Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra aplikované fyziky

Příprava výukového materiálu pro tvorbu GUI v MATLABu

Preparation of educational material for developing of GUI in MATLAB

Diplomová práce

Vedoucí práce:

RNDr. Petr Bartoš, Ph.D.

Autor:

Bc. Jakub Šimek

České Budějovice 2012

Prohlášení:
Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění, souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce.

Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 26. 6. 2012 …….....

Děkuji vedoucímu diplomové práce RNDr. Petru Bartošovi Ph.D. za trpělivost při konzultacích, cenné rady, připomínky a metodické vedení práce.

Anotace

Cílem této diplomové práce bylo vytvoření výukového materiálu pro tvorbu grafických uživatelských rozhraní v prostředí programu MATLAB. Čitatel zde nalezne základní postupy tvorby GUI, získá přehled o jednotlivých grafických objektech a seznámí se s prostředím GUIDE, které je pro vývoj aplikací určeno. Učební text je doplněn o praktické ukázky zdrojových kódů, aby čtenáři co nejvíce přiblížila probíranou problematiku. Celý materiál je doplněn o dvě ilustrativní funkční aplikace s výpisem jejich zdrojových kódů.

Klíčová slova

MATLAB, program, GUI, objekt, aplikace, zdrojový kód, příklad

Abstract

The aim of this thesis is to create an educational material for creating graphical user interfaces in MATLAB. The reader will learn the basic procedures of the GUI creation, get to know the various graphic objects and learn about the GUIDE environment, which is intended for the actual application development. Text is supplemented by practical examples of source code fragments as much as possible. All material is accompanied by two functional sample applications with their source code.

Keywords

MATLAB program, GUI, object, application, source code, example

Obsah

Úvod

Tato diplomová práce je koncipována jako výukový text, který čtenáře seznámí se základy tvorby grafických aplikací v prostředí MATLAB. Ten je dnes jedním z nejrozšířenějších programových balíků pro technické výpočty. Využívá se ve firmách, výzkumných pracovištích a univerzitách po celém světě.

Přestože práce velmi stručně seznamuje se základy práce v programovém prostředí MATLAB, je určena spíše těm čtenářům, kteří již zvládají základní kroky a vyžadují určitou nadstavbu. Cílem je pomoci pokročilejším uživatelům naučit se základní znalosti aplikovat, tvořit více či méně rozsáhlé algoritmy, programovat, vytvářet grafická rozhraní apod.

Práci je možné rozdělit na několik částí. První z nich obsahuje úvod do MATLABu, popis uživatelského prostředí, seznámení s nápovědou a stručné představení základů práce s programem.

Stěžejní částí práce je potom programování, které se zabývá m-soubory, tvorbou skriptů, funkcí a řídícími strukturami. Její zvládnutí je nezbytné pro vytváření uživatelských aplikací, kterou se zabývá třetí část práce.

Zde se čtenář seznámí se systémem Handle Graphic, nástroji pro tvorbu grafických aplikací a postupy při jejich vytváření.

Při psaní byl kladen důraz na názornost probírané problematiky, proto jsou jednotlivé kapitoly doplněny praktickými ukázkami a příklady. Čtenář si tak může vše ihned vyzkoušet přímo v MATLABu.

1 MATLAB

MATLAB(Matrix laboratory) je programové prostředí určené pro vědecké a technické výpočty, které dnes nalézá uplatnění v mnoha vědních oborech. Uživateli poskytuje nástroje pro měření v reálném čase, práci s matematikou, grafikou, přenos dat apod. To vše v graficky příjemném prostředí. Všestrannost programu je zajištěna otevřenou architekturou a velkým množstvím knihoven, tzv. tolboxů, kterými tento výpočetní systém disponuje.

MATLAB má vlastní programovací jazyk a umožňuje data nejen kvalitně zpracovat, ale i zobrazit. Základem filozofie práce v MATLABu je práce s maticemi, což výrazně usnadňuje práci oproti jiným programovacím jazykům jako je C, C++, Fortran nebo Java. Navíc je s těmito jazyky plně kompatibilní.

MATLAB se skládá z těchto základních částí:

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

1.1 Výpočetní jádro

Jak již bylo řečeno, základen práce v programu MATLAB je práce s maticemi. Výpočetní jádro tedy pracuje zejména s operátory pro matice reálných a komplexních čísel. MATLAB ale umožňuje i běžné matematické operace jako násobení, dělení, sčítání a odčítání. Tento režim práce připomíná práci s kalkulátorem.

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, ve kterých každý prvek může být jiného typu.

Uživatel si může zvolit způsob ukládání čísel. MATLAB typicky ukládá čísla s dvojitou přesností (double). Podporuje ale i menší přesnost, aby bylo možné efektivně pracovat i s malou pamětí počítače.
Můžeme také pracovat s vektory a polynomy, pro které je k dispozici několik funkcí (např.: výpočet střední hodnoty, hledání extrémů, výpočet směrodatné odchylky apod.).

V MATLABu je také možnost práce s objekty. Ty uživateli umožňují rozšířit výpočetní prostředí o nové datové typy, na kterých je možno definovat libovolné funkce a operátory.

1.2 Grafický subsystém

Velkou předností programu je grafický subsystém, který nabízí široké spektrum možností prezentace výsledků. Je možné vykreslovat dvourozměrné i třírozměrné grafy, histogramy, koláčové grafy a další.

Program dovoluje i tvorbu grafických prvků jako jsou tlačítka, posuvníky, rolovací menu apod., což umožňuje vytváření plnohodnotných grafických aplikací.

1.3 Otevřená architektura

Mimo vestavěných funkcí MATLAB umožňuje uživateli vytvářet funkce vlastní, které budou přímo odpovídat požadavkům jím vytvářené aplikace. Tyto funkce jsou uloženy v souborech a voláním se nijak neliší od vestavěných funkcí. 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.

1.4 Pracovní nástroje

MATLAB je vytvořen tak, že umožňuje vytvářet plnohodnotné aplikace, obsahuje tedy všechny příkazy pro psaní programů. Přesto je jednoduchý a snadno zvladatelný.

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 ladění 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 jak pro začátečníka, tak i profesionála.

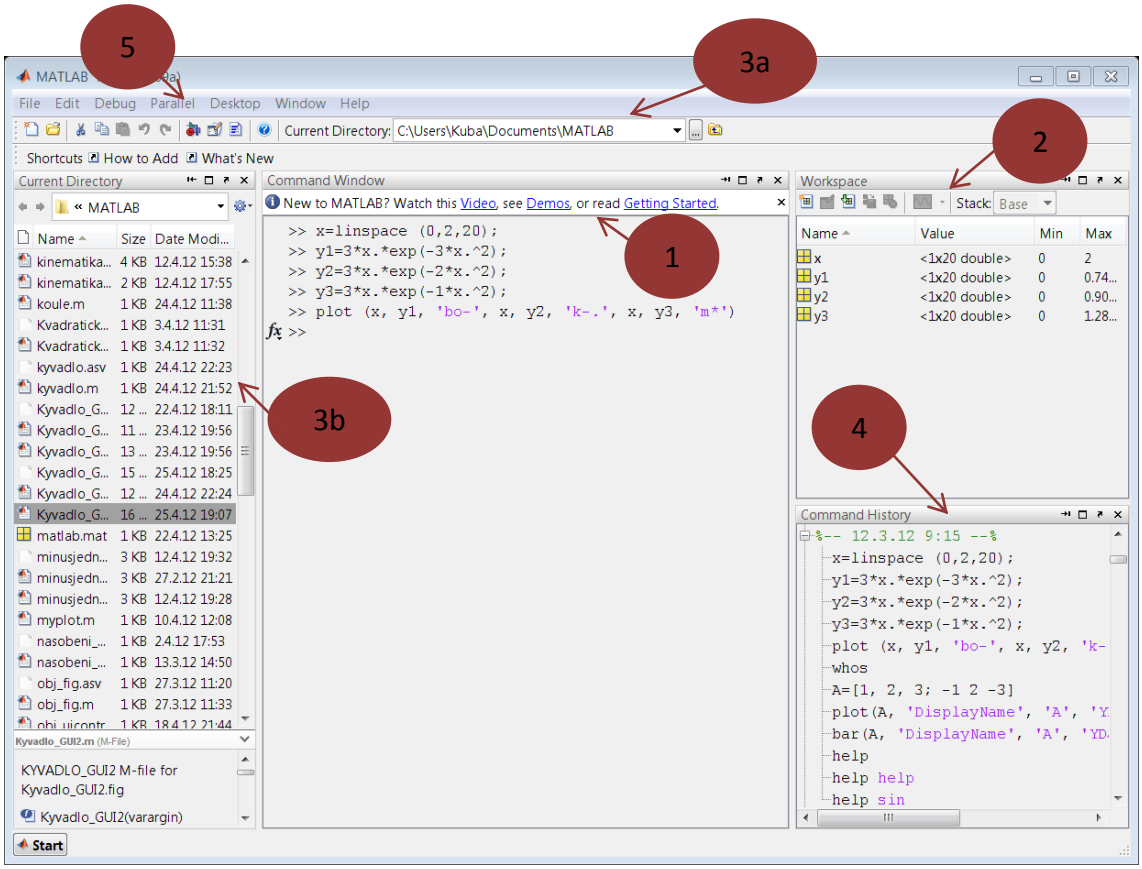
Další významnou předností MATLABu je jeho velmi těsná integrace s programovacím jazykem Java, kde můžou být objekty Java programem MATLAB přímo využity. Kromě toho je možné k MATLABu připojovat také moduly napsané v jazyce C a ve Fortranu.

1.5 Tolboxy

Tolboxy jsou specializované knihovny MATLABu, které zajišťují jeho všestranné použití v různých vědních a technických oborech. Knihovny obsahují předem zpracované funkce určené pro daný vědní obor. Tyto funkce lze navíc dále rozšiřovat a upravovat. Součástí každého tolboxu je i detailní a přehledná dokumentace, která přesně popisuje použité algoritmy a obsahuje také odkazy na vědecké zdroje.

2 Popis základního grafického rozhraní – MATLAB desktop

Po spuštění programu MATLAB se objeví programové prostředí, které můžeme rozdělit na několik částí. Jednotlivé části a jejich funkce si nyní popíšeme:



Obr. 2.1: Grafické rozhraní programu MATLAB

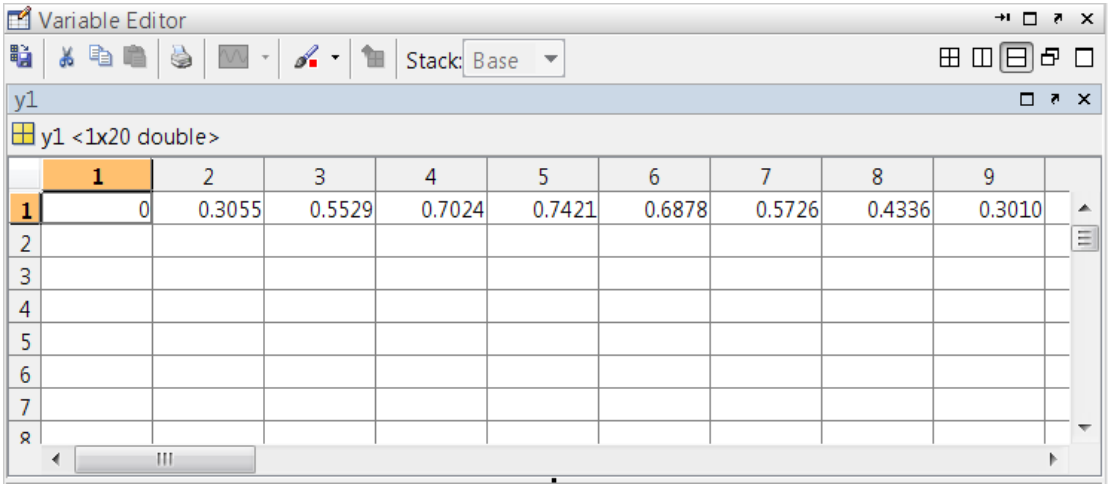
2.1 Command Window (1)

Tato část programu slouží k zadávání jednotlivých příkazů, které jsou po stisknutí klávesy ENTER ihned provedeny. Jsou zde také vypisována chybová hlášení a varování, pokud jednotlivé příkazy nedoplníme středníkem, i obsah proměnných. Nevýhodou tohoto způsobu práce je, že pokud dojde k ukončení programu, jednotlivé proměnné nejsou uloženy. Tento způsob práce je tedy vhodný zejména pokud chceme rychle něco spočítat a jednotlivé proměnné nebudeme potřebovat opakovaně. Každý příkaz je potřeba zadat samostatně a potvrdit ho stiskem klávesy ENTER. Pro práci v okně *Command Window* je možné využít následující klávesy:

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

2.2 Workspace (2)

V tomto okně můžeme nalézt seznam všech proměnných a jejich hodnot načtených v paměti počítače. U každé proměnné je uveden její název, velikost, datový typ a minimální a maximální hodnoty prvku. Protože MATLAB data zpracovává jako matice, jsou jednotlivé prvky proměnné zapsány pomocí sloupců a řádků (**Value → <1x 20 double>**). Proměnné je také možné vypsat pomocí příkazu `who` nebo `whos`, po jehož zadání se v okně *Command Window* objeví úplný seznam všech definovaných proměnných. Obsah jednotlivých proměnných je možné přímo zobrazit a editovat, pokud poklepeme levým tlačítkem myši na některou z nich.



	1	2	3	4	5	6	7	8	9
1	0	0.3055	0.5529	0.7024	0.7421	0.6878	0.5726	0.4336	0.3010
2									
3									
4									
5									
6									
7									
8									

Obr. 2.2: Obsah proměnné y1 z okna Workspace

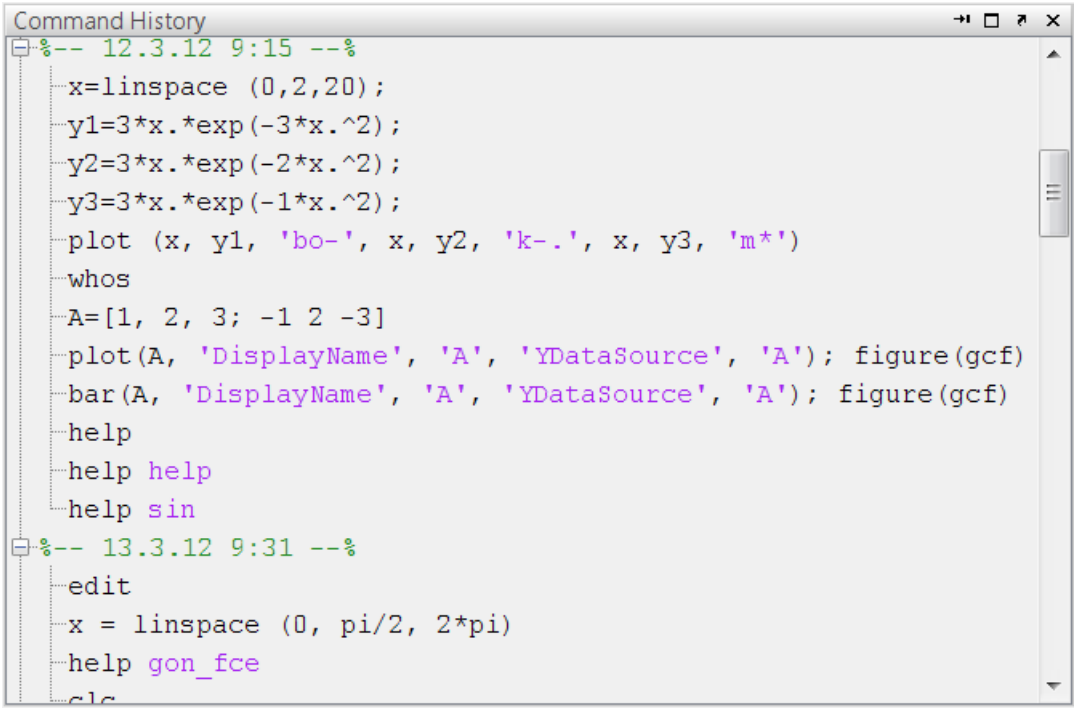
2.3 Current Directory (3a, 3b)

Aktuální adresář je využíván k ukládání rozdělané práce a měly by v něm být vloženy všechny námi vytvořené funkce. Cesta k danému adresáři je zobrazená pomocí příkazového řádku **(3a)**. Pokud bychom chtěli nastavení pracovního adresáře změnit, stačí použít ikonu se třemi tečkami, kde nastavíme novou cestu, nebo cestu přepsat přímo pomocí příkazového řádku. Pracovní adresář můžeme nastavit také pomocí hlavního okna MATLABu, kde k tomu slouží příkaz `cd`. Pokud bychom výsledky naší práce chtěli ukládat například do dokumentů, stačí do okna Command Window zapsat `cd C:\Users\Uzivatel\Documents`. Položka Uživatel pak odpovídá uživatelskému účtu v systému windows. Obsah aktuálně nastaveného adresáře je zobrazen vlevo **(3b)**. Z tohoto adresáře můžeme spouštět a editovat námi vytvořené soubory, jako jsou například skripty a funkce. Pro výpis obsahu aktuálního adresáře můžeme také použít příkaz `dir,` který zapíšeme do hlavního okna *Command Window*.

Pomocí tohoto adresáře můžeme také provádět základní operace se soubory, otevření *(Open)*, spuštění *(Run)*, prohlížení *(Open As Text)*, načtení dat *(Import Data)*. Přístup je k němu přes kliknutí pravým tlačítkem myši na vybraném souboru.

2.4 Command History (4)

Zde je možné nalézt příkazy, které uživatel zadal v okně *Command Window*. Historie příkazů je velice užitečná, pokud některý z příkazů potřebujeme užít opakovaně. Abychom tento příkaz nemuseli do okna *Command Window* zapisovat stále dokola, můžeme využít možnosti listování v historii užitých příkazů. Příkaz vyvoláme pomocí šipek ↑↓ nebo přetažením příkazu z okna *Command History*. Pokud známe název proměnné, můžeme využít také klávesu TAB. Do příkazového řádku v okně *Command Window* stačí zadat počáteční písmeno proměnné a po stisku klávesy TAB se objeví nabídka možností, které začínají tímto písmenem.



Obr. 2.3: Okno Command History

2.5 Menu (5)

Menu prostředí tvoří několik položek a ikon, s různými funkcemi pro ovládání programu. My si zde ukážeme alespoň některé z nich:

- **File** pomocí této nabídky je možné vytvářet například nové m-file soubory, či spustit grafický editor, importovat soubory, apod.
- **Edit** obsahuje např. položky pro kopírování textu přes schránku Windows jako vyjmout, kopírovat, vložit, ale také umožňuje smazat okna Command Window, Command History a Workspace. Dále jsou zde položky *Find* a *Find files*, které umožňují vyhledávání souborů a položek.
- **Debug** tato část menu je určena k ladění zdrojového kódu, může zde zdrojový kód procházet po jednotlivých krocích a snáze tak odhalit případné chyby.
- **Desktop** slouží k nastavení pracovní plochy. Uživatel si zde může určit, která z oken MATLABu budou aktivní a přizpůsobit si tak pracovní plochu dle svých potřeb.

- **Window** je aktivní při práci v grafickém režimu MATLABu.
- **Help** otevírá rozsáhlou nápovědu k systému MATLAB a jeho součástem.

3 Nápověda – MATLAB Help

Díky rozsáhlosti programu a velkého množství příkazů je pro uživatele práce s nápovědou velmi důležitá. Každý uživatel zde najde cenné rady nejen pro zápis syntaxí funkcí, ale i velké množství příkladů.

Pro spuštění nápovědy můžeme využít klávesu F1 nebo položku *Help* v menu. Nápovědu můžeme také spustit zapsáním příkazu `help` do okna *Command Window*. Pokud příkaz `help` zapíšeme bez dalších parametrů, objeví se seznam položek nápovědy. Bylo by zbytečné vyhledávat v těchto položkách, pokud chceme zobrazit nápovědu známé funkce, proto je možné příkaz help zapisovat společně s parametrem např.: `help sin`. V tomto případě dojde k zobrazení nápovědy pouze pro konkrétní funkci.

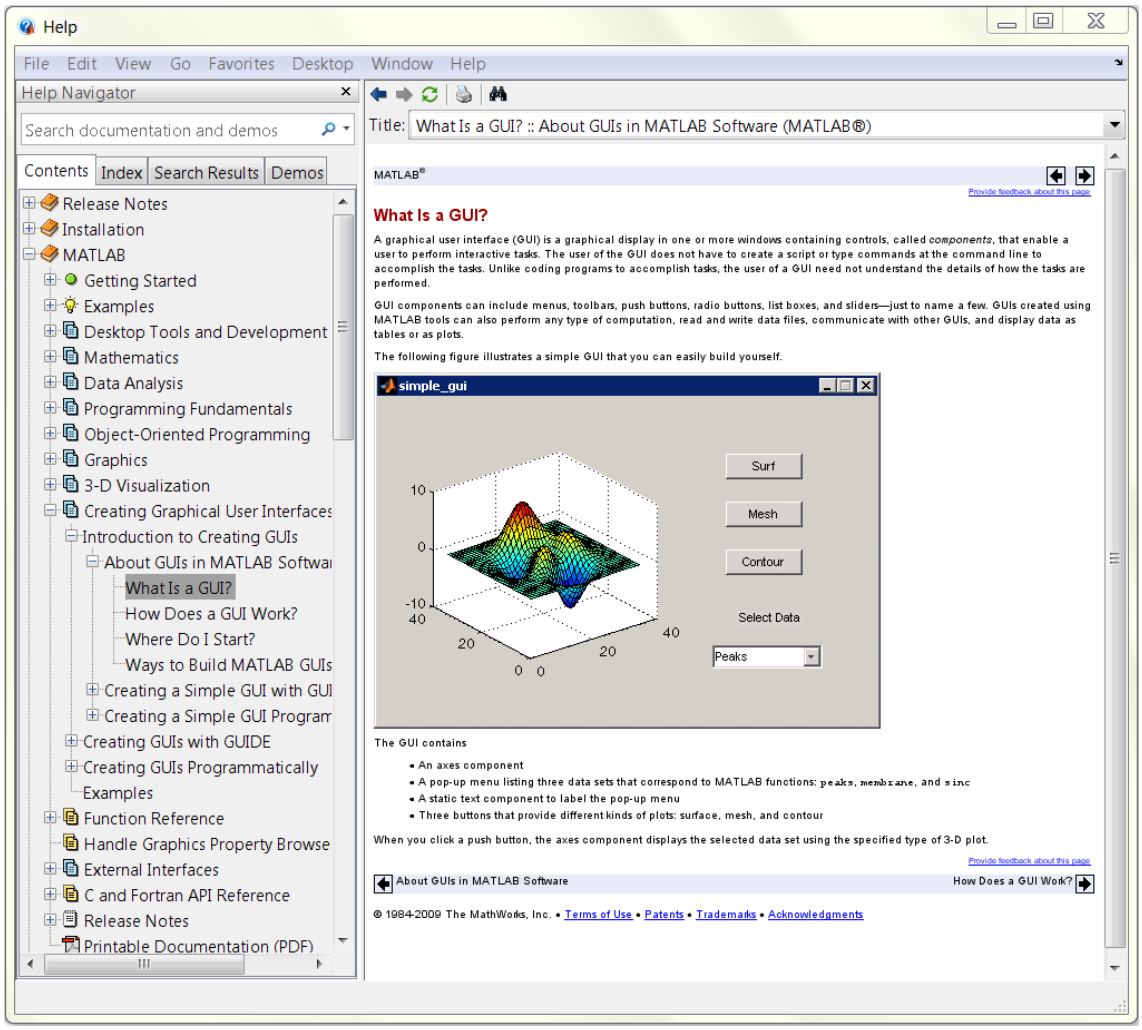
```
>> help sin
 SIN    Sine of argument in radians.
    SIN(X) is the sine of the elements of X.

    See also asin, sind.

    Overloaded methods:
       codistributed/sin

    Reference page in Help browser
       doc sin
```

Obr.3.1: Zobrazení nápovědy funkce sin

Pokud nápovědu vyvoláme stiskem klávesy F1 nebo pomocí menu, objeví se nové okno. V levé části tohoto nově vyvolaného okna můžeme nalézt několik záložek, z nichž nejvýznamnější je záložka *Contents*. Pod touto záložkou se skrývá několik kapitol, které zejména pro začátečníky budou životně důležité. Nalezneme zde kapitoly:

- Začínáme [Getting Started]
- Příklady [Examples]
- Popis prostředí [DesktopTools and Development Environment]
- Matematika[Mathematics]
- Analýza dat [Data Analysis]
- Programování [Programming]
- Grafika[Graphics],
- 3D-vizualizace [3D Visualization]
- Programování grafického prostředí [Creating Graphical User interface]

Dalšími záložkami jsou *index* a *demo*. V záložce *index* nalezneme rejstřík, kde jsou jednotlivé informace seřazeny chronologicky podle abecedy a usnadňují tak vyhledávání. Záložka *demo* pak obsahuje aplikace programované pomocí MATLABu a video ukázky práce s MATLABem.



Obr. 3.2: Nápověda programu MATLAB

Přes toto grafické rozhraní je možné zobrazit i velmi podrobnou nápovědu funkcí, kterými MATLAB disponuje. Pokud bychom chtěli zjistit například syntaxi zápisu funkce *rand*, stačí v hlavním okně zadat příkaz `doc rand`. V pravé části nápovědy se zobrazí podrobná dokumentace, kde uživatel může nalézt nejen syntaxi zápisu funkce ale i její popis a několik příkladů.

4 Základy práce s MATLABem

MATLAB je program, jehož základním stavebním kamenem je matice. Proto se zde veškeré výrazy i takto zpracovávají. Pokud tedy zapíšeme do hlavního okna například číslo 21, MATLAB jej bude chápat jako matici o rozměru 1x1.

4.1 Proměnné

Jakmile uživatel zapíše příkaz do okna *Command Window* a potvrdí klávesou *ENTER*, příkaz se ihned provede. Pokud uživatel neurčí jinak, je výsledek provedené operace ukládán do proměnné `ans`, která je implicitní proměnnou. Někdy je však třeba proměnné uchovat například pro další výpočty. V takových případech je proměnná `ans` nevhodná, protože se přepíše při zadání dalšího výrazu. Pokud chceme výraz uchovat, je třeba mu přiřadit identifikátor, který proměnnou jednoznačně určí. Pokud například zapíšeme `v=5`, vytvořili jsme proměnnou, která je jednoznačně určena písmenem **v** a je v ní uložena hodnota **5.**

Při zápisu proměnných je třeba dodržovat následující pravidla:

- Proměnná může obsahovat maximálně 31 znaků
- Proměnná nesmí začínat číslicí
- Proměnná může obsahovat pouze znaky anglické abecedy (a – z, A – Z), číslice (0 – 9) a podtržítko (_)

```
>> rychlost = 10; %m/s
>> draha = 75; %m
>> cas = draha/rychlost

cas =

    7.5000
```

Obr. 4.1: Příklad definice proměnné

4.2 Příklady definice proměnných

V předchozím příkladu jsme si ukázali, jak vytvořit proměnnou a jak s ní pracovat. Nyní si ještě ukážeme několik dalších příkladů proměnných a způsoby jejich zápisu. MATLAB mimo jiné umožňuje pracovat s:

- Komplexními čísly
- Řetězci
- Vektory
- Maticemi

4.2.1 Komplexní čísla

Proměnné v MATLABU nemusí být pouze reálné, ale mohou být také komplexní. Komplexní čísla je možné zadávat ve složkovém i exponenciálním tvaru, přičemž imaginární jednotku zapisujeme jako `i` nebo `j`.

```
c1=2+3i;              % složkový tvar komplexního čísla
c2=7+4j;
c3=1*exp(2*pi/5*1i); % exponenciální tvar komplexního čísla
```

Obr. 4.2: Zadání komplexního čísla

4.2.2 Řetězce

Řetězce se zapisují jako řada znaků ASCII uzavřených do apostrofů. Je vhodné používat pouze znaky anglické abecedy, protože při použití háčků a čárek bývá problém s jejich zobrazením.

```
% řetězec v MATLABu
text='Ukazka zapisu retezce pomoci MATLABu';
```

Obr. 4.3: Zadání řetězce

4.2.3 Matice

Matice můžeme zadávat několika způsoby:

- Zadání matice po prvcích
- vygenerování příkazem nebo funkcí
- načtení z externího souboru nebo aplikace

Matice se zapisují do hranatých závorek, kdy jednotlivé prvky na řádku jsou odděleny čárkou nebo mezerou a jednotlivé řádky středníkem.

```
A = [1 8 -4; 9 -7 6];                % Matice o rozmeru 2x3
B = [5 -3; 5/4 1; 9 -2; -14 9/10];  % Matice o rozmeru 4x2
```

Obr. 4.4: Zadání matice

4.2.4 Vektory

Vektory zadáváme jako matici typu **1 x n**. V MATLABu vektor můžeme definovat například takto:

```
% vytvoření vektoru
v=[sin(0) sin(pi/2) sin(pi) sin(3*pi/2) sin(2*pi)];
```

Obr. 4.5: Zadání vektoru

Pro vytvoření vektoru velmi často využíváme operátor dvojtečka (`:`), který slouží k vytváření posloupností a konstantním krokem a k tvorbě řad se sestupným či vzestupným uspořádáním prvků. Jeho syntaxe je například: `Jmeno_promenne = pocatecni_hodnota:krok:koncova_hodnota`. Takto vytvořený vektor bude začínat prvkem počáteční hodnota a každá další souřadnice vektoru bude o hodnotu krok větší než ta předchozí. Pro ukončení číselné řady je zde definována koncová hodnota, která určuje maximální možnou hodnotu, které může prvek nabývat.

```
v1= -3:3;   % Vektor od -3 do 3 s krokem 1
v2= 1:3:30; % Vektor od 1 do 30 s krokem 3
```

Obr. 4.6 Vytvoření vektoru pomocí operátoru dvojtečka

Pokud chceme vytvořit vektor se známým počtem ekvidistantně vzdálených prvků, můžeme využít příkaz `linspace`. Jeho syntaxe je: `Jmeno_promenne = linspace(pocatecni_hodnota,koncova_hodnota, pocet)`. Vytvoří se

vektor od počáteční hodnota do koncová hodnota s počtem prvků počet. Pokud chceme vytvořit logaritmickou řadu, je nám k dispozici funkce `logspace`, která má stejnou funkci jako `linspace`, pouze prvky uspořádá logaritmicky.

```
v3= linspace (0,2*pi,13)*180/pi; % Lineární vektor od 0 do 360
v4= logspace (1,3,50);           % Logaritmický vektor od 10 do 10^3
```

Obr. 4.7: Vytvoření vektoru pomocí linspace a logspace

4.3 Formát zobrazených čísel

MATLAB implicitně zobrazuje čísla nebo výrazy na čtyři desetinná místa. Jde o zobrazení v pevné řadové čárce (fix point). Samotné výpočty jsou prováděny s plnou přesností (double). Seznam dostupných možností formátování získáme `help format`. Implicitně je nastaven formát short.

`formatshort`	5 platných míst, 4 desetinná místa, pevná desetinná čárka
`format long`	15 platných míst, 14 desetinná místa, pevná desetinná čárka
`formatshort e`	5 platných míst, 4 desetinná místa, plovoucí desetinná čárka
`format long e`	15 platných míst, 14 desetinná místa, plovoucí desetinná čárka

4.4 Příkaz Clear

Pokud již nechceme některou proměnnou používat, je vhodné ji z důvodu přehlednosti a úspory paměti vymazat. K tomu slouží příkaz `clear`. Jeho syntaxe je `clear_promenna`. Např. pokud bychom chtěli vymazat proměnnou Jméno z předchozí části, zápis by zněl: `clear v3`. V případě, že budeme chtít vymazat všechny proměnné, můžeme použít příkaz `clear all`.

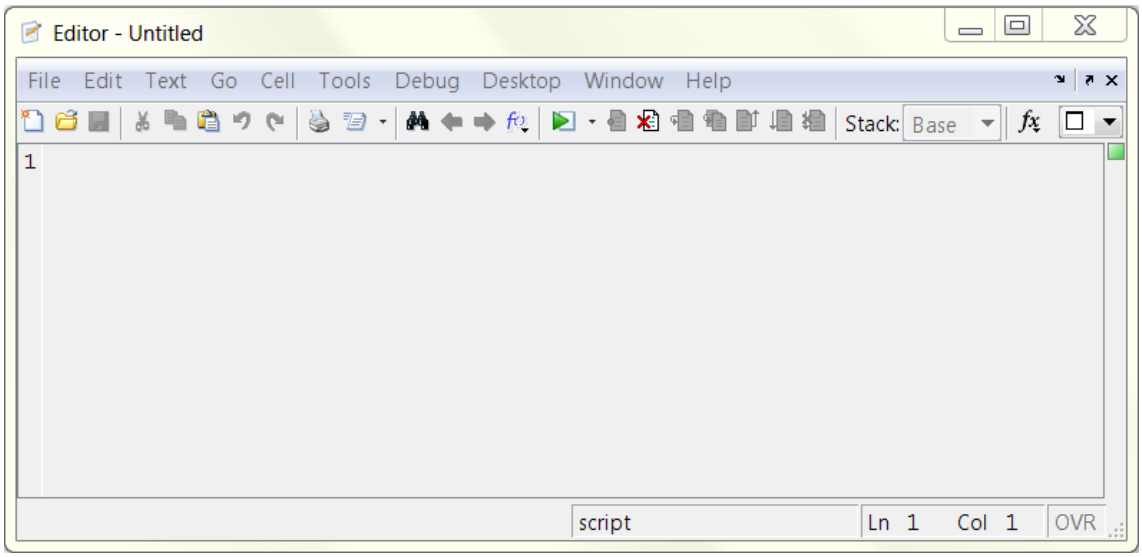
5 Úvod do programování v MATLABu

Jak už bylo uvedeno na začátku, MATLAB nabízí několik způsobů režimu práce. V začátcích zapisujeme příkazy zejména pomocí okna *Command Window*, kdy po zadání příkazu a stisku klávesy ENTER dojde k jejich okamžitému zpracování výpočetním jádrem a zobrazení výsledku. Tento způsob je nevýhodný, pokud píšeme nějaký delší program, nebo program, kde například chceme, aby se některá jeho část provedla opakovaně. Posloupnost příkazů zapisovaných do *Command Window* navíc není možné uložit a po ukončení MATLABu tak dochází ke ztrátě dat.

Pro takovéto případy je MATLAB vybaven textovým editorem (M-file editorem), ve kterém je možné výsledky naší práce uložit a poté opětovně otevřít.

5.1 M-file editor

M-file editor můžeme spustit z hlavního okna MATLABu kliknutím na *File* → *New* → *Blank M-File* nebo zápisem příkazu `edit` do okna *Command Window*. Do nově otevřeného okna je možné ihned zapisovat příkazy. Po vytvoření práce je možné zdrojový kód nejen uložit a opětovně otevřít, ale také odladit a spustit. Pomocí M-file editoru můžeme vytvářet plnohodnotné aplikace, které je možné doplnit i o grafické prvky jako jsou tlačítka, posuvníky, pole pro editaci a zápis textu, rolovací menu, grafy, zaškrtávací pole, kontextová menu apod.



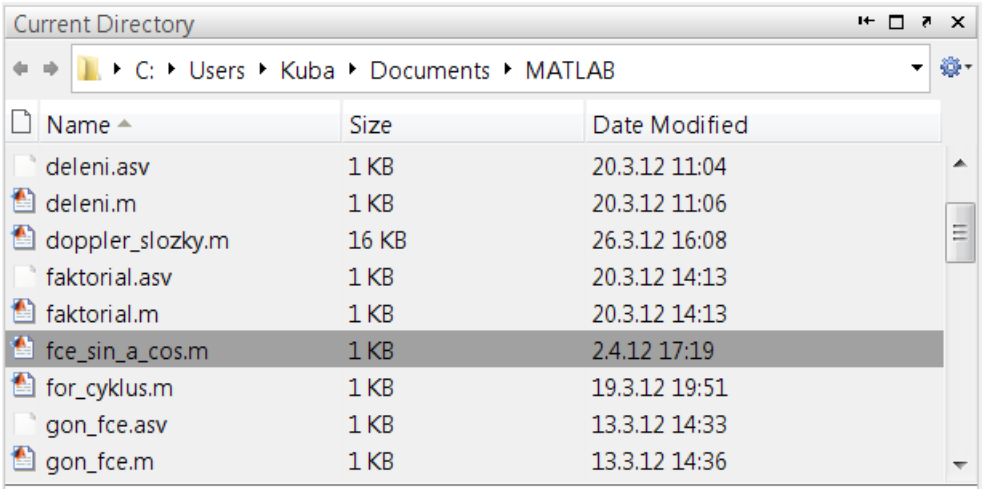
Obr. 5.1: Textový editor programu MATLAB

5.2 Uložení a načtení souboru

Vytvořený soubor je vhodné ihned uložit pomocí příkazu *Save as*. Po kliknutí na tento příkaz je uživatel vyzván k zadání názvu souboru. Zde je třeba dodržet několik pravidel:

- Název souboru může obsahovat maximálně 31 znaků
- Název souboru musí začínat písmenem
- Název souboru by měl obsahovat pouze znaky anglické abecedy (a – z, A – Z) a číslice (0 – 9). Pro oddělení slov je možné použít podtržítko (_)

Jakmile máme soubor správně pojmenovaný, dokončíme celou operaci kliknutím na tlačítko uložit. MATLAB implicitně ukládá soubory do složky `C:\Users\Uzivatel\Documents\MATLAB`, kde položka uživatel odpovídá uživatelskému účtu v systému Windows. Pokud máme správně nastavenou cestu v *Current Directory*, měl by se námi vytvořený soubor ihned objevit v jeho okně. Pro jeho opětovné spuštění stačí pouze dvakrát poklepat levým tlačítkem myši na název souboru.



