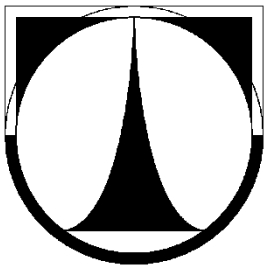
TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta Strojní



BAKALÁŘSKÁ PRÁCE

Webová aplikace pro aproximaci přechodových charakteristik metodou profesora Strejce

Web application to approximate transient characteristics by professor Strejc

Liberec, 2012 Martin Sova

Technická univerzita v Liberci

Fakulta strojní

Katedra aplikované kybernetiky

Jméno a příjmení: Martin Sova
Studijní obor: M2341 - Strojírenství

Zaměření: Inženýrská Informatika

Téma práce:

Webová aplikace pro aproximaci přechodových charakteristik metodou profesora Strejce

Web application to approximate transient characteristics by professor Strejc

Vedoucí bakalářské práce: Ing. Michal Moučka, Ph.D.

Rozsah bakalářské práce:

Počet stran textu:.....53
Počet obrázků:.....24
Počet tabulek:.....2
Počet příloh (vložené CD):.....1

V Liberci dne 8. 5. 2012

Anotace

Bakalářská práce řeší využití vývojového prostředí Matlab a webových serverů při realizaci výukového programu na aproximaci přechodových charakteristik metodou profesora Strejce.

Program využívá prostředí Matlab, programovací jazyk JAVA, technologii servletů a webový server Apache Tomcat. Výsledek této práce by měl sloužit pro výuku aproximace přechodových charakteristik metodou profesora Strejce. Práce také může být návodem, jak vytvořit jiné servlety za použití nástrojů ke zpracování této bakalářské práce.

Annotation

Bachelor thesis solves use of Matlab development environment and web servers in the implementation of educational program to approximate the transient characteristics by Professor Strejc.

The program uses the Matlab development environment, programming language JAVA, technology of servlets and Apache Tomcat. The result of this work should serve to teach the method of approximation of the transient characteristics professor Strejc. Work may also be a instructions, how to create other servlets using tools to handle this bachelor thesis.

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce Ing. Michalu Moučkovi, Ph.D. za jeho cenné rady, konzultace, trpělivost a ochotu při zpracování této bakalářské práce.

Dále bych chtěl poděkovat mé rodině a především mým rodičům za a to, že mi umožnili studium na vysoké škole.

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do její skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

OBSAH

ÚVOD

Tato bakalářská práce popisuje nastudování dynamických webových aplikací na bázi servletů a použití této technologie na webovou aplikaci pro aproximaci přechodových charakteristik metodou profesora Strejce. Aplikace využívá možností vývojového prostředí Matlab, který krok po kroku interaktivním způsobem demonstruje uvedenou aproximační metodu včetně importu dat. Celá aplikace je následně exportována do Javy, kterou vhodně zvolený webový server spouští jako webovou aplikaci. Práce se také zabývá popsáním prvních kroků při studování jazyka Java, naprogramování prvních funkcí ve vývojovém prostředí Matlab a následný export této funkce do podoby, který bude kompatibilní s jazykem Java. Další kroky představují výběr vhodného webového serveru a zprovoznění Java komponenty jako servletu. Dále je v práci nastíněn způsob programování metody prof. Strejce ve vývojovém prostředí Matlab. Programování této „papírové“ metody někomu může přijít jako zbytečné, ale jako demonstrace ukázky možností Matlabu a též jako demonstrace řešení této metody, je zmiňovaná metoda vhodná. Navíc zprovozněním servletu se z tohoto programu stává i nástroj pro výuku této byť staré, nepřesné, ale jednoduché metody pro identifikaci dynamických systémů. Z důvodu nepřesnosti není ani předpoklad, že by se tento program stal fakultou používaným nástrojem pro identifikaci soustav. Proto by tato bakalářská práce jako celek měla sloužit zejména pro výuku a rozšíření pojmů jako Servlet, Matlab Builder JA a metoda prof. Strejce.

