerovaného zdrojového kódu přidáme jednotlivé příkazy, které nám zajistí předávání dat mezi jednotlivými funkcemi. Při vytváření GUI se všechna data shromažďují v proměnné *handles* a nahrávání do této proměnné se děje pomocí funkce *guidata*. Tato funkce také umožňuje uložená data načíst do jiné funkce (obr 33b).

```
153   function pushbutton1_Callback(hObject, eventdata, handles)
154   % hObject    handle to pushbutton1 (see GCBO)
155   % eventdata  reserved - to be defined in a future version of MATLAB
156   % handles    structure with handles and user data (see GUIDATA)
157
158 -  handles=guidata(hObject)
159
```

Obr. 33b: Nagenerovaný zdrojový kódu.

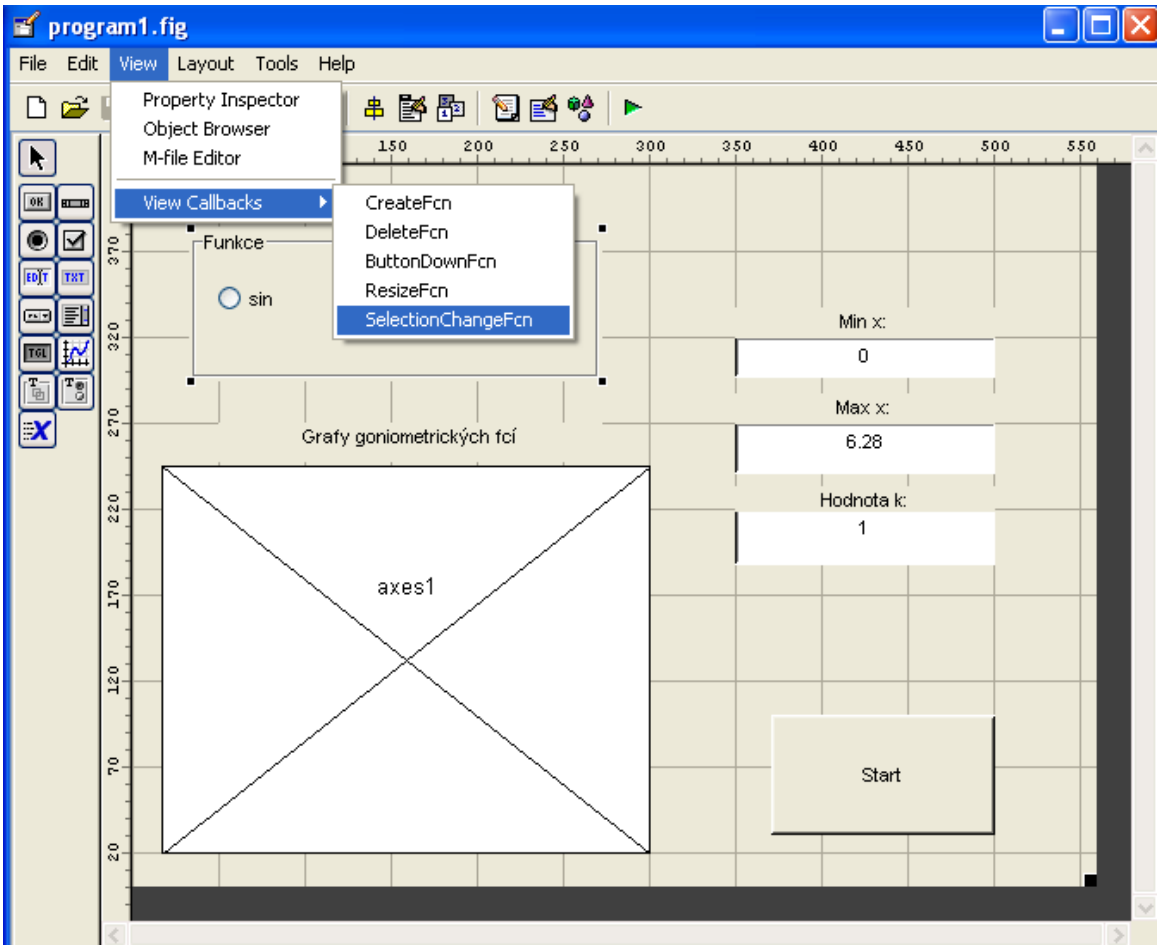
Poté, co jsme ošetřili předávání hodnot jednotlivým proměnným mezi funkcemi a nastavili chování programu, přistoupíme k ošetření jednotlivých událostí při spuštění programu. Například na obr. 34 je zobrazeno ošetření stisku tlačítka *pusbutton1*, které je spojené s vykreslením funkce sinus nebo cosinus, podle toho která funkce je zvolena.



Obr. 34: Nastaveni funkce pro spuštění.

V uvedeném příkladu (obr. 34) jsou na řádku 161 -164 načteny proměnné z proměnné *handles*, které jsou následně využity k vykreslení funkce sinus a cosinus, viz řádky 166 – 174.

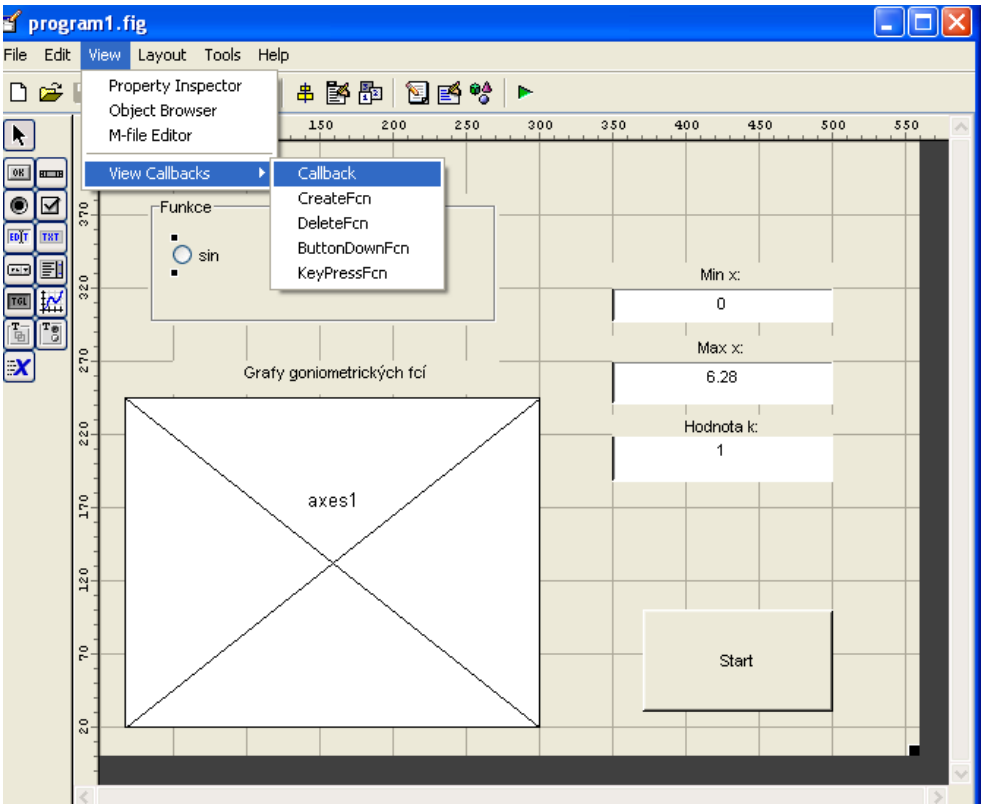
Dále je potřeba ošetřit chování bloku *radiobutton*, které nám zajišťuje přepínání mezi sinem a cosinem. Postupujeme tak, že vytvoříme funkci, která nám přepínání zajistí. Nejprve výběrem položky *View* v menu, přes *view callbacks* zvolíme položku *SelectionChangeFcn*, viz obr. 35.