Obr. 5.2: Soubory vytvořené v textovém editoru

5.3 M-soubory

Výsledky naší práce v M-file editoru jsou ukládány do souborů s příponou **soubor.m** (m-files). MATLAB využívá dva základní typy souborů a to skripty a funkce.

Skripty jsou m-soubory, které slouží k zápisu posloupnosti příkazů bez změn parametrů. Jedná se tedy o algoritmus, u kterého nedochází ke změně vstupních proměnných. Například vykreslení grafu funkce sin.

Funkce se uplatňuje všude tam, kde potřebujeme opakovaně provádět výpočet s různými vstupními hodnotami. Například výpočet kvadratické rovnice, kde postup výpočtu je stále stejný, ale vstupní hodnoty se neustále mění.

Mezi funkcemi a skripty je několik zásadních rozdílů, které si probereme podrobněji v následující kapitole.

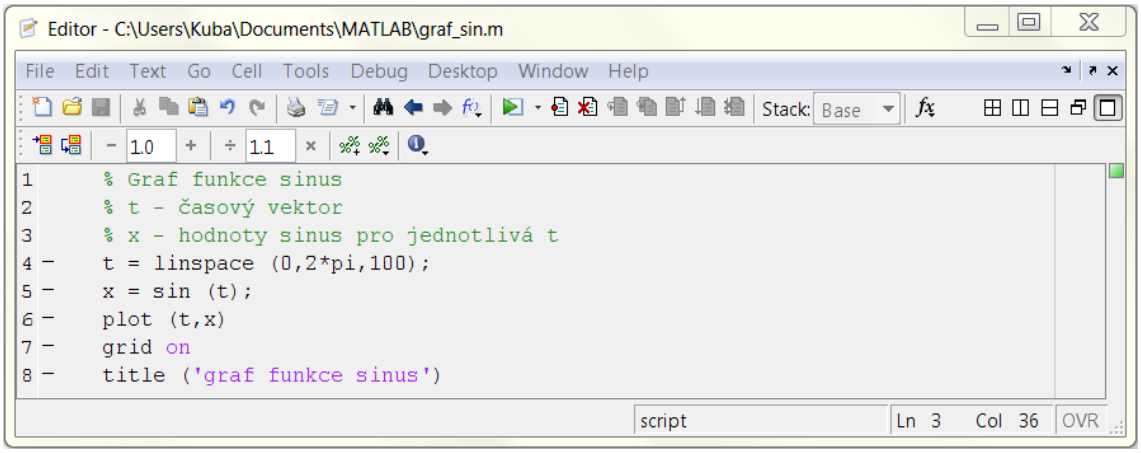
6 Skripty a funkce

Skripty a funkce jsou soubory, ve kterých se jednotlivé příkazy provedou až po jejich spuštění. Tento postup je velmi výhodný zejména při programování grafických aplikací. Jednotlivé příkazy se provádějí ve sledu, v jakém jsou napsány a program tak může běžet po delší dobu, dokud jej uživatel sám neukončí. Zároveň může vytvářet grafy, provádět odezvu na stisk tlačítka, čekat na akci uživatele, či zobrazovat výsledky výpočtů. Můžeme tak naplno využít veškeré možnosti MATLABu k vytvoření plnohodnotné aplikace.

6.1 Tvorba skriptů

Skript je tedy posloupnost příkazů uložených do souboru. Může vytvářet nové proměnné nebo mazat či měnit vybrané. Skripty tedy vytvářejí a pracují s globálními proměnnými. Tyto proměnné jsou na rozdíl od funkcí dostupné všem ostatním m-souborů a po ukončení skriptu zůstávají v okně *Workspace*. Neobsahují také žádné vstupní ani výstupní hodnoty.

Skripty samozřejmě mohou volat jiné skripty nebo funkce, vytvářet grafická okna či vypisovat do *Command Window*. Zde máme ukázku jednoduchého skriptu pro vykreslení grafu funkce sinus. Nejprve vytvoříme vektor v rozsahu od `0` do `2*pi` o sto prvcích. Poté pro jednotlivé prvky spočítáme hodnoty sinu a vykreslíme graf.

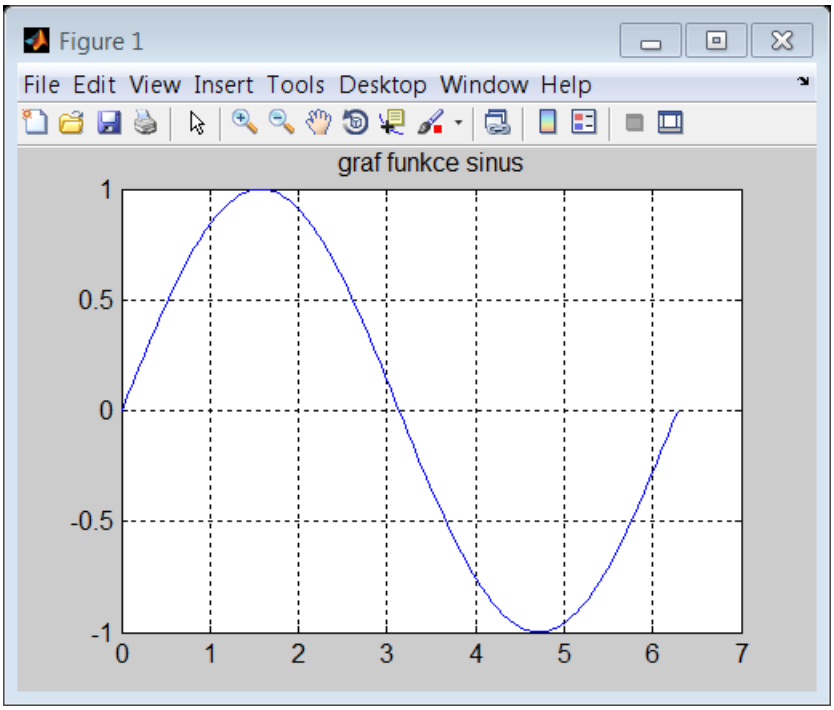


Obr. 6.1: Vytvoření skriptu

Skript můžeme spustit z okna *Command Window* zadáním jeho názvu bez přípony. Například soubor **graf_sin.m** spustíme voláním `graf_sin`

Abychom při tvorbě skriptu nemuseli vždy, když budeme chtít výsledek naší práce vyzkoušet, přepínat do hlavního okna MATLABu, můžeme skript spustit přímo v M-File editoru voláním položek *Debug→Run* nebo stiskem klávesy (*F5*). Popřípadě pomocí ikony run: v menu editoru.

Po proběhnutí našeho skriptu se vykreslí následující graf funkce sinus:



Obr. 6.2: Graf funkce sinus

6.2 Komentáře

Při tvorbě rozsáhlejších zdrojových kódů je vhodné doplnit text komentáři. Jsou to poznámky, které si programátor píše ke zdrojovému kódu, aby byl přehlednější. Jsou také velice užitečné, pokud program otevřeme po nějaké době nebo pokud pracujeme v týmu a námi vytvořený zdrojový kód používá jiný programátor.

Komentáře jsou v textovém editoru uvozeny znakem `%` a text, který se nachází za tímto znakem, nebude MATLABem akceptován. Jednotlivé skripty a funkce je vhodné opatřovat hlavičkou. Jako například u obrázku 6.1. Pokud do okna *Command Window* zapíšeme příkaz `help název_souboru` a potvrdíme ENTER, dojde k výpisu této hlavičky, což urychlí procházení jednotlivých souborů. Výpis hlavičky obrázku 6.1 by vypadal následovně:

```
>> help graf_sin
  Graf funkce sinus
  t - časový vektor
  x - hodnoty sinus pro jednotlivá t
```

Obr. 6.3: Výpis hlavičky skriptu či funkce

6.3 Často využívané příkazy ve skriptech

MATLAB obsahuje několik užitečných příkazů, které můžeme využít při psaní skriptů a funkcí. Některé z nich si nyní ukážeme.

6.3.1 Příkaz disp

Slouží k výpisu hodnoty proměnné bez jejího názvu. Pro ilustraci jsme zvolili příklad násobení matic.

```
% ukázka příkazu disp
% Výpis poměnné C bez jejího názvu
A = [1,2;1,-3;3,-2];
B = [2,-1,2;4,2,-1];
C = A*B;
disp (C)
```

Obr. 6.4: Výpis proměnné pomocí příkazu disp

6.3.2 Příkaz echo

Slouží k výpisu zdrojového kódu z textového editoru v okně *Command Window*. Pokud není příkaz `echo` aktivní (`echo off`), je výpis zdrojového kódu potlačen. Vypisují se pouze proměnné. Pokud však skript doplníme příkazem `echo on`, dojde k plnému výpisu obsahu textového editoru za tímto příkazem.

```
% Coulombuv zákon
% Q1,Q2 - naboje
% r - vzdálenost mezi náboji
echo on;
Q1=5e-6;
Q2=2e-9;
r=6e-9;
Eps0=8.85e-12;
E=1/(4*pi*Eps0)*(Q1*Q2)/r^2;
echo off;
fprintf ('\n');
fprintf('F = %6.2f\n',E )
```

Obr. 6.5: Výpis zdrojového kódu pomocí příkazu echo

Výpis skriptu v okně *Command Window* pak vypadá následovně:

```
Q1=5e-6;
Q2=2e-9;
r=6e-9;
Eps0=8.85e-12;
E=1/(4*pi*Eps0)*(Q1*Q2)/r^2;
echo off;

F = 2497723526238.16
```

Obr. 6.6: Výsledek výpisu zdrojového kódu v okně Command Window

6.3.3 Příkaz input

Příkaz input slouží pro zadání vstupního parametru uživatelem. V následujícím skriptu je příkaz použit pro zadání dvou odvěsen trojúhelníku.

```
% skript pro vypocet prepony trojuhelniku
odv1 = input ('Zadej 1. odvesnu:');
odv2 = input ('Zadej 2. odvesnu:');
Prep = sqrt(odv1^2 + odv2^2);
disp (Prep)
```

Obr. 6.7: Použití příkazu input

Po spuštění programu začne program vykonávat jednotlivé příkazy. Ve chvíli, kdy narazí na příkaz `input`, vyčká na zadání hodnoty uživatelem. Jakmile jsou zadány obě hodnoty, dojde k vypočtení výsledné hodnoty a jejímu zobrazení.

6.3.4 Příkaz pause

Slouží k přerušení běhu programu. Pokud příkaz zadáme bez dalších parametrů, je k odblokováni programu třeba stisk libovolné klávesy. Příkaz lze také zadat ve tvaru `pause(cas_v_sekundach)`. Parametr potom určuje dobu čekání programu.

6.4 Funkce

Pokud chceme využít stejný postup pro více situací, jsou skripty nevyhovující. Řešení nám nabízí funkce. Funkce, stejně jako skripty, v MATLABu vytváříme pomocí vestavěného editoru. Pro funkci jej můžeme spustit pomocí příkazu *File→ New → Function M-File*. Po jeho spuštění se nám automaticky vygeneruje následující zdrojový kód.

```
function [ output_args ] = Untitled( input_args )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here


end
```

Obr. 6.8: Automaticky generovaný zdrojový kód po spuštění funkce

Funkce totiž na rozdíl od skriptu mohou obsahovat uživatelem definovaný počet vstupních a výstupních parametrů. Rozdíl je i v dostupnosti proměnných. U skriptů jsou všechny proměnné globální, u funkce jsou proměnné dostupné pouze pro danou funkci. Existují jen po dobu spuštění funkce.

Každá funkce je uvozena klíčovým slovem `function`, které je povinné. Kromě klíčového slova `function` zde uživatel dále definuje:

- Počet výstupních proměnných [output_args]
- Jméno funkce
- Počet vstupních proměnných (input_args)

Výstupní i vstupní proměnné jsou zde nepovinné. Uživatel totiž může vytvořit funkci, která nebude vracet žádnou výstupní hodnotu nebo nebude potřebovat žádnou hodnotu vstupní. Při definici funkcí se musíme řídit následujícími pravidly:

- výstupní parametry (input_args):
  - je-li jich víc, oddělují se čárkou
  - je-li jen jeden, hranaté závorky nejsou nutné
  - funkce nemusí mít žádný výstup
- jméno funkce:
  - název funkce by měl vystihovat její činnost a musí být stejný jako název ukládaného m-souboru.
  - musí splnit pravidla pro název souboru (kapitola 5.2), jinak se funkci nepodaří spustit
  - název funkce se nesmí shodovat s žádným názvem její proměnné
- vstupní parametry [output_args]
  - je-li jich víc, oddělují se čárkou

- funkce nemusí mít žádný vstup (pak se podobá skriptu, ale má své lokální proměnné (*workspace*)

Zde je ukázka jednoduché funkce pro výpočet hustoty látky. Pokud budeme chtít funkci spustit z hlavního okna MATLABu, stačí pouze zadat název funkce a odpovídající počet vstupních parametrů. V našem případě jsme například zadali `Hustota (135, 10)`. První hodnota odpovídá hmotnosti látky, druhá objemu.

```
function [rho] = Hustota(m,V)
% Výpočet hustoty látky
% m - hmotnost
% V - objem
  if nargin ~=2
     error('Chyba: Potřebuji dva vstupy')
  end
rho = m/V;
end
```

Obr. 6. 9: Příklad definice funkce

```
>> Hustota (135,10)

ans =

   13.5000
```

Obr. 6. 10: Výsledek funkce hustota

6.4.1 Příklady definice funkce

Počet vstupních a výstupních parametrů funkce je možné měnit. Můžeme vytvářet funkce s proměnným počtem vstupních a výstupních parametrů, či některý z parametrů úplně potlačit. Uvedeme si zde příklady několika definic funkce:

- `Fce0paramet` – funkce bez výstupních a vstupních parametrů
- `Fce2vstupy (vstup1,vstup2)` – funkce se dvěma vstupními parametry
- `FceNvstupu (varargin)` – funkce s proměnným počtem vstupních parametrů
- `vystup=Fce1vystup` – funkce s jedním výstupním parametrem
- `[vystup1,vystup2]=Fce2výstupy` – funkce se dvěma výstupními parametry
- `Varargout=FceNvystupu`– funkce s proměnným počtem výstupních parametrů
- `[vystup1,vystup2]=Fce2vstup_vystup (vstup1,vstup2)` – funkce se dvěma vstupními a výstupními parametry

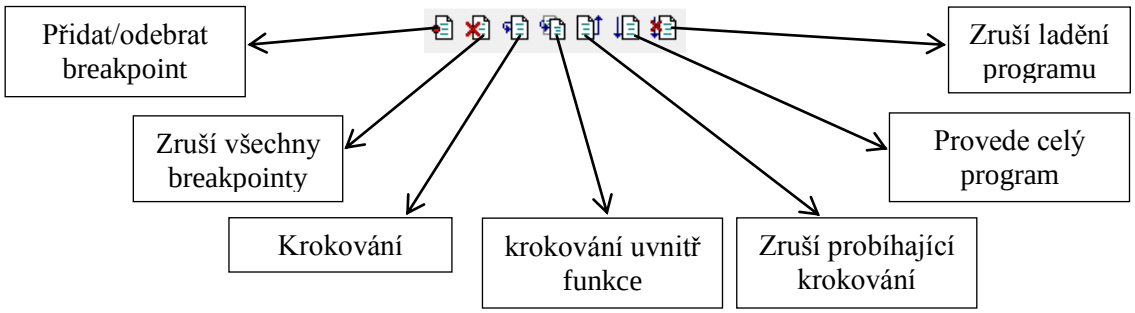
6.5 Ladění funkcí a skriptů

Při programování skriptů a funkcí se nám může stát, že program nepracuje správně. Kromě syntaktických chyb, na které nás program sám upozorní, mohou nastat i chyby jiné, které nemusí být na první pohled zřejmé. V takovém případě přichází na řadu ladění zdrojového kódu. Využíváme k tomu nástroj, který se označuje jako *Debugger*.

Řada programátorů také vytváří „nahrubo" připravený zdrojový kód, který nejprve odladí a teprve posléze ho rozšiřuje.

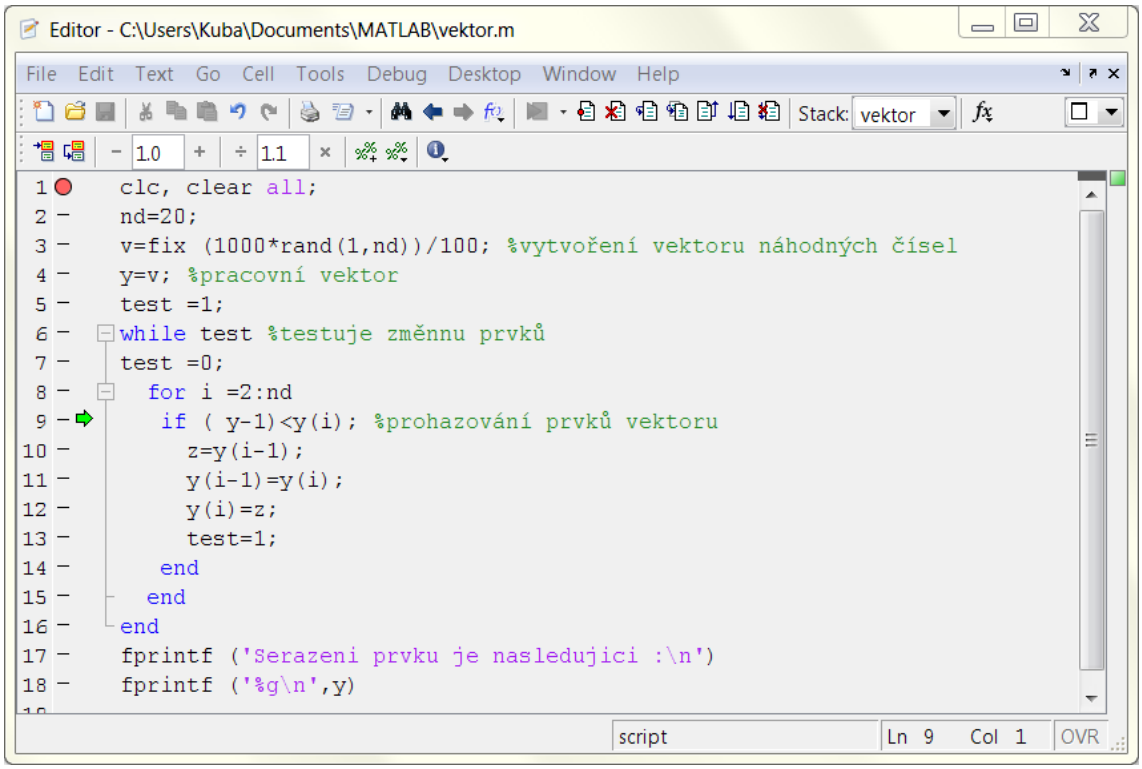
Chceme-li začít s laděním zdrojového kódu, je třeba na některý z řádků umístit bod přerušení, tzv. *breakpoint*. Tento bod je ve zdrojovém kódu označen červenou tečkou (viz obr. 6.12) a slouží k označení místa, odkud bude program postupovat po krocích. Tedy před každým dalším příkazem bude vyžadovat zásah uživatele (ikona krokování, nebo stisk klávesy F10).

Jakmile máme umístěný *breakpoint*, můžeme program spustit. Ikony, které nebyly před tímto krokem aktivní, nyní můžeme využít k ladění programu.



Obr. 6.11: Nástroje pro ladění zdrojového kódu

Po startu programu se objeví malá zelená šipka (obr. 6.12), která označuje místo programu, ve kterém se právě nacházíme. Skutečnost, že MATLAB přešel do režimu ladění programu, vidíme i v okně *Command Window*, kde se MATLAB hlásí velkým písmenem **K**. Nyní můžeme pomocí ikon z obrázku 6.11 zahájit vlastní ladění programu. V části programu, kde již proběhlo krokování, můžeme také zjistit aktuální hodnoty proměnných. Stačí pouze najet myší na proměnnou, jejíž hodnotu chceme zjistit. Po chvilce se objeví žlutý rámeček s informacemi o názvu proměnné a jejím obsahu.



Obr. 6.12: Ladění zdrojového kódu programu

7 Řídící struktury

Řídící struktury můžeme rozdělit do dvou skupin a to na podmínky a cykly. Podmínky využijeme, pokud například chceme, aby se určitý příkaz či skupina příkazů provedla jen za určitých okolností, tedy při splnění podmínky. Naproti tomu cykly slouží k opakovanému provádění příkazu nebo skupiny příkazů.

7.1 Příkaz if

Příkaz slouží k větvení programu. Pokud chceme například, aby program vykonal nějaké příkazy pouze při splnění určité logické podmínky, využijeme právě příkaz `if`. Základní struktura příkazu vypadá takto:

```
if vyraz
    prikazy
end
```

Obr. 7.1: Konstrukce podmínky if

Pokud probíhající program narazí na příkaz `if`, nejprve vyhodnotí podmínku `vyraz`. Pokud je podmínka splněna, program pokračuje ve vykonávání příkazů umístěných v těle podmínky. Pokud podmínka není splněna, hlavní program pokračuje ve vykonávání příkazů za výrazem `end`. V následující ukázce uživatel nejprve zadá číslo a pokud je podmínka splněna, vypíše se hlášení, že zadané číslo je kladné.

```
% Větvení programu - podmínka if
a = input ('Zadej číslo:');
if a >= 0
    disp ('Zadané číslo je kladné')
end
```

Obr. 7.2: Příklad větvení if

Podmínku větvení `if` můžeme ještě rozšířit o výraz `else`. V případě použití této konstrukce má příkaz následující podobu:

```
if vyraz
    prikazy
else
    prikazy
end
```

Obr. 7.3: Konstrukce podmínky if – else

Program opět nejprve vyhodnotí podmínku, a pokud je podmínka splněna, provede příkazy v těle podmínky `if`. V případě, že podmínka splněna není, provedou se příkazy v části `else`. Předchozí ukázku jsme nyní rozšířili o tuto část. Pokud tedy uživatel zadá číslo, které bude větší nebo rovno 0, vypíše se hlášení, že zadané číslo je kladné. V opačném případě se provede příkaz v části `else`, tedy výpis: Zadané číslo je záporné.

```
% Větvení programu if - else
a = input ('Zadej číslo: ');
if a >= 0
    disp ('Zadané číslo je kladné')
else
    disp ('Zadané číslo je záporné')
end
```

Obr. 7.4:Příklad větvení if–else

Každý příkaz větvení může obsahovat pouze jedno `if` a jedno `else`. Tento stav nemusí být vždy dostačující, proto je MATLAB vybaven konstrukcí, která zajistí rozšíření příkazu `if` o další podmínku.

```
if vyraz1
    prikazy
elseif vyraz2
    prikazy
else
    prikazy
end
```

Obr.7.5: Konstrukce podmínky if – elseif – else

Program nejprve vyhodnotí první podmínku, a pokud je podmínka splněna, provede příkazy v těle podmínky `if`. Pokud splněna není, program otestuje `vyraz2` a v případě jeho splnění provede příkazy obsažené v těle podmínky `elseif`. Pokud není ani tato podmínka splněna, provedou se příkazy obsažené v části `else`. Předchozí příklad nyní rozšíříme o tuto podmínku.

```
% Větvení programu if - elseif - else
a = input ('Zadej číslo: ');
if a>0
    disp ('Zadané číslo je kladné')
elseif a==0
    disp ('Zadané číslo je 0')
else
    disp ('Zadané číslo je záporné')
end
```

Obr.7.6: Příklad větvení if – elseif – else

Uživatel opět zadá číslo, které program vyhodnotí a podle podmínky, která bude zadanému výrazu vyhovovat, vypíše hlášení, zda je číslo kladné, záporné nebo rovno 0.

Při vytváření řídících příkazů je možné využívat několik logických operátorů, které se mohou při sestavování podmínek různě spojovat. Pokud bychom chtěli například zapsat, že číslo `a` musí být větší než 0 a současně menší nebo rovno 10, vypadala by podmínka následovně: `a > 0 && a <= 10`. Přehled logických výrazů, které je možné pří vytváření podmínek využívat, je uvedený v následující tabulce:

>	Je větší
<	Je menší
>=	Je větší nebo rovno
<=	Je menší nebo rovno
==	Je rovno (= je pouze přiřazení)
~=	Nerovná se
&&	Logický součin (musí být splněny obě podmínky)
||	Logický součet (musí být splněna alespoň jedna podmínky)

Tab. 7.1: Přehled nejpoužívanějších logických výrazů

Jednotlivé podmínky je do sebe možné také vnořit. Využití vnořeného příkazu si ukážeme na příkladu kvadratické rovnice. Po zadání vstupních koeficientů program nejprve vyhodnotí první podmínku. Pokud se koeficient a ~=0, dojde k výpočtu dvou kořenů rovnice. Jestliže je podmínka prvního větvení splněna, tedy a == 0, pokračuje program vyhodnocením podmínky vnořeného cyklu. Pokud je podmínka splněna, program vypíše hlášení, že rovnice nemá řešení. Při jejím nesplnění se vypočte sdružený kořen kvadratické rovnice.

```
function Kvadraticka_rce (a,b,c)
%Funkce pro výpočet kvadratické rovnice
% a,b,c - koeficienty
% D - diskriminant
% x,x1,x2 - kořeny rovnice
a = input ('Zadej a = ');
b = input ('Zadej b = ');
c = input ('Zadej c = ');
if a==0
    if b==0
        fprintf ('Rovnice nemá řešení')
    else
        x=-c/b;
        fprintf ('Rovnice má jeden kořen:\n x= %1.2f\n',x)
    end
else
  D = b^2 - 4*a*c;
  x1=(-b+sqrt(D))/(2*a);
  x2=(b+sqrt(D))/(2*a);
  fprintf ('Rovnice má dva kořeny:\n')
  fprintf ('x1= %1.2f\n',x1)
  fprintf ('x2= %1.2f\n',x2)
end
```

Obr. 7.7: Kvadratická rovnice

7.2 Příkaz switch – case

Tento druh větvení programu používáme, pokud chceme jeden výraz porovnat s několika hodnotami. Program opět nejprve vyhodnotí podmínku, poté najde odpovídající část `case`, pro kterou je tato podmínka splněna a provede příkazy obsažené v této části větvení. V případě, že podmínka není splněna ani pro jednu část `case`, provedou se příkazy za klíčovým slovem `otherwise`.

```
switch vyraz
    case podminka_1
        prikazy
    case podminka_2
        prikazy
    case podminka_3
        prikazy
    otherwise
        prikazy
end
```

Obr. 7.8: Konstrukce příkazu switch - case

Pro ukázku práce s příkazem switch – case můžeme využít následující příklad, který slouží ke změně řádků matice.

```
M = [2, 7, -4; 8, -1, 3; 3, -5, -2];
r1 = input ('Řádek pro přehození:');
r2 = input ('Řádek pro přehození:');
pom=1;

switch pom
    case (r1==1 && r2==2) || (r1==2 && r2==1)
        M ([1,2],:)= M ([2,1],:);
        disp (M)
    case (r1==1 && r2==3) || (r1==3 && r2==1)
        M ([1,3],:)= M ([3,1],:);
        disp (M)
    case (r1==2 && r2==3) || (r1==3 && r2==2)
        M ([2,3],:)= M ([3,2],:);
        disp (M)
    otherwise
        disp ('Nesprávná vstupní hodnota')
end
```

Obr. 7.9: Ukázka větvení switch - case

7.3 Cyklus for

V programu je někdy třeba určitou část zdrojového kódu provést opakovaně. Pro tento účel je MATLAB vybaven cykly, které provádějí opakování části zdrojového kódu, dokud je daná podmínka splněna. Cyklus for používáme, pokud předem známe počet opakování. Jeho konstrukce je následující:

```
for n = vektor
    prikazy
end
```

Obr. 7.10: Konstrukce cyklu for

Příkaz následující po `for n = vektor` se provádí pro každý prvek vektoru. Hodnota `n` se tedy s každým během smyčky zvětšuje o 1, dokud nedojde k horní hranici. Poté je příkaz for ukončen a program pokračuje za blokem `for`. Tedy za klíčovým slovem `end`.

```
function [fak]=faktorial(n)
% Vypocet faktorialu (n!)
 % fak ... výsledná hodnota
% n ... pocet
fak=1;
for i=1:n
   fak=fak*i;
end
```

Obr. 7.11: Výpočet faktoriálu pomocí cyklu for

Jako příklad nám může sloužit funkce pro výpočet faktoriálu, kde nejprve uživatel zadá číslo, čímž určí počet opakování cyklu `for`. V každém běhu cyklu se pak hodnota `i` zvýší o jedna a hodnota `fak` je nahrazena hodnotou `fak*i`.

7.4 Cyklus while

Tento cyklus nemá předem určeno, kolikrát se část programu obsažená v těle cyklu bude opakovat. Počet opakování je závislý na vstupní podmínce cyklu a trvá tak dlouho, dokud je podmínka splněna. Konstrukce cyklu vypadá následovně.:

```
while vyraz
    prikazy
end
```

Obr. 7.12: Konstrukce cyklu while

Program tedy nejprve otestuje podmínku `vyraz`, a pokud je tato podmínka splněna, pokračuje ve vykonávání příkazů v těle cyklu. Jestliže splněna není, příkazy obsažené v těle cyklu se neprovedou a program pokračuje za klíčovým slovem `end`. Následující program slouží k součtu `n` přirozených čísel.

```
function [s]=soucet_prirozenych_cisel(cislo)
% Soucet prvnich n prirozenych cisel
% s ... výsledek součtu
% cislo ... zadaný počet čísel
i=1;
s=0;
while i<=cislo
   s=s+i;
   i=i+1;
end
```

Obr. 7.13:Příklad užití cyklu while

Uživatel nejprve zadá počet čísel číselné řady, které budeme sčítat. Nastavíme počáteční hodnoty. V každém kroku cyklu dojde k přičtení hodnoty `s` a iterační hodnoty `i`. Tento postup se opakuje do té doby, dokud hodnota `i <= číslo`. Tedy dokud není podmínka splněna.

8 Integrály a derivace

Integrály a derivace nám pomáhají matematicky popsat mnoho dějů se kterými se člověk potkává v běžném životě. Určitý integrál je možné interpretovat jako plochu pod křivkou, derivaci jako směrnici ke křivce v určitém bodě. Systém MATLAB umožňuje výpočet integrálů a derivací spojité funkce. Ta může být zadána například výrazem nebo tabulkou.

8.1 Výpočet určitého integrálu

Pro výpočet integrálu MATLAB využívá metod tzv. numerické integrace, kdy se snažíme nahradit integrál jiným druhem výpočtu, při kterém se snažíme zajistit, aby se získaná hodnota od skutečné hodnoty integrálu lišila co nejméně.

Pro výpočet určitého integrálu disponuje MATLAB funkcemi `quad` a `quadl`. Tyto dvě funkce využívají pro výpočet plochy pod křivkou adaptivní Simpsonovo pravidlo. Podle tohoto pravidla je křivka rozdělena na intervaly a výsledná hodnota je určena jako součet ploch pod parabolou jednotlivých vzorků.

Výpočet určitého integrálu si můžeme ukázat na přikladu funkce:

$$\int_{-2}^{3} (x+3)^2 \,\mathrm{d}\,x \tag{1}$$

Ještě před použitím příkazu `quad` si vytvoříme funkci, která bude obsahovat námi integrovaný výraz. Funkce bude mít jeden vstupní a jeden výstupní parametr a pojmenujeme si ji například `int_x_plus3`.

Celou funkci uložíme do zvláštního m-souboru:

```
function [y] = int_x_plus3 (x)
y=(x+3).^2;
```

Obr. 8.1: Vytvoření funkce pro integraci

Nyní již můžeme přistoupit k samotné integraci. Výpočet provedeme pomocí příkazu `quad (@int_x_plus3, -2, 3)`. Z příkazu můžeme vidět, že první parametr je odkaz na námi vytvořenou funkci, další dva jsou dolní a horní mez integrace.

```
>> quad (@int_x_plus3, -2, 3)

ans =

   71.6667
```

Obr. 8.2: Integrál funkce $(x+3)^2$

Funkci je možné také definovat jako anonymní. Výhodou tohoto zápisu je, že vše je obsaženo v okně *Command Window* či skriptu a není třeba vytvářet zvláštní m-soubor.

```
>> x_plus3=@(x) (x+3).^2; %Anonymní funkce
>> quad (x_plus3, -2, 3) %Výpočet integrálu

ans =

   71.6667
```

Obr. 8.3: Anonymní funkce

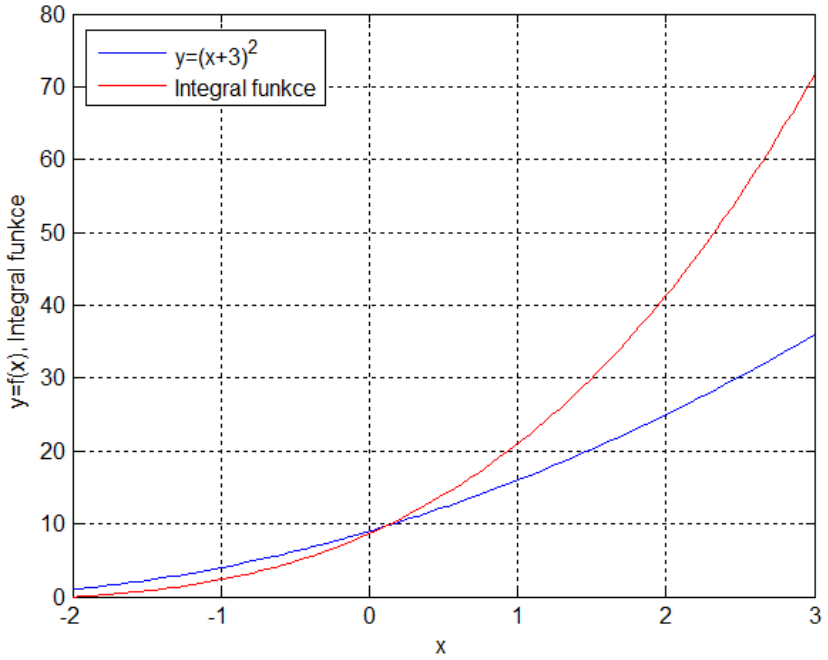
Nyní máme určenou hodnotu integrálu na intervalu od -2 do 3. Pokud bychom chtěli průběh integrace zobrazit graficky, můžeme postupovat například takto:

```
x_na3=@(x) (x+3).^2; %anonymni funkce
quad (x_na3, -2, 3) %urcity integral od -2 do 3

x=linspace(-2, 3, 100); %vektor funkce
y=x_na3(x);


%Vypocet urciteho integralu
y_itegrace=zeros (length(x),1);
for i=1:1:length(x)
    y_integrace(i) = quad(x_na3, -2, x(i));
end

%Graficky vystup
plot (x,y,'b',x,y_integrace,'r')
xlabel ('x')
ylabel ('y=f(x), Integral funkce')
legend ('(y=x+3)^2','Integral funkce')
grid on
```

Obr. 8.4: Grafické znázornění integrace

Určitý integrál představuje plochu pod modrou křivkou. Červená křivka zobrazuje hodnotu určitého integrálu v mezích od -2 do 3.

Funkce `quadl` má stejnou syntaxi zápisu jako funkce `quad`. Lze jej využít u výpočtů, kde jsou vysoké nároky na přesnost integrace nebo tam, kde integrace s `quad` selže.



Obr. 8.5: Průběh integrované funkce a jejího integrálu

8.2 Výpočet dvojného a trojného integrálu

Pro výpočet dvojného a trojného integrálu MATLAB disponuje funkcemi `dblquad` a `triplequad`. Výpočet si opět ukážeme na příkladu:

$$\int_0^3 \int_1^2 \left(x^2 \cdot y\right) \mathrm{d}\, x \,\mathrm{d}\, y \tag{2}$$

Postupovat můžeme stejně jako u výpočtu jednoduchého integrálu. Tedy nejprve si vytvořit do samostatného m-souboru funkci, kterou posléze integrujeme, nebo využít funkci anonymní. Protože první z postupů je analogický jako v předchozím případě, ukážeme si zde výpočet pouze s využitím funkce anonymní.

```
>> int_x2y=@(x,y) x.^2*y; %Anonymní funkce
>> dblquad (int_x2y, 0, 3, 1, 2) %Výpočet dvojného integrálu

ans =

   13.5000
```

Obr. 8.6: Průběh integrované funkce a jejího integrálu

První parametr příkazu `dblquad` je opět odkaz na anonymní funkci, další parametry jsou potom horní a dolní meze jednotlivých integrálů.

Dvojný integrál nám nejčastěji slouží pro výpočty objemu pod plochou. Pokud bychom chtěli určovat například hmotnost nějakého tělesa, je třeba využít integrál trojný. Postup je opět analogický jako u funkce `dblquad`. Vzroste pouze počet parametrů příkazu `triplequad` a anonymní funkce se rozšíří o jednu proměnnou.

8.3 Výpočet derivace funkce

Na rozdíl od určitého integrálu, který je vyjádřen jako plocha pod křivkou na určitém intervalu, tedy množství, je derivace spojena s bodem na křivce. Představuje směrnici tečny v tomto bodě. Numerický výpočet hodnoty derivace v bodě je tedy mnohem komplikovanější.

Derivace funkce je definována vztahem:

$$\frac{\mathrm{d}\,y}{\mathrm{d}\,x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{3}$$

MATLAB umožňuje výpočet numerické derivace, proto je předchozí výraz třeba aproximovat vztahem:

$$\frac{\mathrm{d}\,y}{\mathrm{d}\,x} \approx \frac{\Delta y}{\Delta x} = \frac{f(x+h) - f(x)}{h} \tag{4}$$

Máme-li zadaný vektor prvků, můžeme vypočítat diference mezi jednotlivými prvky. Za tímto účelem je zde k dispozici funkce `diff`.

```
>> v=[5, 1, -8, 2, 3, -1]

v =

     5     1    -8     2     3    -1

>> diff(v)%diference mezi prvky vektoru

ans =

    -4    -9    10     1    -4
```

Obr. 8.7: Příkaz diff

Výsledný vektor je o jeden prvek menší než původní vektor, protože funkce počítá rozdíl mezi jednotlivými prvky.