Pro splnění daného úkolu se musí postupovat podle určitého plánu:

a. Nastudování jazyka JAVA a Java Servlety
    - Vytvoření a spuštění základních JAVA programů
    - Zprovoznění webového serveru a Java Servletů
b. Nastudování prostředí Matlab
    - Naprogramování několika funkcí
    - Seznámení s Matlab Builder JA
c. Nastudování aproximace přechodových charakteristik
d. Naprogramování metody prof. Strejce v prostředí Matlab
e. Vytvoření JAVA komponenty
f. Naprogramování servletu
g. Zprovoznění a odladění servletu

1 JAVA

Programovací jazyk Java je objektově orientovaný jazyk a patří k nejpoužívanějším jazykům, a to hlavně z důvodu přenositelnosti mezi platformami. Dnes se s Javou můžeme setkat od systémů s čipovými kartami (JavaCard) přes mobilní zařízení (Java ME (Micro Edition)) a desktopové systémy (Java SE (Standard Edition)) až po distribuované systémy (podpora aplikací v síti) pracující po celém světě (Java EE (Enterprise Edition)). Přenositelnost se samozřejmě vztahuje pouze na platformy, pro které je připravená Java platforma. Samozřejmě existuje široké spektrum programovacích jazyků, z nichž mnohé jsou kvalitní. Java má ovšem silné argumenty, proč použít právě ji, jako například. [1]:

a. Java má jednoduchou syntax.
b. Sama se stará o uvolňování nepotřebné paměti.
c. Použitelná jak pro malé tak i velmi rozsáhlé projekty.
d. Přenositelnost.
e. Vhodná pro začátečníky.
f. Je opravdu široce používaná, tudíž jsou pro ni i velmi kvalitní vývojová prostředí a můžeme se inspirovat z tisíců otevřených Java projektů.

Ovšem Java, stejně jako každý jiný programovací jazyk, není vhodná za všech okolností. Javu není vhodné použít zejména pro aplikace jako Real-time systémy (systémy, kde je zapotřebí garantovat odezvu v daném časovém horizontu), drivery (ovladače), součásti operačních systémů a výpočetně velmi náročné aplikace. Obecně má Java nižší výkon než jazyk C. Dá se říct, že tedy Java není vhodná pro nízkoúrovňové programování, kde je vhodnější jazyk C [1].

Již zmíněnou přenositelnost zajišťuje tzv. kompiler (překladač), který kompiluje (překládá) zdrojové kódy do tzv. java bytecodu. Při spuštění Java programu se zkompilovaný kód převede na strojový kód procesoru s ohledem na operační systém, toto převedení zajišťuje JVM (Java Virtual Machine). Java kód je tedy možné napsat jen jednou a pak spustit na jakékoliv platformě. K spuštění Java kódu potřebujeme tzv. sadu standardních knihoven API (Application Programming Interface). JVM a API tvoří celek a je distribuován jako JRE (Java Runtime Environment). Pro kompilaci programu je třeba samozřejmě kompilátor, který je součástí JDK (Java Development Kit). JDK také obsahuje další vývojové nástroje, jako JRE (JVM a API), kompiler, debbuger pro

ladění programů, nástroj javadoc pro generování dokumentace, nástroj pro vytváření JAR archivů a další nástroje. Takže pokud budeme chtít Java program jen spustit, stačí nám JRE, ale pokud budeme chtít programy vytvářet a kompilovat, budeme potřebovat JDK [1].

1.1 Instalace a nastavení systémových proměnných

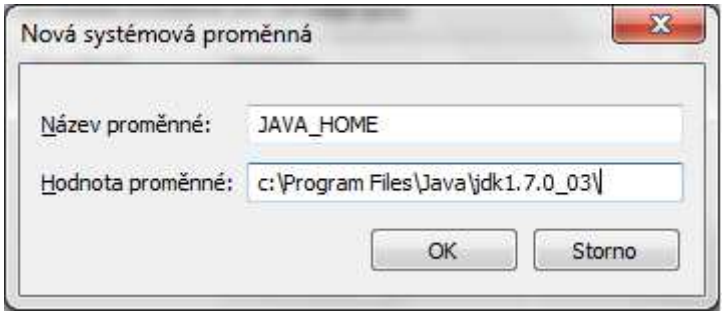
Pokud bude potřeba Javu používat pro vývoj, bude nezbytné stáhnout Java JDK. Častým problémem začátečníků je, který JDK vybrat, protože na stránkách http://www.oracle.com/technetwork/java/index.html je několik verzí Java JDK. Tato práce se zaměřuje na vývoj servetů pro desktopové systémy a proto je potřeba Java SE. Na uvedených stránkách se stáhne Java Platform (JDK) 7u3 (v této době je to aktuální verze), po potvrzení licence se stáhne verze pro používaný operační systém. V této práci se Java instaluje na Windows x64 (64-bit). Nainstaluje se podle zvyklostí z MS Windows. Současně se nainstaluje i Java FX (platforma pro vývoj „bohatých webových aplikací“). Instalátor následně otevře i webovou stránku s registrací, ale tu můžeme v klidu přeskočit. Počítač, na kterém jsou všechny úkony demonstrované, používá Windows 7 v 64bitové edici.