Obr. 35: Vytvoření přepínání.

Stejným způsobem si uděláme funkci pro vytvoření funkce, ve které si poté přednastavíme proměnou funkce na sin. Oproti předchozímu případu se tento postup liší pouze v tom, že místo SelectionChangeFcn si vybereme CreatFcn, tuto část uděláme úplně nakonec. Tímto se nám vytvořily v M-file editoru funkce, jejichž chování musíme teprve naprogramovat.

Při vytváření funkce *Callback* pro vykreslení sinu, nejprve vybereme v pracovním okně položku pro sinus a následně přes *View- View Callbacks a Callback* (obr. 36) spustíme funkci, kterou doprogramujeme. Obdobně pokračujeme i u druhého radio buttonu, který máme popsaný jako cos.



Obr. 36: Nastavení radio buttonu.

Po stisku tohoto tlačítka (obr 35- selectChangeFcn), se nám do zdrojového kódu programu přidá nová funkce *uipanel1_SelectionChangeFcn*. Následně vytvoříme sérii příkazů, která zjistí výběr zvolené hodnoty, jak je zobrazeno na obr 37.

C:\Documents and Settings\Tomas -\Plocha\mat\program1.m

File Edit Text Cell Tools Debug Desktop Window Help

Stack: Base

```
179   function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
180   % hObject    handle to uipanel1 (see GCBO)
181   % eventdata  reserved - to be defined in a future version of MATLAB
182   % handles    structure with handles and user data (see GUIDATA)
183
184 - get(hObject,'Tag')
185 - switch get(hObject,'Tag')
186 -     case 'radiobutton1'
187 -         funkce='sin'    % sinus
188 -     case 'radiobutton2'
189 -         funkce='cos'    % cosinus
190 - end
191
192 - handles.funkce= funkce
193 - guidata(hObject, handles)
194
```

Obr. 37: Dopsání funkce.

Při vytváření jsme postupovali tak, že jsme dopsali řádky 184- 193 přičemž, jsme nejprve načetli hodnotu *tag* našeho objektu, a pomocí přepínače *switch* jsme rozhodli o tom, které z daných políček je zaškrtnuto (obr. 37- řádky 185-190). Pomocí příkazů na řádcích 192-193 jsme uvedené parametry uložili do proměnné *handles*, čímž může být exportována do dalších funkcí v programu.

Řádky 204 až 214 (obr. 38) jsou příkazy, které reagují na zaškrtnutí určitého radiobuttonu, a zpětně volají funkci, která rozlišuje a ukládá buď sin, nebo cos do proměnné funkce.

```
196   % --- Executes on button press in radiobutton1.
197   function radiobutton1_Callback(hObject, eventdata, handles)
198   % hObject    handle to radiobutton1 (see GCBO)
199   % eventdata  reserved - to be defined in a future version of MATLAB
200   % handles    structure with handles and user data (see GUIDATA)
201
202   % Hint: get(hObject,'Value') returns toggle state of radiobutton1
203
204 -  uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
205
206   % --- Executes on button press in radiobutton2.
207   function radiobutton2_Callback(hObject, eventdata, handles)
208   % hObject    handle to radiobutton2 (see GCBO)
209   % eventdata  reserved - to be defined in a future version of MATLAB
210   % handles    structure with handles and user data (see GUIDATA)
211
212   % Hint: get(hObject,'Value') returns toggle state of radiobutton2
213
214 -  uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
215
```

Obr. 38: Radio button.

Na závěr je potřeba přednastavit funkci, která bude brána jako implicitní. V našem případě byla nastavena funkce sinus. V programu to provedeme tak, že přidáme řádek 222 a 223 (viz obr. 39). Tímto dosáhneme toho, že když stiskneme tlačítko START, bez změny jakéhokoliv parametru (jako je tomu například při spuštění programu), vykreslí se nám sinus.

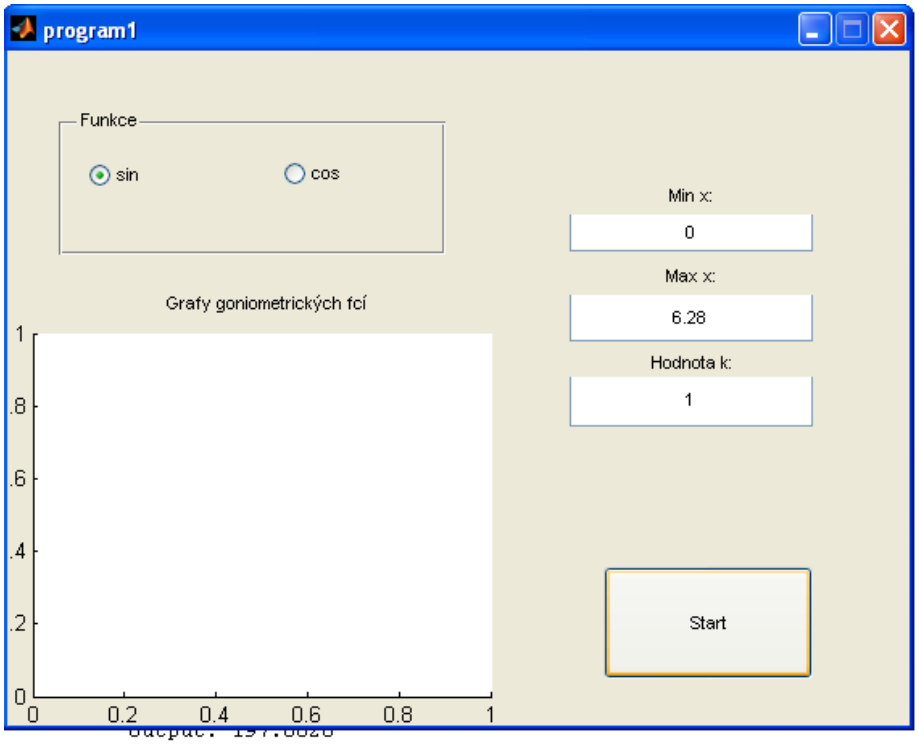
```
216   % --- Executes during object creation, after setting all properties.
217   function uipanel1_CreateFcn(hObject, eventdata, handles)
218   % hObject    handle to uipanel1 (see GCBO)
219   % eventdata  reserved - to be defined in a future version of MATLAB
220   % handles    empty - handles not created until after all CreateFcns called
221
222 -  handles.funkce='sin'
223 -  guidata(hObject, handles)
```

Obr. 39: Přednastavení sin.

Nyní je program vytvořen, a my ho můžeme spustit pomocí tlačítka „zelená šipka“ (obr. 40).

Obr. 40:Spuštění.