V následujícím případě si ukážeme výpočet derivace funkce zadané polynomem čtvrtého řádu.

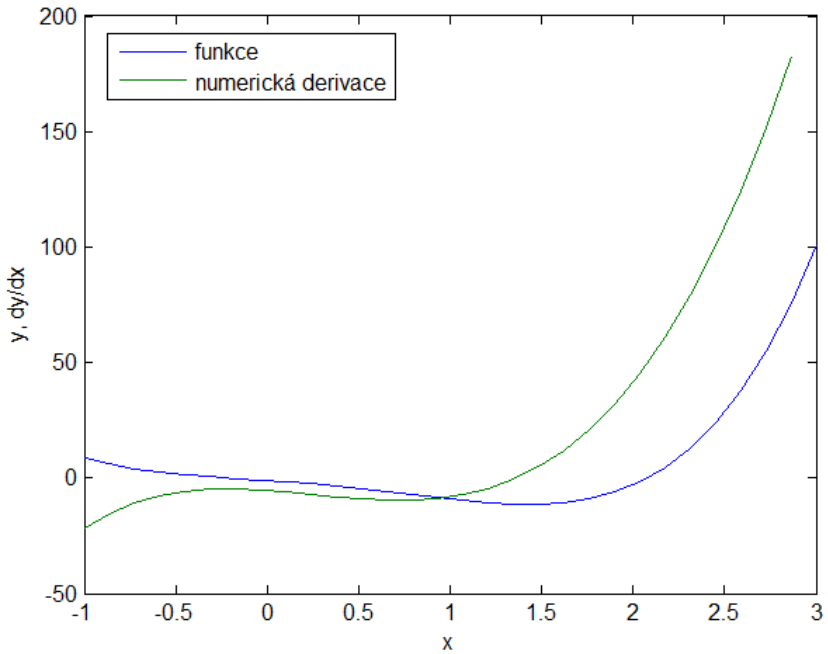
$$y = 3x^4 - 4x^3 - 2x^2 - 5x - 1 \tag{5}$$

Po zadání polynomu si vytvoříme vektor x, pro který si na intervalu od -1 do 3 vypočteme v třiceti bodech hodnoty polynomu. Dále pomocí výrazu $\mathrm{d}\,p = \Delta p / \Delta x$ vypočteme derivaci funkce pomocí příkazu `diff`.

```
%polynom
y=[3, -4, -2, -5, -1];
%interval a hodnota polynomu na tomto intervalu
x=linspace (-1, 3, 30);
p=polyval(y,x);
%numerický výpočet derivace
dp=diff(p)./diff(x);
%grafický výstup
plot (x,p, x(1:length(x)-1),dp)
xlabel('x')
ylabel('y, dy/dx')
legend('funkce','numerická derivace')
```

Obr. 8.8: Výpočet derivace funkce

Na obrázku 8.9 můžeme vidět grafické znázornění funkce a její derivace na intervalu od -1 do 3.



Obr. 8.9: Graf funkce a její derivace

9 Diferenciální rovnice

Pomocí diferenciálních rovnic lze popsat mnoho problémů reálného světa. Jedná se o matematické rovnice, kde jako proměnné vystupují derivace funkcí. Reálné problémy často vedou na nelineární diferenciální rovnice, případně jejich soustavy, které je nutné řešit numericky. V MATLABu je možné řešit soustavy diferenciálních rovnic prvního řádu se známými počátečními podmínkami. Tyto rovnice mají tvar:

$$\frac{\mathrm{d}\,x}{\mathrm{d}\,t} = f(x,t), x(t_0) = x_0 \tag{1}$$

9.1 Obyčejné diferenciální rovnice prvního řádu

Z rovnice 9.1 je zřejmé, že obyčejné diferenciální rovnice obsahují jednu (nebo více derivací) jedné závislé proměnné x a nezávisle proměnnou t (nejčastěji čas). Pokud je diferenciální rovnice n-tého řádu, je třeba ji pro zpracování v MATLABu převést na soustavu n rovnic prvního řádu.

$$\begin{aligned} y &= f\left(t, x, x', x'', \ldots x^{(n-1)}\right) \\ y_1' &= y_2 \\ y_2' &= y_3 \\ &\vdots \\ y_n' &= f\left(t, x, x', x'', \ldots x_n\right) \end{aligned} \tag{2}$$

MATLAB disponuje několika řešiteli diferenciálních rovnic, které se liší použitou metodou integrace (viz. kapitola 9.2).

Jak jednoduše vyřešit diferenciální rovnici pomocí MATLABu si ukážeme na příkladu kmitání harmonického oscilátoru, který je tvořen závažím zavěšeným na pružině. Tuhost pružiny je *k=0,5* a hmotnost *m=1* kg. Závaží se po nárazu zdola začne pohybovat a v prvním okamžiku je jeho rychlost *v=1* ms^{-1}. Naším úkolem je popsat dráhu a rychlost pohybu závaží a vykreslit jejich časovou závislost.

Diferenciální rovnice popisující kmitání závaží na pružině má tvar:

$$m\frac{\mathrm{d}^2 s}{\mathrm{d}\,t^2} = -ks \tag{3}$$

Abychom tuto diferenciální rovnici mohli vyřešit pomocí MATLABu, je třeba ji převést na dvě rovnice prvního řádu. Rovnice si tedy upravíme do následujícího tvaru:

$$\frac{\mathrm{d}\,s}{\mathrm{d}t} = v \tag{4}$$

$$\frac{\mathrm{d}\,v}{\mathrm{d}\,t} = -\frac{k}{m}\cdot s \tag{5}$$

Nyní si spustíme prázdný m-file, ve kterém vytvoříme funkci vracející derivace funkcí v určitém čase. Tato funkce odpovídá funkci $f(t,x)$ v obecném tvaru diferenciální rovnice.

```
%Soustava diferenciálních rovnic popisujících pohyb
%ds/dt = v
%dv/dt = -k/m * s
function [dxdt] = dif_rce (t,x)
    dxdt=zeros(2,1);
    dxdt(1)=x(2);
    dxdt(2)=-0.5/1*x(1);
```

Obr. 9.1: Funkce pro výpočet derivace

Řešení diferenciální rovnice provedeme pomocí funkce `ode45` na intervalu 0 až 15 sekund. První parametr je odkaz na funkci `dif_rce`, druhý je vektor obsahující interval řešení `[0, t_konec]` a třetí je vektor s počátečními podmínkami `[s0, v0]`.

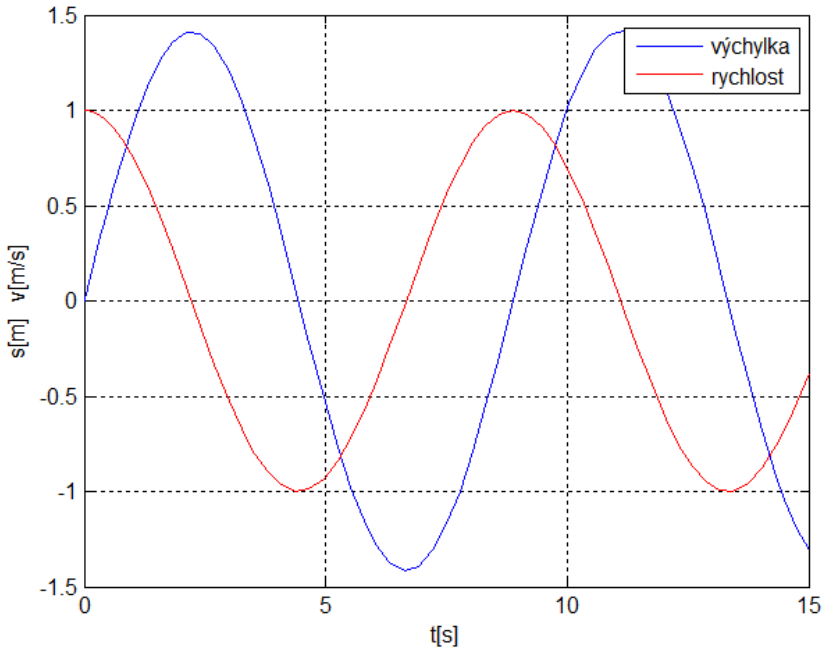
```
%řešení dif. rovnice
[t, x]= ode45(@dif_rce, [0, t_konec], [s0, v0]);
```

Obr. 9.2: řešení diferenciální rovnice

Výstupem funkce `ode45` je vektor, který obsahuje prvky, pro které bylo prováděno řešení a matice obsahuje sloupcové vektory s řešením. V našem případě vektor `t` obsahuje čas a matice `x` obsahuje polohy závaží a jeho rychlosti v odpovídajícím čase. Zdrojový kód ještě doplníme o grafický výstup a celé řešení znázorníme graficky.

```
%Pohyb závaží na pružině
%Parametry závaží
s0=0; %m
v0=1; %m/s

%cas reseni
t_konec=15;

%řešení dif. rovnice
[t, x]= ode45(@dif_rce, [0, t_konec], [s0, v0]);

%převod na fyzikální veličiny
s=x(:,1);
v=x(:,2);

%Grafický výstup
plot (t,s,'b',t,v,'r')
grid on
xlabel ('t[s]')
ylabel ('s[m]   v[m/s]')
legend ('výchylka','rychlost')
```

Obr. 9.3: Zpracování a znázornění výsledků



Obr. 9.3: Grafické znázornění výchylky a rychlosti oscilátoru

9.2 Řešitelé pro diferenciální rovnice

MATLAB obsahuje několik řešitelů diferenciálních rovnic, které se liší v přesnosti a použitých metodách integrace. Zde si uvedeme jejich stručný popis.

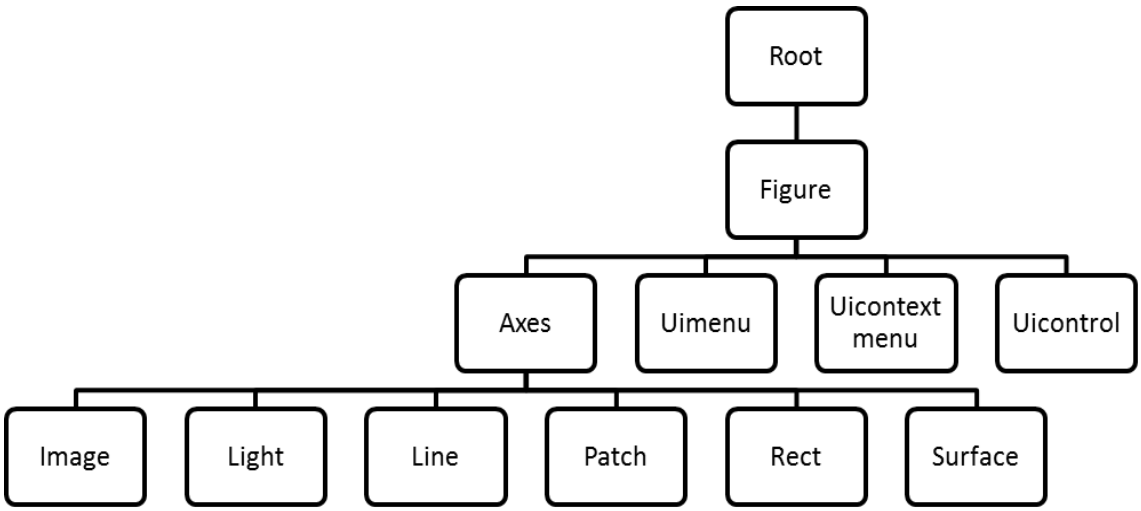
`Ode23`	Řešení založené na explicitní formulaci Runge-Kutteho metody (2. řád s odhadem chyby 3.řádu). Použitelný pro méně náročné stiff systémy, kde není požadována vysoká přesnost, případně pro funkce, které nejsou hladké, např. nespojité funkce
`Ode23s`	Implicitně jednokrokový Rosenbrockův řešič druhého řádu, použitelný pro méně náročné stiff systémy s nižší přesností řešení.
`Ode23t`	Implicitně jednokrokové lichoběžníkové pravidlo, vhodné pro středně náročné stiff systémy a pro kombinace soustav algebraicko diferenciálních rovnic
`Ode23tb`	Podobný jako ode23s. Může být efektivní pro nižší přesnosti
`Ode45`	Řešitel založený na explicitní formulaci Runge-Kutteho metody (4.řád s odhadem chyby 5. řádu). Vhodný pro většinu problémů. Typicky první řešitel pro nový problém.
`Ode113`	Vícekrokový řešitel Adams-Bashforth-Moulton s proměnným řádem (první až třináctý řád). Vhodný pro vysokou přesnost řešení, nelze použít pro nespojité funkce.
`Ode15s`	Implicitně vícekrokový řešitel proměnného řádu. Pokud řešitel ode45 selže nebo je neefektivní, měl by se vyzkoušet tento řešitel.

10 Handle Graphics

Součástí programu MATLAB je grafický systém nazvaný handle graphics, který je učen k efektivnímu využití grafických objektů. Uživatel může mnohem lépe pracovat s grafikou a využívat příkazy pro 2D a 3D vizualizaci dat, zpracování signálů, animace a grafiku obecně. Pochopením tohoto grafického systému umožní uživateli programovat uživatelské aplikace (GUI) a rozšířit znalosti o tvorbě grafů.

10.1 Hierarchie grafických objektů

Systém handle graphics disponuje celou řadou grafických objektů, které jsou hierarchicky uspořádány. Objekty tedy vytvářejí stromovou strukturu a jednotlivé objekty jsou vzájemně v podřízeném nebo nadřízeném vztahu. Kořen stromu hierarchického uspořádání představuje objekt *Root*, kterému jsou podřízeny všechny ostatní objekty.



Obr. 10.1: Hierarchické uspořádání grafických objektů programu MATLAB

Hierarchické uspořádání je velmi výhodné, pokud chceme současně změnit určitou vlastnost několika podřízených grafických objektů. Například změnit velikost textu několika tlačítek. Díky hierarchickému uspořádání můžeme využít tzv. dědičnost. Tedy schopnost podřízených grafických objektů přejímat vlastnosti od nadřazených objektů. V našem případě jsme chtěli měnit velikost textu u tlačítek, tedy u objektu *Uicontrol*. Na této úrovni bychom museli měnit velikost textu každého tlačítka zvlášť, což je neefektivní. Při využití dědičnosti stačí tuto vlastnost změnit pouze jednou u grafického objektu *Figure*.

10.2 Grafický objekt Figure

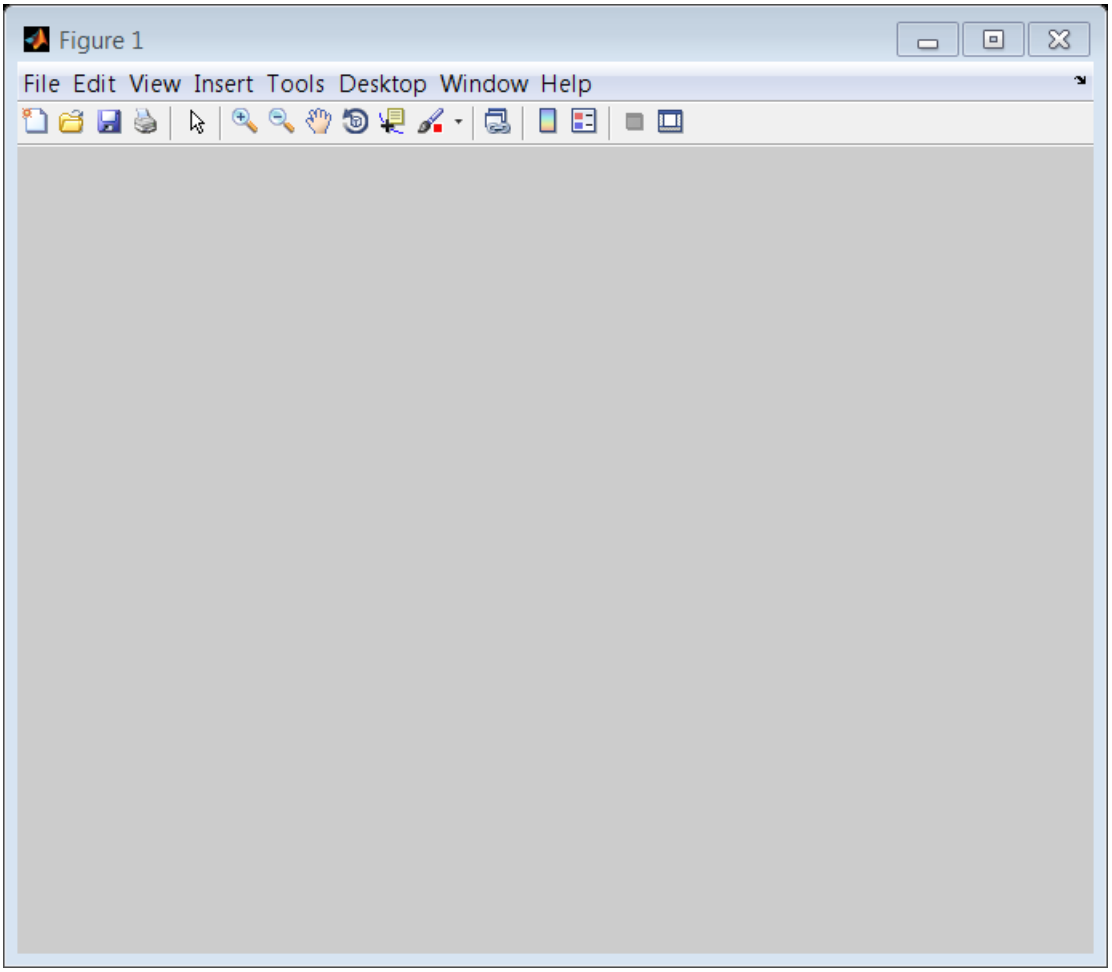
Jednotlivé příkazy pro grafické objekty budeme zapisovat do m-souborů, o jejichž výhodách jsme již mluvili, případně je možné příkazy zapisovat i do okna *Command Window*.

Pro vytvoření grafického objektu spustíme textový editor (příkaz `edit`), do kterého zapíšeme příkaz `h=figure`.

```
% Grafický objekt figure
h=figure;
```

Obr. 10. 2: Vytvoření grafického objektu figure

Pokud takto vytvořený skript uložíme a spustíme, výsledkem bude nové okno s názvem *Figure1*, do kterého je možné umísťovat další podřízené objekty (viz. hierarchie grafických objektů).



Obr. 10.3: Grafický objekt Figure

Proměnná `h` je tzv. handle, který jednoznačně určuje grafický objekt a umožňuje tak snadno měnit jeho parametry. Každý grafický objekt má toto handle jiné.

10.2.1 Příkazy set a get

Při vytvoření okna Figure má každé okno implicitně nastavené určité parametry, jako velikost, pozice na obrazovce, barva pozadí, název apod. Pro výpis aktuálního nastavení můžeme využít příkaz `get(h)`, který do okna *Command Window* vypíše seznam parametrů, které lze u tohoto objektu nastavit. Protože jsme využili příkaz `get(h)`, jednoznačně jsme určili, že půjde o vlastnosti grafického objektu *Figure*. Pokud nás zajímá pouze určitý parametr, například barva pozadí, stačí příkaz `get` zapsat pouze s tímto parametrem tedy: `get(h,'Color')`.

```
% Grafický objekt figure
h=figure;
get(h,'Color')
```

Obr. 10.4: Použití příkazu get

Výsledkem je vektor typu **1 x 3**, jehož prvky nabývají hodnot od 0 do 1 a každá složka vektoru odpovídá hodnotě barevných škál RGB.

```
ans =

    0.8000    0.8000    0.8000
```

Obr. 10. 5: Výsledek výpisu vlastnosti color pomocí příkazu get

Jednotlivé parametry grafického objektu je možné měnit pomocí příkazu `set`. V předchozím příkladu jsme si ukázali, jak zjistit barvu pozadí okna *Figure*. Tuto barvu nyní pomocí příkazu `set` změníme.

```
% Grafický objekt figure
h=figure;
set(h,'Color',[0.5 0.7 0.3])
get(h,'Color')
```

Obr. 10.6: Změna barvy pozadí pomocí příkazu set

Příkaz `get` jsme opětovně využili proto, aby bylo vidět, jak se změnil vektor barevných škál.

```
ans =

    0.5000    0.7000    0.3000
```

Obr. 10.7: Výpis vlastnosti color po aplikaci příkazu set

Příkazy `set` a `get` lze tedy využívat k zjištění aktuálního nastavení vlastností grafických objektů a také k jejich modifikaci tak, aby odpovídaly přání programátora.

10.2.2 Další nastavení vlastností objektu figure

Pokud jsme si pomocí příkazu `get` nechali vypsat všechny vlastnosti objektu *Figure*, zjistili jsme, že seznam je poměrně obsáhlý. My si nyní ukážeme možnosti nastavení nejpoužívanějších z nich.

10.2.3 Nastavení velikosti a umístění okna

Pro nastavení slouží vlastnost `position`. Nastavení je možné opět provést pomocí příkazu `set` a to následovně: `set(h,'position',[300 300 400 200])`. První dvě čísla určují pozici od levého dolního okraje obrazovky, třetí a čtvrté pak výšku a šířku objektu v bodech.

10.2.4 Nastavení jména

Jméno objektu je implicitně nastaveno na *Figure1*. Pokud chceme toto implicitní nastavení zrušit, stačí využít příkaz `set(h,'NumberTitle','off')`. Okno můžeme pojmenovat i vlastním názvem. My si naše grafické okno pojmenujeme například Test. Pro pojmenování využijeme opět příkaz `set`: `set(h,'Name','Test')`.

10.2.5 Skrytí řádky s menu

Pro ukrytí menu využijeme vlastnosti `Menubar`, kterou lze nastavit na hodnotu `figure` nebo `none`. Hodnota `figure` je nastavena implicitně. Pokud budeme chtít lištu s menu skrýt, stačí nastavit hodnotu na none. Program tedy doplníme o: `set(h,'Menubar','none').`

10.2.6 Viditelnost objektu

K nastavení viditelnosti slouží vlastnost `Visible`. Tato položka je implicitně nastavena do stavu zapnuto `on`. Pokud chceme některý z objektů skrýt například na určitou dobu, stačí tuto vlastnost přepnout do stavu `off`: `set(h,'Visible','off').`

10.3 Grafický objekt Uicontrol

Jak napovídá naše hierarchická struktura, jsou objekty *Uicontrol* podřízeny grafickému objektu *Figure*, jsou tedy jeho potomky. Jednotlivé objekty jsou také vytvářeny uvnitř objektu *Figure*. Objekty *Uicontrol* představují tlačítka, zaškrtávací pole, textová pole, pole pro editovatelné texty, posuvníky, rámy a menu.

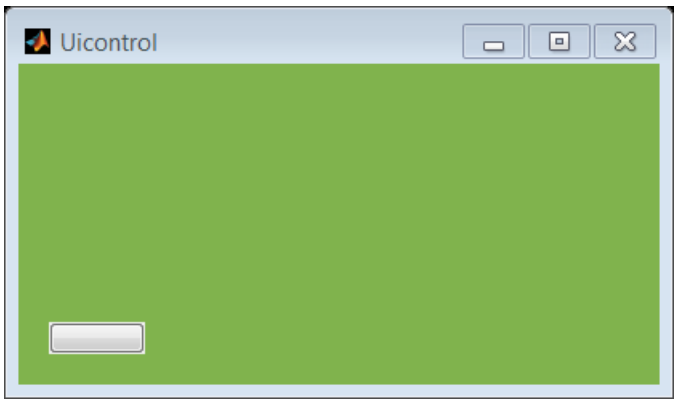
10.3.1 Vytvoření grafického objektu Uicontrol

Pro vytvoření objektu *Uicontrol* využijeme opět proměnnou handle (`h1`), pomocí které je možné objekt jednoznačně identifikovat a usnadní nám změnu vlastností objektu. Objekt *Uicontrol* vytvoříme tety takto: `h1=uicontrol`.

```
% Grafický objekt Uicontrol
h=figure ('Color',[0.5 0.7 0.3],'position',[800 500 400 200],...
    'Name','Uicontrol','NumberTitle','off','Menubar','none');
h1=uicontrol;
```

Obr. 10.8: Vytvoření objektu Uicontrol

Protože jsme náš objekt blíže neurčili, využije se opět implicitní nastavení. Implicitně je zde nastaveno tlačítko (*Pushbutton*). Spuštění výše uvedeného zdrojového kódu se objeví objekt *Figure*,ve kterém je umístěn jeden objekt *Uicontrol*.



Obr. 10.9: Grafický objekt Uicontrol

10.3.2 Nastavení vlastností objektu Uicontrol

U jednotlivých objektů je opět možné měnit celou řadu vlastností. Jejich kompletní seznam získáme zápisem příkazu `get(h1)`. Protože je seznam docela obsáhlý, zaměříme se především na ty, které jsou při programování aplikací nejvíce využívané.

```
BackgroundColor = [0.9 0.9 0.9]
Callback =
CData = []
Enable = on
Extent = [0 0 56 32]
FontAngle = italic
FontName = MS Sans Serif
FontSize = [14]
FontUnits = points
FontWeight = bold
ForegroundColor = [0 0 1]
HorizontalAlignment = center
KeyPressFcn =
ListboxTop = [1]
Max = [1]
Min = [0]
Position = [310 25 65 30]
String = Start
Style = pushbutton
SliderStep = [0.01 0.1]
TooltipString =
Units = pixels
Value = [0]

BeingDeleted = off
ButtonDownFcn =
Children = []
Clipping = on
CreateFcn =
DeleteFcn =
BusyAction = queue
HandleVisibility = on
HitTest = on
Interruptible = on
Parent = [1]
Selected = off
SelectionHighlight = on
Tag =
Type = uicontrol
UIContextMenu = []
UserData = []
```

Obr. 10. 10: Vlastnosti grafického objektu Uicontrol

10.3.3 Text tlačítka

Každé tlačítko je třeba doplnit textem, který by měl výstižně popisovat, co se po stisku tlačítka stane. My si zde naše tlačítko doplníme nápisem Start. Pro jeho zobrazení využijeme řetězec `string`. Nastavení textu tlačítka ve zdrojovém kódu je možné provést takto: `set (h1,'String','Start')`.

10.3.4 Změna velikosti, tloušťky a sklonu písma

Pro změnu velikosti písma můžeme využít vlastnost `Fontsize`, ke které je pak třeba přiřadit odpovídající velikost jako například ve Wordu: `set(h1,'FontSize',14)`.

Pro změnu tloušťky písma a nastavení jeho sklonu jsme využili příkazy `set (h1,'FontWeight','bold')` a `set (h1,'Fontangle','italic')`. Pokud bychom chtěli zjistit i další možnosti nastavení libovolné vlastnosti, stačí tuto vlastnost zapsat bez parametru. Chceme například zjistit další možnosti nastavení sklonu písma. Program doplníme o příkaz `set (h1,'Fontangle')`.

Odezvou nám potom bude výpis možností v okně *Command Window*.

```
  [ {normal} | italic | oblique ]
```

Obr. 10.11: Odezva na dotaz set (h1,'Fontangle')

10.3.5 Změna barvy textu a pozadí

Změna barvy pozadí je řízena vlastností `BackgroundColor`, barva textu v tlačítku vlastností `ForegroundColor`. Barvu můžeme opět zadávat jako vektor složek RGB nebo pomocí anglických názvů. Zde si pozadí nastavíme na světle šedou pomocí příkazu `set (h1,'BackgroundColor',[0.9 0.9 0.9])` Pro změnu barvy písma na modrou nám bude sloužit příkaz `set (h1,'ForegroundColor',[0 0 0.9])` nebo `set (h1,'ForegroundColor','Blue')`.

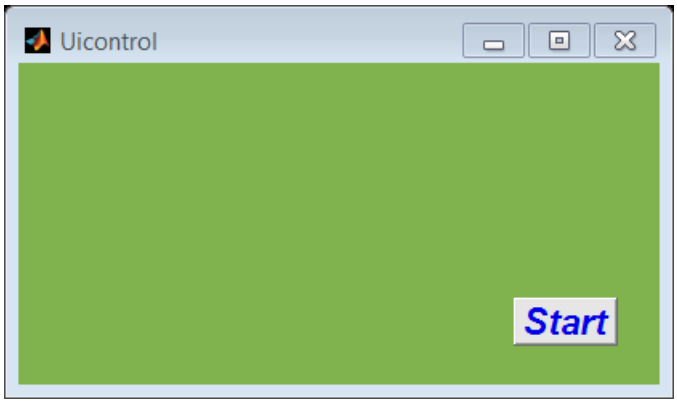
10.3.6 Změna velikosti a umístění

Jestliže chceme změnit umístění grafického objektu, využijeme stejně jako u objektu *Figure* vlastnost `position`. Do m-souboru tedy zapíšeme následující příkazy: `set (h1,'position',[310 25 65 30])`. První dvě číslice určují pozici grafického objektu *Uicontrol*, která se zde počítá od levého dolního okraje grafického objektu *Figure*, další dvě pak šířku a výšku vytvořeného tlačítka. Jednotlivé změny vlastností grafického objektu tak, jak byly v textu popsány, ukazuje následující zdrojový kód.

```
% Grafický objekt Uicontrol
h=figure ('Color',[0.5 0.7 0.3],'position',[800 500 400 200],...
    'Name','Uicontrol','NumberTitle','off','Menubar','none');
h1=uicontrol ('String','Start','FontSize',14,'FontWeight','bold',...
       'Fontangle','italic','BackgroundColor',[0.9 0.9 0.9],...
       'ForegroundColor','Blue','position',[310 25 65 30]);
```

Obr. 10. 12: Objekt Pushbutton

Výsledná grafická aplikace vypadá následovně.



Obr. 10.13: Vlastnosti objektu Uicontrol

10.4 Změna stylu grafických objektů

MATLAB neposkytuje grafické objekty jen v podobě tlačítek, ale disponuje celou řadou dalších objektů, které lze při návrhu aplikací využít. Ke změně grafického objektu slouží vlastnost `style`, která je implicitně nastavena právě na tlačítko (*Pushbuton*). Seznam všech objektů, které lze pomocí vlastnosti style nastavit, získáme, pokud zapíšeme `set(h1,'Style')`. K dispozici máme celkem deset možností: *Pushbutton, Togglebutton, Radiobutton, Checkbox, Edit, Text, Slider, Frame, Listbox, Popupmenu*.

Některé vlastnosti, jako velikost, umístění, barva, jsou společné pro všechny objekty a jejich nastavení bylo vysvětleno již v kapitole 10.3.

Některé vlastnosti jsou však typické pouze pro určité objekty. Nastavení jednotlivých objektů si nyní probereme podrobněji.

10.4.1 Toglebutton

Togglebutton je grafický objekt, který je vzhledem velmi podobný běžnému tlačítku. Rozdíl spočívá v trvání stavu zapnuto a vypnuto. Běžné tlačítko se po stisku ihned vrátí do původního stavu a je připraveno na další stisknutí. Při použití tlačítka *Togglebutton* je stav po stisknutí trvale udržován. Pokud chceme tlačítko vrátit do stavu vypnuto, je třeba ho stisknout opětovně.

```
% Grafický objekt Uicontrol
h=figure ('Color',[0.5 0.7 0.3],'position',[800 500 300 100],...
    'Name','Uicontrol','NumberTitle','off','Menubar','none');
h1=uicontrol ('Style','Toggle','String','Toggle off','FontSize',12,...
       'BackgroundColor',[0.9 0.9 0.9],'ForegroundColor',[0 0 0.9],...
       'position',[180 40 100 30],'FontWeigh','bold');
h2=uicontrol ('Style','Toggle','String','Toggle on','FontSize',12,...
       'BackgroundColor',[0.9 0.9 0.9],'ForegroundColor',[0 0 0.9],...
       'position',[20 40 100 30],'FontWeigh','bold');
```

Obr. 10.14: Objekt Togglebutton

Zde jsme vlastnost `style` nastavili na `Toggle`, čímž jsme určili objekt jako *Togglebutton*. Nastavení zbylých parametrů je stejné jako u tlačítka *Puschbutton*, proto je zde není třeba znovu vysvětlovat. Výsledek naší práce vidíme na obrázku 10.15.

Obr. 10.15: Ukázka objektu Togglebutton

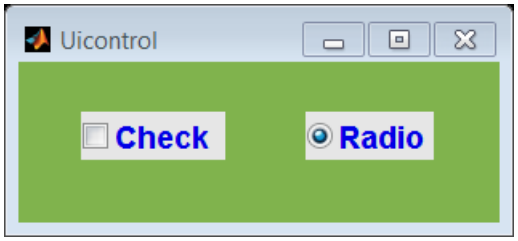
10.4.2 Radiobutton a checkbox

Tyto dva objekty mohou sloužit k nastavení různých parametrů aplikací, které budeme vytvářet. Abychom určili druh objektu, využili jsme opět vlastnost `style`. V prvním případě jsme ji přiřadili parametr `Radiobutton`, v druhém `Checkbox`. Kromě stylu objektu jsme m-soubor doplnili o další již známé vlastnosti. Výsledný kód je na obrázku 10.16.

```
% Grafický objekt Uicontrol
h=figure ('Color',[0.5 0.7 0.3],'position',[800 500 300 100],...
    'Name','Uicontrol','NumberTitle','off','Menubar','none');
h1=uicontrol ('Style','Radiobutton','String','Radio','FontSize',12,...
       'BackgroundColor',[0.9 0.9 0.9],'ForegroundColor',[0 0 0.9],...
       'position',[180 40 80 30],'FontWeigh','bold');
h2=uicontrol ('Style','Checkbox','String','Check','FontSize',12,...
       'BackgroundColor',[0.9 0.9 0.9],'ForegroundColor',[0 0 0.9],...
       'position',[40 40 90 30],'FontWeigh','bold');
set(h1,'Value',1);
```

Obr. 10.16: Objekty Radiobutton a Checkbox

U objektu *Radiobutton* a *Checkbox* je možné nastavit, zda jsou pole zatržené či nikoliv. Tuto vlastnost řídí hodnota `value`, která je implicitně nastavena na 0, tlačítko je nezatržené. Pokud budeme chtít nastavit tlačítko jako zatržené, stačí využít příkaz `set(h1,'Value',1)` pro Radiobutton a `set(h2,'Value',1)` pro Checkbox.



Obr. 10.17: Ukázka objektu Radiobutton a Checkbox

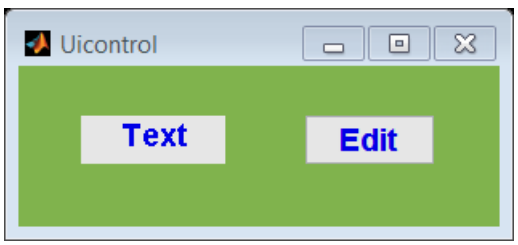
10.4.3 Edit a text

Objekt *Text* slouží pouze k zobrazení textu. Například pokud chceme popsat nějaký jiný objekt. Položka *Edit* umožňuje text nejen zobrazit, ale také editovat. Stačí pouze kliknout levým tlačítkem myši dovnitř položky.

```
% Grafický objekt Uicontrol
h=figure ('Color',[0.5 0.7 0.3],'position',[800 500 300 100],...
    'Name','Uicontrol','NumberTitle','off','Menubar','none');
h1=uicontrol ('Style','Edit','String','Edit','FontSize',12,...
       'BackgroundColor',[0.9 0.9 0.9],'ForegroundColor',[0 0 0.9],...
       'position',[180 40 80 30],'FontWeigh','bold');
h2=uicontrol ('Style','Text','String','Text','FontSize',12,...
       'BackgroundColor',[0.9 0.9 0.9],'ForegroundColor',[0 0 0.9],...
       'position',[40 40 90 30],'FontWeigh','bold');
```

Obr. 10.18: Nastavení objektu Edit a Text

Obsah m-souboru je na obrázku 10.18. Po jeho spuštění se objeví grafické okno, které je na obrázku 10.19.



Obr. 10.19: Ukázka objektu edit a text

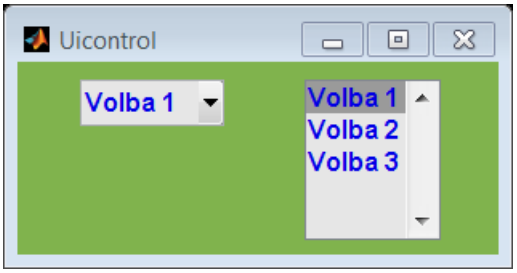
10.4.4 Listbox a Popupmenu

Tyto dvě položky je vhodné použít například v případech, kdy chceme vybírat z předem známého počtu možností. Objekt *Listbox* je tvořen jako seznam, kde jsou jednotlivé položky trvale zobrazené. U položky *Popupmenu* je seznam položek tvořen jako rozvinovací nabídka, která se zobrazí po kliknutí na šipku.

```
% Grafický objekt Uicontrol
h=figure ('Color',[0.5 0.7 0.3],'position',[800 500 300 120],...
    'Name','Uicontrol','NumberTitle','off','Menubar','none');
h1=uicontrol ('Style','Popupmenu','FontSize',10,'FontWeigh','bold',...
       'BackgroundColor',[0.9 0.9 0.9],'ForegroundColor',[0 0 0.9],...
       'position',[180 10 90 100],'String','Volba 1|Volba 2|Volba 3');
h2=uicontrol ('Style','Listbox','FontSize',10,'FontWeigh','bold',...
       'BackgroundColor',[0.9 0.9 0.9],'ForegroundColor',[0 0 0.9],...
       'position',[40 20 85 90],'String','Volba 1|Volba 2|Volba 3');
```

Obr. 10.20: Vytvoření objektu Listbox a Popupmenu

Informace o nastavení objektu zjistíme pomocí příkazu `get`. Mezi vlastnostmi objektu *Listbox* nebo *Popupmenu*, najdeme také položku `Value`, která určuje právě vybranou nabídku. Jednotlivé nabídky jsou číslovány odshora. První položka v seznamu má tedy hodnotu `value 1`. Pokud by byla vybrána například třetí nabídka, bude mít položka `value` hodnotu `3` atd. Aktuální hodnotu vlastnosti `value` můžeme zjistit pomocí příkazu `get(h1,'Value')`.