Po nainstalování je nutné nastavit několik systémových proměnných, z důvodu, aby kompilátor věděl, kde najde soubory tříd, knihoven atd.. Ve Windows 7 se systémové proměnné mohou nastavit přes příkazový řádek nebo pohodlněji přímo ve Windows, cesta k proměnným je Start > pravý klik na Počítač > Vlastnosti > Upřesnit nastavení systému > Proměnné prostředí, a zde v položce Systémové proměnné vytvoříme několik proměnných [1].

JAVA_HOME

Tato proměnná musí obsahovat kořenový adresář instalace Javy (obr. 1.1-1):

```
c:\Program Files\Java\jdk1.7.0_03\
```

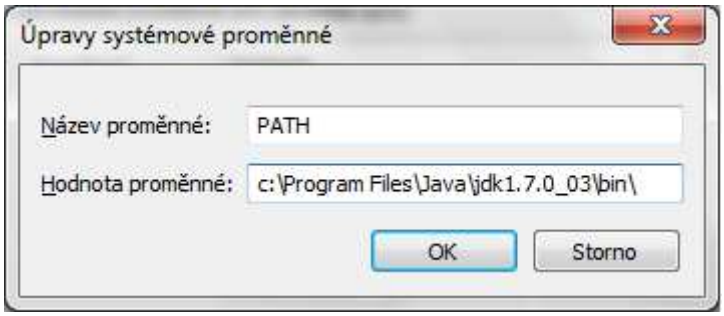


Obr. 1.1-1: Založení nové systémové proměnné

PATH

Tato proměnná musí odkazovat na adresáře vývojových nástrojů (obr. 1.1-2):

c:\Program Files\Java\jdk1.7.0_03\bin\



Obr. 1.1-2: Úprava nové systémové proměnné

CLASSPATH

Tato proměnná odkazuje na cesty ke spustitelným souborům, uživatelským knihovnám jako JAR soubory a podobně. Využita je pak zejména s nastavením Apache Tomcat.

Po nastavení všech proměnných potvrdíme OK a tímto je nakonfigurovaná Java.

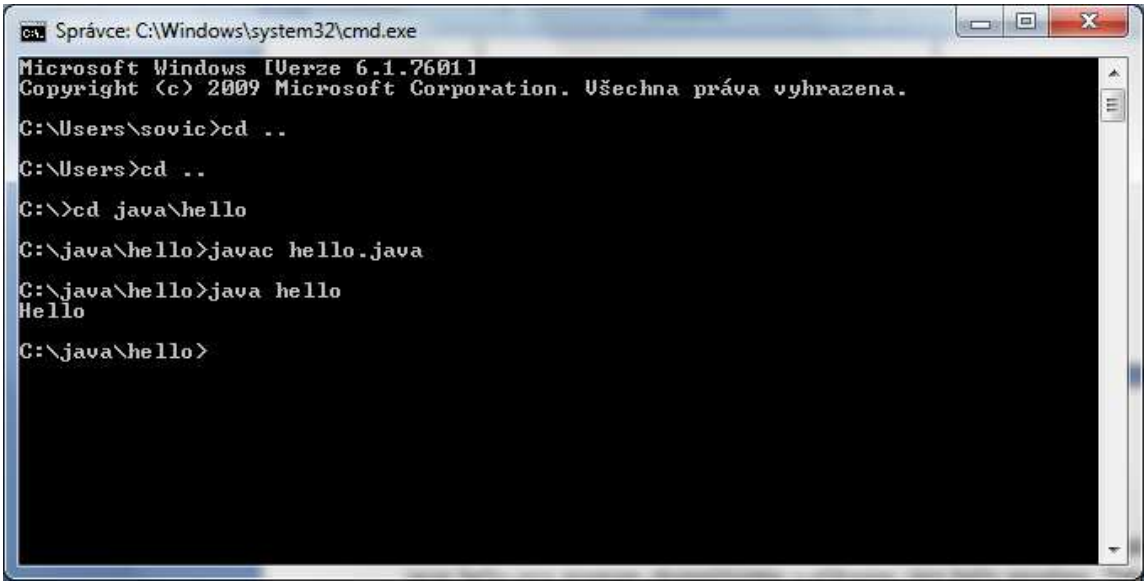
1.2 První Java program

Pro potvrzení správnosti konfigurace se například vytvoří soubor s názvem hello.java v adresáři C:\java\hello\ do kterého se vloží kód:

```
public class hello {
```

```
  public static void main(String[] args) {
    System.out.println("Hello");
  }
}
```