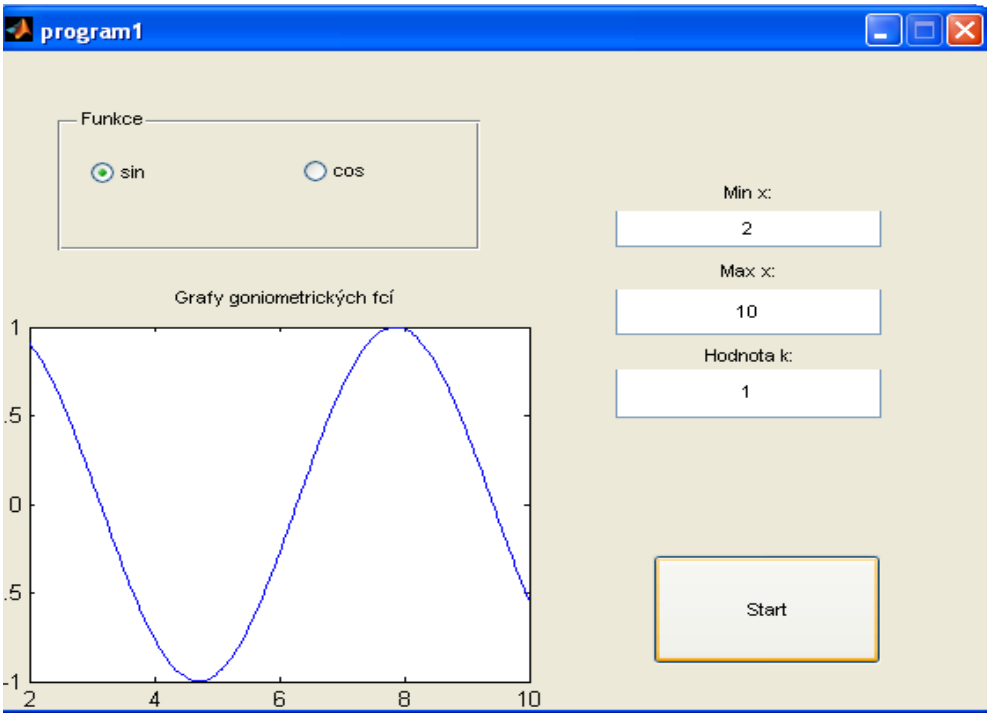
Pokud by v programu byla nějaká chyba, zobrazí se nám v pracovním okně MATLABu chybové hlášení. V případě naší ukázky je chyba znázorněna na obr. 41.



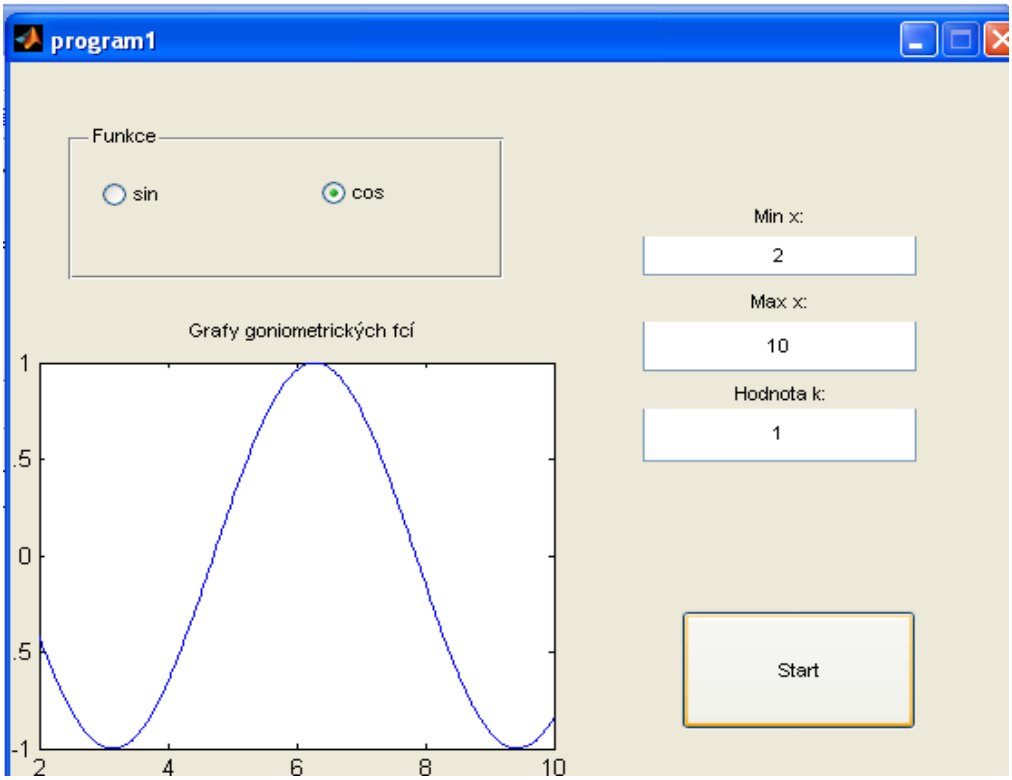

```
??? Undefined function or variable 'K'.

Error in ==> program1>pushbutton1_Callback at 168
    y=sin(K*x);

Error in ==> gui_mainfcn at 75
        feval(varargin{:});

Error in ==> program1 at 42
    gui_mainfcn(gui_State, varargin{:});

??? Error while evaluating uicontrol Callback.
```

Obr. 41:Chybové hlášení.

Ze zápisu je patrné, že chyba se nachází na řádku 168. Proto v m-file editor opravíme zadaný příkaz a program znovu spustíme. Tak postupujeme tak dlouho, dokud je program bez chyby, a vykresluje, jak sin (obr. 42), tak cos (obr. 43).



Obr. 42: Sinus.



Obr. 43:Cosinus.

4.2. Kód pro sinus a cosinus

```
function varargout = program1(varargin)
% PROGRAM1 M-file for program1.fig
%      PROGRAM1, by itself, creates a new PROGRAM1 or raises the
existing
%      singleton*.
%
%      H = PROGRAM1 returns the handle to a new PROGRAM1 or the handle
to
%      the existing singleton*.
%
%      PROGRAM1('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in PROGRAM1.M with the given input
arguments.
%
%      PROGRAM1('Property','Value',...) creates a new PROGRAM1 or
raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before program1_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to program1_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help program1

% Last Modified by GUIDE v2.5 30-Oct-2009 14:39:41

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @program1_OpeningFcn, ...
                   'gui_OutputFcn',  @program1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before program1 is made visible.
```

```
function program1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to program1 (see VARARGIN)

% Choose default command line output for program1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes program1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = program1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1
as a double

handles.minX=str2double(get(hObject,'String'));
guidata(hObject, handles)

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

handles.minX=0;
guidata(hObject, handles)
```

```
function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2
as a double
handles.maxX=str2double(get(hObject,'String'));
guidata(hObject, handles)

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

handles.maxX=6.28;
guidata(hObject, handles)

function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3
as a double
handles.hodnotaK=str2double(get(hObject,'String'));
guidata(hObject, handles)

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

handles.hodnotaK=1;
guidata(hObject, handles)
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
handles=guidata(hObject)

% nacitani hodnot
minX=handles.minX;
maxX=handles.maxX;
hodnotaK=handles.hodnotaK;
funkce=handles.funkce;

x=linspace(minX, maxX, 201);
if funkce=='sin'
    y=sin(hodnotaK*x);
else funkce=='cos'
    y=cos(hodnotaK*x);
end

plot(x,y)
axis([minX, maxX, -1, 1])



% --------------------------------------------------------------------
function uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to uipanel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

get(hObject,'Tag')
switch get(hObject,'Tag')
    case 'radiobutton1'
        funkce='sin'    % sinus
    case 'radiobutton2'
        funkce='cos'    % cosinus
end

handles.funkce= funkce
guidata(hObject, handles)


% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1

uipanel1_SelectionChangeFcn(hObject, eventdata, handles)

% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton2

uipanel1_SelectionChangeFcn(hObject, eventdata, handles)
```

```
% --- Executes during object creation, after setting all properties.
function uipanel1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to uipanel1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

handles.funkce='sin'
guidata(hObject, handles)
```

5. Dopplerův jev

Tento jev dostal název podle svého objevitele Christiana Andreasa Dopplera. Christian Andreas Doppler se narodil 29. Listopadu 1803 v Salzburgu a zemřel 17. března roku 1855 v Benátkách. Byl to význačný rakouský fyzik a matematik.