Obr. 10.21: Listbox a Popupmenu

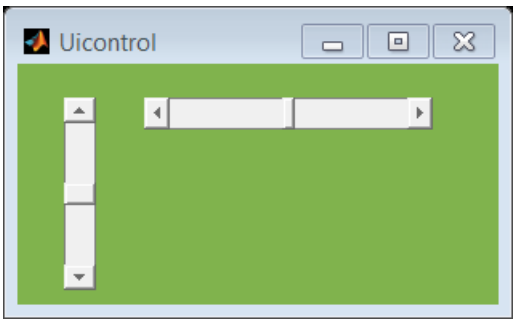
10.4.5 Slider

Slider neboli posuvník, je velmi často využívaným objektem při programování grafických aplikací. Posuvníky můžeme vytvářet buď svislé nebo vodorovné. Zda je posuvník svislý nebo vodorovný, určuje vlastnost `position`. První dvě položky určují pozici v okně figure, další dvě pak šířku a výšku. Pokud tedy zadáme výšku posuvníku větší než šířku, je orientován svisle. Pokud bude větší jeho šířka, je orientován vodorovně. Zdrojový kód pro vytvoření posuvníků je následující:

```
% Grafický objekt Uicontrol - Slider
h=figure ('Color',[0.5 0.7 0.3],'position',[800 500 300 150],...
    'Name','Uicontrol','NumberTitle','off','Menubar','none');
h1=uicontrol ('Style','Slider','position',[30 10 20 120],...
    'Min',10,'Max',20,'Value',15,'SliderStep',[1/(20-10) 2/(20-10)]);
h2=uicontrol ('Style','Slider','position',[80 110 180 20],...
    'Min',0,'Max',50,'Value',25,'SliderStep',[1/50 2/50]);
```

Obr. 10.22: Vytvoření svislého a vodorovného posuvníku

Při vytváření posuvníku je třeba zadat jeho minimální a maximální hodnotu, tedy čísla, kterých bude posuvník nabývat v krajních polohách. To provedeme pomocí parametrů `min` a `max`, které nám tyto hodnoty jednoznačně určí. Dále je vhodné přiřadit hodnotu jezdci posuvníku. Pokud bychom chtěli jezdce umístit do některé z krajních poloh, odpovídá položka `value` hodnotě `min` nebo `max`. My jsme si oba jezdce umístili doprostřed posuvníku. Vlastnost `SliderStep` slouží k definici kroku jezdce, tedy vzdálenosti, o kterou se jezdec posune při kliknutí na jednu z krajních šipek nebo do dráhy jezdce. Tato položka je tvořena jako vektor o dvou prvcích.



Obr. 10.23: Posuvník Slider

Aktuální polohu jezdce, či minimální a maximální hodnotu, můžeme opět zjistit pomocí známého příkazu `get`. Například pro zjištění polohy jezdce prvního posuvníku by vypadal následovně: `get(h1,'Value')`.

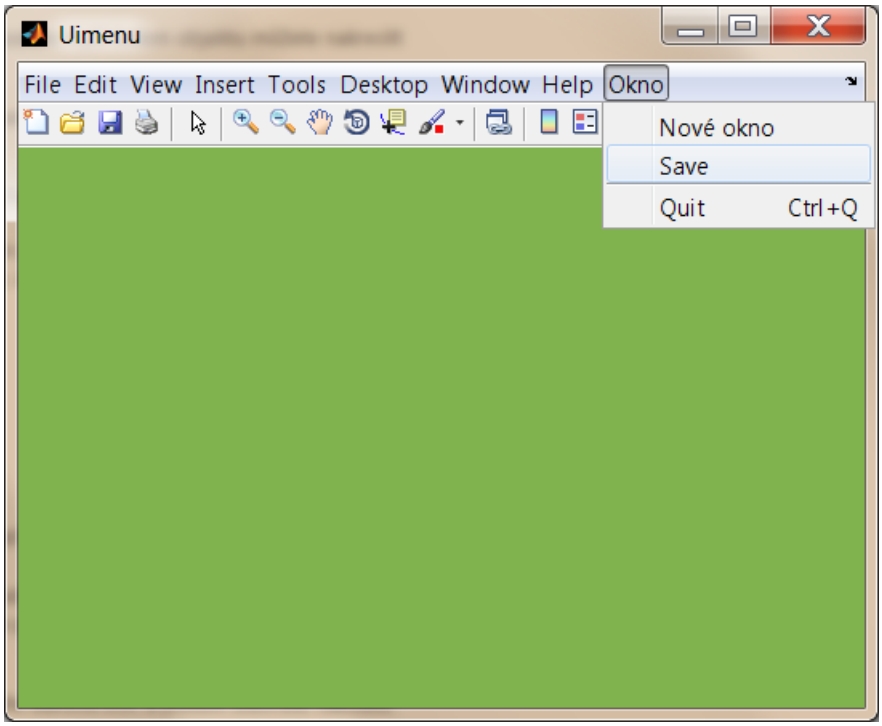
10.5 Grafický objekt Uimenu

Při vytváření vlastních grafických aplikací je někdy vhodné aplikaci doplnit o vlastní menu. Menu je položka, která se vytvoří v hlavní liště programu (obr.10.24). Pokud na ni klikneme levým tlačítkem myši, objeví se seznam dalších položek, které tvoří tzv. submenu. V MATLABu jsou tyto položky realizovány pomocí grafického objektu *Uimenu*, který je podle hierarchie grafických objektů na stejné úrovni jako objekt *Uicontrol*. Jsou mu tedy nadřazeny grafické objekty *Figure* a *Root*. Menu vzniká jako součást grafického objektu *Figure*.

```
%Grafický objekt Uimenu
h=figure ('Color',[0.5 0.7 0.3],'position',[800 500 500 350],...
    'Name','Uimenu','NumberTitle','off');
h1=uimenu ('Label','Okno');
    uimenu(h1,'Label','Nové okno','Callback','figure');
    uimenu(h1,'Label','Save','Callback','save');
    uimenu(h1,'Label','Quit','Callback','exit',...
              'Separator','on','Accelerator','Q');
```

Obr. 10.24: Vytvoření objektu Uimenu

Pro vytvoření vlastního menu slouží řádek`h1=uimenu('Label','Okno')`, který zajistí přidání nové nabídky na hlavní panel. Položky, které se zobrazí po kliknutí na námi vytvořené menu okno, jsou definovány pomocí proměnné handle položky `uimenu`. Poslední čtyři řádky zdrojového kódu tedy vytvářejí submenu, jehož funkce si můžeme vyzkoušet kliknutím na jednotlivé položky.



Obr. 10.25: Nově vytvořené menu

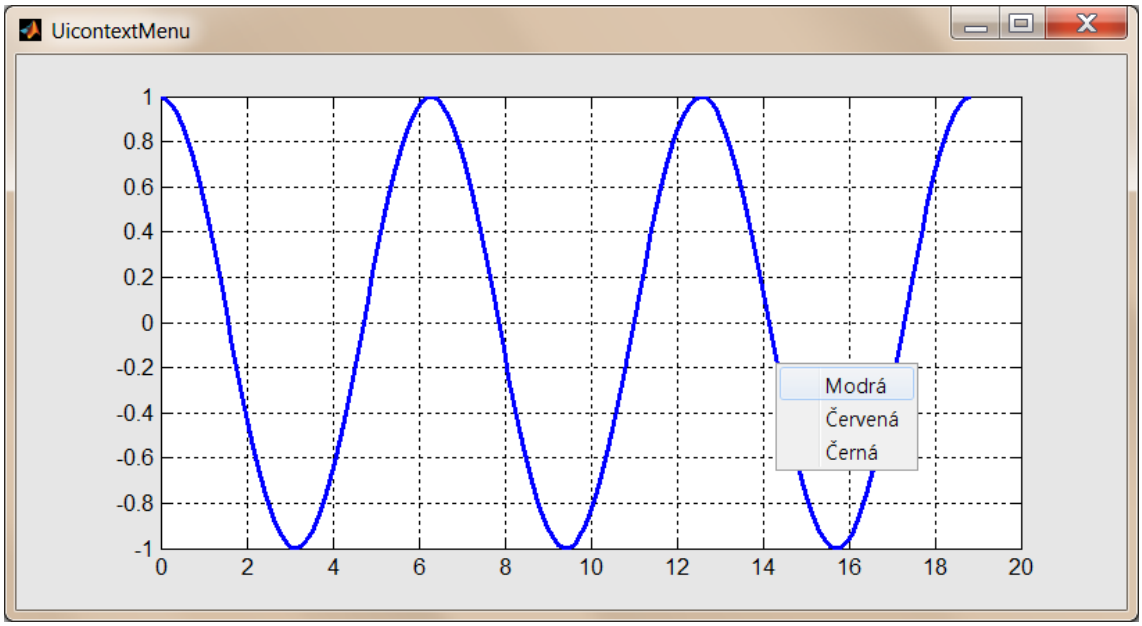
10.6 Grafický objekt Uicontextmenu

Tento druh menu využíváme, pokud chceme měnit vlastnosti některých objektů. Je to tedy menu, které se zobrazí po kliknutí pravého tlačítka myši na vybraný objekt, například graf. Na rozdíl od *Uimenu*, které je součástí menu objektu *Figure*, je *Uicontextmenu* pevně svázáno s objektem. Provázanost si můžeme ukázat i na následujícím příkladu, ve kterém jsme vytvořili jednoduchý graf funkce cosinus.

```
%Grafický objekt UicontextMenu
t=linspace (0,6*pi,150);
u=cos(t);
h=figure ('Color',[0.9 0.9 0.9],'position',[800 500 800 400],...
    'Name','UicontextMenu','NumberTitle','off','Menubar','none');
Zmena=uicontextmenu;
h1=plot(t,u,'LineWidth',2,'UicontextMenu',Zmena);
grid on;
Nabidka1=['set(h1,''Color'',''b'')'];
Nabidka2=['set(h1,''Color'',''r'')'];
Nabidka3=['set(h1,''Color'',''k'')'];
Barva1 = uimenu(Zmena,'Label','Modrá','Callback',Nabidka1);
Barva2 = uimenu(Zmena,'Label','Červená','Callback',Nabidka2);
Barva3 = uimenu(Zmena,'Label','Černá','Callback',Nabidka3);
```

Obr. 10.26: Grafický objekt Uicontextmenu

Pokud klikneme pravým tlačítkem myši na čáru grafu, zobrazí se nabídka, pomocí které můžeme měnit barvu čáry, klikneme-li mimo křivku, stav aplikace se nezmění. Jednotlivé položky menu jsou definovány pomocí tří posledních řádků. Akci, která se provede po kliknutí na jednu z položek, zajišťuje funkce Callback, která pomocí příkazu `set` provede změnu barvy čáry. Výsledek naší práce je na obrázku 10.27.



Obr. 10.27: Menu vytvořené uživatelem

10.7 Grafický objekt Axes

Jednou z největších zbraní MATLABU je práce s grafikou. Můžeme si vybírat z velkého množství typů grafů, ať už chceme naše data zobrazovat dvojrozměrně či trojrozměrně. Grafy je možné opatřovat názvy, měnit popisky a rozsah os, nastavovat barvu a způsob vykreslování jednotlivých bodů či čar, přidávat další objekty apod. Všechny tyto možnosti řídí grafický objekt *Axes*, kterému jsou podřízeny další objekty.

Jako příklad si vytvoříme jednoduchý graf, kterému nezapomeneme přiřadit proměnnou handle tak, abychom s ním v rámci objektu *Axes* mohli pracovat.

```
%Grafický objekt Axes
x=linspace (0,2,100);
y1=3*x.*exp(-3*x.^2);
y2=3*x.*exp(-2*x.^3);
y3=3*x.*exp(-1*x.^4);
h=plot (x, y1, 'b', x, y2, 'k', x, y3, 'm');
grid on;
```

Obr. 10.28: Graf pro manipulaci s objekty Axes

Pokud se v naší aplikaci vyskytne více grafických objektů *Axes*, je vhodné vědět, jak určit proměnnou handle tohoto grafického objektu. K tomu lze využít příkaz `h=gca`. Výsledkem je hodnota, která nám jednoznačně určí grafický objekt.

```
h =

  170.0026
```

obr. 10.29: Zjištění handle grafického objektu Axes

V našem grafu se nyní pomocí proměnné handle (`h`), pokusíme změnit styl čáry prvního grafu z plné na přerušovanou. Vlastnost sloužící ke změně stylu čáry zjistíme opět pomocí příkazu `get(h)`. Změnu stylu čáry grafu zajišťuje položka `LineStyle`, která je nyní nastavena na plnou čáru `get(h,'LineStyle')`. Vlastnost tedy změníme

pomocí příkazu `set` na čáru přerušovanou: `set(h,'LineStyle','--')` Pokud si nyní spustíme aplikaci, vidíme, že výsledek je jiný, než jsme očekávali.

Pokusíme se nyní určit proměnné handle grafických objektů. Pro objekt *Axes* ji získáme příkazem `h=gca`. Tento objekt však není jediný. Podle hierarchického uspořádání může objekt *Axes* obsahovat další podřízené objekty, tzv. potomky. Pro zjištění těchto objektů je možné využít příkaz `children`. Zdrojový kód tedy rozšíříme o následující příkazy:

```
%Grafický objekt Axes
x=linspace (0,2,100);
y1=3*x.*exp(-3*x.^2);
y2=3*x.*exp(-2*x.^3);
y3=3*x.*exp(-1*x.^4);
h=plot (x, y1, 'b', x, y2, 'k', x, y3, 'm');
grid on;
h=gca
h=get(h,'children')
```

Obr. 10. 30: Užití příkazu children

Výsledkem je následující odezva v okně *Command Window*:

```
h =

  170.0092


h =

  173.0095
  172.0095
  171.0115
```

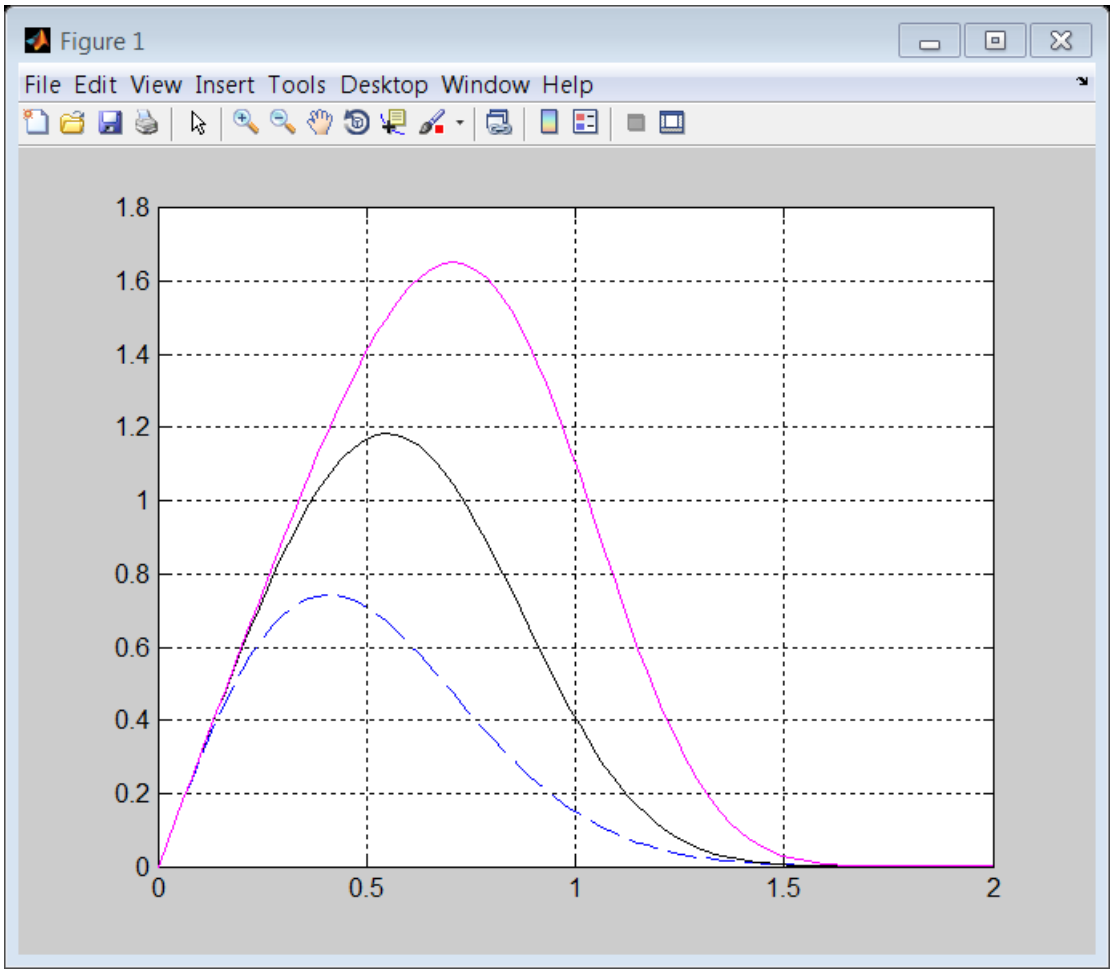
Obr. 10. 41: Odezva na příkaz children

Objekt *Axes* obsahuje tři další objekty, které řídí vykreslení jednotlivých grafů (funkce `plot`). Nyní již pouze stačí nastavit změnu stylu čáry u příslušné křivky.

```
%Grafický objekt Axes
x=linspace (0,2,100);
y1=3*x.*exp(-3*x.^2);
y2=3*x.*exp(-2*x.^3);
y3=3*x.*exp(-1*x.^4);
h=plot (x, y1, 'b', x, y2, 'k', x, y3, 'm');
grid on;
h=gca;
h=get(h,'children');
set(h(3),'LineStyle','--');
```

Obr. 10. 52: Změna stylu čáry pomocí handle

Výsledná podoba grafu je následující:



Obr. 10. 63: Výsledný graf

Kromě proměnné `children`, je v MATLABu k dispozici ještě proměnná `parent`, která slouží k určování nadřazených objektů (rodičů). Výhodou těchto proměnných je, že lze pomocí nich měnit vlastnosti celých skupin objektů a výrazně tak ulehčit práci programátora.

11 Nástroj GUIDE

V moderních operačních systémech jsou téměř veškeré aplikace vytvářeny jako grafické uživatelské rozhraní. Jsou to tedy aplikace, které obsahují grafické ovládací prvky a zprostředkovávají vazbu mezi uživatelem a programem. Tyto aplikace se často označují zkratkou GUI z anglického (Graphical User Interface). Většina uživatelů PC přijde do styku právě pouze s GUI a podle jednoduchosti ovládání a vzhledu se rozhodují, zda budou či nebudou aplikaci užívat.

V předchozí kapitole jsme se seznámili se základními grafickými prvky a ukázali si několik jednoduchých příkladů, jak tyto grafické prvky vytvořit. Tyto příklady jsme museli cele naprogramovat. Při tvorbě GUI v MATLABu je možné použít editor, který nám pomůže jednotlivé grafické prvky vytvořit. Tento editor je v systému MATLAB integrován a nazývá se GUIDE (Graphical User Interface Development Environment).

Editor umožňuje snadné rozmístění grafických prvků, které doplní automaticky generovaným zdrojovým kódem. Řešení je tedy časově méně náročné. Nevýhodou však je, že automaticky generovaný zdrojový kód není optimální. Na rozdíl od aplikace, kterou si uživatel vytvoří sám, obsahuje nepotřebné části zdrojového kódu.

11.1 Zásady návrhu GUI

Hlavním posláním grafického uživatelského rozhraní je co nejvíce uživateli zpříjemnit práci s aplikací. Proto bychom tvorbě aplikace měli věnovat dostatečnou pozornost a při návrhu se řídit několika zásadami. Dobře vytvořenou grafickou aplikací totiž můžeme mnohé získat. Námi vytvořené aplikace by měly obsahovat:

- Intuitivní ovládání
- Standardní postupy
- Vzhled (přehlednost GUI)

11.1.1 Intuitivní ovládání

Dnešní aplikace jsou konstruovány tak, aby po jejich spuštění mohl uživatel s aplikací ihned pracovat a to bez potřeby detailně studovat její ovládání nebo třeba číst manuál. Uživatel tedy aplikaci do jisté míry ovládá intuitivně. Pokud vytváří GUI, je třeba tuto podmínku mít na paměti a například ovládací prvky seskupovat do logických bloků. Jako příklad můžeme použít aplikaci z kapitoly 12.1. Zde uživatel nejprve zadává rychlost a až poté po stisku tlačítka je zobrazen výsledek. Blok pro zadání rychlosti je úmyslně umístěn výše než blok s výsledkem. Je logické data nejprve načíst a až poté provést výpočet a zobrazit výsledek.

11.1.2 Standardní postupy

Při spouštění různých aplikací si můžeme všimnout, že některé znaky jsou pro všechny aplikace společné. Jako příklad můžeme uvést klávesovou zkratku *Ctrl+C*, která slouží ke kopírování dat do schránky. Tato zkratka je standardem, který se dodržuje ve všech aplikacích. Pokud tedy vytváříme vlastní aplikaci, měli bychom tento standard také dodržet a nepřiřazovat zkratce *Ctrl+C* například akci uložení. Dalším příkladem může být například položka soubor v menu různých aplikací. Jistě jste si všimli, že tato

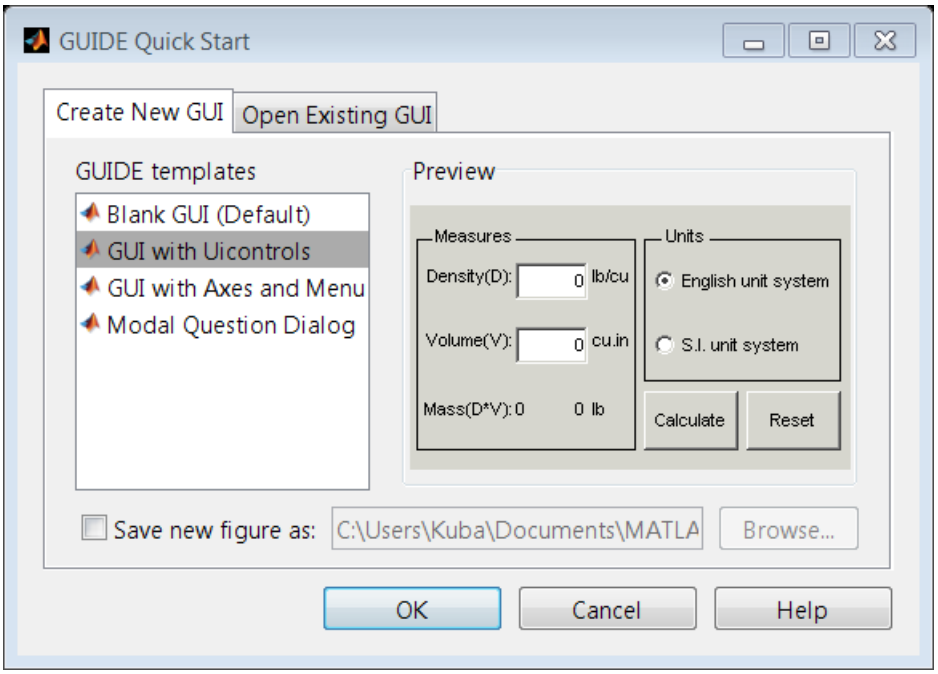
položka bývá v menu vždy jako první. Při programování bychom se tedy měli řídit těmito zažitými pravidly.

11.1.3 Vzhled

Vizuální stránka grafického uživatelského rozhraní do velké míry přispívá k oblíbenosti aplikace. Vzhled je také to první, co uživatel po spuštění aplikace zkoumá. Pokud jsou například ovládací prvky rozmístěny různě po ploše, navíc nepříliš dobře zarovnané, budí aplikace dojem neprofesionality. Pokud programátor neumí ani zarovnat jednotlivé prvky, jak asi umí programovat?

11.2 Spuštění editoru GUIDE

Editor je možné spustit zápisem příkazu `guide` do okna *Command Window* nebo pomocí položek *File* → *New* →*GUI*. Po jeho spuštění se zobrazí okno *GUIDE Quick Start* (obr. 11.1).



Obr. 11.1: Spuštění GUIDE editoru

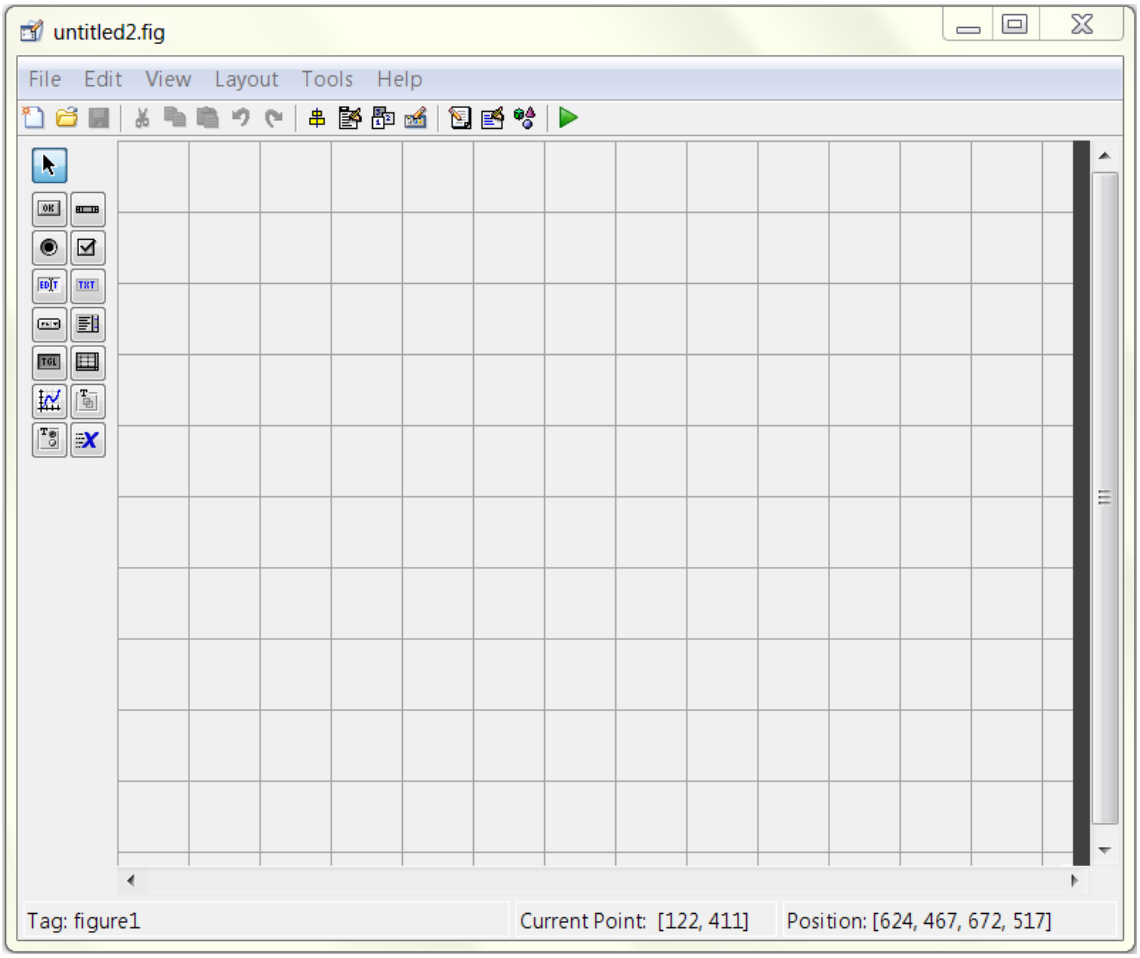
Toto okno nám nabízí celkem čtyři možnosti tvorby grafického uživatelského rozhraní. Uživatel má na výběr:

- Prázdnou aplikaci (*Blank GUI*)
- Aplikaci s ovládacími prvky Uicontrol(*GUI with Uicontrols*)
- Aplikaci s objektem Axes a menu (*GUI with Axes and Menu*)
- Vytváření dotazů pro chod aplikace (*Model Question Dialog*)

Pro naše potřeby budeme nejčastěji volit prázdnou aplikaci, tedy možnost (*Blank GUI*). Kromě typu aplikace si v tomto okně také můžeme vybrat, zda bude jeho práce hned po startu uložena *Save on startup as* (Při startu uložit jako…) nebo zda se aplikace uloží až při prvním spuštění.

Nyní vybereme možnost *Blank GUI* a klikneme na tlačítko *OK*. Po této akci se objeví průvodce s názvem untitled.

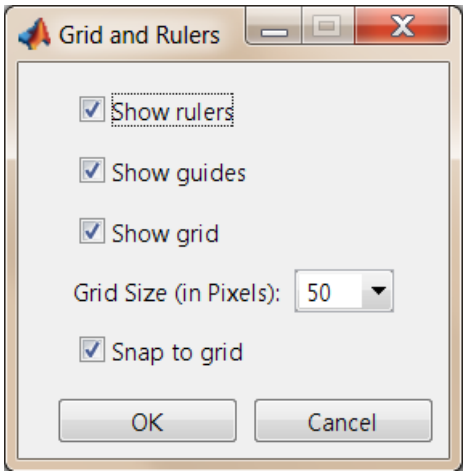
Průvodce má podobu standardního okna s řadou rozbalovacích menu, ze kterých jsou přístupné všechny možnosti a nastavení a pruh ikon, které obsahují nejdůležitější a často používané povely. Nejdůležitější součástí GUIDE editoru je paleta objektů tzv. *Component Palette*, kde jsou všechny objekty *Uicontrol*. Pracovní plochu tvoří čtvercová síť – *Layout Area*, která umožňuje hrubě určit pozice jednotlivých objektů, které na pracovní plochu umístíme. Velikost čtvercové sítě můžeme měnit pomocí čtverečku v pravém dolním rohu pracovní oblasti.



Obr. 11.2: Prostředí GUIDE editoru

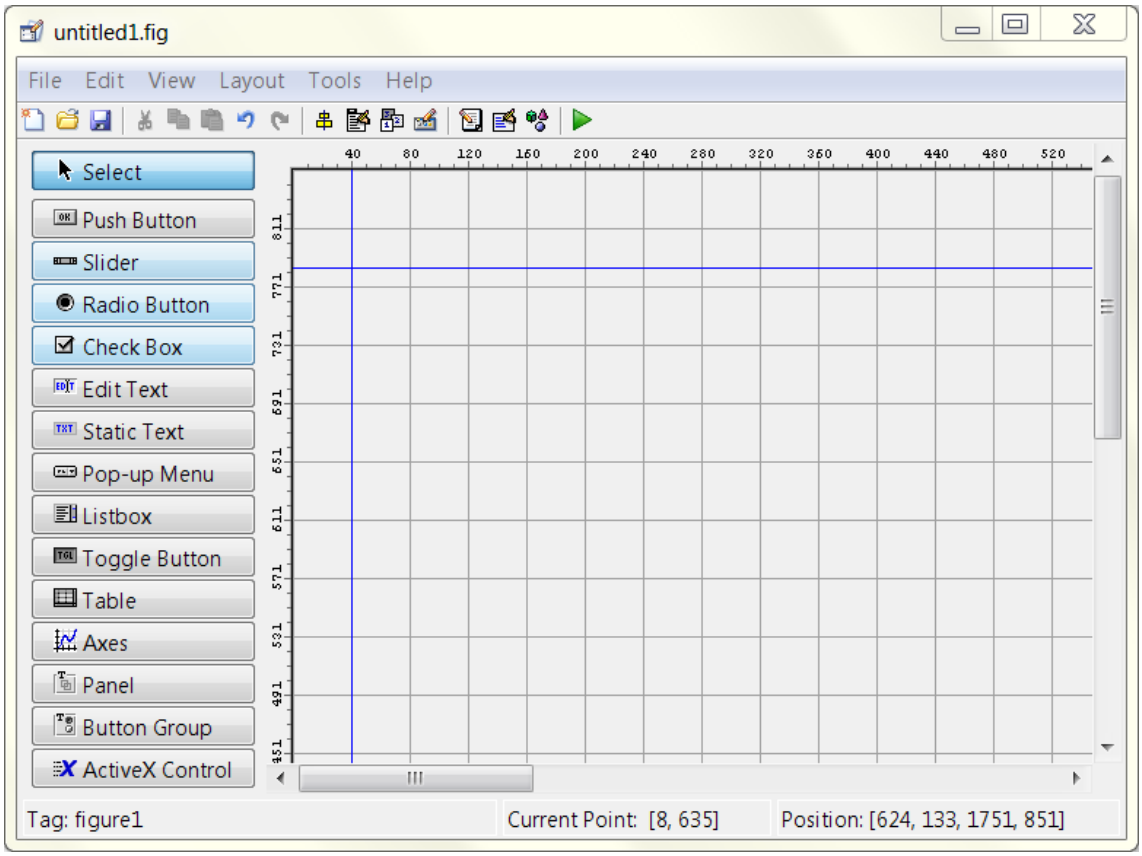
11.3 Nastavení GUIDE editoru

Před zahájením práce v GUIDE editoru je vhodné provést několik drobných změn. Nejprve si upravíme vzhled *Layout Area*, tedy pracovní plochy, na kterou umísťujeme jednotlivé objekty. Změny provedeme kliknutím na *Tools* v menu, kde si vybereme položku *Grid and Rulers* (obr. 11.3). Zde je vhodné zaškrtnout položku *Show rulers*, která slouží k zobrazení pravítka. Dále je dobré změnit velikost čtvercové sítě. My jsme zde velikost čtverce zmenšili na 50 pixelů.



Obr. 11.3: Grid and Rulers

Další nastavení provedeme kliknutím na *File → Preferences*. Zde zaškrtneme položku *Show names in componet palette*, která zajistí zobrazení názvů u palety objektů. (obr. 11.4).



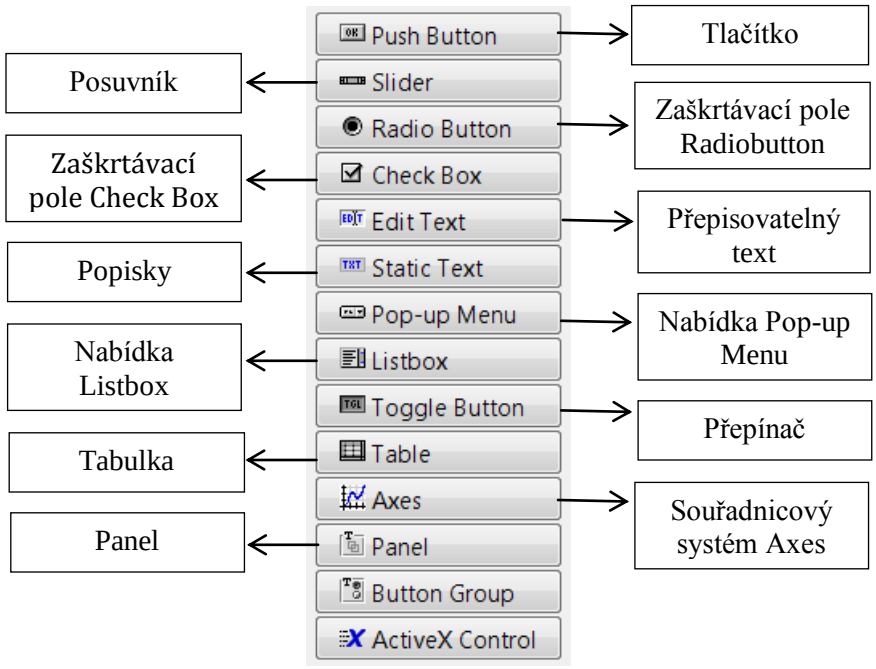
Obr. 11.4: Nastavení pracovního prostředí GUIDE editoru

11.3.1 Vodící linky

Pro přesnější umístění jednotlivých objektů můžeme využít vodící linky, ke kterým je možné jednotlivé objekty přichytit. Vodící linku vytvoříme najetím kurzoru myši na pravítko. Ve chvíli, kdy se kurzor změní ve dvoušipku, stačí stisknout levé tlačítko myši a táhnout zpět do Layout Area. V místě, kde chceme vodící linku umístit, pustíme pravé

tlačítko myši. Pokud s umístěním linky nejsme spokojeni, stačí na ni umístit kurzor, stisknout pravé tlačítko a tažením linku přemístit (Obr. 11.4).

11.4 Component Palette



Obr. 11.5: Paleta grafických prvků – Componenent Palette

11.4.1 Push Button – tlačítko

Tento prvek využíváme nejčastěji pro zahájení nějaké akce. Můžeme ho využít například pro start aplikace nebo provedení výpočtu, vykreslení grafu apod.

11.4.2 Slider – posuvník

Posuvník může sloužit například ke změně hodnot v určitém rozsahu. Programátor zde nadefinuje maximální a minimální hodnotu, čímž určí rozsah, ve kterém bude veličina měněna. Dále zde může určit krok, čímž lze například zajistit, aby veličina nabývala jen celočíselných hodnot.

11.4.3 Radio Button – zaškrtávací pole

Toto pole může sloužit například k výběru mezi akcemi, které bude aplikace provádět. Pokud budeme chtít například zobrazit graf funkce sinus nebo cosinus, můžeme k tomu využít právě pole *Radio Button*.

11.4.4 Check Box – zaškrtávací pole

Tento prvek má obdobnou funkci jako *Radio Button*. Můžeme ho využít pro zapnutí či vypnutí mřížky grafu, viditelnost či neviditelnost objektu nebo nastavení dalších podmínek aplikace, kde se rozhodujeme mezi dvěma akcemi.

11.4.5 Edit Text – přepisovatelný text

Tuto komponentu můžeme použít pro zadávání vstupních hodnot uživatelem. Pokud budeme například chtít provést nějaký výpočet, kde si uživatel sám zadá vstupní hodnoty, využijeme k tomu právě *Edit Text*.