Soubor se uloží a spustí se Příkazový řádek. V něm je nutné soubor přesunout do adresáře s uloženým souborem a příkazem javac hello.java se program zkompiluje a příkazem java hello se program spustí. Takto by měl vypadat výsledek (obr 1.2-1):



Obr. 1.2-1: Příkazový řádek

Po zobrazení řetězce Hello je potvrzena funkčnost a mohou se bez problému vyvíjet jiné aplikace. Pro náročnější vývoj je vhodné používat nějaké vývojové prostředí, jako je Eclipse či NetBeans IDE. Ale pro začátečníka se více hodí nějaký textový editor s výrazněnou syntax, jako je Notepad++ či PSPad. Z autorovy vlastní zkušenosti se doporučuje zejména kombinace NetBeans IDE a Notepad++.

1.3 Servlety

Servlet je program, který běží na webovém serveru a působí jako střední vrstva mezi požadavkem přicházejícím z webového prohlížeče nebo do jiného http klienta a databází či aplikací na serveru HTTP. Servlety jsou odpovědí technologie Java na programování pomocí CGI – Common Gateway Interface. Jejich úkolem je [2]:

a. Přečíst všechna data odeslaná uživatelem - tyto údaje se obvykle vkládají do formuláře na webové stránce.
b. Vyhledat všechny další informace o požadavku, které jsou uloženy v požadavku HTTP - například informace o prohlížeči, cookies, hostitelském jménu žádajícího klienta.
c. Tvorba výsledků - tento proces může vyžadovat například komunikaci s databází.
d. Zformátovat výsledky uvnitř dokumentů - zejména uložení informací do stránky HTML.
e. Nastavení vhodných parametrů odezvy HTTP - sdělení prohlížeči, jaký typ dokumentu se vrací (např. HTML), nastavení parametrů cookies atd.

f. Odeslání dokumentu zpět ke klientovi - tento dokument lze poslat v textovém formátu (HTML), binárním formátu (obrázky GIF) nebo v komprimovaném formátu (GZIP), který překrývá jiný formát.

Na mnohé požadavky od klientů lze zodpovědět vrácením statických dokumentů a tyto dokumenty budou obslouženy serverem bez použití servletů. Ale v mnoha případech statický výsledek není dostačující a na každý požadavek je potřeba vytvořit novou stránku. Důvody, proč webové stránky musí být vytvořeny za běhu jsou např. [2]:

a. Webová stránka je založena na datech předložených uživatelem
   - Např. on-line obchod.
b. Webová stránka je odvozena z dat, která se často mění
   - Stránky o počasí či zprávy
c. Webová stránka používá informace ze společných databází nebo z jiných serverových zdrojů
   - Např. stránka elektronického obchodu, která uvádí cenu a dostupnost položek, které jsou na prodej.

Jak již bylo zmíněno, velkou výhodou jazyka Java je její rozšířenost a přenositelnost. Tyto výhody přebírají servlety a proto servlety napsané například pro Apache, mohou běžet třeba i například na IBM WebSphere. Servlety mají také rozsáhlou infrastrukturu

pro automatickou syntaktickou analýzu a dekódování formulářových dat HTML, dekódování a nastavování záhlaví HTTP, obsluhu cookies, sledování sezení a mnoho dalších funkcí na vysoké úrovni. Kromě toho se programátor jazyka Java již nemusí učit jiný jazyk, když ví, že technologie Java poskytuje přenositelný a opakovaně použitelný kód. Další obrovskou výhodou servletů je jejich cena. Pro osobní použití či máloobsažné stránky stačí volně šiřitelné webové servery, pro obsáhlejší stránky je vhodné použít komerční produkty, které jsou relativně drahé, avšak pokud máme již nějaký komerční webový server, přidání servletové podpory stojí velmi málo.