Obr. 44: Christian Andreas Doppler

Střední školu vystudoval v Linci, po jejímž absolvování v roce 1822 zahájil studium na vídeňské polytechnice. O tři roky později studoval na vídeňské univerzitě. V roce 1892 se stal asistentem matematiky u profesora Adama Burga na vídeňské technice. V roce 1835 se Doppler stal profesorem matematiky v Praze. V roce 1837 byl na technice pověřen konáním přednášek z geodézie, aby o 4 roky později byl jmenován profesorem elementární matematiky. Roku 1850 se stal ředitelem fyzikálního institutu na univerzitě ve Vídni, ale roku 1852 musel požádat o dovolenou ze zdravotních důvodů. Odjel se léčit do Itálie, kde následně zemřel.[8]

Pokud se kolem pozorovatele pohybuje zdroj zvuku (např. automobil) vnímá pozorovatel při přibližování zdroje vyšší frekvenci, než je skutečná frekvence zdroje. Poté co se zdroj a pozorovatel minou, vnímaná frekvence poklesne a je dokonce nižší, než kterou vydává zdroj. Obdobný jev lze pozorovat také v případě, kdy je zdroj v klidu

a pozorovatel se pohybuje, nebo v případě, kdy se současně pohybuje jak zdroj, tak pozorovatel. Popsaný jev nastává vždy při vzájemném pohybu zdroje zvuku a pozorovatele a je tím výraznější, čím rychleji se zdroj zvuku vzhledem k pozorovateli pohybuje. Pozorovatel pak vnímá zvuk jiné frekvence, než je frekvence kmitání zdroje. Podstatou tohoto jevu v polovině 19. století Ch. J. Doppler, po němž je jev pojmenován.

Nejdůležitější jsou následující případy Dopplerova jevu:

Zdroj zvuku Z je v klidu a přijímač zvuku (P_1, popř. P_2) se pohybuje po vzájemné spojnici konstantní rychlostí *u*, která je menší než rychlost zvuku *v*.
Zdroj Z vysílá vlnění o vlnové délce $\lambda = v/f$ (*f* je frekvence zdroje vlnění). Na obrázku 46 je znázorněn systém vlnoploch. Jestliže jsou přijímače zvukového vlnění P_1 a P_2 v klidu, dospěje k nim za jednotku času stejný počet vlnoploch a přijímače registrují zvukové vlnění stejné frekvence $f = v/\lambda$.

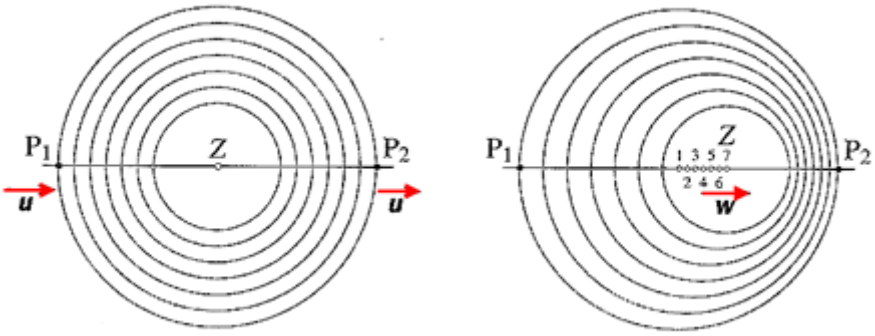
Jestliže se přijímač P_1 přibližuje rychlostí *u* ke zdroji zvuku, dospěje k němu za jednotku času větší počet vlnoploch a přijímač registruje vyšší frekvenci zvuku.

$$f_1 = \frac{v+u}{\lambda} = \frac{v+u}{v} f = \left(1 + \frac{u}{v}\right) f \qquad (1)$$

Pro názornou představu: při přibližování ke zdroji při rychlosti asi 70 km/h vzroste vnímaná frekvence zhruba o 6 %.
Přijímač P_2, který se od zdroje zvuku rychlostí *u* vzdaluje, zachytí méně zvukových vln a registruje nižší frekvenci zvuku:

$$f_2 = \frac{v-u}{\lambda} = \frac{v-u}{v} f = \left(1 - \frac{u}{v}\right) f \qquad (2)$$

Kdybychom se vzdalovali rychlostí zvuku, pak bychom žádný zvuk neslyšeli, protože z předchozího vzorce by frekvence byla nulová. Jestliže bychom se vzdalovali rychlostí větší než rychlost zvuku, pak bychom slyšeli zvyšující se frekvenci.



Obr. 46: Systém vlnoploch v okolí pohybujícího se zdroje

Druhým význačným případem je, jestliže je přijímač zvuku v klidu a zdroj zvuku se pohybuje rychlostí *w* směrem od přijímače P_1 k přijímači P_2.

Zdroj zvuku se od přijímače P_1 vzdaluje a to se projevuje jako zvětšení vlnové délky zvukového vlnění (zvětšuje se vzdálenost mezi jednotlivými vlnoplochami). Vlnová délka $\lambda_1 = (v + w)/f$ a přijímač P_1 registruje frekvenci

$$f_1 = \frac{v}{\lambda_1} = \frac{v}{v+w} f = \left(1 + \frac{w}{v}\right)^{-1} f, \qquad (3)$$

která je nižší než frekvence zdroje.

Naopak v místě přijímače P_2 je vlnová délka zvukového vlnění kratší $\lambda_2 = (v - w)/f$ a přijímač P_2 registruje frekvenci

$$f_1 = \frac{v}{\lambda_1} = \frac{v}{v-w} f = \left(1 - \frac{w}{v}\right)^{-1} f \qquad (4)$$

která je vyšší než frekvence zdroje.