11.4.6 Static Text – Popisky

Tento prvek používáme nejčastěji k popisu dalších prvků. *Static Text* používáme nejčastěji k lepší přehlednosti aplikace a zajištění lepší orientace pro uživatele.

11.4.7 Pop-up Menu – nabídka

Používáme, pokud chceme vybírat z určité skupiny prvků podobného druhu. Například pokud chceme vybírat mezi indexy lomu různých materiálů. Podmínkou je, že uživatel může vybírat pouze z prvků, které jsou v menu k dispozici.

11.4.8 List Box – nabídka

List Box má stejnou funkci jako *Pop-up Menu*. Rozdíl je pouze ve způsobu zobrazení dat. Zatímco u *Pop-up Menu* prvky vybíráme po kliknutí na šipku, která zobrazí jejich nabídku, u *List Boxu* mezi prvky listujeme.

11.4.9 Togle Button – přepínač

Je tlačítko, které slouží k přepínání mezi hodnotami. Můžeme jej využít například k přepínání mezi stupni a radiány.

11.4.10 Table – tabulka

Tato komponenta slouží k zadávání většího množství dat. Například pokud budeme chtít pracovat s maticemi či statistickými daty, můžeme pro jejich zadání využít tabulku.

11.4.11 Axes – souřadnicový systém

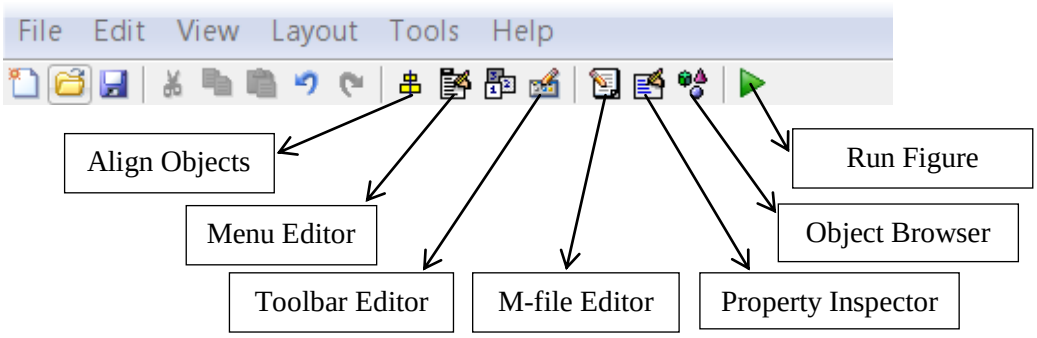
Při tvorbě aplikací nám slouží k vyhrazení prostoru, ve kterém se bude zobrazovat graf. Lze pomocí něj také ovládat všechny podřízené objekty, například funkci Plot.

11.4.12 Panel

Panel je komponenta, která slouží zejména k seskupování objektů například ovládacích prvků. Slouží k zajištění lepší přehlednosti aplikace.

11.5 Menu GUIDE editoru

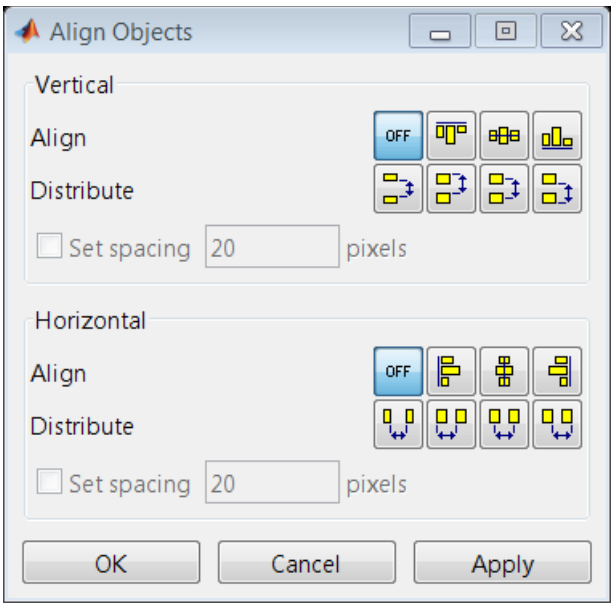
Editor GUIDE obsahuje několik ikon, které jsou typické pouze pro tento editor. My si jednotlivé ikony nyní postupně probereme a vysvětlíme některé jejich funkce.



Obr. 11.6: Menu GUIDE editoru

11.5.1 Align Objects

V předchozí kapitole jsme objekty zarovnávali pomocí položky `position`. V GUIDE editoru je k tomuto účelu určen nástroj *Align Objects*, který po přetažení objektů na pracovní plochu (*Layout area*) umožní jejich přesné zarovnání.

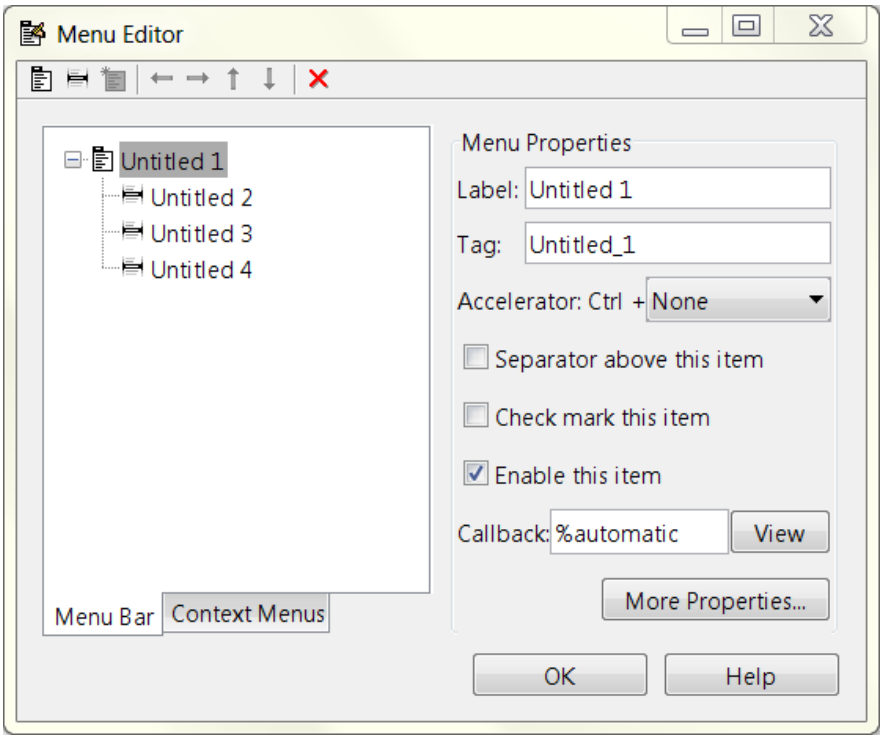


Obr. 11.7: Align Objects

Jednotlivé komponenty je možné zarovnávat jak v horizontálním, tak ve vertikálním směru. Ve vertikálním směru je možné objekty zarovnávat k hornímu a dolnímu okraji nebo na střed. V horizontálním směru pak k levému nebo pravému okraji a na střed. Kromě zarovnání objektů je zde možné nastavit i vzdálenost mezi jednotlivými objekty.

11.5.2 Menu Editor

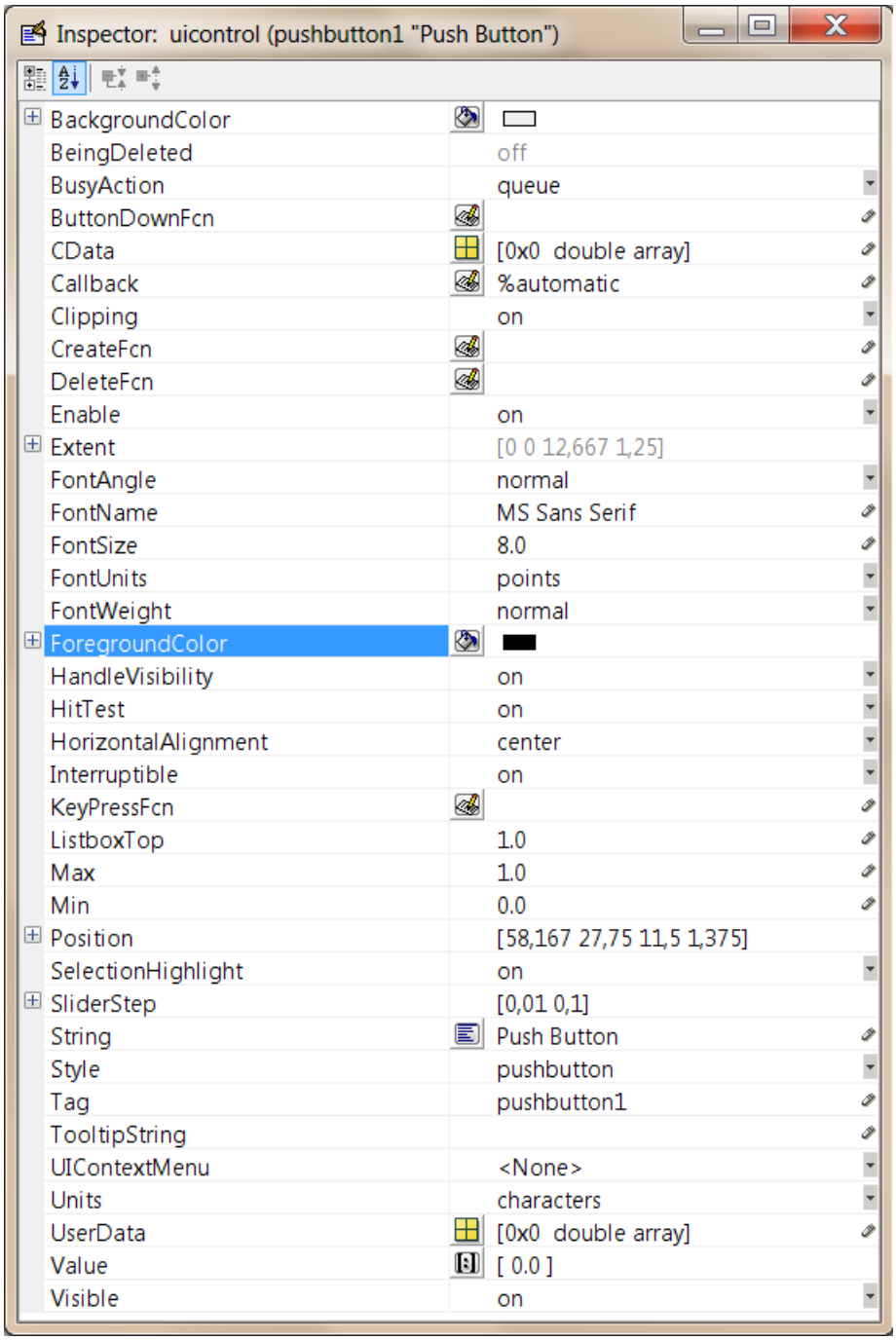
Po kliknutí na tuto ikonu se otevře nové okno, ve kterém uživatel může vytvářet vlastní menu. Pracujeme zde vlastně s objekty *Uimenu* a *Uikontextmenu*. Můžeme tedy vytvářet nabídky, které se zobrazí na hlavním panelu nebo po kliknutí na pravé tlačítko myši.



Obr. 11.8: Menu Editor

11.5.3 Property Inspector

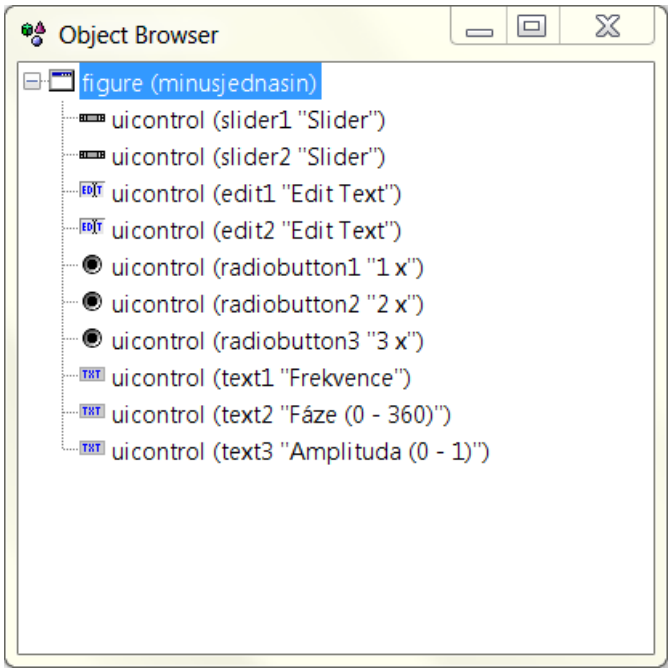
Tako ikona v menu slouží k zobrazení vlastností jednotlivých objektů. Kromě menu lze *Property Inspector* spustit také dvojím poklepáním na vybraný objekt. Při podrobnějším prozkoumání zjistíme, že se zde nacházejí některé položky, které jsou nám již známé z předchozí kapitoly. Některé položky jsou pro všechny objekty stejné, některé jsou typické pouze pro daný objekt. Výčet je u všech objektů poměrně obsáhlý, proto jsme se seznámili jen s těmi důležitými.



Obr. 11.9: Property Inspector

11.5.4 Object Browser

Při vytváření aplikací skládáme jednotlivé objekty na pracovní plochu GUIDE editoru. Pokud chceme zobrazit seznam všech použitých objektů a jejich hierarchii, můžeme použít právě *Object Browser*. V následující ukázce máme několik posuvníků, zaškrtávacích polí a textových či editovatelných objektů. Je zde také patrná jejich podřízenost objektu *Figure*.



Obr. 11.10: Objekt Browser

11.5.5 Run

Poslední ikona slouží ke spuštění námi vytvořené aplikace. Pokud jsme v okně *GUIDE Quick Start* nezaškrtli volbu *Save on startup as*, MATLAB nás nyní vyzve k uložení naší práce. Aplikace je uložena do dvou souborů s příponou **soubor.fig** a **soubor.m**.

fig – file: zde je uložena grafická podoba aplikace (kompletní popis grafické části GUI a jeho součástí). Obsah tohoto souboru můžeme zobrazit dvojím poklepáním pravého tlačítka myši v okně *Current Directory*.

m – file:je soubor, který se vygeneruje automaticky při prvním spuštění. V souboru jsou části procedur a funkcí, které se zde vytvářejí automaticky v závislosti na akcích, které jsme provedli v GUIDE editoru. Vznikají zde i prázdné Callback funkce určené pro obsluhu jednotlivých objektů. Zde si zdrojový kód doplníme sami tak, aby aplikace prováděla to, co od ní požadujeme.

11.6 Zpětnovazební funkce jednotlivých Uicontrol objektů.

Zpětnovazební funkce (Callback funkce) jsou volány, pokud uživatel provede v aplikaci nějakou akci. Například stisk tlačítka nebo výběr položky z menu. Jak bude aplikace na tuto akci reagovat, určuje programátor zápisem příkazů do příslušné Callback funkce. Je dobré si uvědomit, že i přesto, že jsou tyto funkce generovány automaticky GUIDE editorem, nijak se neliší od funkcí, které jsme vytvářeli v kapitole 6.

Při návrhu aplikací můžeme m-soubor doplnit o další zpětnovazební funkce. Pokud v GUIDE editoru klikneme pravým tlačítkem na vybraný objekt a zvolíme položku *View Callbacks*, objeví se další možnosti zpětnovazebních funkcí, vybereme-li některou z nich, funkce se ihned přidá do m-souboru.

- **CreateFcn** je funkce, která se spouští při startu aplikace, můžeme zde definovat počáteční podmínky
- **DeleteFcn**je funkce, která se spouští, když je okno smazáno
- **CloseRequestFcn** je funkce, která je vyvolána v okamžiku zavírání aplikace.
- **KeyPressFcn** je funkce vyvolaná stiskem tlačítka, pokud je kurzor v okně aplikace.
- **ResizeFcn** se zavolá, pokud uživatel změní velikost okna.
- **WindowButtonDownFcn** je spuštěna po kliknutí myši, pokud je kurzor nad pozadím obrázku neaktivním prvkem uživatelského rozhraní nebo pozadím os (tzn. není vyvolána při kliknutí na aktivní ovládací prvky).
- **WindowButtonMotionFcn** je spuštěna, pokud uživatel pohybuje myší se stisknutým tlačítkem v rámci okna (neplatí pro menu okna).
- **WindowButtonUpFcn** je zavolána poté, co uživatel uvolní tlačítko myši. Předchozí stisk tlačítka musí být v rámci obrázku.

U jednotlivých objektů je vhodné dodržovat určité postupy při programování jejich zpětnovazebních funkcí. Nyní si je představíme.

11.6.1 Zpětná vazba tlačítka Push Button

Tlačítko *Push Button* je zpravidla určeno k provedení nějaké akce. Například ke startu aplikace. Jeho zpětnovazební funkce Callback má následující tvar:

```
function pushbutton_Callback(hObject, eventdata, handles)
prikazy %provedou se po stisku tlačítka
```

11.6.2 Zpětná vazba tlačítka Toggle Button

Jak už bylo řečeno na začátku kapitoly, jedná se o dvoustavové tlačítko, jehož stavy jsou řízeny parametry `min` a `max`. Pokud je hodnota parametru `min=1`, je tlačítko rozepnuté, pokud je `max=1`, je tlačítko sepnuté.

```
function togglebutton_Callback(hObject, eventdata, handles)
stav_tlacitka=get(hObject,'Value');
if stav_tlacitka==get(hObject,'Max')
prikazy %provedou se při stisknutém tlačítku
elseif stav_tlacitka==get(hObject,'Min')
prikazy %provedou se při uvolněném tlačítku
end
```

11.6.3 Zpětná vazba zaškrtávacího pole Radio Button

Toto zaškrtávací pole se může nacházet ve stavu sepnuto a vypnuto. Zda je pole aktivní či nikoli, řídí opět vlastnost `Value`, která nabývá hodnot 1 při sepnutém stavu a 0 pokud tlačítko není vybráno. Tlačítko *Radio Button* se často využívá, pokud chceme vybírat z několika možností a pokud chceme, aby mohla být zvolena právě jedna možnost.

```
function vypne_radio(off)
set(off,'Value',0)
```

Nejprve vytvoříme funkci, která nastaví stav tlačítka na vypnuto. Tuto funkci pak můžeme využít v následující konstrukci, která zajistí sepnutí vždy pouze jednoho z tlačítek *Radio Button*.

```
function radiobutton1_Callback(hObject, eventdata, handles)
off=[handles.radiobutton2,handles.radiobutton3];
vypne_radio(off)
```

11.6.4 Zpětná vazba zaškrtávacího pole Check Box

Jedná se opět o dvoustavové tlačítko, jehož stav je označen zaškrtnutým nebo prázdným polem. Stav tlačítka řídí opět položka `value` s parametry `max` a `min`. Při sepnutém stavu je `max=1`, v opačném případě je `min=1`.

```
function checkbox_Callback(hObject, eventdata, handles)
if get(hObject,'Value')==get(hObject,'Max')
prikazy %Je-li pole zatržené proveď příkazy
else
prikazy%Je-li pole prázdné proveď příkazy
end
```

11.6.5 Zpětná vazba editovatelného pole Edit Text

Pole nejčastěji využíváme pro zadání vstupních hodnot uživatelem. Hodnota je ukládána jako řetězec, se kterou je potom dále pracováno.

```
function edit_Callback(hObject, eventdata, handles)
retezec=get(hObject,'String')
```

Pokud chceme zadávat numerické hodnoty, je vhodné vstupní řetězec ihned převést na číslo, aby s ním bylo možné ihned provádět výpočty.

```
function edit_Callback(hObject, eventdata, handles)
retezec=str2double(get(hObject,'String'))
```

11.6.6 Zpětná vazba popisků Static Text

Nejčastěji jej využijeme k popisu jednotlivých komponent, ale můžeme jej využít například i k zobrazení výsledků či dalších informací.

11.6.7 Zpětná vazba posuvníku Slider

O posuvníku jsme se v textu zmínili již několikrát. V MATLABu jej nejčastěji využíváme společně s komponentou Edit text k nastavení vstupních parametrů. Posuvník nám umožňuje přesně stanovit minimální a maximální rozsah hodnot a díky tomu jednoznačně určit interval, který bude uživatel moci nastavit. Aktuální pozici posuvníku řídí hodnota `value`. Tuto hodnotu je třeba zjišťovat před každou změnou pozice posuvníku.

```
functionslider_Callback(hObject, eventdata, handles)
    hodnota=get(handles.slider,'Value')
```

V položce `value` je uložena minimální – min a maximální – max hodnota posuvníku. Dále je zde definován krok `SliderStep`, který určuje o kolik se posune pozice

posuvníku při kliknutí na jednu z šipek nebo jeho tažení a dále je zde definován posun při kliknutí do dráhy jezdce. Pokud bychom chtěli například nastavit rozsah posuvníku od 1 do 100, krok při kliknutí nebo tažení 0,5, krok při kliknutí do dráhy jezdce 5 a počáteční polohu jezdce na 10, můžeme funkci Callback doplnit o:

```
function slider_Callback(hObject, eventdata, handles)
    hodnota=get(handles.slider,'Value')
slider_step(1)=0.5/(100-1)
slider_step(2)=5/(100-1)
set(handles.slider,'sliderstep',slider_step,'max',100,...
'min',1,'Value',10)
```

Toto nastavení můžeme provést i pomocí *Property Inspectoru*.

11.6.8 Zpětná vazba menu Popup Menu

Popup Menu slouží k výběru z několika známých položek. V tomto menu se po rozkliknutí šipky objeví nabídka, ze které si uživatel vybere právě jednu z možností. Jednotlivé položky jsou v menu definovány pomocí vlastnosti `String`. Každé položce je poté přiřazen index. První položka má hodnotu 1, další 2 atd. Hodnoty indexů jsou opět uchovávány pomocí `value`. Výběr jednotlivých položek lze zajistit pomocí přepínače `switch`.

```
function popupmenu1_Callback(hObject, eventdata, handles)
hodnota=get(hObject,'Value')
switch hodnota
    case 1
prikazy %provede se při výběru první položky
    case 2
prikazy %provede se při výběru druhé položky
    case 3
prikazy %provede se při výběru třetí položky
```

11.6.9 Zpětná vazba menu List Box

V tomto seznamu je možné vybírat jednu, ale také více položek současně. Jednotlivé položky jsou opět ukládány jako řetězce `String`. Pořadí v seznamu opět určuje index. Položka, která má index roven 1 (první položka v seznamu), je automaticky zvýrazněna. Možnost výběru jedné nebo více položek je možné řídit pomocí vlastností min a max.

- Je-li `Max-Min>1` lze vybrat více prvků najednou
- Je-li `Max-Min<=1` lze vybrat pouze jeden prvek

List Box rozlišuje mezi jednoduchým kliknutím a dvojklikem na položku a nastavuje podle toho vlastnost obrázku `SelectionType` na normal nebo open. MATLAB vyhodnocuje zpětnou vazbu *List Boxu* po uvolnění tlačítka myši nebo po uvolnění klávesy, které mění vlastnost `value`. Příkazy zpětné vazby se vykonají po kliknutí nebo dvojkliku na některou z položek. Pokud je povolen výběr více položek, je třeba zajistit, aby zpětná vazba ihned nevykonala požadované úkoly. To lze zajistit přidáním

tlačítka *Pusch Button*, které nám bude sloužit k potvrzení ukončení výběru z nabídky *List Box*.

12 Postup při tvorbě GUI aplikací

Ještě předtím, než se pustíme do vlastního návrhu aplikace pomocí GUIDE editoru, je třeba si vše dobře promyslet. Zejména u složitějších aplikací není vhodné začít ihned navrhovat bez hlubšího prostudování problému. Postup při navrhování GUI aplikace si můžeme rozdělit do několika etap.

1. Rozmyslet si, jak bude naše aplikace vypadat, které veličiny bude třeba ovládat a jaké k tomu použijeme ovládací prvky. Jak budou jednotlivé ovládací prvky rozmístěny, aby aplikace byla uživatelsky přívětivá a přehledná. Vhodný je i náčrtek rozložení prvků.
2. Do *Layout Area* umístíme jednotlivé objekty, upravíme jejich velikost, vzhled a zarovnání. Ovládací prvky je vhodné seskupovat do bloků a přisoudit jim stejné rozměry. Pokud tedy například použijeme několik posuvníků, je vhodné jim nastavit stejnou výšku a šířku a umístit je pod sebe.
3. Jakmile máme připravený návrh, uložíme aplikaci, tedy vygenerujeme dva soubory s příponou **soubor.fig** a **soubor.m** a spustíme ji. Zde si prohlédneme skutečný vzhled aplikace, a pokud jsme se vzhledem spokojeni, můžeme se pustit do úpravy zdrojového kódu.
4. Otevřeme si soubor s příponou soubor.m, do kterého doplníme příslušné zpětnovazební (Callback) funkce.
5. Vytvořenou aplikaci otestujeme pomocí Debugger a pokud je vše v pořádku, tedy aplikace dělá to, co od ní požadujeme, jsme hotovi.

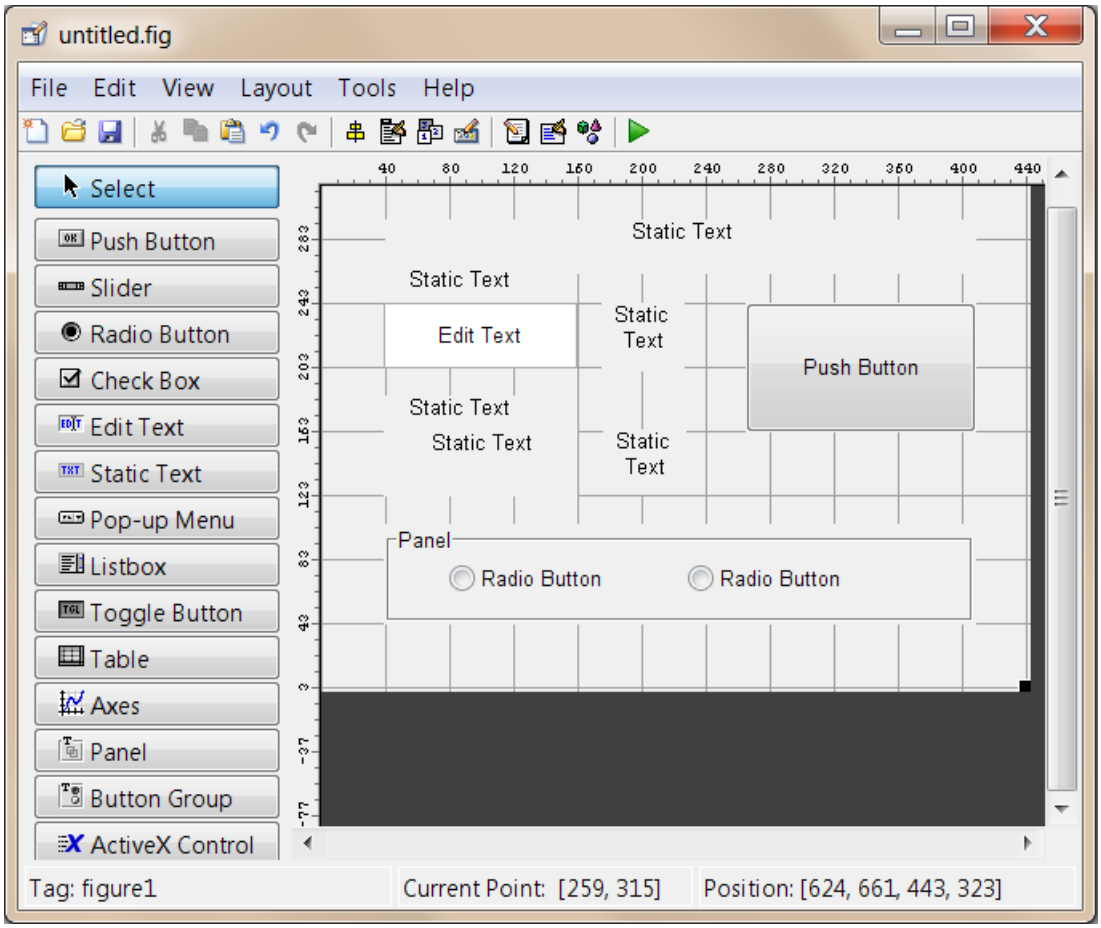
12.1 Návrh aplikace pomocí GUIDE

Návrh aplikace budeme demonstrovat na jednoduchém příkladu aplikace pro převod rychlosti, ve kterém uživatel zadá hodnotu a po stisku tlačítka se zobrazí výsledek. Uživatel se zde také bude moci rozhodnout, zda chce převádět z metrů za sekundu na kilometry za hodinu, nebo opačně.

Pokud jsme dodrželi postup z úvodu kapitoly a máme promyšleno, jak má naše aplikace vypadat, můžeme se pustit do vlastní realizace. Nejprve pomocí palety komponent (*Component Palette*) vybereme příslušné objekty a umístíme je do pracovní plochy *Layout Area*.

Objekty můžeme na pracovní plochu umístit tak, že klikneme na příslušný objekt, například tlačítko (*Push Button*) a kliknutím do pracovní plochy se tento objekt na pracovní ploše ihned objeví. Druhou možností je přetažení objektů z palety komponent do pracovní plochy. Objekt, který jsme právě umístili na pracovní plochu, nebo objekt, který máme právě vybrán, je označen pomocí vodících bodů a lze s ním dále manipulovat. Můžeme měnit jeho velikost a také umístění na pracovní ploše.

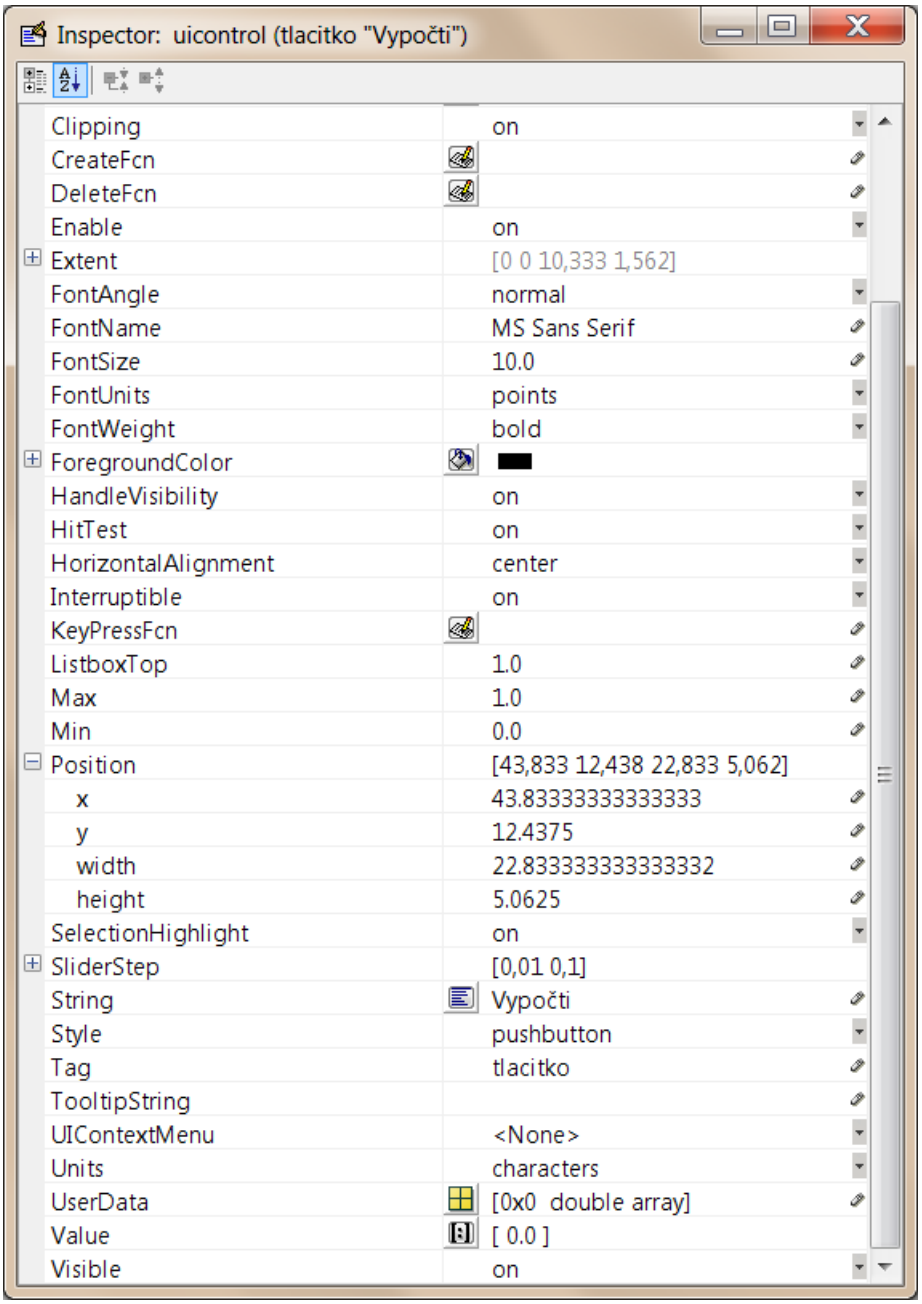
Pro náš návrh aplikace využijeme následující objekty: 1x Edit Text, 6x Static Text, 1x Push Button, 1x Panel a 2x Radio Button. Jednotlivé objekty umístíme do *Layout Area* a pomocí *Align Objects* zarovnáme. Výsledek vidíme na obrázku 12.1.



Obr. 12. 1: Rozvržení GUI aplikace

Pokud jsme s rozvržením jednotlivých objektů spokojeni, můžeme se pustit do změny jejich vlastností. Nastavení jednotlivých objektů provádíme pomocí *Property Inspectoru*. Pokud budeme chtít vyvolat *Property Inspector* například tlačítka (*Push Button*), stačí na něj pouze dvakrát poklepat. Otevře se nám již známé okno, ve kterém se ihned můžeme pustit do změny vlastností tlačítka. Nejdůležitější položkou je vlastnost `Tag`, která slouží jako identifikátor objektu a také později vytvořené zpětnovazební funkce. My si zde tuto vlastnost nastavíme na `tlacitko` (obr. 12.2). Identifikátor objektu je také uveden v záhlaví každého *Property Inspectoru*, což výrazně usnadní orientaci mezi vyvolanými *Property Inspectory* a jejich objekty.

V našem návrhu jsme dále pokračovali změnou textu zobrazeného na tlačítku na vypočti. K tomu slouží vlastnost `String`. Vhodné je také změnit velikost písma, aby byl text lépe čitelný. `FontSize` nastavíme na deset bodů a zvýrazníme pomocí `FontWeight`, kde provedeme změnu z `normal` na `bold`. Kromě těchto vlastností můžeme samozřejmě nastavit spoustu dalších, jako barvu písma, barvu pozadí, typ písma, výšku a šířku objektu, jeho pozici, minimální a maximální hodnoty apod. Funkce jednotlivých vlastností si můžete vyzkoušet sami. Výsledné nastavení tlačítka můžeme vidět na obrázku 12.2.

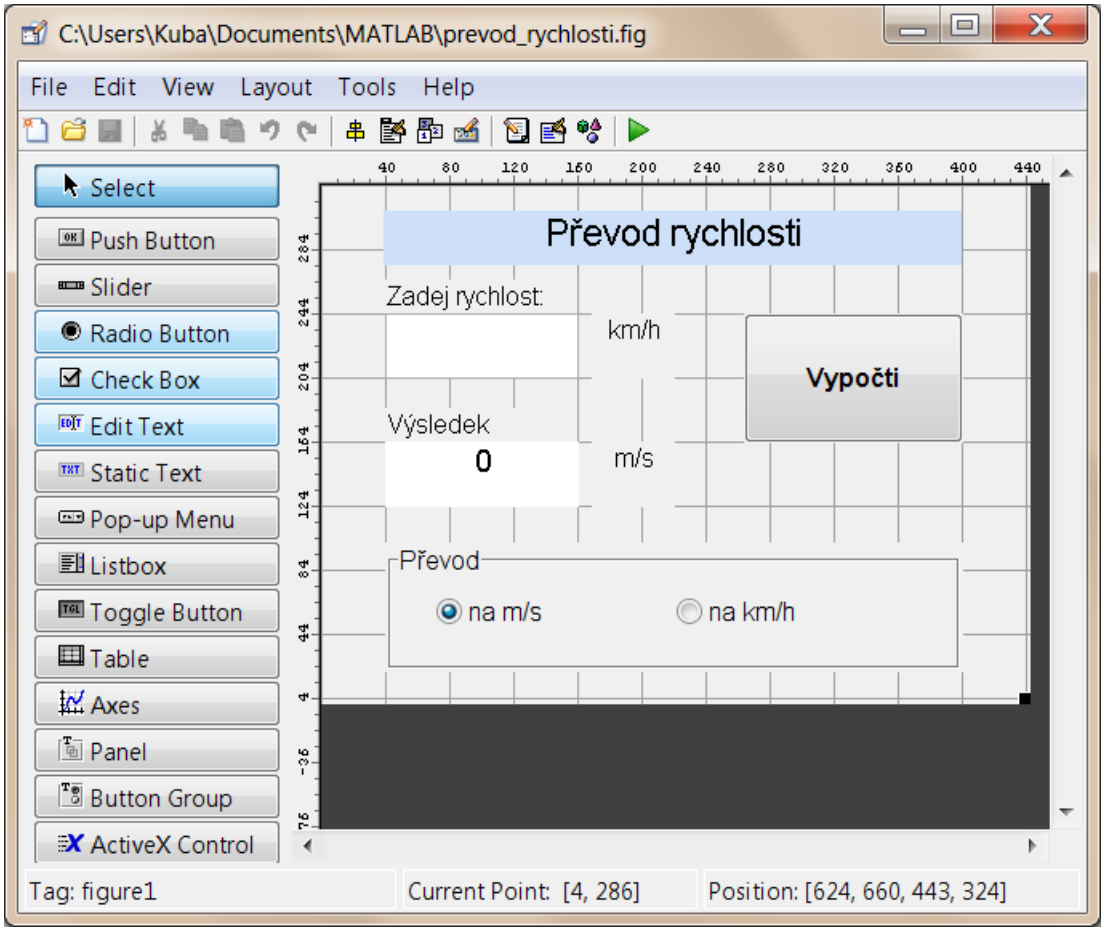


Obr. 12. 2: Nastavení vlastností tlačítka Push Button

Některé položky v *Property Inspectoru* obsahují ještě rozbalovací nabídky, jako například `Position`. Pokud si tuto nabídku otevřeme, zjistíme, že obsahuje položky `x, y, width, height`. Funkce této vlastnosti je nám již známá z kapitoly 10, kde jsme pozici každého objektu a jeho rozměry popisovali pomocí vektoru o čtyřech prvcích. V *Property Inspectoru* je tento vektor tvořen pomocí položek `x` a `y`, které se vztahují k umístění objektu v *Layout Area* a `width` a `height`, které určuje jeho šířku a výšku. Dalšími rozbalovacími nabídkami jsou například `BackgroundColor` a `ForegroundColor`, jejichž funkce si již zvládneme prostudovat sami.