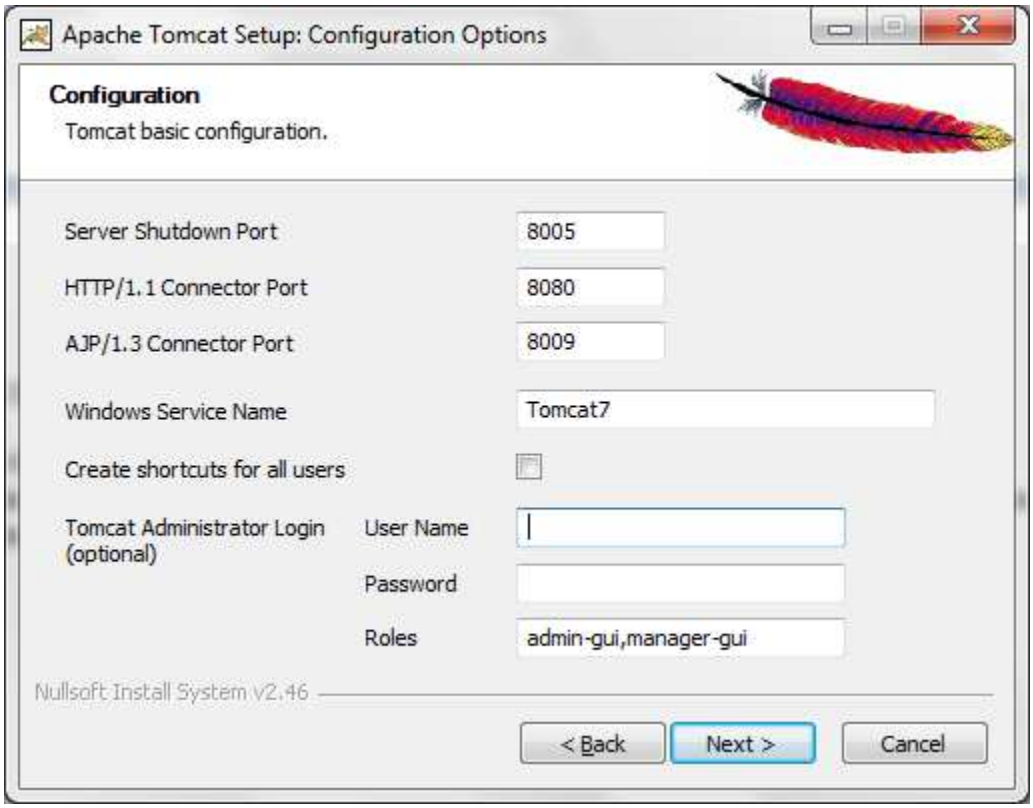
2 APLIKAČNÍ SERVER

Pro nasazení Java komponenty na web musí mít uživatel nainstalovaný nějaký aplikační server s podporou servletů a JavaServer Pages. Existuje mnoho různých serverů a konfigurace je pro každý z nich jiná. Pro tuto bakalářskou práci bylo zvoleno, zejména z finančních důvodů, open-source řešení. Ze stále aktuálních a známých open-source serverů jsou k dispozici například Apache Tomcat a GlassFish. Pro vývoj zadané webové aplikace byl zvolen Apache Tomcat, z důvodu, že je stále aktualizovaný, jednoduchý, transparentní, má menší náročnost na výpočetní výkon, podstatně rychleji se v něm vyvíjí a ve verzi Apache Tomcat 7 podporuje i specifikace Servlet 3.0. Bohužel Tomcat vyžaduje větší úsilí pro konfiguraci než některé komerční produkty. Existuje ovšem spousta dalších open-source webových serverů, ale Apache Tomcat je nejrozšířenější a tudíž má největší podporu jak komunity, tak autorů. Open-source servery se v některých věcech plně vyrovnají i komerčním řešením, jako je například WebSphere Application Server od IBM, který vyniká zejména robustností a bezpečností, ale je bohužel drahý [2].

2.1 Instalace Apache

Apache Tomcat je možné získat například na stránkách http://tomcat.apache.org/ kde by v sekci Tomcat 7.0 měla být ke stažení nejnovější verze. Dobré řešení je stáhnout rovnou 32-bit/64-bit Windows Service Installer a nainstalovat dle pokynů.