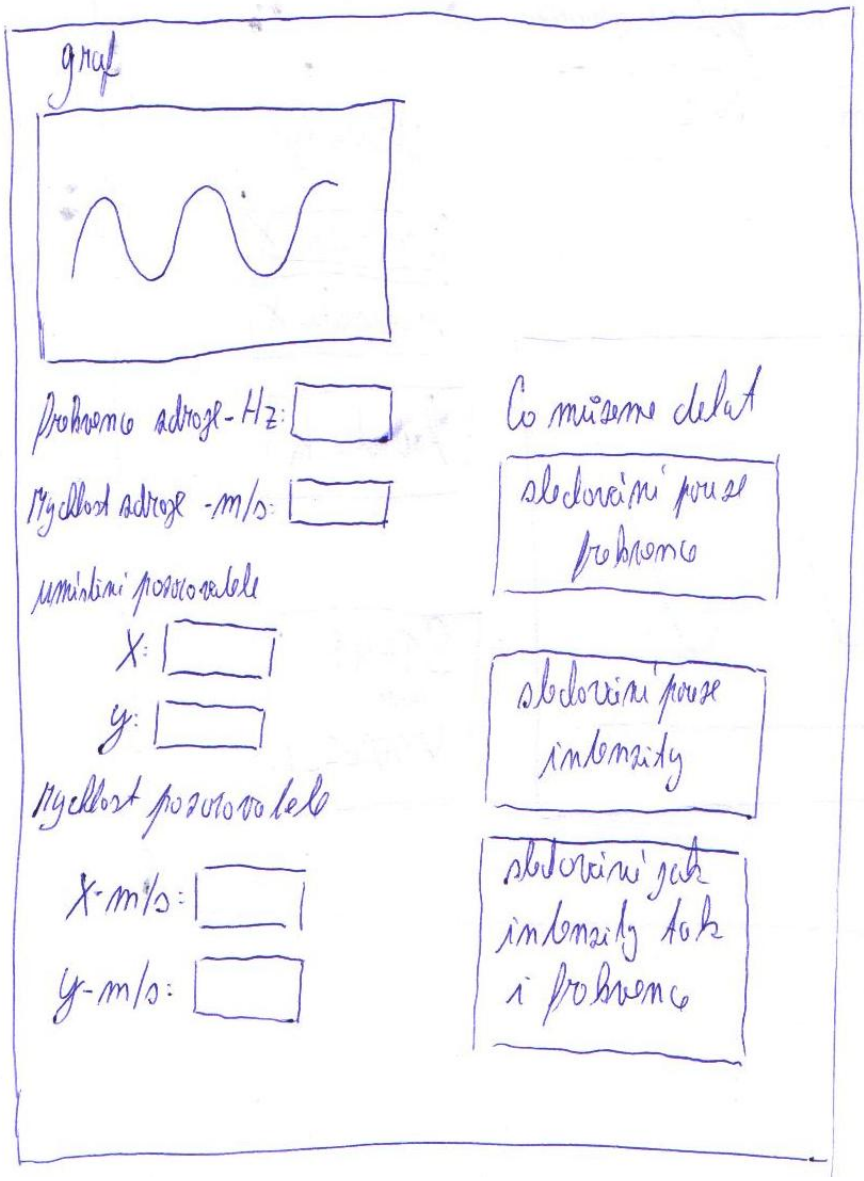
Dopplerův jev lze pozorovat i u policejních aut. Auta policie jsou vybavena radary RAMER-7, které pracují v mikrovlnném pásmu 34,0 a 34,3 GHz. Úzký radarový paprsek má velmi malý vysílací výkon, rychlost se měří na vzdálenost kolem 35 metrů.

Obr. 47: Radar umístěný na policejním automobilu

Radar (obr. 47) je obvykle zabudován v policejním automobilu, měření se tak může provádět ze stojícího i jedoucího auta. Průběh a výsledky kontroly jsou zdokumentovány digitální kamerou a přeneseny do počítače. Pro noční měření je radar s kamerou doplněn bleskem[9].

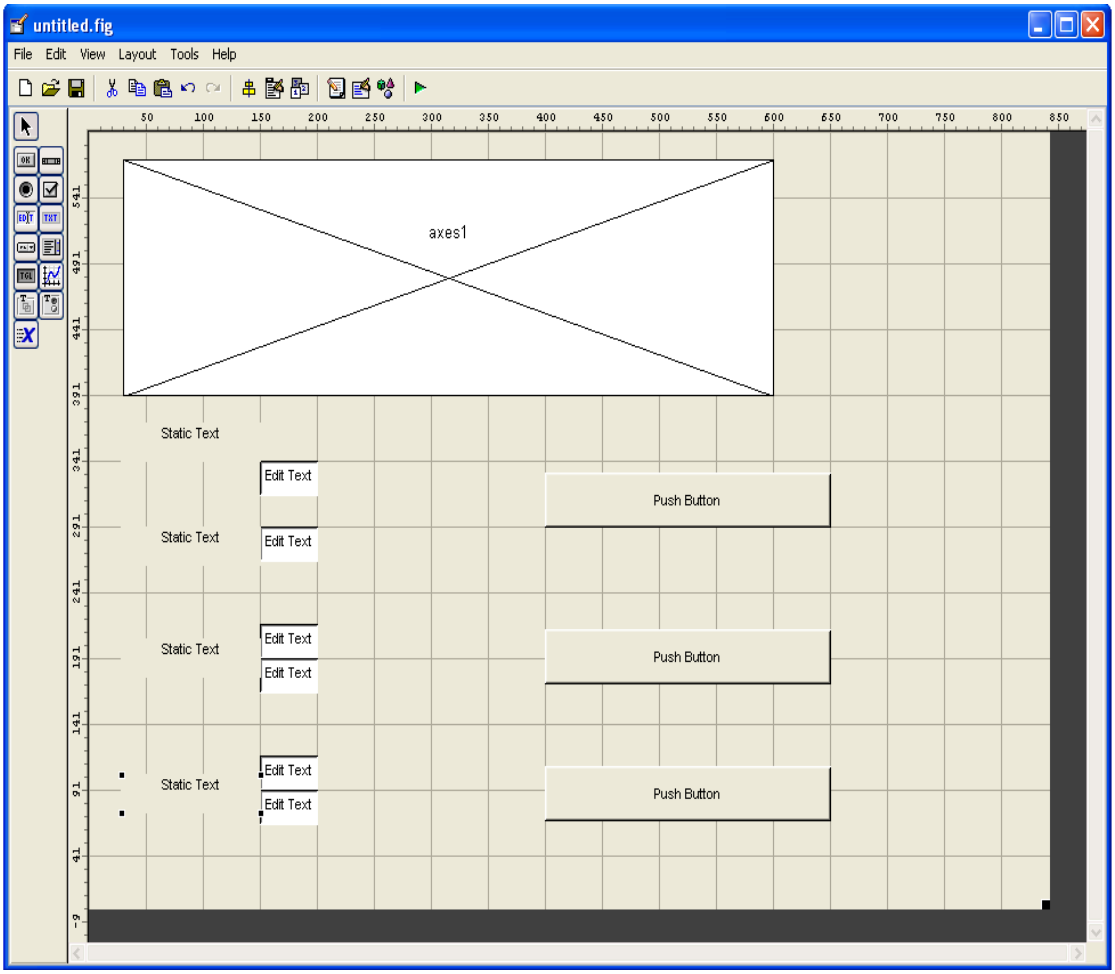
5.2. GUI Doppler

V této kapitole přistoupíme k vytvoření aplikace. Budeme postupovat obdobným způsobem, jako tomu bylo u aplikace pro vykreslování funkcí sinus a cosinus. Nejdříve si tedy uděláme náčrt toho, jak by naše aplikace měla vypadat, viz obr. 48. Jak již bylo řečeno, program bude mít za úkol simulovat frekvenci pohybujícího se zdroje vůči pozorovateli, který se také může pohybovat. Výstupem bude zvukový signál postupně se měnící frekvencí a intenzitou. Dále bude výstup zobrazen i graficky.