Nyní můžeme provést nastavení u všech ostatní objektů, které jsme umístili na pracovní plochu GUIDE editoru. Vyvoláme si tedy postupně příslušné *Property Inspectory*

a provedeme změny jednotlivých vlastností. Výsledné nastavení může vypadat například takto:



Obr. 12. 3: Nastavení vzhledu aplikace pomocí GUIDE editoru

Nyní máme připravený vzhled aplikace. Abychom mohli pokračovat v práci, je třeba aktuální stav uložit pomocí *File →Save as*. Dotaz na uložení se vygeneruje také, pokud se aplikaci v tomto stavu pokusíme spustit. Zadáme tedy název souboru, v našem případě **prevod_rychlosti** a uložíme. Pokud se nyní podíváme do *Current Directory*, zjistíme, že MATLAB projekt uloží do dvou souborů. Do souboru **prevod_rychlosti.fig** se uloží aktuální vzhled a automaticky vygenerovaný zdrojový kód se vloží do souboru **prevod_rychlosti.m** . Vytvořený m-soubor se také ihned automaticky spustí spolu s naší GUI aplikací.

Jestliže vzhled naší aplikace neodpovídá našim představám, můžeme provést ještě dodatečné úpravy. Pro opětovné spuštění GUIDE editoru přejdeme do hlavního okna MATLABu, kde si v *Current Directory* najdeme soubor **prevod_rychlosti.fig** a klikneme na něj pravým tlačítkem myši. V menu, které se nám objevilo, zvolíme položku *Open in GUIDE*. Soubor můžeme otevřít také pomocí *GUIDE Quick Start*, kde si přepneme na záložku *Open Existing GUI* a vybereme odpovídající soubor. „Doladíme“ vzhled a uložíme změny.

Nyní již máme vzhled aplikace podle našich představ a můžeme se pustit do vzájemného provázání jednotlivých grafických objektů. Otevřeme si soubor

prevod_rychlosti.m, kde je zatím pouze automaticky generovaný zdrojový kód jednotlivých objektů. Pokud si takto připravenou aplikaci spustíme (stiskneme *Run* nebo klávesu *F5*), ve vytvořeném GUI můžeme zapisovat do pole *Edit Text* a klikat na jednotlivá tlačítka, ale to je asi tak vše. Nyní je třeba vytvořit zpětné vazby mezi objekty. V našem případě budeme chtít, aby uživatel zadal rychlost a po stisku tlačítka vypočti se zobrazil výsledek. Dále aby bylo možné převádět nejen na metry za sekundu ale i na kilometry za hodinu.

```
function varargout = prevod_rychlosti(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @prevod_rychlosti_OpeningFcn, ...
                   'gui_OutputFcn',  @prevod_rychlosti_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 prevod_rychlosti is made visible.
function prevod_rychlosti_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% Choose default command line output for prevod_rychlosti
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

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


% --- Outputs from this function are returned to the command line.
function varargout = prevod_rychlosti_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);

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

Obr. 12. 4: Automaticky generovaný zdrojový kód aplikace prevod_rychlosti

V souboru **prevod_rychlosti.m** pomocí kolečka rolujeme do jeho spodní části. Zde je automaticky vytvořeno několik funkcí, které nám budou sloužit k nastavení zpětné vazby mezi objekty. Jednotlivé propojení grafických objektů zajišťuje funkce Callback, která se v našem příkladu automaticky vygenerovala pro objekty *Edit Text*, kde budou zapisovány hodnoty pro převod, tlačítko *Pusch Button*, které bude sloužit k výpočtu

a zobrazení výsledků a dvě tlačítka *Radio Button*, které budeme používat pro přepínání převodního vztahu na kilometry za hodinu nebo metry za sekundu.

```
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

% --- 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

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

% --- 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

% --- 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
```

Obr. 12. 5: Zpětnovazební funkce objektů aplikace prevod_rychlosti

Jednotlivé Callback funkce jsou složeny z několika parametrů a mají následující formu: `function ObjectName_Callback(hObject, eventdata, handles)`. Položka `ObjectName` je identifikátor objektu, pro který budeme zpětnou vazbu vytvářet. Identifikátor každého objektu můžeme zjistit pomocí vlastnosti `Tag` v *Property Inspectoru* vybraného objektu. Pokud si tedy budeme vytvářet vlastní identifikátory, je nutné zajistit, aby zde žádný z názvů nebyl dvakrát.
`hObject` je proměnná handle zpětnovazebního objektu. Jeho hodnotu můžeme zjistit pomocí `gcbo` (get handle to current callback object).

Handles je struktura, kde jsou uložena všechna nastavení objektů obsažených v GUI. Obsahuje handle jednotlivých objektů, na které je odkazováno pomocí identifikátoru. Tato struktura se vytvoří při spuštění GUI a je uložena v aplikačních datech obrázku užitím GUIDATA. Kopie struktury se předává každé zpětné vazbě Callback. Pokud budeme tuto strukturu během činnosti zpětné vazby měnit, je třeba každou změnu uložit, aby další zpětné vazby tuto změnu rozeznaly. Pro uložení změn můžeme použitím příkazu `guidata(hObject, handles)`, který přepíše původní kód struktury.
Kromě zpětnovazebních funkcí zde můžeme najít ještě funkci, která slouží k nastavení počátečních podmínek objektu. Tako funkce má obdobnou formu jako funkce Callback. Liší se pouze klíčovým slovem `CreateFcn`: `function ObjectName_CreateFcn(hObject, eventdata, handles)`. Zde je možné například nastavit text, který se zobrazí v objektu *Edit Text* při startu aplikace, nebo určit, které tlačítko *Radio Button* bude stisknuté. Funkce tedy složí k nastavení počátečních podmínek aplikace.

Nyní, když známe význam jednotlivých funkcí, můžeme se pustit do vlastního vytváření zpětných vazeb. Tyto funkce jsou totiž pouze kostrou, do které je třeba doplnit příslušný zdrojový kód.

V naší aplikaci nejprve doplníme zdrojový kód pro pole *Edit Text*, které má identifikátor `edit1`. Najdeme si tedy funkci `function edit1_Callback(hObject, eventdata, handles)` a doplníme ji o následující pžíkazy:

```
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)
val=str2double(get(hObject,'String'));
if isnumeric(val)
    handles.val=val;
else
    handles.numberOfErrors = handles.numberOfErrors +1;
    set(handles.edit1,'String',...
        [handles.errorString,num2str(handles.numberOfErrors)]);
end
guidata(hObject,handles)
```

Obr. 12. 6: Zdrojový kód pole Edit Text

Editovatelný text pracuje s řetězci. Pokud do pole budou zadávány numerické hodnoty pro výpočty, musíme zadanou hodnotu převést na číslo. Tento převod uskuteční řádek `val=str2double(get(hObject,'String'))`, který číselnou hodnotu uloží do proměnné `val`. Dále je zde podmínka, která testuje, zda je proměnná `val` číslo. Pokud ano, uloží se proměnná `val` do struktury `handles`, aby s ní bylo možno pracovat i u jiných objektů. Pokud hodnota nevyhovuje, provede se větev `else` a program zobrazí chybu. V textovém poli se objeví hlášení NaN.

Pro vynulování proměnné val při startu aplikace můžeme využít funkci `function edit1_CreateFcn(hObject, eventdata, handles)`, která je určena k nastavení počátečních podmínek při startu aplikace. Hodnotu proměnné `val` zde nastavíme na `0` a uložíme do struktury handles.

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

Obr. 12. 7: Nastavení počátečních podmínek objektu Edit Text

Nyní se můžeme pustit do nastavení zpětných vazeb pro tlačítko, po jehož stisknutí se zobrazí výsledek. V tomto případě jsme změnili identifikátor funkce na `tlacitko`. Změna se po uložení ihned projeví a název zpětnovazební funkce odpovídá identifikátoru. Funkci doplníme o následující příkazy:

```
function tlacitko_Callback(hObject, eventdata, handles)
% hObject    handle to tlacitko (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if get(handles.radiobutton1,'Value')==1
    set(handles.text1,'String',handles.val/3.6);
else
    set(handles.text1,'String',handles.val*3.6);
end
```

Obr. 12. 8: Zdrojový kód funkce tlacitko

Tlačítko naší aplikace bude sloužit k realizaci vlastního výpočtu a zobrazení výsledků. Nejprve je však třeba určit, který z převodů se bude realizovat. Za tímto účelem je zde opět podmínka, která testuje, zda je pole *radiobutton1* aktivní. Hodnota `value` je `1`. Pokud je podmínka splněna do objektu *StaticText1*, se vypíše výsledek podílu čísla zadaného do *Edit text* a čísla 3,6. Pokud je aktivní pole *radiobutton2*,vypíše se výsledek součinu čísla zadaného do *Edit text* a čísla 3,6.

Nyní si vytvoříme funkci, která zajistí, že aktivní bude vždy pouze jedno z polí *Radio Button*. Pokud tedy klikneme na neaktivní pole *Radio Button*, hodnota `value` druhého pole se automaticky nastaví na 0.

```
function mutual_exclude(off)
    set(off,'Value',0)
```

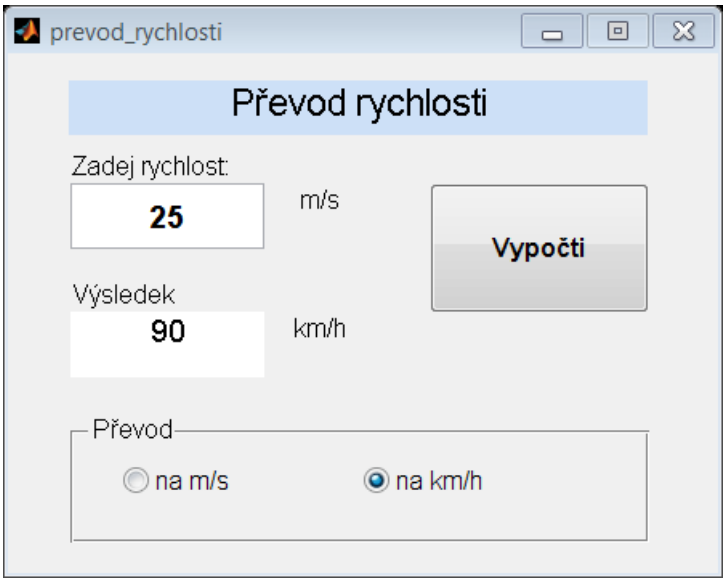
Obr. 12. 9: Funkce pro aktivaci pouze jednoho z polí Radio Button

Tuto funkci nyní využijeme při vytváření zdrojového kódu funkcí `radiobutton1` a `radiobutton2`, které budou sloužit ke změně převodního vztahu z metrů za sekundu na kilometry za hodinu a opačně.

```
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)
off = [handles.radiobutton2];
mutual_exclude(off)
set(handles.text2,'String','m/s');
set(handles.text3,'String','km/h');
% Hint: get(hObject,'Value') returns toggle state of radiobutton1

% --- 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)
off = [handles.radiobutton1];
mutual_exclude(off)
set(handles.text2,'String','km/h');
set(handles.text3,'String','m/s');
% Hint: get(hObject,'Value') returns toggle state of radiobutton2
```

Obr. 12. 10: Zdrojový kód funkce radiobutton1 a radiobutton2

V obou případech nejprve zavoláme funkci `mutual_exclude`, která zajistí, že bude aktivní pouze jedno pole *Radio Button*. Pokud tedy uživatel stiskne *radiobutton1*, *radiobutton2* se automaticky nastaví do stavu vypnuto a opačně. Dále je zde třeba ošetřit nastavení jednotek. Tedy položky *text2* a *text3*. To provedeme pomocí dvou příkazů `set`, které zajistí jejich prohození. Tím je aplikace hotová a můžeme ji vyzkoušet.



Obr. 12. 11: Aplikace prevod_rychlosti

13 Modelování pohybu kyvadla

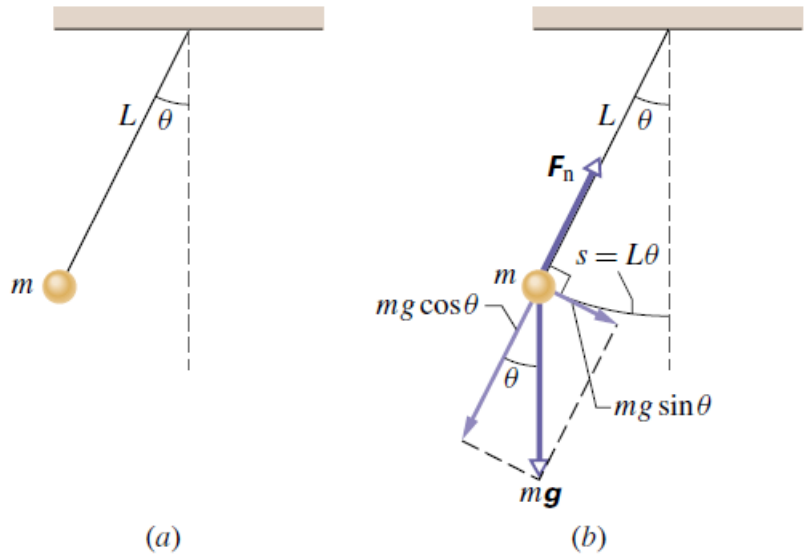
13.1 Úvod do problémů

Soustava, která umožňuje vykonat kmitavý pohyb, se nazývá mechanický oscilátor. Typickým příkladem mechanického oscilátoru je závaží zavěšené na pružině. V případě, že dojde k vychýlení závaží z klidového stavu, začne po jeho uvolnění měnit svou polohu ve vertikálním směru střídavě nahoru a dolů nebo-li bude kmitat.

Za ideálních podmínek, kdy bude gravitační pole homogenní, pružina lineární (síla vyvolaná pružinou je přímo úměrná výchylce) a proti pohybu závaží nebude působit žádný odpor, vznikne harmonický oscilátor. To znamená, že závaží bude kmitat harmonicky.

13.2 Matematické kyvadlo

Matematické kyvadlo je idealizovaný model oscilátoru. Jedná se o hmotný bod m (závaží kyvadla), který je zavěšený na nehmotném pevném vlákně délky L. V soustavě se dále zanedbává odpor prostředí a tření v závěsu kyvadla. Situace je znázorněna na obrázku 13.1.



Obr. 13. 1: Matematické kyvadlo

Pokud dojde k vychýlení závaží a jeho uvolnění, kyvadlo se volně houpe tam a zpět v horizontálním směru, tj. doleva a doprava od svislé přímky vedené závěsem kyvadla. Tato přímka pak představuje rovnovážnou polohu kyvadla, tedy stav, kdy je kyvadlo v klidu.

Na hmotný bod kyvadla působí tíhová síla $m \cdot g$ kterou můžeme rozložit na dvě složky. Složka $m \cdot g \cdot \sin\theta$ působí proti směru pohybu kyvadla a snaží se o jeho návrat do rovnovážné polohy. Složka $m \cdot g \cdot \cos\theta$ představuje sílu, kterou hmotný bod působí na závěs. Proti této síle působí síla F_n, která je stejně velká, ale opačného směru. Výchylka kyvadla je tedy určena pouze tečnou složkou tíhové síly. Směr okamžité rychlosti

kyvadla je tečna k oblouku opisovaného hmotným bodem. Z druhého Newtonova pohybového zákona platí:

$$F = m \cdot a \tag{1}$$

Protože se při pohybu kyvadla uplatňuje pouze tečná složka tíhové síly, můžeme rovnici (1) upravit do tvaru:

$$F = m \cdot a = -m \cdot g \cdot \sin\theta \tag{2}$$

$$a = -g \cdot \sin\theta \tag{3}$$

Směr síly a směr pohybu hmotného bodu je vždy opačný, proto je i zrychlení uvedeno se záporným znaménkem. Zrychlení ovlivňuje změnu úhlu θ. Podle vzorce obloukové délky s je oblouková délka:

$$s = L \cdot \theta \tag{4}$$

Rychlost a zrychlení hmotného bodu můžeme určit pomocí první a druhé derivace:

$$v = \frac{\mathrm{d}s}{\mathrm{d}t} \tag{5}$$

$$v = L \cdot \frac{\mathrm{d}\theta}{\mathrm{d}t} \tag{6}$$

$$a = \frac{d^2\theta}{\mathrm{d}t^2} \tag{7}$$

Po dosazení z rovnice (3):

$$L \cdot \frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} = -g \cdot \sin\theta \tag{8}$$

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \frac{g}{L} \cdot \sin\theta = 0 \tag{9}$$

Označíme li $\omega^2 = \frac{g}{L}$ dostáváme pohybovou rovnici matematického kyvadla:

$$\frac{d^2\theta}{\mathrm{d}t^2} + \omega^2 \cdot \sin\theta = 0 \tag{10}$$

Pro zjednodušení řešení diferenciální rovnice se nyní omezíme pouze na malé výchylky. Nechť tedy pro úhel θ platí: $\theta \leq 5°$.

$$\sin\theta \approx \theta \tag{11}$$

Úpravou rovnici (10) nyní dostáváme diferenciální rovnici druhého řádu:

$$\frac{\mathrm{d}^2\theta}{\mathrm{d}t^2} + \omega^2 \cdot \theta = 0 \tag{12}$$

Obecným řešením této rovnice je vztah:

$$y(t) = C_1 \cdot \cos(\omega_0 \cdot t) + C_2 \cdot \sin(\omega_0 \cdot t) \quad (13)$$

S počátečními podmínkami $y = y_0$ a $\frac{dy}{dt} = 0$ je řešení této rovnice:

$$y = y_0 \cdot \cos(\omega_0 \cdot t) \quad (14)$$

Rovnice $\omega^2 = \frac{g}{L}$ se nazývá vlastní frekvence kyvadla. Pokud tuto rovnici dále upravíme, získáme rovnici pro periodu kmitů matematického kyvadla.

$$\left(\frac{2 \cdot \pi}{T}\right)^2 = \frac{g}{L} \quad (11)$$

$$T = 2 \cdot \pi \cdot \sqrt{\frac{L}{g}} \quad (12)$$

13.3 Aplikace Kyvadlo

Úkolem naší aplikace bude simulovat pohyb kyvadla a jeho úhlové výchylky. Uživatel si zde dále bude moci zobrazit aktuální výchylku, síly působící na kyvadlo a jeho okamžitou rychlost a zrychlení. Dále zde bude možné nastavit délku závěsu kyvadla, hmotnost a počáteční úhel vychýlení.

Nyní máme stanoveno, co bude cílem naší aplikace a můžeme se pustit do vlastního návrhu. Ještě před tím, než však spustíme GUIDE editor, je vhodné si promyslet, jak bude naše aplikace vypadat. Kde budou rozmístěny jednotlivé objekty na pracovní ploše *Layout Area*, které ovládací prvky, budou uživateli sloužit k nastavení veličin a jak budou tyto prvky rozmístěny.

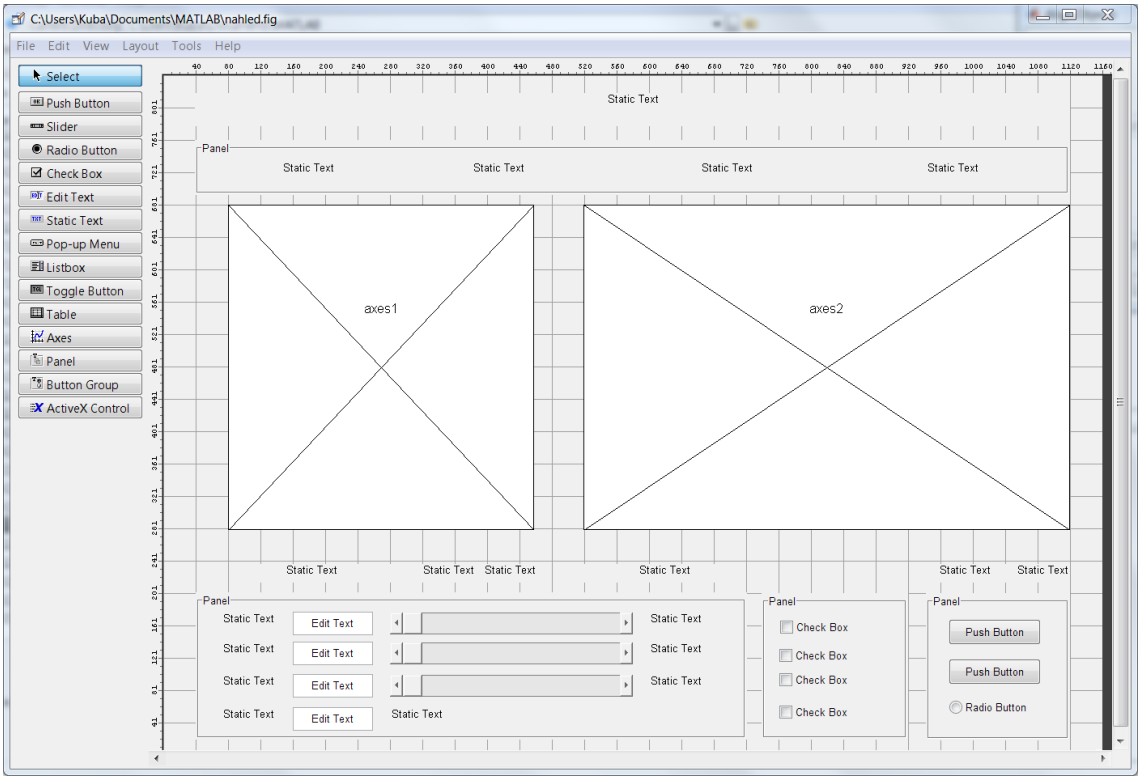
Máme-li vše připravené, můžeme spustit GUIDE editor a začít umisťovat jednotlivé komponenty do pracovní plochy *Layout Area*. Mějme na paměti, že jednotlivé ovládací prvky by měly být seskupeny do logických celků, aby se uživateli aplikace co nejlépe ovládala. Proto si jednotlivé ovládací prvky rozmístíme na tři komponenty *Panel*. První bude sloužit k nastavení parametrů kyvadla (délka závěsu, hmotnost, úhel), druhý k zobrazení průběhů (výchylka, rychlost, zrychlení, síla) a třetí bude sloužit k vlastnímu ovládání, tedy spuštění, zpomalení a zastavení aplikace (obr.13.2).

Pro nastavení parametrů kyvadla využijeme editovatelná pole a posuvníky, aby uživatel mohl vstupní hodnoty zadávat pomocí klávesnice nebo kliknutím myši. Pro každý z parametrů kyvadla bude k dispozici jeden objekt *Edit Text* a jeden objekt *Slider*. Celkem tedy použijeme tři posuvníky a tři editovatelná pole. Panel ještě doplníme informací o gravitačním zrychlení, které bude konstantní. K jeho zobrazení nám tedy postačí objekt *Static Text*.

Druhý panel bude sloužit ke zobrazení průběhů kyvadla. Bude obsahovat čtyři zatrhávací pole *Check Box*, která budou sloužit k zobrazení vypočtených průběhů, tedy výchylky, rychlosti, zrychlení a tečné složky síly.

Třetí skupinu ovládacích prvků budou tvořit dvě tlačítka *Push Button* a jedno pole *Radio Button*. Zde bude možné aplikaci spustit, zastavit a zpomalit.

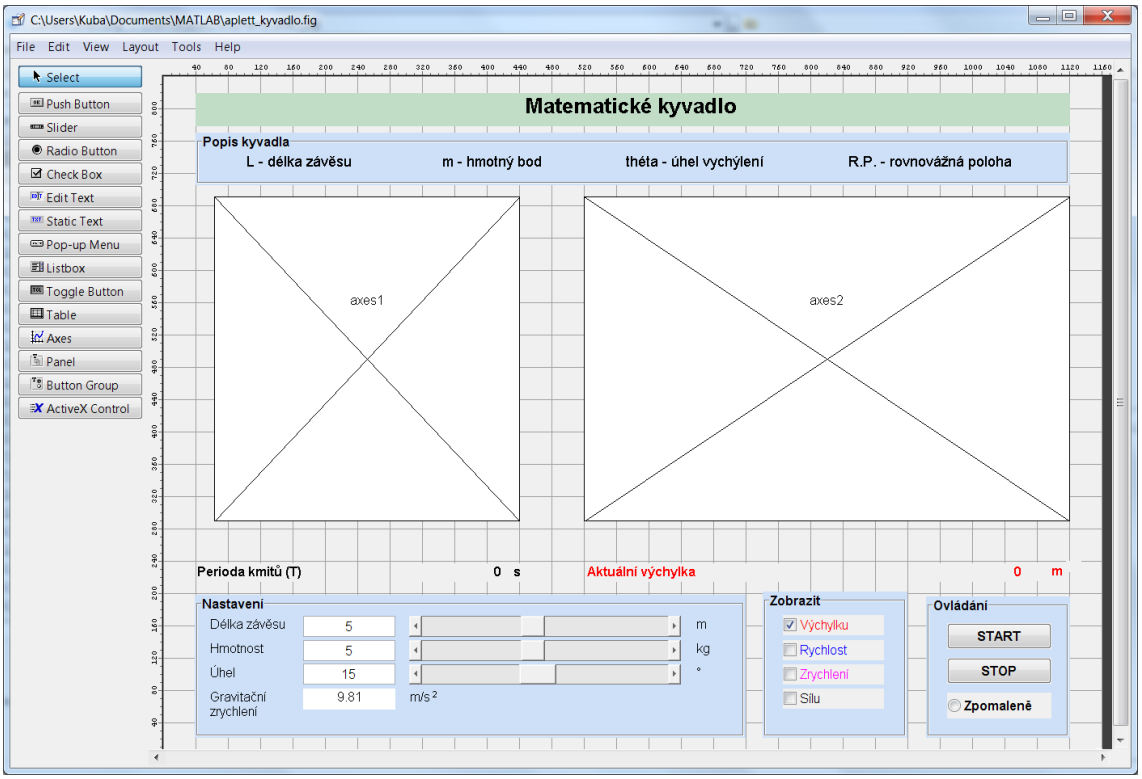
Návrh ještě doplníme o dva objekty Axes, které budou sloužit k znázornění pohybu kyvadla a jednotlivých průběhů a samozřejmě objekty *Static Text*, které využijeme k popisu veličin a jejich jednotek.



Obr. 13. 2: Rozložení objektů na pracovní ploše

Nyní máme na pracovní ploše (*Layout Area*) umístěny všechny prky potřebné pro naši aplikaci a můžeme se pustit do jejich úprav. Ty budeme provádět pomocí jejich *Property Inspectoru*. Jednotlivé komponenty popíšeme, nadefinujeme jejich vzhled, velikosti a umístění a nastavíme počáteční hodnoty. Jak změnit nastavení vlastnosti jednotlivých objektů jsme si popsali již v předchozí kapitole, proto není nutné se jimi zde znovu podrobně zabývat. Neměli bychom také zapomenout všechny objekty patřičně zarovnat pomocí nástroje *Align Objects*.

Jsme-li s dosavadními výsledky naší práce spokojeni, můžeme GUI aplikaci uložit například pod názvem **aplett_kyvadlo** a pustit se do úprav automaticky generovaného zdrojového kódu.



Obr. 13. 3: Nastavení vlastností objektů

Automaticky generovaný zdrojový kód se nachází v souboru **aplett_kyvadlo.m**, kam nyní začneme doplňovat naše příkazy. Nejprve je třeba upravit funkci, která se volá po startu aplikace `aplett_kyvadlo_OpeningFcn`. Zde je třeba nadefinovat vstupní hodnoty a zajistit jejich zpracování, aby se nám kyvadlo ihned po startu aplikace zobrazilo.

```
%********************** Uvodní nastavení parametrů **********************
function aplett_kyvadlo_OpeningFcn(hObject, eventdata, handles, varargin)


delka_zavesu=5;                         %vstupní hodnoty
hmotnost=5;
uhel=15;

axes(handles.axes1);                    %ulože a zobrazení dat
Vektor=[delka_zavesu hmotnost uhel];
set(handles.axes1,'UserData',Vektor)
kresli_kyvadlo(delka_zavesu,hmotnost,uhel) %zobrazení kyvadla
popis (delka_zavesu,hmotnost,uhel)

axes(handles.axes2);                    %nastavení grafu
   xlabel('t [s]')
   ylabel('y [m]')
   grid on

handles.output = hObject;
guidata(hObject, handles);
```

Obr. 13. 4: funkce aplett_kyvadlo_OpeningFcn

Za tímto účelem jsou zde volány funkce `kresli_kyvadlo` (obr. 13.5) a `popis` (obr 13.6), které pomocí objektu *axes1* zajistí zobrazení kyvadla a přidání popisků.

```
% *************** Vypocet poloh a vykresleni kyvadla *********************
function kresli_kyvadlo(l,m,fi)
%Funkce slouzi k vypoctu poloh a zobrazeni poloh zavesu, hmotneho bodu a
%uchyceni kyvadla
    if fi>0                     %vektory oblouku kyvadla
        oblouk=0:0.5:fi;        %kladná hodnota
    else
        oblouk=0:-0.5:fi;       %záporná hodnota
    end
    oblouk_x=cos((270-oblouk)*pi/180)*l;  %Souřadnice oblouku
    oblouk_y=sin((270-oblouk)*pi/180)*l;

    uhel_rad=(270-fi)*pi/180; %vypocet uhlu v radianech
    x=cos(uhel_rad)*l;        %Souradnice polohy
    y=sin(uhel_rad)*l;
    delka=360;                %Pro vypocet kruznice (hmotneho bodu)

    axis equal;               %Graficky vystup
    axis ([-6 6 -11.5 0.5]);
    axis off

    cla                       %smazani predchozich grafu
    hold on

      fill([-6 -6 6 6],[0 0.3 0.3 0],[0.9 0.5 0.1],'LineWidth',1,...
           'LineStyle','-');
      line ([0 0],[0 -l],'LineWidth',1,'LineStyle','-.');
      plot (0,-l,'o','MarkerSize',9.5)
      line ([0 x],[0 y],'LineWidth',1.5);
      plot(oblouk_x,oblouk_y,'LineStyle','-.')
      [xp,yp]=kruznice(x,y,0.25,delka);
      patch (xp, yp,[0.9 0.9 0.2]);
    hold off
```

Obr. 13. 5: funkce kresli_kyvadlo

```
%************************ Popis veličin kyvadla ************************
function popis (l,m,fi)
    uhel_rad=(270-fi)*pi/180; %vypocet uhlu v radianech
    x=cos(uhel_rad)*l;        %Souradnice polohy
    y=sin(uhel_rad)*l;
    text (x/2-0.7,y/2,'\fontsize{12} L')    %popis kyvadla
    text (x-1.2,y-0.7,'\fontsize{12} m')
    text (0.2,-l/4,'\fontsize{12} \theta')
    text (-0.6,-l-0.8,'\fontsize{12} R.P.')
```

Obr. 13. 6: funkce popis

Protože budeme chtít, aby se po každém zásahu uživatele, po každé změně parametrů kyvadla tyto změny ihned projevily, je funkce `aplett_kyvadlo_OpeningFcn` doplněna o proměnnou `Vektor`, která zajistí, že jsou vždy k dispozici aktuální hodnoty proměnných `delka_zavesu`, `hmotnost`, `uhel`. Funkce `kresli_kyvadlo`

a `popis`, které jsou volány při každé změně parametrů kyvadla, tak vždy zobrazí jeho aktuální polohu.

Máme-li zajištěno zobrazení při startu aplikace, můžeme se pustit do tvorby kódu pro ovládací prvky kyvadla, tedy objekty *Edit Text* a *Slider*. Zde je nutné zajistit, aby změna parametru posuvníku způsobila i změnu parametrů editovatelného pole a opačně. Ve zpětné vazbě `slider1_Callback` je proto nastaven handle pole `edit1` na řetězec `String`, který zobrazuje aktuální hodnotu `Value` pozice Slideru (obr.13.7). Zpětná vazba `edit1_Callback`, která zajišťuje nastavení jezdce posuvníku při zadání hodnoty do pole *Edit Text*, bude o trošku složitější. Musíme totiž ošetřit stav, kdy uživatel nezadá numerickou hodnotu, nebo zadá numerickou hodnotu, která bude mimo rozsah posuvníku. O to se stará podmínka `if - else`, která při nesprávném zadání zobrazí chybové hlášení a vyzve k opětovnému zadání parametru kyvadla (obr.13.8).

O to, aby se změna nastavení vlastností kyvadla ihned projevila, se opět stará proměnná `Vektor`. Zde jsou nejprve načteny původní parametry kyvadla, které jsou poté přepsány nově nastavenou hodnotou. Nová data jsou následně opět uložena a zobrazena.

```
%******************** Posuvník pro nastavení délky závěsu ******************
function slider1_Callback(hObject, eventdata, handles)
  %Propojení posuvníku a editovatelného pole
  set(handles.edit1,'String',...
      num2str(get(handles.slider1,'Value')));

  %Načtení dat po změně délky závěsu
  axes(handles.axes1);
    Vektor=get(handles.axes1,'UserData');
    delka_zavesu=get(hObject,'Value');
    hmotnost=Vektor(2);
    uhel=Vektor(3);
    Vektor(1)=delka_zavesu;

  %Uložení a zobrazení změny délky závěsu
    set(handles.axes1,'UserData',Vektor);
    kresli_kyvadlo(delka_zavesu,hmotnost,uhel)
    popis (delka_zavesu,hmotnost,uhel)
    guidata(hObject, handles);
```

Obr. 13. 7: funkce pro posuvník

```
%************ Editovatelné pole pro nastavení délky závěsu ***************
function edit1_Callback(hObject, eventdata, handles)
  %Načtení délky závěsu editovatelné pole
  val=str2double(get(handles.edit1,'String'));

  if isnumeric(val)&& length(val)==1 &&...
     val>=get(handles.slider1,'Min') &&...
     val<=get(handles.slider1,'Max')
        set(handles.slider1,'Value',val);

    %Načtení dat po změně délky závěsu
    axes(handles.axes1);
      Vektor=get(handles.axes1,'UserData');
      delka_zavesu=val;
      hmotnost=Vektor(2);
      uhel=Vektor(3);
      Vektor(1)=val;

      %Uložení a zobrazení délky závěsu
      set(handles.axes1,'UserData',Vektor);
      kresli_kyvadlo(delka_zavesu,hmotnost,uhel)
      popis (delka_zavesu,hmotnost,uhel)
  else
    %Při chybném zadání vstupu
    warndlg('Zadej delku v rozsahu 1-10','Chyba vstupu')
    set(handles.edit1,'String','');
  end

    guidata(hObject, handles);
```

Obr. 13. 8: funkce pro editovatelné pole

Funkce na obrázku 13.7 a 13.8 jsou určeny pro nastavení délky závěsu kyvadla. Funkce pro nastavení hmotnosti a úhlu, jsou však téměř totožné (liší se pouze indexy objektů), proto je zde uvádět nebudeme. Jejich zdrojový kód je možné nalézt v příloze této práce, kde je kompletní výpis kódu aplikace **aplett_kyvadlo**.

Dále je třeba zajistit, aby bylo možné vybrat vždy pouze jedno ze zatrhávacích polí *Check Box*. K tomu nám slouží funkce `pouze_jedno`, která je volaná ve zpětné vazbě jednotlivých polí *Check Box*.

```
%********************* Povoluje vybrat pouze jeno pole ********************
function pouze_jedno(off)
  set(off,'Value',0)
```

Obr. 13. 9: funkce pro výběr pouze jednoho pole

Ta je ještě doplněna o několik změn řetězce `String` objektů *Static Text*, aby zobrazené hodnoty odpovídaly právě vybranému poli *Check Box* a grafu zobrazenému v *Axes2*.

```
%********************** Zatrhávací pole výchylka ***********************
function checkbox1_Callback(hObject, eventdata, handles)
  off=[handles.checkbox2,handles.checkbox3,handles.checkbox4];
  pouze_jedno(off) %zatržené checkbox1

   %Zobrazení aktuální veličiny
   set(handles.text8,'String','Okamžitá výchylka','ForegroundColor','red')
   set(handles.text9,'ForegroundColor','red')
   set(handles.text19,'String','m','ForegroundColor','red')
   set(handles.text23,'String','')

   axes(handles.axes2)
   xlabel('t [s]')
   ylabel('y [m]')

   guidata(hObject, handles);
```

Obr. 13. 10: zpětná vazba pole Check Box

Opět zde uvádíme pouze zpětnou vazbu pro výchylku kyvadla. Zdrojový kód pro rychlost, zrychlení a sílu je téměř totožný a můžeme ho nalézt v příloze.

Nyní je třeba vytvořit funkci pro zobrazení průběhu kyvadla. K tomu využijeme diferenciální rovnici, s jejíž pomocí vypočteme hodnoty grafu pro interval `[0, T/2]`. Tento interval jsme zvolili záměrně, aby bylo možné dobře sladit pohyb kyvadla s aktuálně zobrazenou hodnotou v grafu při řešení obsluhy tlačítka START.