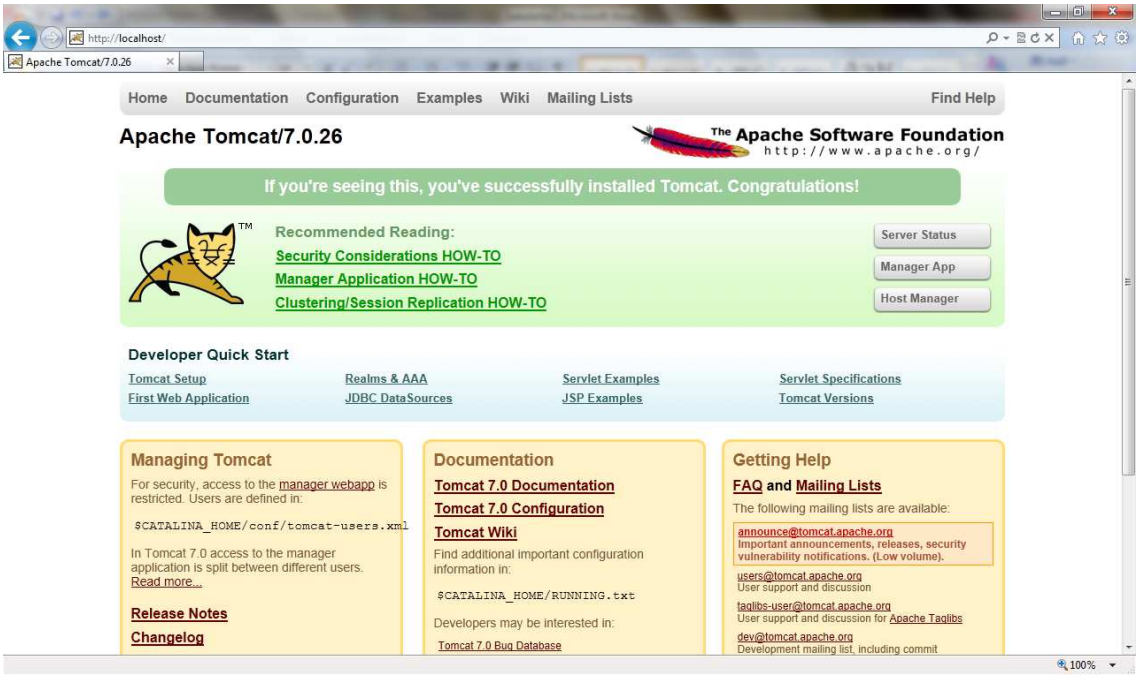
Není-li nainstalován na PC jiný webový server, je vhodné přepnout HTTP/1.1 Conncetor Port na hodnotu 80, což je standardní číslo portu http. Hodnota 8080 je v instalaci proto, aby se vyloučily konflikty s jinými webovými servery. Dále se vyplní User Name a Password. V kolonce Roles se mohou vyplnit role uživatele. Další role najdeme v příslušných dokumentací k potřebným programům (obr. 2.1-1):



Obr. 2.1-1: Nastavení Apache Tomcat

Tlačítkem Next se provede přesun na další okno instalace a instalátor požádá o cestu k JVM, ale standardně si ji (instalátor) najde sám. V případě instalace na vývojovém PC to je C:\Program Files\Java\jre7. Pak je potřeba zvolit cestu kam se má Apache Tomcat nainstalovat. Vhodné je ponechat standardní cestu a následně spustit instalaci. Na konci instalace se instalátor zeptá, zda má být spuštěn Tomcat. Pro ověření funkčnosti je nutné Tomcat spustit a do prohlížeče zadat adresu http://localhost. V případě, že byl ponechán port 8080 je rozhraní Apache Tomcat vyvoláváno adresou http://localhost:8080.

Potvrzením adresy by se měla zobrazit stránka představující funkčnost Apache Tomcat (obr. 2.1-2):



Obr. 2.1-2: Funkční Apache Tomcat

2.2 Vytvoření prvních servletů a konfigurace Apache Tomcat

V dostupné literatuře jsou k nalezení pouze univerzální postupy jak vytvořit a zprovoznit první servlety, které ne vždy odpovídají představám o funkčnosti a bohužel zaberou spoustu času na vyzkoušení. Proto se v této práci uvádí podrobný návod jak krok za krokem zprovoznit servlet pro systém Windows 7 v 64bitové edici.