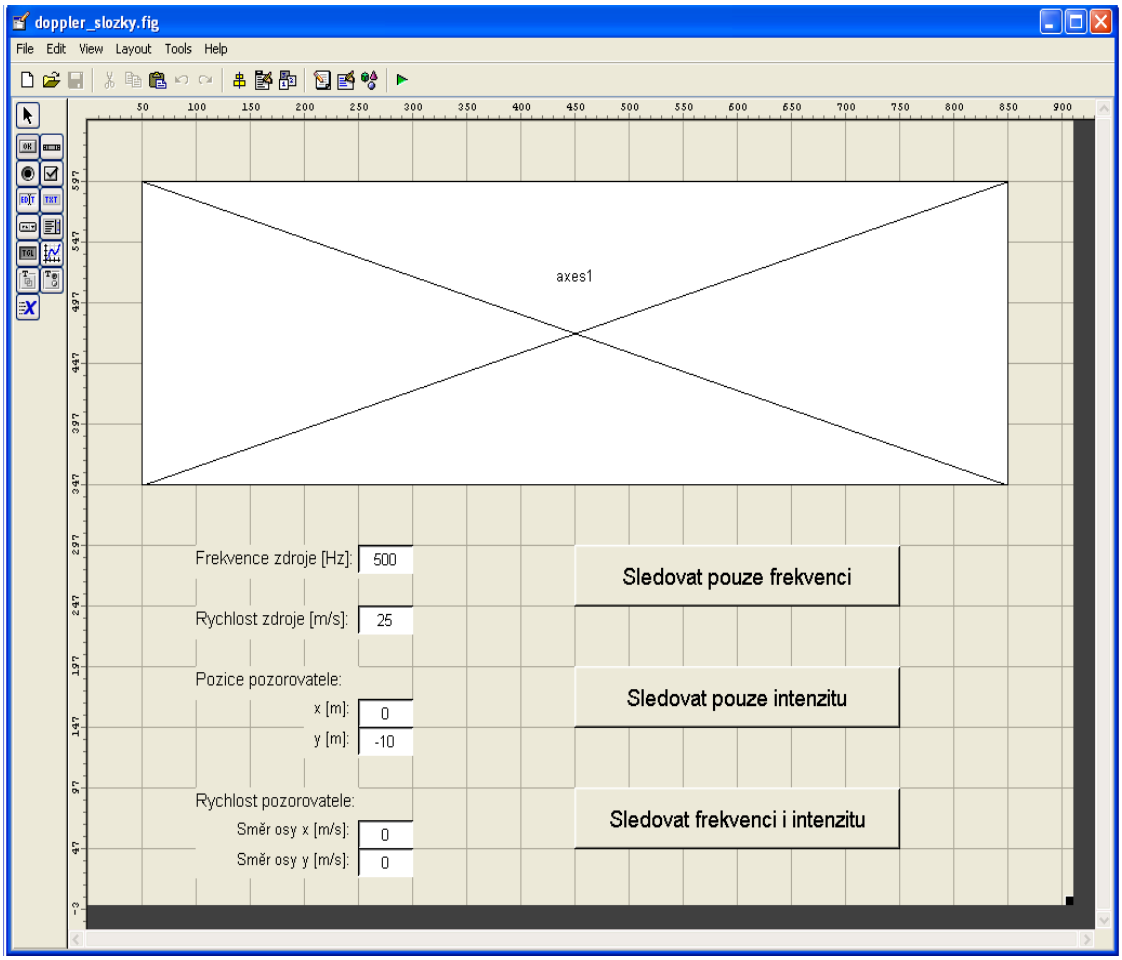


Obr. 48: Náčrt aplikace Doppler

Poté co jsme si rozložili rozmístění jednotlivých prvků, spustíme MATLAB a zvolíme vytváření nového grafického rozhraní. Potřebné komponenty přesuneme z panelů nástrojů na požadované. Jednotlivé komponenty pojmenujeme, určíme popisky a nadefinujeme tlačítka pro vykonávání námi požadované funkce. Nezapomeneme nastavit počáteční hodnoty (provedení těchto kroků bylo uvedeno již v předchozí kapitole). Na obrázku 49 je zobrazeno výsledné rozložení aplikace. V této fázi si program uložíme v našem případě pod jménem doppler_slozky.



Obr. 49: Počáteční vzhled.



Obr. 50. Výsledný vzhled

Nyní máme vytvořené grafické rozhraní aplikace a musíme ji oživit. Jak jsme si ukázali dříve, spustíme m-file editor do kterého dopíšeme požadované instrukce programu. Pro názornost si ukážeme popis jednotlivých příkazů pro edit1, které jsme si dopisovali, (obr. 51) a část pushbutton1 (obr. 52).

Na obrázku 52 vidíme zaškrtnuty dvě části kódu, které vypadají docela podobně. První část na řádkách 5 a 6 nám edituje proměnou f. Která je na řádkách 22 a 23 přednastavena na hodnotu 500. Je to vstupní frekvence, která se objeví po spuštění programu. Tuto frekvenci můžeme změnit na jinou hodnotu, nebo si můžeme nastavit jinou vstupní přednastavenou hodnotu a to tak, že přepíšeme v řádku 23 hodnotu 500 na požadovanou hodnotu.

```
97  function edit1_Callback(hObject, eventdata, handles)
98  % hObject    handle to edit1 (see GCBO)
99  % eventdata  reserved - to be defined in a future version of MATLAB
00  % handles    structure with handles and user data (see GUIDATA)
01
02  % Hints: get(hObject,'String') returns contents of edit1 as text
03  %        str2double(get(hObject,'String')) returns contents of edit1 as a double
04
05  handles.f=str2double(get(hObject,'String'));
06  guidata(hObject, handles)
07
08
09
10  % --- Executes during object creation, after setting all properties.
11  function edit1_CreateFcn(hObject, eventdata, handles)
12  % hObject    handle to edit1 (see GCBO)
13  % eventdata  reserved - to be defined in a future version of MATLAB
14  % handles    empty - handles not created until after all CreateFcns called
15
16  % Hint: edit controls usually have a white background on Windows.
17  %       See ISPC and COMPUTER.
18  if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
19      set(hObject,'BackgroundColor','white');
20  end
21
22  handles.f=500;
23  guidata(hObject, handles)
```

Obr. 52: Dopsáni kódu pro edit1

Na obrázku 53 je znázorněna část napsaného kódu pro *pushbutton1*. Tato funkce začíná nejdříve načítáním vstupních hodnot, které vidíme na řádcích 256 – 263. Na těchto řádcích máme příkazy pro načtení frekvence klaksonu, rychlosti zdroje, poloha pozorovatele a jeho rychlost v jednotlivých osách.

Část zdrojového kódu uvedena od řádku 265 odpovídá vlastnímu výpočtu, kde nejdříve máme nastaveny počáteční hodnoty. Řádky 267 – 270 zajistí nastavení počátečních hodnot. Následné v řádcích 273 – 292 jsou dopočítány ostatní potřebné parametry, které slouží pro vynesení grafu. Komentáře k jednotlivým řádkům jsou uvedeny za znakem procenta, což nám také slouží k lepší přehlednosti zdrojového kódu.