```
%*************** Soustava dif. rovnic popisujicich kyvadlo*****************
function [dxdt] = dif_rce(t,x)
%ds/dt = v
%dv/dt = -g/l*s
global dlk;
g=9.81;
    dxdt=zeros(2,1);
    dxdt(1)=x(2);
    dxdt(2)=-g/dlk*x(1);
```

Obr. 13. 11: Soustava diferenciálních rovnic kyvadla

```
%****************** Funkce pro vykreslení průběhů ***********************
function [t,s,v,a,Ft] = kresli_graf (l,m,fi,g)
s0=(fi*pi/180)*l; %m
v0=0; %m/s

%cas reseni
T=2*pi*sqrt(l/g);

%omega
w=(2*pi)/T;

%reseni diferencialni rovnice
options=odeset('RelTol',1e-9);
[t, x]=ode45(@dif_rce, [0, T/2], [s0, v0], options);

%prevod na fyzikalni veliciny
s=x(:,1);
v=x(:,2);
a=-s*w^2;
Ft=-m*g*s/l;
```

Obr. 13. 12: Funkce pro výpočet bodů grafu

Závěrečným krokem bude vytvoření obsluhy tlačítek START a STOP. Po stisku tlačítka START začne naše kyvadlo kmitat a zároveň dojde k vykreslení aktuálně vybraného průběhu, tedy výchylky. Zde je třeba zamezit přístupu uživatele k nastavení parametrů, aby nebylo možné je během pohybu kyvadla měnit. O to se zde stará funkce `zakaz_nastaveni`.

```
%******************** Zakáže manipulaci s ovládacími prvky ***************
function zakaz_nastaveni(off)
  set (off,'Enable','off')
```

Obr. 13. 13: Funkce zamezující přístup k nastavení parametrů kyvadla

Tuto funkci využijeme ve zpětné vazbě tlačítka START (*pushbutton1*).

```
%Zakže manipulaci s parametry kyvadla (slider)
off=[handles.slider1,handles.slider2,handles.slider3...
     handles.edit1,handles.edit2,handles.edit3];
zakaz_nastaveni(off)

%Zakáže manipulaci s parametry kyvadla (edit text)
off=[handles.slider1,handles.slider2,handles.slider3...
     handles.edit1,handles.edit2,handles.edit3];
zakaz_nastaveni(off)
```

Obr. 13. 14: Volání funkce při obsluze tlačítka START

Nepřetržité kmitání kyvadla zajistíme pomocí cyklu `while`, do kterého budou vnořeny dva cykly `for`. Jeden pro pohyb kyvadla z kladné amplitudy do záporné a druhý pro pohyb ze záporné do kladné. V každém z cyklu je třeba zajistit zobrazení polohy kyvadla a aktuálního grafu. Zde je důvod řešení diferenciální rovnice pouze na intervalu `[0, T/2]`. Jeden `for` cyklus je pouze pro polovinu periody kmitu kyvadla.

```
  %Funkce pro vzkreslení grafu od 0 do T/2
  [t,s,v,a,Ft]=kresli_graf(l,m,fi,g);

  %určí a vytvoří vektor pro polovinu amplitudy (jeden for cyklus)
  poloha=length(t);
  uhel=linspace(fi,-fi,poloha);

%Vytvoří vektory pro zobrazení dvou period kyvadla
t=[t;t+max(t);t+2*max(t);t+3*max(t)];
s=[s;-s;s;-s];
v=[v;-v;v;-v];
a=[a;-a;a;-a];
Ft=[Ft;-Ft;Ft;-Ft];
```

Obr. 13. 15: Výpočet hodnot pro dvě periody grafu

Graficky zobrazovat pouze polovinu periody by nebylo příliš smysluplné, proto jsou výstupy funkce `kresli_graf` po zjištění délky vektoru `t`, která je využita ve `for` cyklu, upraveny na `2*T`. Graficky se tedy zobrazí dvě periody kyvadla.

Protože si můžeme vybrat ze čtyř možností zobrazení, je uvnitř každého cyklu ještě větvení `if - elseif - else`, který zajistí zobrazení vybraného grafu.

```
while get(hObject,'Value')==1 %Cyklus pro další periodu
  if krok >= length(t)
    krok=0;     %nuloje krok po překročení délky vektoru t
    t=t+2*T;    %Zvětší čas o 2*T
  end
    %Cyklus pro první polovinu periody
    for n=1:poloha      %iterační hodnota pro kyvadlo
        krok=krok+1;    %iterační hodnota pro graf
            %zobrazení kyvadla
            axes(handles.axes1);
            kresli_kyvadlo(l,m,uhel(n))
            %Zobrazení průběhů
            axes(handles.axes2);
                %výchylka
                if get(handles.checkbox1,'Value')==1
                    cla
                    hold on
                    plot(t,s,'r','LineWidth',1.2)
                    axis ([min(t) max(t) min(s) max(s)])
                    plot(t(krok),s(krok),'o','MarkerSize',8,...
                        'MarkerFaceColor','b')
                    xlabel('t [s]')
                    ylabel('y [m]')
                    grid on
                    hold off
                    set(handles.text9,'String',s(krok),...
                        'ForegroundColor','red')
                %rychlost
                elseif get(handles.checkbox2,'Value')==1
                    cla
                    hold on
                    plot(t,v,'b','LineWidth',1.2)
```

Obr. 13. 16: Část zpětné vazby tlačítka START

Zdrojový kód zpětné vazby tlačítka START je velmi obsáhlý, proto zde uvádíme pouze jeho nejdůležitější části. Zobrazení ostatních grafů, druhý cyklus for a řešení zpomalení kyvadla můžeme nalézt opět v příloze.

Zpětná vazba tlačítka STOP nám bude sloužit k vrácení kyvadla do výchozího bodu a opětovnému zpřístupnění nastavení. Za tímto účelem je zde vytvořena funkce `povol_nastaveni`.

```
%******************** Povolí manipulaci s ovládacími prvky ***************
function povol_nastaveni(on)
  set (on,'Enable','on')
```

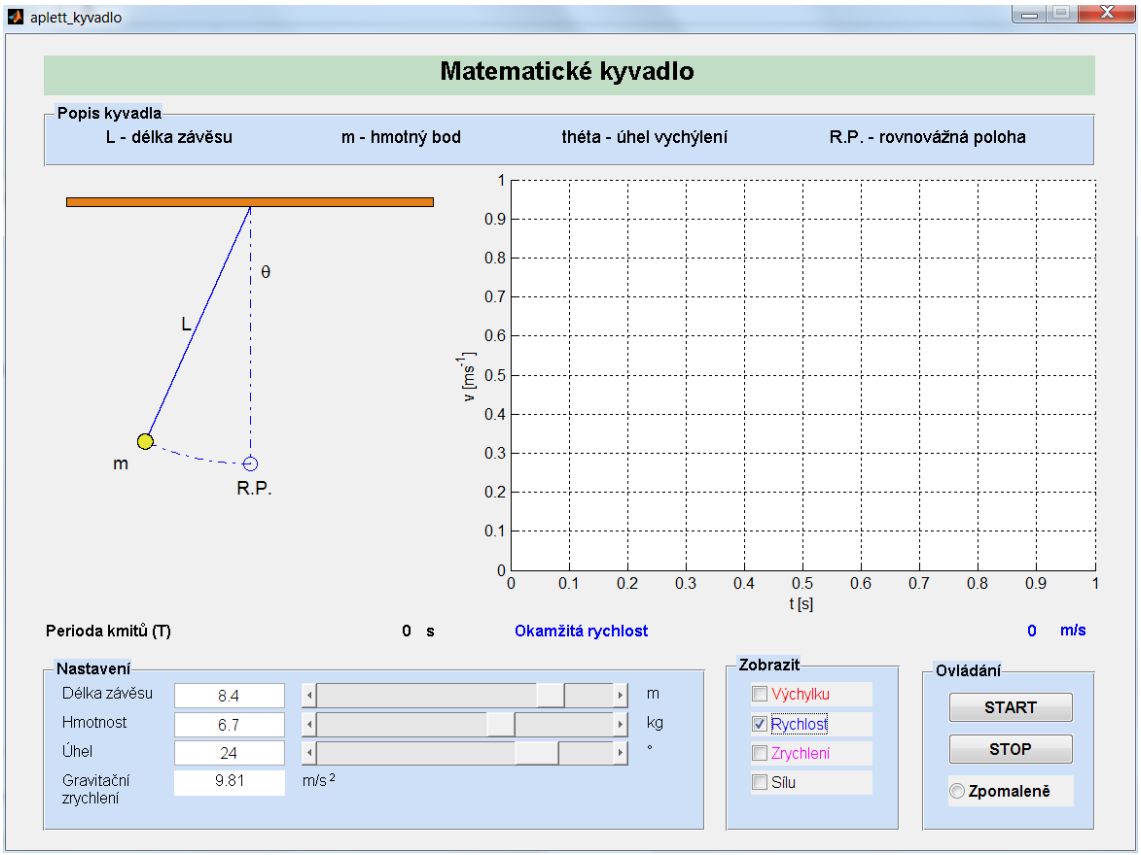
Obr. 13. 17: Funkce povolující přístup k nastavení parametrů kyvadla

Aby bylo možné vrátit kyvadlo do výchozího bodu, je zde opět využita proměnná `vektor`, ze které jsou načteny a zobrazeny poslední hodnoty nastavení kyvadla. Vynulovány jsou také objekty *text7* a *text9*, které slouží k číselnému zobrazení aktuální hodnoty výchylky, rychlosti, zrychlení a síly.

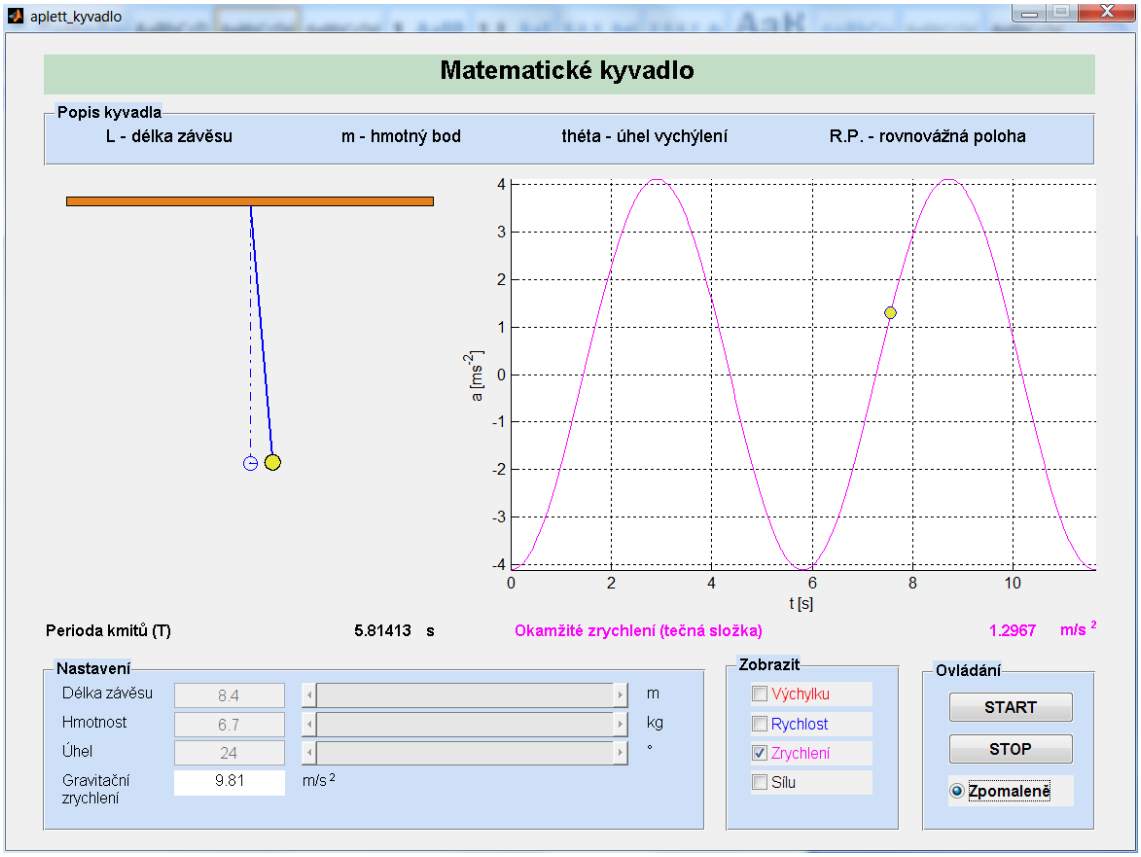
```
% ************************** Tlačítko STOP. ****************************
function pushbutton2_Callback(hObject, eventdata, handles)

    %nolování zobrazovaných hodnot
    set(handles.pushbutton1,'Value',0)
    set(handles.text7,'String',0)
    set(handles.text9,'String',0)

    %povolí nastavení vstupních veličin
    on=[handles.slider1,handles.slider2,handles.slider3...
        handles.edit1,handles.edit2,handles.edit3];
    povol_nastaveni(on)

    %Načte poslední nastavení veličin
    axes(handles.axes1);
    Vektor=get(handles.axes1,'UserData');
    delka_zavesu=Vektor(1);
    hmotnost=Vektor(2);
    uhel=Vektor(3);

    %Uloží a zobrazí naposledy načtené veličiny
    set(handles.axes1,'UserData',Vektor);
    kresli_kyvadlo(delka_zavesu,hmotnost,uhel)
    popis (delka_zavesu,hmotnost,uhel)

    guidata(hObject, handles);
```

Obr. 13. 18: Zpětná vazba tlačítka STOP

Nyní již stačí aplikaci pouze uložit a užít si výsledky naší práce.



Obr. 13. 19: Ukázka aplikace aplett_kyvadlo



Obr. 13. 20: Ukázka aplikace aplett_kyvadlo

14 Závěr

Tato diplomová práce je koncipována jako učební text, jehož podrobným prostudováním by měl čtenář získat základní představu o tvorbě GUI aplikací v prostředí MATLAB a měl by být schopný vytvářet své vlastní návrhy. Protože se jedná o velmi obsáhlou problematiku, omezili jsme se pouze na základní postupy. Možnosti vytváření vlastních aplikací jsou však téměř nevyčerpatelné.

Pochopením základních režimů práce, tvorbou skriptů a funkcí, seznámením se systémem handle graphic a především vytváření vlastních aplikací pomocí GUIDE editoru, nám pomůže stát se do jisté míry nezávislými programátory. Jednotlivé kapitoly jsou také koncipovány tak, aby směřovaly právě k tomuto cíli.

Vše je doplněno o dvě praktické ukázky návrhu aplikace, které by čtenáři měly pomoci pochopit problematiku práce.

Věřím, že tato práce bude přínosem pro ty, kteří se chtějí zabývat tvorbou vlastních aplikací v prostředí MATLAB.

15 Seznam použité a doporučené literatury

Použitá literatura

[1] Karel Zaplatílek, Bohuslav Doňar : MATLAB pro začátečníky, 2.vydání, Praha, Ben 2005, počet str. 151, ISBN 80-7300-175-6

[2] Karel Zaplatílek, Bohuslav Doňar : MATLAB tvorba uživatelských aplikací, Praha, Ben 2004, počet str. 209, ISBN 80-7300-133-0

[3] Pavel Kaban: Výpočty a simulace v programech MATLAB a Simulink, Brno ComputerPress, počet str. 220, ISBN 80-251-1301-9

[4] Stejskal, V., Okrouhlík, M.: Kmitání s Matlabem, Vydavatelství ČVUT, Praha, 2002, 376 stran, ISBN 80-01-02435-0.

[5] Halliday D.,Resnick R., Walker J.: Fyzika, Brno, Vutium, počet str. 1193, ISBN 80-214-1869-9

WWW zdroje

[1] Seznámení s MATLABem – elementární funkce MATLABu. [online]. [cit. 2012-2-21]. Dostupné z: <http://www.fch.vutbr.cz/~polcerova/pc/texty/7kap1.pdf>

[2] MATLAB. Jazyk pro technické výpočty. [online]. [cit. 2012-3-1]. Dostupné z: <http://www.humusoft.cz/produkty/matlab/matlab/>

[3] Kocková, H., Hynčík, L. Modelování MATLABem 2. Doprovodný učební text ke cvičením. [online]. [cit. 2012-3-11]. Dostupné z: <http://www.kme.zcu.cz/download/predmety/ModelovaniMATLABem2.pdf>

[4] Heringová B., Hora, P. MATLAB. Díl I. – Práce s programem. [online]. [cit. 2012-3-4]. Dostupné z: <http://www.cdm.cas.cz/czech/hora/vyuka/mvs/tutorial.pdf>

[5] Heringová B., Hora, P. MATLAB. Díl I. – Práce s programem. [online]. [cit. 2012-3-19]. Dostupné z: <http://www.cdm.cas.cz/czech/hora/vyuka/mvs/reference.pdf>

[6] Čapek, M., Hamouz, P. Skripty a funkce v MATLABu, cykly, uživatelsky definované funkce a další. [online]. [cit. 2012-4-7]. Dostupné z: <http://www.elmag.org/lib/exe/fetch.php/wiki:user:capek:maa_pr11_funkce.pdf>

[7] Kupka, L. Matlab & Simulink, studijní materiály pro předmět Základy kybernetiky. [online]. [cit. 2012-4-22]. Dostupné z: <http://www.mti.tul.cz/files/zky/MATLABaSimulink.pdf>

[8] Hrabec, J. Matlab. Úvod do programového prostředí. [online]. [cit. 2012-4-22]. Dostupné z: <http://www.umel.feec.vutbr.cz/VIT/images/pdf/studijni_materialy/ing/Matlab_P.pdf>

Jiné zdroje

[1] Nápověda programu MATLAB – MATLAB Help

Příloha A

Kompletní zdrojový kód aplikace aplett_kyvadlo.

```
function varargout = aplett_kyvadlo(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @aplett_kyvadlo_OpeningFcn, ...
                   'gui_OutputFcn',  @aplett_kyvadlo_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


%********************* Úvodní nastavení parametrů *******************
function aplett_kyvadlo_OpeningFcn(hObject, eventdata, handles,
varargin)

delka_zavesu=5;                          %vstupní hodnoty
hmotnost=5;
uhel=15;

axes(handles.axes1);                     %uložení a zobrazení dat
Vektor=[delka_zavesu hmotnost uhel];
set(handles.axes1,'UserData',Vektor)
kresli_kyvadlo(delka_zavesu,hmotnost,uhel) %zobrazení kyvadla
popis (delka_zavesu,hmotnost,uhel)

axes(handles.axes2);                     %nastavení grafu
   xlabel('t [s]')
   ylabel('y [m]')
   grid on

handles.output = hObject;
guidata(hObject, handles);

%**********************************************************************
function varargout = aplett_kyvadlo_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;

%******************* Povoluje vybrat pouze jedno pole ***************
function pouze_jedno(off)
  set(off,'Value',0)
```

```
%****************** Povolí manipulaci s ovládacími prvky *************
function povol_nastaveni(on)
  set (on,'Enable','on')

%****************** Zakáže manipulaci s ovládacími prvky *************
function zakaz_nastaveni(off)
  set (off,'Enable','off')


%********************* Zatrhávací pole výchylka **********************
function checkbox1_Callback(hObject, eventdata, handles)

  off=[handles.checkbox2,handles.checkbox3,handles.checkbox4];
  pouze_jedno(off) %zatržené checkbox1

   %Zobrazení aktuální veličiny
   set(handles.text8,'String','Okamžitá
výchylka','ForegroundColor','red')
   set(handles.text9,'ForegroundColor','red')
   set(handles.text19,'String','m','ForegroundColor','red')
   set(handles.text23,'String','')

   axes(handles.axes2)
   xlabel('t [s]')
   ylabel('y [m]')

   guidata(hObject, handles);

%********************* Zatrhávací pole rychlost **********************
function checkbox2_Callback(hObject, eventdata, handles)

  off=[handles.checkbox1,handles.checkbox3,handles.checkbox4];
  pouze_jedno(off)%zatržené checkbox2

  %Zobrazení aktuální veličiny
  set(handles.text8,'String','Okamžitá
rychlost','ForegroundColor','blue')
  set(handles.text9,'ForegroundColor','blue')
  set(handles.text19,'String','m/s','ForegroundColor','blue')
  set(handles.text23,'String','')

  axes(handles.axes2)
  xlabel('t [s]')
  ylabel('v [ms^-^1]')

  guidata(hObject, handles);

%******************** Zatrhávací pole zrychlení **********************
function checkbox3_Callback(hObject, eventdata, handles)

  off=[handles.checkbox1,handles.checkbox2,handles.checkbox4];
  pouze_jedno(off)%zatržené checkbox3

  %Zobrazení aktuální veličiny
  set(handles.text8,'String','Okamžité zrychlení (tečná složka)',...
      'ForegroundColor','magenta')
  set(handles.text9,'ForegroundColor','magenta')
```

```
  set(handles.text19,'String','m/s','ForegroundColor','magenta')
  set(handles.text23,'String','2')

  axes(handles.axes2)
  xlabel('t [s]')
  ylabel('a [ms^-^2]')

  guidata(hObject, handles);

%********************* Zatrhávací pole síla *************************
function checkbox4_Callback(hObject, eventdata, handles)

  off=[handles.checkbox1,handles.checkbox2,handles.checkbox3];
  pouze_jedno(off)%zatržené checkbox4

   set(handles.text8,'String','Síla (tečná složka)',...
       'ForegroundColor','black')
   set(handles.text9,'ForegroundColor','black')
   set(handles.text19,'String','N','ForegroundColor','black')
   set(handles.text23,'String','')

   axes(handles.axes2)
   xlabel('t [s]')
   ylabel('F_t [N]')

  guidata(hObject, handles);

 %******************* Zpomalení kmitání kyvadla **********************
 function radiobutton7_Callback(hObject, eventdata, handles)

 guidata(hObject, handles);

%***************** Posuvník pro nastavení délky závěsu ***************
function slider1_Callback(hObject, eventdata, handles)

  %Propojení posuvníku a editovatelného pole
  set(handles.edit1,'String',...
      num2str(get(handles.slider1,'Value')));

  %Načtení dat po změně délky závěsu
  axes(handles.axes1);
    Vektor=get(handles.axes1,'UserData');
    delka_zavesu=get(hObject,'Value');
    hmotnost=Vektor(2);
    uhel=Vektor(3);
    Vektor(1)=delka_zavesu;

  %Uložení a zobrazení změny délky závěsu
    set(handles.axes1,'UserData',Vektor);
    kresli_kyvadlo(delka_zavesu,hmotnost,uhel)
    popis (delka_zavesu,hmotnost,uhel)
    guidata(hObject, handles);

%******** Nastavení délky závěsu při startu aplikace (slider) ********
function slider1_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
set(hObject,'Min',1,'Max',10,'Value',5,...
    'SliderStep',[0.1/(10-1) 1/(10-1)]);
guidata(hObject, handles)

%******************* Posuvník pro nastavení hmotnosti ***************
function slider2_Callback(hObject, eventdata, handles)

  %Propojení posuvníku a editovatelného pole
  set(handles.edit2,'String',...
      num2str(get(handles.slider2,'Value')));

  %Načtení dat po změně hmotnosti
  axes(handles.axes1);
    Vektor=get(handles.axes1,'UserData');
    delka_zavesu=Vektor(1);
    hmotnost=get(hObject,'Value');
    uhel=Vektor(3);
    Vektor(2)=hmotnost;

  %Uložení a zobrazení změny mnotnosti
    set(handles.axes1,'UserData',Vektor);
    kresli_kyvadlo(delka_zavesu,hmotnost,uhel)
    popis (delka_zavesu,hmotnost,uhel)
    guidata(hObject, handles);

%********** Nastavení hmotnosti při startu aplikace (slider) *********
function slider2_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
set(hObject,'Min',1,'Max',10,'Value',5,...
    'SliderStep',[0.1/(10-1) 1/(10-1)]);
guidata(hObject, handles)

%******************** Posuvník pro nastavení úhlu *******************
function slider3_Callback(hObject, eventdata, handles)

  %Propojení posuvníku a editovatelného pole
  set(handles.edit3,'String',...
      num2str(get(handles.slider3,'Value')));

  %Načtení dat po změně úhlu
  axes(handles.axes1);
    Vektor=get(handles.axes1,'UserData');
    delka_zavesu=Vektor(1);
    hmotnost=Vektor(2);
    uhel=get(hObject,'Value');
    Vektor(3)=uhel;

  %Uložení a zobrazení změny úhlu
    set(handles.axes1,'UserData',Vektor);
    kresli_kyvadlo(delka_zavesu,hmotnost,uhel)
    popis (delka_zavesu,hmotnost,uhel)
    guidata(hObject, handles);
```

```
%************* Nastavení úhlu při startu aplikace (slider) ***********
function slider3_CreateFcn(hObject, eventdata, handles)

if isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
set(hObject,'Min',2,'Max',30,'Value',15,...
    'SliderStep',[1/(30-2) 5/(30-2)]);
guidata(hObject, handles)

%********** Editovatelné pole pro nastavení délky závěsu *************
function edit1_Callback(hObject, eventdata, handles)

  %Načtení délky závěsu editovatelné pole
  val=str2double(get(handles.edit1,'String'));

  if isnumeric(val)&& length(val)==1 &&...
     val>=get(handles.slider1,'Min') &&...
     val<=get(handles.slider1,'Max')
        set(handles.slider1,'Value',val);

    %Načtení dat po změně délky závěsu
    axes(handles.axes1);
      Vektor=get(handles.axes1,'UserData');
      delka_zavesu=val;
      hmotnost=Vektor(2);
      uhel=Vektor(3);
      Vektor(1)=val;

      %Uložení a zobrazení délky závěsu
      set(handles.axes1,'UserData',Vektor);
      kresli_kyvadlo(delka_zavesu,hmotnost,uhel)
      popis (delka_zavesu,hmotnost,uhel)
  else
    %Při chybném zadání vstupu
    warndlg('Zadej delku v rozsahu 1-10','Chyba vstupu')
    set(handles.edit1,'String','');
  end

    guidata(hObject, handles);

%******* Nastavení délky závěsu při startu aplikace (edit text) ******
function edit1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

guidata(hObject, handles);

%************ Editovatelné pole pro nastavení hmotnosti **************
function edit2_Callback(hObject, eventdata, handles)

  %Načtení hmotnosti editovatelné pole
  val=str2double(get(handles.edit2,'String'));

  if isnumeric(val)&& length(val)==1 &&...
```

```
    val>=get(handles.slider2,'Min') &&...
    val<=get(handles.slider2,'Max')
       set(handles.slider2,'Value',val);

    %Načtení dat po změně hmotnosti
    axes(handles.axes1);
      Vektor=get(handles.axes1,'UserData');
      delka_zavesu=Vektor(1);
      hmotnost=val;
      uhel=Vektor(3);
      Vektor(2)=val;

      %Uložení a zobrazení hmotnosti
      set(handles.axes1,'UserData',Vektor);
      kresli_kyvadlo(delka_zavesu,hmotnost,uhel)
      popis (delka_zavesu,hmotnost,uhel)
  else
    %Při chybném zadání vstupu
    warndlg('Zadej hmotnost v rozsahu 1-10','Chyba vstupu')
    set(handles.edit1,'String','');
  end

    guidata(hObject, handles);

%******** Nastavení hmotnosti při startu aplikace (edit text) ********
function edit2_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

  guidata(hObject, handles);

%*************** Editovatelné pole pro nastavení úhlu ***************
function edit3_Callback(hObject, eventdata, handles)

  %Načtení úhlu editovatelné pole
  val=str2double(get(handles.edit3,'String'));

  if isnumeric(val)&& length(val)==1 &&...
     val>=get(handles.slider3,'Min') &&...
     val<=get(handles.slider3,'Max')
        set(handles.slider3,'Value',val);

     %Načtení dat po změně úhlu
     axes(handles.axes1);
       Vektor=get(handles.axes1,'UserData');
       delka_zavesu=Vektor(1);
       hmotnost=Vektor(2);
       uhel=val;
       Vektor(3)=val;

       %Uložení a zobrazení úhlu
       set(handles.axes1,'UserData',Vektor);
       kresli_kyvadlo(delka_zavesu,hmotnost,uhel)
       popis (delka_zavesu,hmotnost,uhel)
  else
    %Při chybném zadání vstupu
```

```
    warndlg('Zadej úhel v rozsahu 1-30','Chyba vstupu')
    set(handles.edit3,'String','');
  end

    guidata(hObject, handles);

%*********** Nastavení úhlu při startu aplikace (edit text) **********
function edit3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),...
        get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

  guidata(hObject, handles);

% ********************** Tlačítko START. ***************************
function pushbutton1_Callback(hObject, eventdata, handles)
  global dlk;

  l=get(handles.slider1,'Value');            %Načtení vstupních hodnot
  m=get(handles.slider2,'Value');
  fi=get(handles.slider3,'Value');

  dlk=l;
  g=9.81;
  krok=0;
  T=2*pi*sqrt(l/g);

  set(handles.text7,'String',T);

  %Zakže manipulaci s parametry kyvadla (slider)
  off=[handles.slider1,handles.slider2,handles.slider3...
       handles.edit1,handles.edit2,handles.edit3];
  zakaz_nastaveni(off)

  %Zakáže manipulaci s parametry kyvadla (edit text)
  off=[handles.slider1,handles.slider2,handles.slider3...
       handles.edit1,handles.edit2,handles.edit3];
  zakaz_nastaveni(off)

  %Funkce pro vykreslení grafu od 0 do T/2
  [t,s,v,a,Ft]=kresli_graf(l,m,fi,g,T);

  %určí a vytvoří vektor pro polovinu amplitudy (jeden for cyklus)
  poloha=length(t);
  uhel=linspace(fi,-fi,poloha);

%Vytvoří vektory pro zobrazení dvou period kyvadla
t=[t;t+max(t);t+2*max(t);t+3*max(t)];
s=[s;-s;s;-s];
v=[v;-v;v;-v];
a=[a;-a;a;-a];
Ft=[Ft;-Ft;Ft;-Ft];

format short;
while get(hObject,'Value')==1 %Cyklus pro další periodu
  if krok >= length(t)
    krok=0;     %nuloje krok po překročení délky vektoru t
```

```
  t=t+2*T;     %Zvětší čas o 2*T
end
  %Cyklus pro první polovinu periody
  for n=1:poloha      %iterační hodnota pro kyvadlo
      krok=krok+1;    %iterační hodnota pro graf
          %zobrazení kyvadla
          axes(handles.axes1);
          kresli_kyvadlo(l,m,uhel(n))
          %Zobrazení průběhů
          axes(handles.axes2);
              %výchylka
              if get(handles.checkbox1,'Value')==1
                  cla
                  hold on
                  plot(t,s,'r','LineWidth',1.2)
                  axis ([min(t) max(t) min(s) max(s)])
                  plot(t(krok),s(krok),'o','MarkerSize',8,...
                      'MarkerFaceColor',[0.9 0.9 0.2])
                  xlabel('t [s]')
                  ylabel('y [m]')
                  grid on
                  hold off
                  set(handles.text9,'String',s(krok),...
                      'ForegroundColor','red')
              %rychlost
              elseif get(handles.checkbox2,'Value')==1
                  cla
                  hold on
                  plot(t,v,'b','LineWidth',1.2)
                  axis ([min(t) max(t) min(v) max(v)])
                  plot(t(krok),v(krok),'o','MarkerSize',8,...
                      'MarkerFaceColor',[0.9 0.9 0.2])
                  xlabel('t [s]')
                  ylabel('v [ms^-^1]')
                  grid on
                  hold off
                  set(handles.text9,'String',v(krok),...
                      'ForegroundColor','blue')

              %zrychlení
              elseif get(handles.checkbox3,'Value')==1
                  cla
                  hold on
                  plot(t,a,'m','LineWidth',1.2)
                  axis ([min(t) max(t) min(a) max(a)])
                  plot(t(krok),a(krok),'o','MarkerSize',8,...
                      'MarkerFaceColor',[0.9 0.9 0.2])
                  xlabel('t [s]')
                  ylabel('a [ms^-^2]')
                  grid on
                  hold off
                  set(handles.text9,'String',a(krok),...
                      'ForegroundColor','magenta')

              %síla
              elseif get(handles.checkbox4,'Value')==1
                  cla
                  hold on
                  plot(t,Ft,'k','LineWidth',1.2)
                  axis ([min(t) max(t) min(Ft) max(Ft)])
                  plot(t(krok),Ft(krok),'o','MarkerSize',8,...
```

```
                    'MarkerFaceColor',[0.9 0.9 0.2])
                xlabel('t [s]')
                ylabel('F_t [N]')
                grid on
                hold off
                set(handles.text9,'String',Ft(krok),...
                    'ForegroundColor','black')
            end

    %Zpomalení kyvadla
    if get(handles.radiobutton7,'Value')==1
        pause (0.3)
    else
        pause (0.01)
    end
    if get(hObject,'Value')~=1 %obsluha tlačítka STOP
        axes(handles.axes2)
        cla
        return
    end
end

%Cyklus pro druhou polovinu periody
for k=[poloha:-1:1] %iterační hodnota pro kyvadlo
    krok=krok+1;    %iterační hodnota pro graf
        %zobrazení kyvadla
        axes(handles.axes1);
        kresli_kyvadlo(l,m,uhel(k))

        %zobrazení průběhů
        axes(handles.axes2);
            %výchylka
            if get(handles.checkbox1,'Value')==1
                cla
                hold on
                plot(t,s,'r','LineWidth',1.2)
                axis ([min(t) max(t) min(s) max(s)])
                plot(t(krok),s(krok),'o','MarkerSize',8,...
                    'MarkerFaceColor',[0.9 0.9 0.2])
                xlabel('t [s]')
                ylabel('y [m]')
                grid on
                hold off
                set(handles.text9,'String',s(krok),...
                    'ForegroundColor','red')

            %rychlost
            elseif get(handles.checkbox2,'Value')==1
                cla
                hold on
                plot(t,v,'b','LineWidth',1.2)
                axis ([min(t) max(t) min(v) max(v)])
                plot(t(krok),v(krok),'o','MarkerSize',8,...
                    'MarkerFaceColor',[0.9 0.9 0.2])
                xlabel('t [s]')
                ylabel('v [ms^-^1]')
                grid on
                hold off
                set(handles.text9,'String',v(krok),...
                    'ForegroundColor','blue')
```

```
                %zrychlení
                elseif get(handles.checkbox3,'Value')==1
                    cla
                    hold on
                    plot(t,a,'m','LineWidth',1.2)
                    axis ([min(t) max(t) min(a) max(a)])
                    plot(t(krok),a(krok),'o','MarkerSize',8,...
                        'MarkerFaceColor',[0.9 0.9 0.2])
                    xlabel('t [s]')
                    ylabel('a [ms^-^2]')
                    grid on
                    hold off
                    set(handles.text9,'String',a(krok),...
                        'ForegroundColor','magenta')

                %síla
                elseif get(handles.checkbox4,'Value')==1
                    cla
                    hold on
                    plot(t,Ft,'k','LineWidth',1.2)
                    axis ([min(t) max(t) min(Ft) max(Ft)])
                    plot(t(krok),Ft(krok),'o','MarkerSize',8,...
                        'MarkerFaceColor',[0.9 0.9 0.2])
                    xlabel('t [s]')
                    ylabel('F_t [N]')
                    grid on
                    hold off
                    set(handles.text9,'String',Ft(krok),...
                        'ForegroundColor','black')
                end

            %zpomalení kyvadla
            if get(handles.radiobutton7,'Value')==1
                pause (0.3)
            else
                pause (0.01)
            end
            if get(hObject,'Value')~=1 %obsluha tlačítka STOP
                axes(handles.axes2)
                cla
                return
            end
        end
    end

  guidata(hObject, handles);

% ************************ Tlačítko STOP. ***************************
function pushbutton2_Callback(hObject, eventdata, handles)

    %nolování zobrazovaných hodnot
    set(handles.pushbutton1,'Value',0)
    set(handles.text7,'String',0)
    set(handles.text9,'String',0)

    %povolí nastavení vstupních veličin
    on=[handles.slider1,handles.slider2,handles.slider3...
         handles.edit1,handles.edit2,handles.edit3];
    povol_nastaveni(on)
```

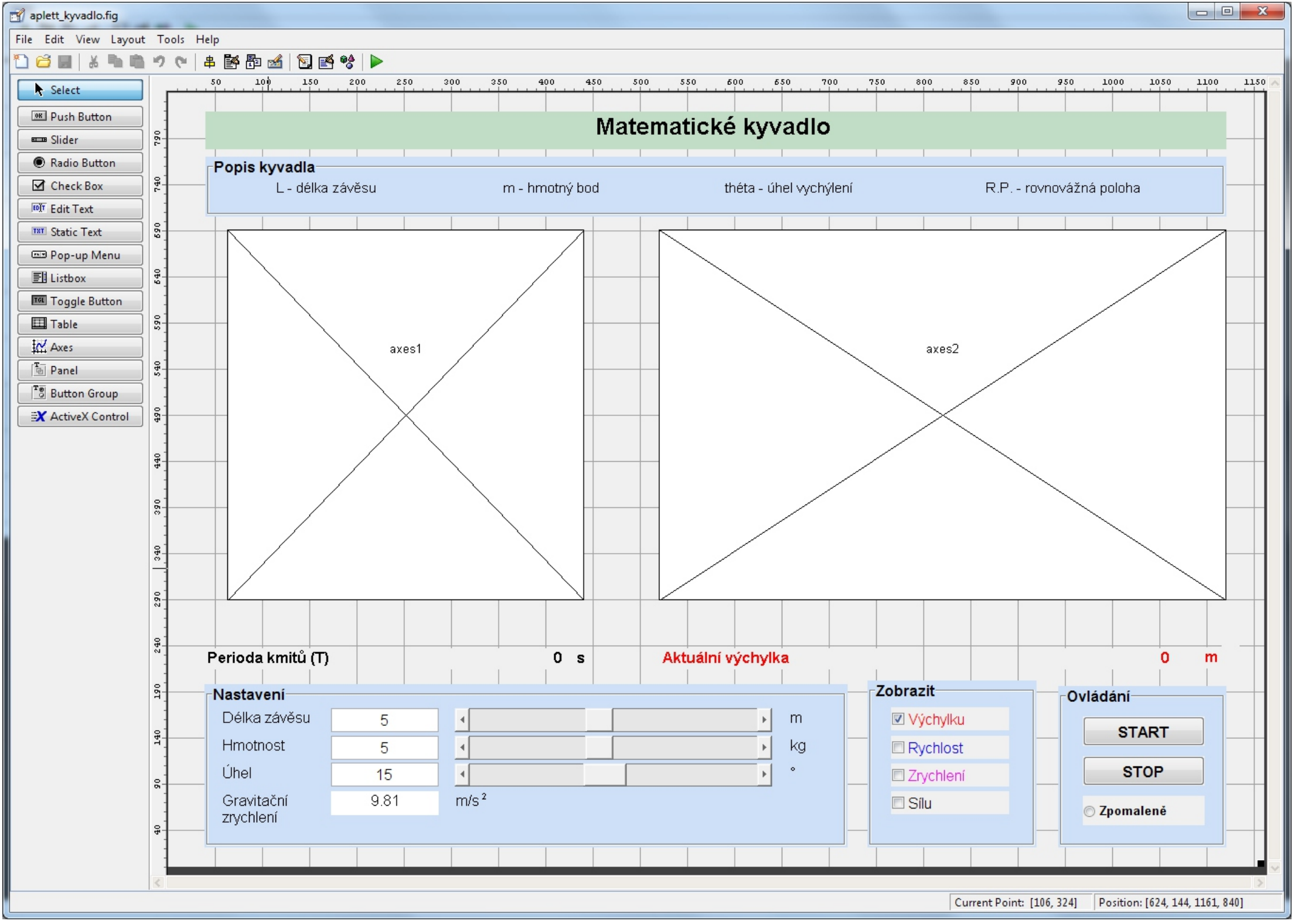
```
    %Načte poslední nastavení veličin
    axes(handles.axes1);
    Vektor=get(handles.axes1,'UserData');
    delka_zavesu=Vektor(1);
    hmotnost=Vektor(2);
    uhel=Vektor(3);

    %Uloží a zobrazí naposledy načtené veličiny
    set(handles.axes1,'UserData',Vektor);
    kresli_kyvadlo(delka_zavesu,hmotnost,uhel)
    popis (delka_zavesu,hmotnost,uhel)

    guidata(hObject, handles);

% ************* Vypočet poloh a vykresleni kyvadla ******************
function kresli_kyvadlo(l,m,fi)
%Funkce slouží k vypočtu poloh a zobrazeni poloh závěsu, hmotného bodu
%a uchyceni kyvadla
    if fi>0                     %vektory oblouku kyvadla
        oblouk=0:0.5:fi;        %kladná hodnota
    else
        oblouk=0:-0.5:fi;       %záporná hodnota
    end
    oblouk_x=cos((270-oblouk)*pi/180)*l;  %Souřadnice oblouku
    oblouk_y=sin((270-oblouk)*pi/180)*l;

    uhel_rad=(270-fi)*pi/180; %vypočet uhlu v radiánech
    x=cos(uhel_rad)*l;        %Souřadnice polohy
    y=sin(uhel_rad)*l;
    delka=360;                %Pro vypočet kružnice (hmotného bodu)

    axis equal;               %Graficky vystup
    axis ([-6 6 -11.5 0.5]);
    axis off

    cla                       %smazáni předchozích grafu
    hold on

      fill([-6 -6 6 6],[0 0.3 0.3 0],[0.9 0.5 0.1],'LineWidth',1,...
           'LineStyle','-');
      line ([0 0],[0 -l],'LineWidth',1,'LineStyle','-.');
      plot (0,-l,'o','MarkerSize',9.5)
      line ([0 x],[0 y],'LineWidth',1.5);
      plot(oblouk_x,oblouk_y,'LineStyle','-.')
      [xp,yp]=kruznice(x,y,0.25,delka);
      patch (xp, yp,[0.9 0.9 0.2]);
    hold off

%************* Kružnice pro zobrazeni hmotného bodu *****************
function [x,y]=kruznice(x0,y0,r,bodu)

    uhel=linspace(0,2*pi,bodu);
    x=x0+r*cos(uhel);
    y=y0+r*sin(uhel);
%********************** Popis veličin kyvadla ***********************
function popis (l,m,fi)
    uhel_rad=(270-fi)*pi/180; %vypočet uhlu v radiánech
    x=cos(uhel_rad)*l;        %Souřadnice polohy
```

```
    y=sin(uhel_rad)*l;
    text (x/2-0.7,y/2,'\fontsize{12} L')    %popis kyvadla
    text (x-1.2,y-0.7,'\fontsize{12} m')
    text (0.2,-l/4,'\fontsize{12} \theta')
    text (-0.6,-l-0.8,'\fontsize{12} R.P.')