Před prvním psaním servletů je potřeba nakonfigurovat proměnné tak, aby kompilátor věděl, odkud má brát třídy, servlety a JSP. V nabídce Proměnné prostředí, v kolonce Systémové proměnné se upraví (či vytvoří, pokud ještě není vytvořená) proměnná **CLASSPATH** a přejde se do adresáře instalace knihoven Apache Tomcat. Cesta je v tomto případě C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\. Zde jsou veškeré knihovny, které bude používat kompilátor a Apache Tomcat. Nelze jen zadat cestu do složky, ale přímo odkazy na soubory knihoven. Musejí se tedy zkopírovat celé cesty a za každým odkazem dopsat potřebné knihovny. Hodnoty se oddělují znakem středník „;“. Jako první by měla být přidaná knihovna servlet-api.jar. Celá cesta v CLASSPATH může vypadat takto [1]:

C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\servlet-api.jar.

Pro zprovoznění prvního servletu se přejde do složky webapps v nainstalovaném adresáři s Apache Tomcat. Zde se vytvoří složka helloworld a v ní se vytvoří složka WEB-INF a v ní složka classes. Ve složce WEB-INF se vytvoří soubor s názvem web.xml a po otevření se do něj vloží následující kód a uloží se [2]:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

 <!--<!DOCTYPE web-app  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd"> -->

 <web-app>
  <servlet>
   <servlet-name>hello</servlet-name>
   <servlet-class>helloworld</servlet-class>
  </servlet>
  <servlet-mapping>
  <servlet-name>hello</servlet-name>
  <url-pattern>/helloworld</url-pattern>
  </servlet-mapping>
 </web-app>
```

Ve složce classes se vytvoří soubor s názvem helloworld.java. Opět se otevře, vloží následující kód a uloží se:

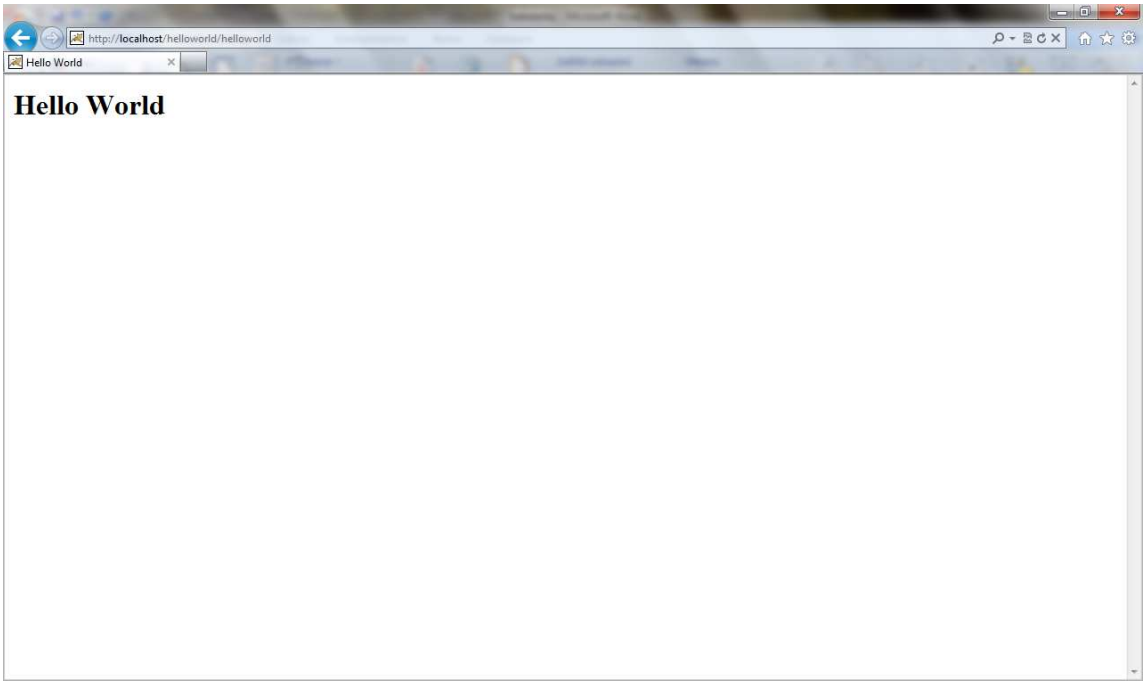
```
import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;

 public class helloworld extends HttpServlet{
   public void doGet(HttpServletRequest request,     HttpServletResponse response)
   throws ServletException,IOException{
   response.setContentType("text/html");
   PrintWriter print = response.getWriter();
   print.println("<html>");
   print.println("<head><title>Hello World</title></title>");
   print.println("<body>");
   print.println("<h1>Hello World</h1>");
```