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


handles=guidata(hObject)

% nacitani hodnot
f=handles.f;                        % frekvence klaksonu
w=handles.w;    % rychlost zdroje
x0=handles.x0;      % x-ova souradnice pozorovatele v case 0
y0=handles.y0;      % y-ova souradnice pozorovatele v case 0
vx=handles.vx;   % rychlost pozorovatele ve smeru osy x
vy=handles.vy;    % rychlost pozorovatele ve smeru osy y


% vlastni vypocet

xz0=-200;   % x-ova souradnice zdroje v case 0
c=340;          % rychlost zvuku
t=0:0.0001:15;  % cas
T=1/f;          % zakladni perioda pro klakson


a=x0+vx*t-xz0-w*t;      % koeficienty do vzorcu
b=y0+vy*t;
t1=(-(vx*a+vy*b)-sqrt((vx*a+vy*b).^2-(vx^2+vy^2-c^2).*(a.^2+b.^2)))/(vx^2+vy^2-c^2);
                        % cas ktery potrebuje zvuk na to, aby dorazil od zdroje k
                        % pozorovateli
t=t+T;

a=x0+vx*t-xz0-w*t;      % vypocet tehoz, ale nyni pro nasledujici periodu
b=y0+vy*t;
t2=(-(vx*a+vy*b)-sqrt((vx*a+vy*b).^2-(vx^2+vy^2-c^2)*(a.^2+b.^2)))/(vx^2+vy^2-c^2);


t=t-T;
tt=t2-t1;       % diference casu - to, o co se pozorovateli zkrati perioda zdroje

fp=ones(1,length(tt))./(tt+T);  % frekvence vnimana pozorovatelem


% % % intenzita=600./((x0+vx*(t+t1)-xz0-w*t).^2+(y0+vy*(t+t1)).^2);   % vypocet intenzity zvuku
u=interp1(2*pi*f*(t+t1), sin(2*pi*f*t),2*pi*f*t);    % funkcionalni zobrazeni funkce sinus definovane
                                % na ekvidistantne delenych bodech t+t1 na ekvidistantni sit t
```

Obr. 53: Část funkce pro pushbutton1.

5.3. Zdrojový kód Dopplera

```
function varargout = doppler_slozky(varargin)
% DOPPLER_SLOZKY M-file for doppler_slozky.fig
%      DOPPLER_SLOZKY, by itself, creates a new DOPPLER_SLOZKY or
raises the existing
%      singleton*.
%
%      H = DOPPLER_SLOZKY returns the handle to a new DOPPLER_SLOZKY
or the handle to
%      the existing singleton*.
%
%      DOPPLER_SLOZKY('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in DOPPLER_SLOZKY.M with the given
input arguments.
%
%      DOPPLER_SLOZKY('Property','Value',...) creates a new
DOPPLER_SLOZKY or raises the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before doppler_slozky_OpeningFunction gets
called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to doppler_slozky_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help doppler_slozky

% Last Modified by GUIDE v2.5 09-Nov-2009 18:53:44

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @doppler_slozky_OpeningFcn, ...
                   'gui_OutputFcn',  @doppler_slozky_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
else
    gui_mainfcn(gui_State, varargin{:})
end
% End initialization code - DO NOT EDIT


% --- Executes just before doppler_slozky is made visible.
```

```
function doppler_slozky_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to doppler_slozky (see VARARGIN)

% Choose default command line output for doppler_slozky
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes doppler_slozky wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = doppler_slozky_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1
as a double

handles.f=str2double(get(hObject,'String'));
guidata(hObject, handles)



% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
handles.f=500;
guidata(hObject, handles)

function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3
as a double

handles.w=str2double(get(hObject,'String'));
guidata(hObject, handles)


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

handles.w=25;
guidata(hObject, handles)


function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4
as a double

handles.x0=str2double(get(hObject,'String'));
guidata(hObject, handles)


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end


handles.x0=0;
guidata(hObject, handles)

function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5
as a double

handles.y0=str2double(get(hObject,'String'));
guidata(hObject, handles)



% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

handles.y0=-10;
guidata(hObject, handles)



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6
as a double

handles.vx=str2double(get(hObject,'String'));
guidata(hObject, handles)


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

handles.vx=0;
guidata(hObject, handles)



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7
as a double

handles.vy=str2double(get(hObject,'String'));
guidata(hObject, handles)


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

handles.vy=str2double(get(hObject,'String'));
guidata(hObject, handles)


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


handles=guidata(hObject)

% nacitani hodnot
f=handles.f;                        % frekvence klaksonu
w=handles.w;    % rychlost zdroje
x0=handles.x0;      % x-ova souradnice pozorovatele v case 0
y0=handles.y0;      % y-ova souradnice pozorovatele v case 0
```

```
vx=handles.vx;    % rychlost pozorovatele ve smeru osy x
vy=handles.vy;     % rychlost pozorovatele ve smeru osy y


% vlastni vypocet

xz0=-200;    % x-ova souradnice zdroje v case 0
c=340;           % rychlost zvuku
t=0:0.0001:15;  % cas
T=1/f;           % zakladni perioda pro klakson


a=x0+vx*t-xz0-w*t;       % koeficienty do vzorcu
b=y0+vy*t;
t1=(-(vx*a+vy*b)-sqrt((vx*a+vy*b).^2-(vx^2+vy^2-
c^2).*(a.^2+b.^2)))/(vx^2+vy^2-c^2);
                         % cas ktery potrebuje zvuk na to, aby dorazil
od zdroje k
                         % pozorovateli
t=t+T;

a=x0+vx*t-xz0-w*t;       % vypocet tehoz, ale nyni pro nasledujici
periodu
b=y0+vy*t;
t2=(-(vx*a+vy*b)-sqrt((vx*a+vy*b).^2-(vx^2+vy^2-
c^2)*(a.^2+b.^2)))/(vx^2+vy^2-c^2);


t=t-T;
tt=t2-t1;        % diference casu - to, o co se pozorovateli zkrati
perioda zdroje

fp=ones(1,length(tt))./(tt+T);  % frekvence vnimana pozorovatelem


% % % intenzita=600./((x0+vx*(t+t1)-xz0-w*t).^2+(y0+vy*(t+t1)).^2);
% vypocet intenzity zvuku
u=interp1(2*pi*f*(t+t1), sin(2*pi*f*t),2*pi*f*t);    % funkcionalni
zobrazeni funkce sinus definovane
                                  % na ekvidistantne delenych bodech
t+t1 na ekvidistantni sit t




% graficky vystup
%prvni graf
axes1(handles.frekvence_graf);   % Select the proper axes
 plot(t,fp)
 minimum_y=0.95*min(fp)
 maximum_y=1.05*max(fp)
set(handles.frekvence_graf,'yLim', [minimum_y maximum_y]);
line([0 15],[f f],'color',[0 0 0])
 %druhy graf
 axes1(handles.intenzita_graf);   % Select the proper axes
plot(t,intenzita)
 set(handles.intenzita_graf);
 %treti graf
```

```
 axes1(handles.vychylka_graf);    % Select the proper axes
 plot(t-t1,u)
set(handles.vychylka_graf,'xLim', [0 15]);


wavplay(u, 10000,'async')       % vyvolani zvuku se soucasnym
odstranenim blokovani prikazoveho radku


h=handles.axes1;
h1=plot(h, x0, y0, 'o', 'markersize', 12, 'MarkerFaceColor', 'b')
hold on
h2=plot(h, xz0, 0,'or', 'markersize', 12, 'MarkerFaceColor', 'r')
axis([xz0, -xz0, 1.5*y0-1, -1.5*y0+1])
for i=1:78:length(t)
    set(h2, 'xdata', xz0+w*t(i))
    drawnow
end