%***************** Funkce pro vykreslení průběhů *********************
function [t,s,v,a,Ft] = kresli_graf (l,m,fi,g,T)
s0=(fi*pi/180)*l; %m
v0=0; %m/s

%omega
w=(2*pi)/T;

%řešeni diferenciální rovnice
options=odeset('RelTol',1e-9);
[t, x]=ode45(@dif_rce, [0, T/2], [s0, v0], options);

%převod na fyzikální veličiny
s=x(:,1);
v=x(:,2);
a=-s*w^2;
Ft=-m*g*s/l;

%************** Soustava dif. rovnic popisujících kyvadlo *************
function [dxdt] = dif_rce(t,x)
%ds/dt = v
%dv/dt = -g/l*s
global dlk;
g=9.81;
    dxdt=zeros(2,1);
    dxdt(1)=x(2);
    dxdt(2)=-g/dlk*x(1);
```

Příloha B

Přílohu B tvoří CD přiložené k diplomové práci které obsahuje:

- Soubor s kompletním zdrojovým kódem aplikace aplet_kyvadlo.m
- Soubor s rozmístěním a úpravou komponent aplikace aplet_kyvadlo.fig
- Soubor aplet_kyvadlo.asv
- Několik obrázků z běhu aplikace aplet_kyvadlo
- Tuto diplomovou práci v elektronické podobě

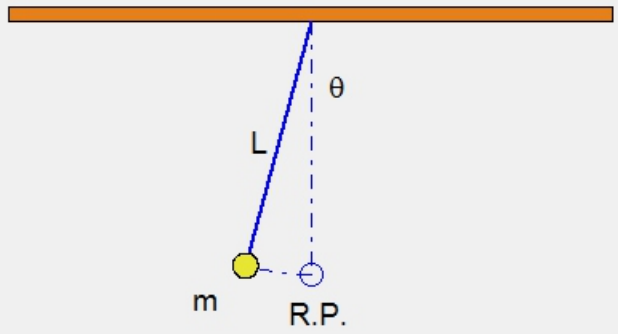

aplett_kyvadlo.fig
File Edit View Layout Tools Help
Select
Push Button
Slider
Radio Button
Check Box
Edit Text
Static Text
Pop-up Menu
Listbox
Toggle Button
Table
Axes
Panel
Button Group
ActiveX Control
Matematické kyvadlo
Popis kyvadla
L - délka závěsu
m - hmotný bod
théta - úhel vychýlení
R.P. - rovnovážná poloha
axes1
axes2
Perioda kmitů (T)
0 s
Aktuální výchylka
0 m
Nastavení
Délka závěsu
5
m
Hmotnost
5
kg
Úhel
15
°
Gravitační zrychlení
9.81
m/s 2
Zobrazit
Výchylku
Rychlost
Zrychlení
Sílu
Ovládání
START
STOP
Zpomaleně
Current Point: [106, 324]
Position: [624, 144, 1161, 840]

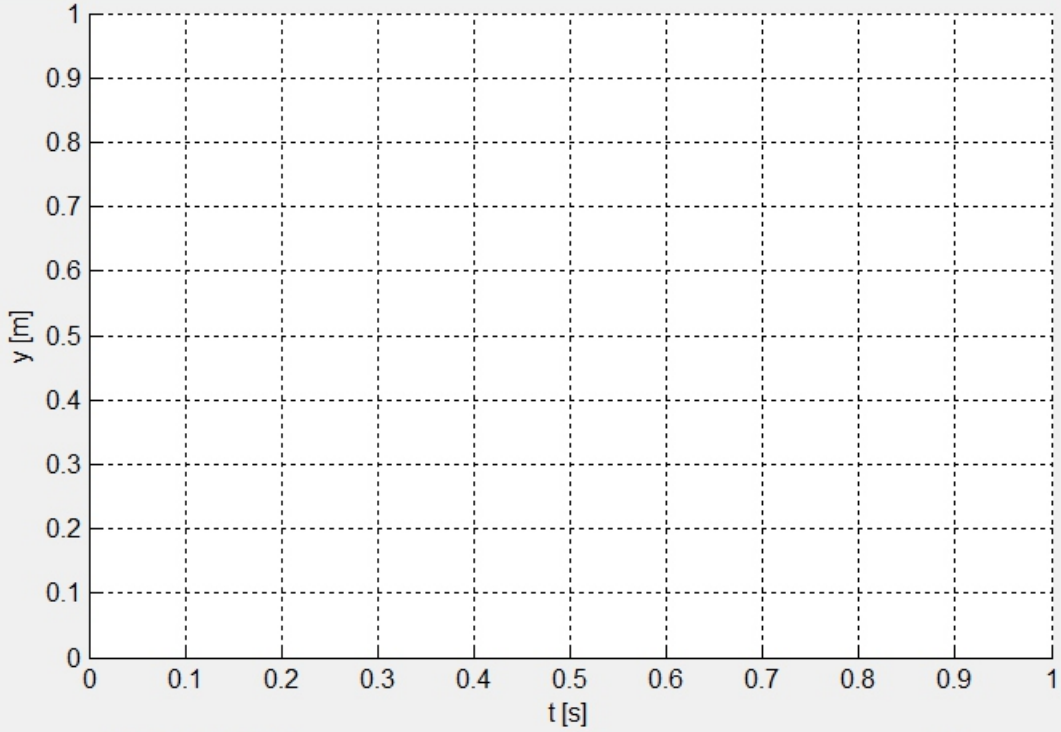
aplett_kyvadlo

Matematické kyvadlo

Popis kyvadla

L - délka závěsu | m - hmotný bod | théta - úhel vychýlení | R.P. - rovnovážná poloha





Perioda kmitů (T) 0 s **Aktuální výchylka** 0 m

Nastavení

Délka závěsu	5	m
Hmotnost	5	kg
Úhel	15	°
Gravitační zrychlení	9.81	m/s^2

Zobrazit

- [x] Výchylku
- [ ] Rychlost
- [ ] Zrychlení
- [ ] Sílu

Ovládání

START

STOP

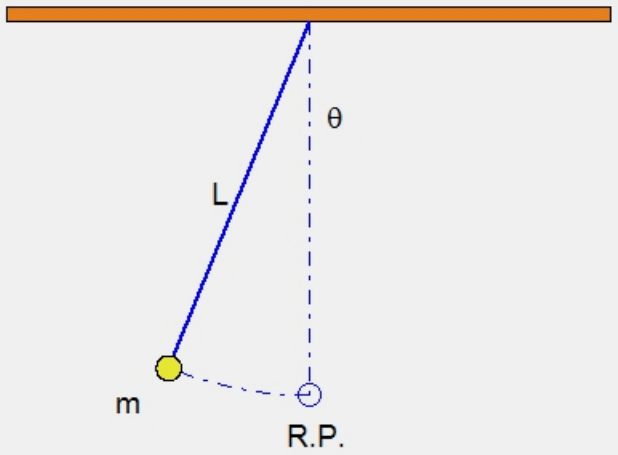
( ) Zpomaleně

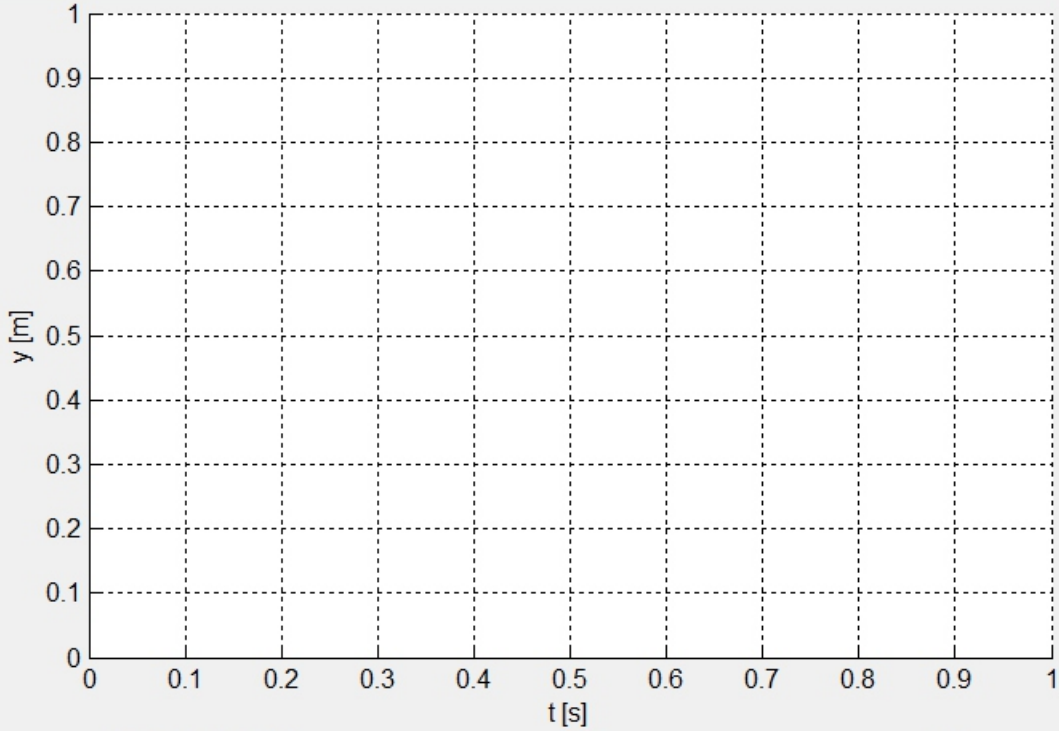
aplett_kyvadlo

Matematické kyvadlo

Popis kyvadla

L - délka závěsu | m - hmotný bod | théta - úhel vychýlení | R.P. - rovnovážná poloha





Perioda kmitů (T) 0 s **Aktuální výchylka** 0 m

Nastavení

Délka závěsu	7.4	m
Hmotnost	5.7	kg
Úhel	22	°
Gravitační zrychlení	9.81	m/s²

Zobrazit

- [x] Výchylku
- [ ] Rychlost
- [ ] Zrychlení
- [ ] Sílu

Ovládání

START

STOP

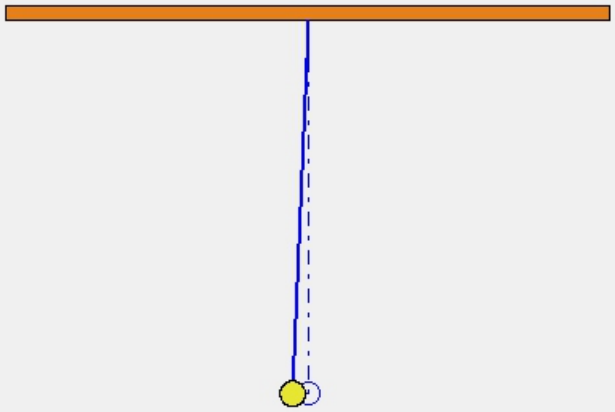
( ) Zpomaleně

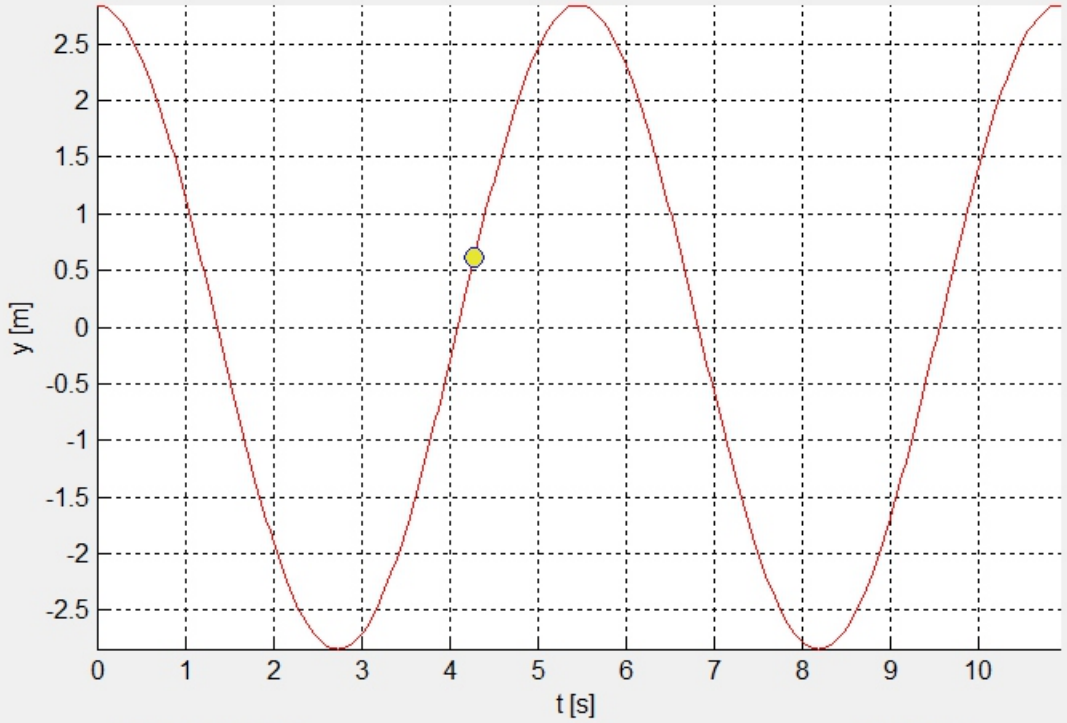
aplett_kyvadlo

Matematické kyvadlo

Popis kyvadla

L - délka závěsu | m - hmotný bod | théta - úhel vychýlení | R.P. - rovnovážná poloha





Perioda kmitů (T) 5.45709 s **Aktuální výchylka** 0.613976 m

Nastavení

Délka závěsu	7.4	m
Hmotnost	5.7	kg
Úhel	22	°
Gravitační zrychlení	9.81	m/s 2

Zobrazit

- [x] Výchylku
- [ ] Rychlost
- [ ] Zrychlení
- [ ] Sílu

Ovládání

START

STOP

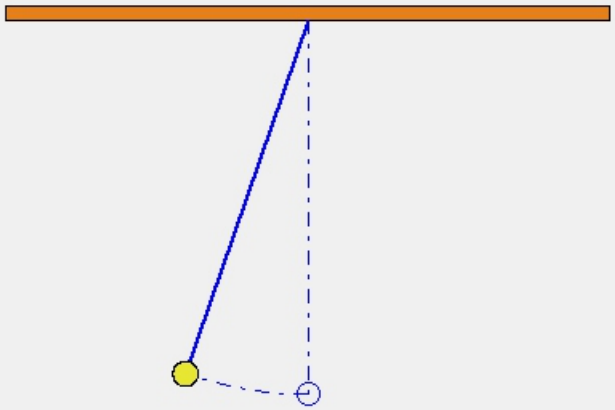
- [ ] Zpomaleně

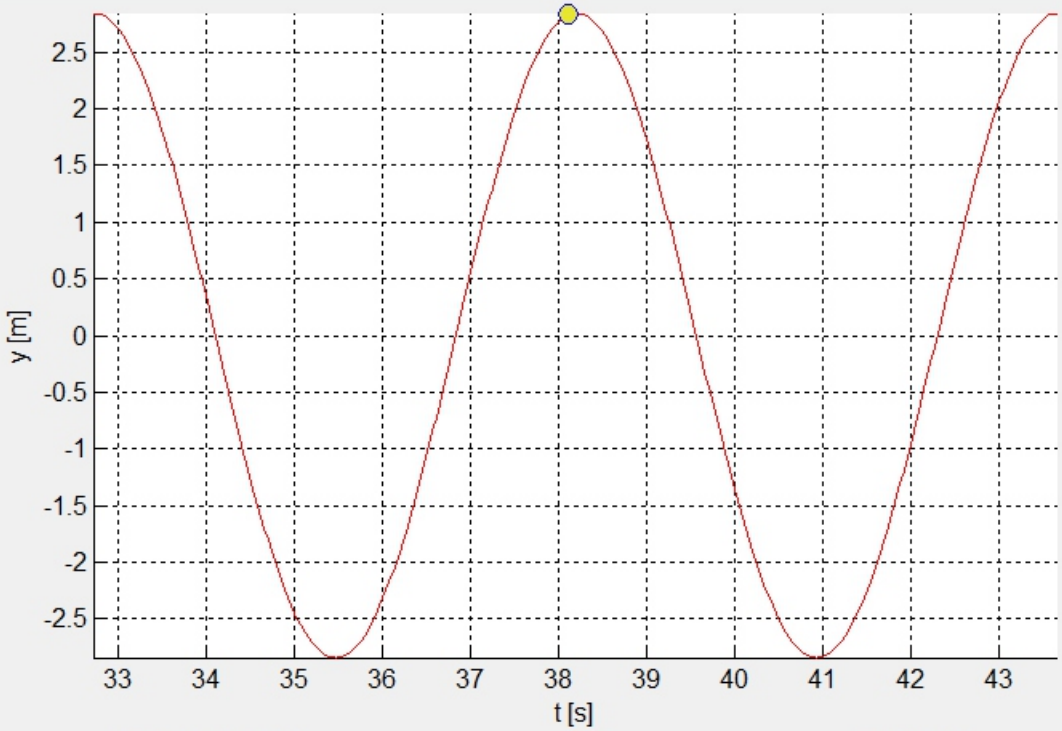
aplett_kyvadlo

Matematické kyvadlo

Popis kyvadla

L - délka závěsu | m - hmotný bod | théta - úhel vychýlení | R.P. - rovnovážná poloha





Perioda kmitů (T) 5.45709 s

Aktuální výchylka 2.82759 m

Nastavení

Délka závěsu	7.4		m
Hmotnost	5.7		kg
Úhel	22		°
Gravitační zrychlení	9.81	m/s^2	

Zobrazit

- [x] Výchylku
- [ ] Rychlost
- [ ] Zrychlení
- [ ] Sílu

Ovládání

START

STOP

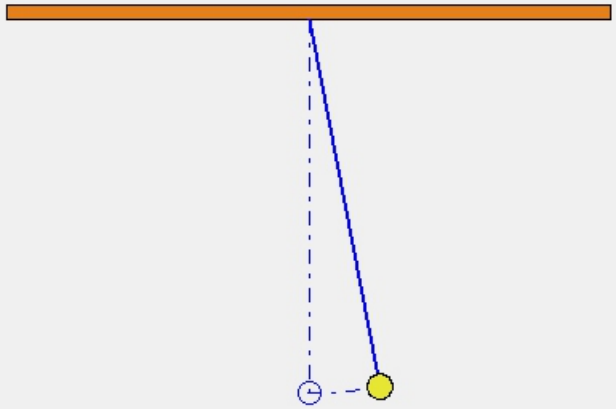
- ( ) Zpomaleně

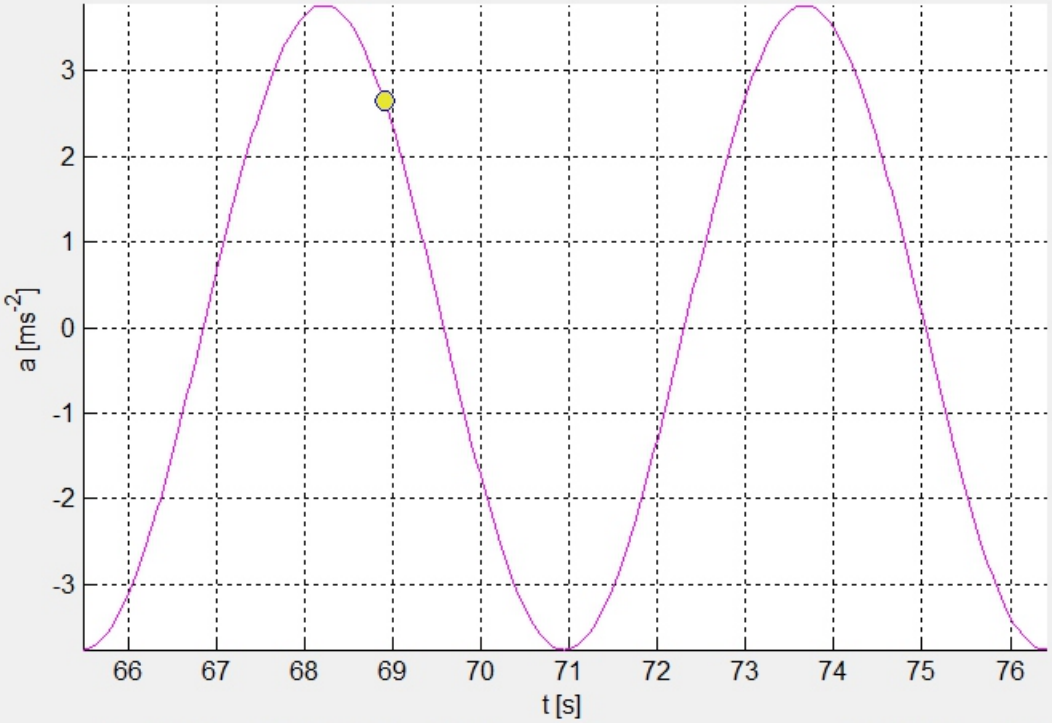
aplett_kyvadlo

Matematické kyvadlo

Popis kyvadla

L - délka závěsu | m - hmotný bod | théta - úhel vychýlení | R.P. - rovnovážná poloha





Perioda kmitů (T) 5.45709 s

Okamžité zrychlení (tečná složka) 2.65176 m/s 2

Nastavení

Délka závěsu	7.4	m
Hmotnost	5.7	kg
Úhel	22	°
Gravitační zrychlení	9.81	m/s 2

Zobrazit

- [ ] Výchylku
- [ ] Rychlost
- [x] Zrychlení
- [ ] Sílu

Ovládání

START

STOP

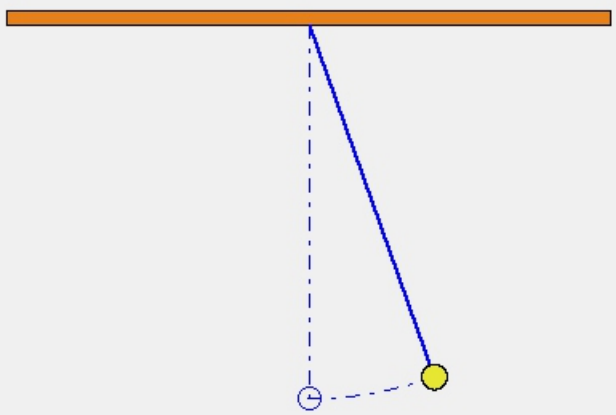
(•) Zpomaleně

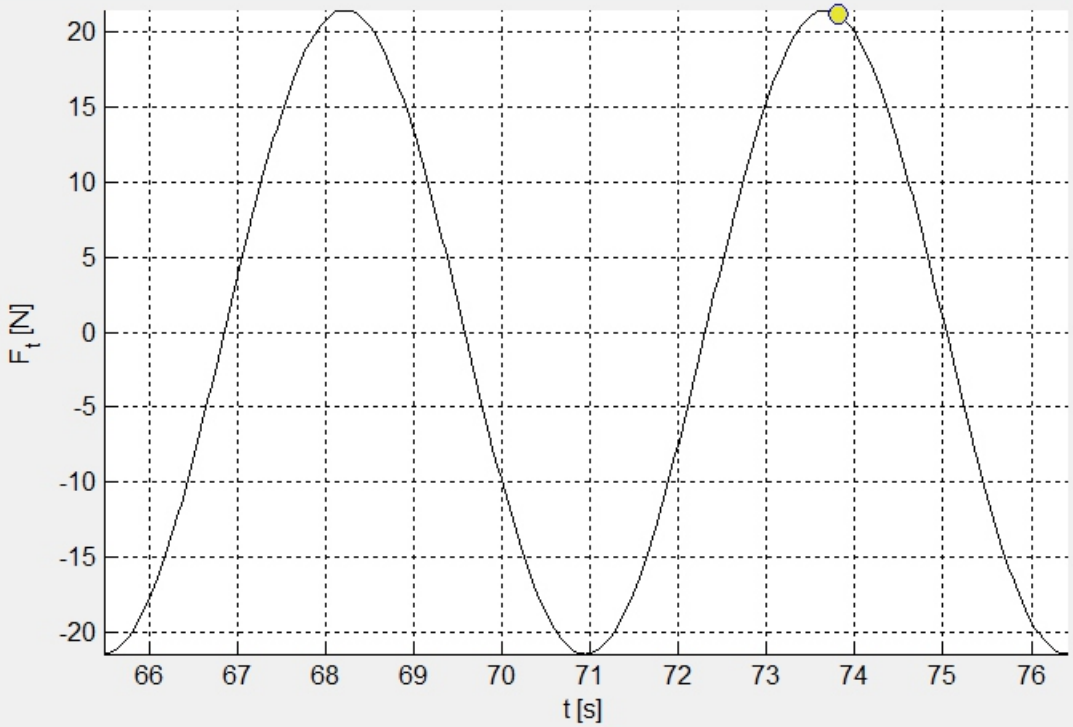
aplett_kyvadlo

Matematické kyvadlo

Popis kyvadla

L - délka závěsu | m - hmotný bod | théta - úhel vychýlení | R.P. - rovnovážná poloha





Perioda kmitů (T) 5.45709 s **Síla (tečná složka)** 21.1769 N

Nastavení

Délka závěsu	7.4	m
Hmotnost	5.7	kg
Úhel	22	°
Gravitační zrychlení	9.81	m/s^2

Zobrazit

- [ ] Výchylku
- [ ] Rychlost
- [ ] Zrychlení
- [x] Sílu

Ovládání

START

STOP

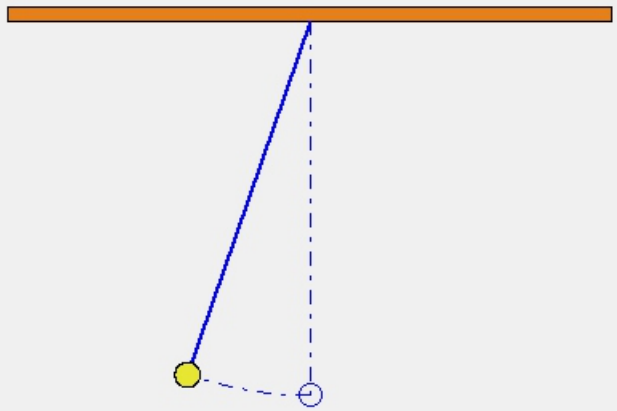
(•) Zpomaleně

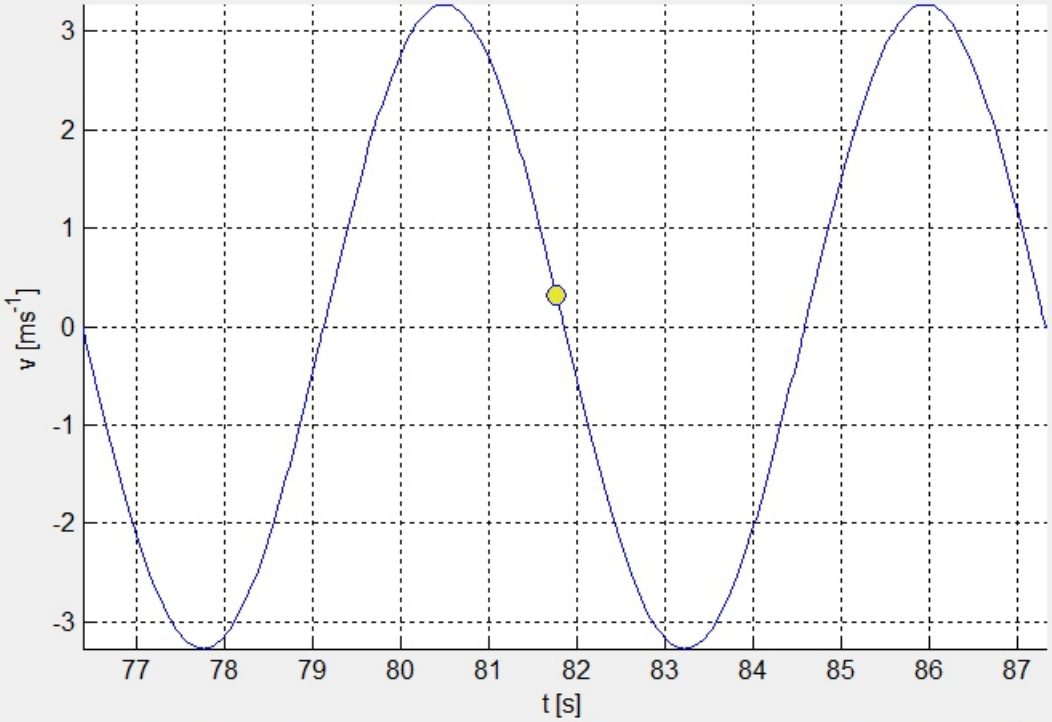
aplett_kyvadlo

Matematické kyvadlo

Popis kyvadla

L - délka závěsu | m - hmotný bod | théta - úhel vychýlení | R.P. - rovnovážná poloha





Perioda kmitů (T) **5.45709 s** **Okamžitá rychlost** **0.322053 m/s**

Nastavení

Délka závěsu	7.4		m
Hmotnost	5.7		kg
Úhel	22		°
Gravitační zrychlení	9.81	m/s 2	

Zobrazit

- ☐ Výchylku
- ☑ Rychlost
- ☐ Zrychlení
- ☐ Sílu

Ovládání

START

STOP

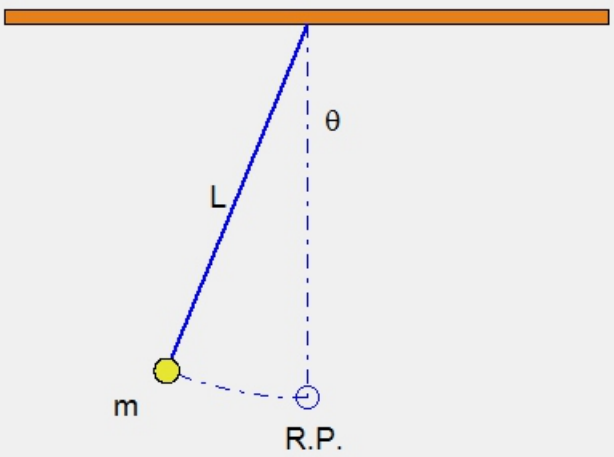
◯ Zpomaleně

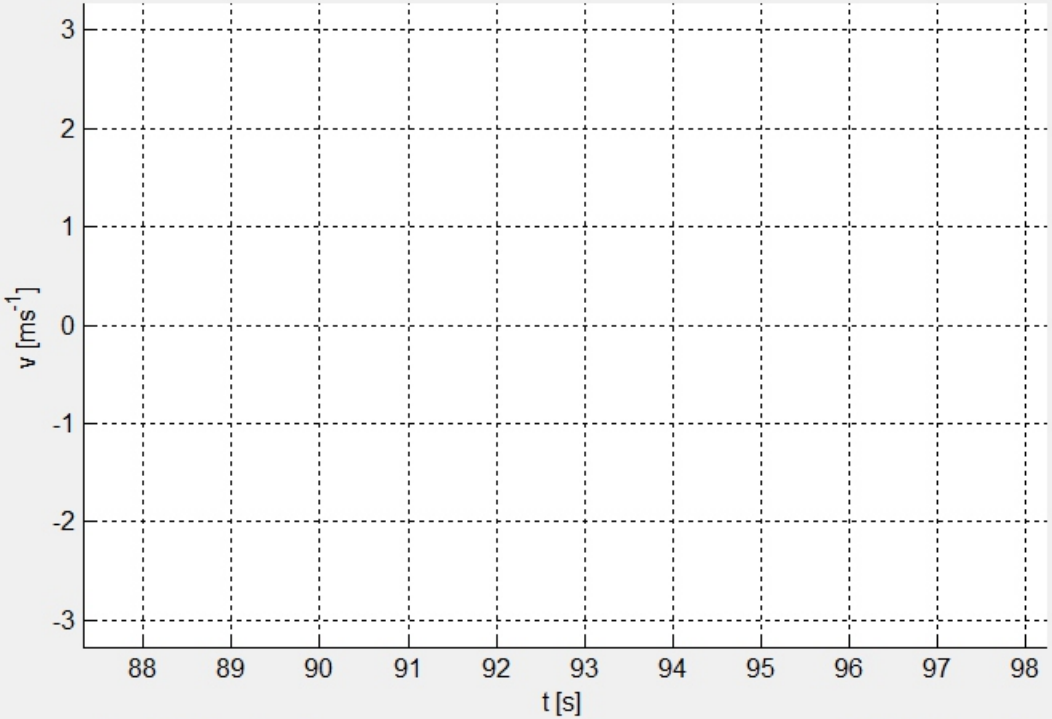
aplett_kyvadlo

Matematické kyvadlo

Popis kyvadla

L - délka závěsu | m - hmotný bod | théta - úhel vychýlení | R.P. - rovnovážná poloha





Perioda kmitů (T) 0 s

Okamžitá rychlost 0.046043 m/s

Nastavení

Délka závěsu	7.4		m
Hmotnost	5.7		kg
Úhel	22		°
Gravitační zrychlení	9.81	m/s 2	

Zobrazit

- [ ] Výchylku
- [x] Rychlost
- [ ] Zrychlení
- [ ] Sílu

Ovládání

START

STOP

( ) Zpomaleně