delete(h1)
delete(h2)
```

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles=guidata(hObject)

% nacitani hodnot
f=handles.f;                        % frekvence klaksonu
w=handles.w;    % rychlost zdroje
x0=handles.x0;      % x-ova souradnice pozorovatele v case 0
y0=handles.y0;      % y-ova souradnice pozorovatele v case 0
vx=handles.vx;   % rychlost pozorovatele ve smeru osy x
vy=handles.vy;    % rychlost pozorovatele ve smeru osy y


% vlastni vypocet

xz0=-200;   % x-ova souradnice zdroje v case 0
c=340;          % rychlost zvuku
t=0:0.0001:15;  % cas
T=1/f;          % zakladni perioda pro klakson


a=x0+vx*t-xz0-w*t;      % koeficienty do vzorcu
b=y0+vy*t;
t1=(-(vx*a+vy*b)-sqrt((vx*a+vy*b).^2-(vx^2+vy^2-
c^2).*(a.^2+b.^2)))/(vx^2+vy^2-c^2);
```

```
                        % cas ktery potrebuje zvuk na to, aby dorazil
od zdroje k
                        % pozorovateli
t=t+T;

a=x0+vx*t-xz0-w*t;      % vypocet tehoz, ale nyni pro nasledujici
periodu
b=y0+vy*t;
t2=(-(vx*a+vy*b)-sqrt((vx*a+vy*b).^2-(vx^2+vy^2-
c^2)*(a.^2+b.^2)))/(vx^2+vy^2-c^2);


t=t-T;
tt=t2-t1;       % diference casu - to, o co se pozorovateli zkrati
perioda zdroje

fp=ones(1,length(tt))./(tt+T);  % frekvence vnimana pozorovatelem


intenzita=600./((x0+vx*(t+t1)-xz0-w*t).^2+(y0+vy*(t+t1)).^2);   %
vypocet intenzity zvuku
u=interp1(2*pi*f*(t), intenzita.*sin(2*pi*f*t),2*pi*f*t);    %
funkcionalni zobrazeni funkce sinus definovane
                            % na ekvidistantne delenych bodech
t+t1 na ekvidistantni sit t

wavplay(u, 10000,'async')       % vyvolani zvuku se soucasnym
odstranenim blokovani prikazoveho radku

h=handles.axes1;
h1=plot(h, x0, y0, 'o', 'markersize', 12, 'MarkerFaceColor', 'b')
hold on
h2=plot(h, xz0, 0,'or', 'markersize', 12, 'MarkerFaceColor', 'r')
axis([xz0, -xz0, 1.5*y0-1, -1.5*y0+1])
for i=1:78:length(t)
    set(h2, 'xdata', xz0+w*t(i))
    drawnow
end

delete(h1)
delete(h2)


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


handles=guidata(hObject)

% nacitani hodnot
f=handles.f;                        % frekvence klaksonu
w=handles.w;    % rychlost zdroje
x0=handles.x0;      % x-ova souradnice pozorovatele v case 0
y0=handles.y0;      % y-ova souradnice pozorovatele v case 0
vx=handles.vx;   % rychlost pozorovatele ve smeru osy x
vy=handles.vy;    % rychlost pozorovatele ve smeru osy y
```

```
% vlastni vypocet

xz0=-200;   % x-ova souradnice zdroje v case 0
c=340;          % rychlost zvuku
t=0:0.0001:15;  % cas
T=1/f;          % zakladni perioda pro klakson


a=x0+vx*t-xz0-w*t;      % koeficienty do vzorcu
b=y0+vy*t;
t1=(-(vx*a+vy*b)-sqrt((vx*a+vy*b).^2-(vx^2+vy^2-
c^2).*(a.^2+b.^2)))/(vx^2+vy^2-c^2);
                        % cas ktery potrebuje zvuk na to, aby dorazil
od zdroje k
                        % pozorovateli
t=t+T;

a=x0+vx*t-xz0-w*t;      % vypocet tehoz, ale nyni pro nasledujici
periodu
b=y0+vy*t;
t2=(-(vx*a+vy*b)-sqrt((vx*a+vy*b).^2-(vx^2+vy^2-
c^2)*(a.^2+b.^2)))/(vx^2+vy^2-c^2);


t=t-T;
tt=t2-t1;       % diference casu - to, o co se pozorovateli zkrati
perioda zdroje

fp=ones(1,length(tt))./(tt+T);  % frekvence vnimana pozorovatelem


intenzita=600./((x0+vx*(t+t1)-xz0-w*t).^2+(y0+vy*(t+t1)).^2);   %
vypocet intenzity zvuku
u=interp1(2*pi*f*(t+t1), intenzita.*sin(2*pi*f*t),2*pi*f*t);    %
funkcionalni zobrazeni funkce sinus definovane
                                % na ekvidistantne delenych bodech
t+t1 na ekvidistantni sit t




% % % % % % graficky vystup
% % % % % % prvni graf
axes(handles.frekvence_graf);   % Select the proper axes
plot(t,fp)
minimum_y=0.95*min(fp)
maximum_y=1.05*max(fp)
set(handles.frekvence_graf,'yLim', [minimum_y maximum_y]);
line([0 15],[f f],'color',[0 0 0])

% % % % % % druhy graf
axes(handles.intenzita_graf);   % Select the proper axes
 plot(t,intenzita)
 set(handles.intenzita_graf);

% % % % % % treti graf
```

```
 axes(handles.vychylka_graf);    % Select the proper axes
plot(t-t1,u)
 set(handles.vychylka_graf,'xLim', [0 15]);


wavplay(u, 10000,'async')       % vyvolani zvuku se soucasnym
odstranenim blokovani prikazoveho radku


h=handles.axes1;
h1=plot(h, x0, y0, 'o', 'markersize', 12, 'MarkerFaceColor', 'b')
hold on
h2=plot(h, xz0, 0,'or', 'markersize', 12, 'MarkerFaceColor', 'r')
axis([xz0, -xz0, 1.5*y0-1, -1.5*y0+1])
for i=1:78:length(t)
    set(h2, 'xdata', xz0+w*t(i))
    drawnow
end

delete(h1)
delete(h2)
```

Závěr

Ve své práci jsem se pokusil shrnout postup práce při vytváření grafického rozhraní pro aplikace vytvořené v programovém balíku MATLAB. V první části práce jsem shrnul základní postupy práce v programu, následně jsem popsal základní prvky používané při vytváření GUI a v poslední části jsem uvedl dvě konkrétní ukázky-aplikaci pro vykreslení funkce sinus a cosinus a aplikaci pro simulaci Dopplerova jevu.

Materiály pro svou práci jsem čerpal z mnoha zdrojů, a to jak českých tak zahraničních. Vzhledem k charakteru problematiky byla většina z nich prezentována na internetu.

Mohu konstatovat, že práce v programu MATLAB je zajímavá a intuitivní, vytváření programů je na rozdíl od programování v jiných jazycích mnohem jednodušší a výsledný zdrojový kód aplikace je lépe čitelný. Myslím, že právě tato vlastnost dělá program přístupný širokému spektru uživatelů.

Věřím, že předložená práce poslouží dalším zájemcům o tvorbu grafického rozhraní aplikací v programu MATLAB jako prostředek pro prvotní orientaci v této zajímavé problematice.

Seznam použité liteatury:

[1] MATLAB – adresa: http://en.wikipedia.org/wiki/MATLAB

[2] MATLAB - adresa:

http://www.humusoft.cz/produkty/matlab/matlab/

[3] Mgr. Majerová, D – výukové materiály z aresy:

http://uprt.vscht.cz/majerova/matlab/

[4] Uraj, J : Využití Grafical User Interface programového balíku

MATLAB při výuce fyziky

[5] Brabec, F: Graphical user interface pro MATLAB v7.1.

[6] Kupka, L.: Matlab, Simulink – úvod do používání

[7] Zaplatílek, K. – Doňar,B.: Matlab – tvorba uživatelských aplikací

[8] EDUTORIUM – adresa: http://www.techmania.cz/edutorium

[9] Ministerstvo obrany ČR – adresa:

http://www.army.cz/images/id_8001_9000/8753/radar/k33.htm

[10] MATLAB – adresa: http://www.math.ufl.edu/help/matlab-tutorial/

[11] Humusoft - http://www.humusoft.cz

[12] Görner, V.: Programový systém MATLAB

[13] Help programu MATLAB