```
  print.println("</body></html>");
  }
}
```

Po kompilaci by měla být následující struktura složek a souborů.:

```
|-- helloworld
|   `-- WEB-INF
|       |-- classes
|       |   |-- helloworld.class
|       |   `--helloworld.java
|       `-- web.xml
```

Ve Windows vpravo dole na liště v oznamovací oblasti je ikona Apache Tomcat, kde se po kliknutí pravého tlačítka myši otevře nabídka, kde je položka Stop service. Po vypnutí služby se opět Tomcat tím samým způsobem zapne, až na to, že se klikne na Start Service. Toto by se mělo dělat po každé kompilaci a jakékoliv změně ve složce servletu. Pro kontrolu se do prohlížeče zadá adresa servletu http://localhost/helloworld/helloworld a měl by být viděn výsledek (obr. 2.2-1):



Obr. 2.2-1: První program

Tento postup byl velice podrobný a jeho dodržením se ušetří spousta času. Velice důležité je zachovat strukturu složek a souborů. Pokud se nezachová, servlet nebude pracovat tak jak by měl a zbytečně se bude hledat chyba v kódu nebo v Apache Tomcat. Tato struktura je vhodná pro servlety, které generují přímo webovou stránku. Také se můžou využít specifikace Servlet 3.0 a místo tvorby web.xml se použijí anotace.

Struktura složek a souborů může vypadat i takto:

```
|-- ROOT
|--obrazky
|   `--hlavicka.jpg
|-- WEB-INF
|   |-- classes
|   |   |-- program.class
|   |   `--program.java
|   `-- lib
|       `-- knihovna.jar
|
`-- index.jsp
```

Pokud je programátorovi alespoň trochu znám jazyk Java, a dodrží tento postup při tvorbě prvních servletů, neměl by mít problém vytvářet vlastní servlety v relativně krátkém čase.

3 MATLAB

Programovací jazyk MATLAB je určen pro vědeckotechnické účely, modelování, analýzu a zpracování dat, měření a zpracování signálu apod. Využití je obrovské a ve spojení s webovým serverem se jedná o opravdu mocný nástroj. Jedním z cílů této práce je také vytvořit spojení Matlabu a webového serveru. Aplikace vytvořená v Matlabu se převede na Java komponentu, kterou lze pak nasadit na webový/aplikační server, kde se spouští a provádí požadované výpočty. Toto spojení ovšem s sebou nese opět nějaká úskalí, kterými si musí každý vývojář projít.

Pro vývoj Java komponenty je samozřejmě potřeba MathWorks Matlab, který je buď nainstalovaný, nebo se nainstaluje dle pokynů instalátoru. Důležitou součástí Matlabu je Matlab Compiler (překladač) a Matlab Builder JA. Matlab Compiler je nástroj, který spolupracuje s Matlab toolboxy a umožňuje vytvářet komponenty, které mohou komunikovat s externími aplikacemi. Mohou se vytvářet také komponenty pro Excel či C/C++ nebo .NET platformu. Ovšem tato práce se zabývá vytvořením Java komponenty, pro kterou je potřeba právě Matlab Builder JA, který umožňuje vytvoření Java třídy z Matlabovských souborů tzv. m-filů.

Pro spuštění Java komponenty není de facto potřeba mít Matlab nainstalovaný, stačí pouze nainstalovaný tzv. Matlab Component Runtime (MCR). MCR představuje výpočtové jádro Matlabu a poskytuje knihovny Matlabu pro vytvořené komponenty různých technologií. MCR umožňuje spouštět komponenty i na počítačích, kde není Matlab k dispozici. Ovšem výhoda servletů je, že běží na straně webového serveru, tudíž není nutné mít nainstalovaný Matlab či MCR na koncovém počítači [5], [11].

3.1 Vytvoření a zprovoznění komponenty

Pro jednoduchou ukázku se vytvoří Java komponenta a spustí se na serveru. Například se vytvoří Java komponenta pro zaokrouhlení čísla funkcí round(x) [5]:

a. V Matlabu se vytvoří nový m-file, do kterého se vloží tento kód a pojmenuje se jako zaok.m [3],[4] :

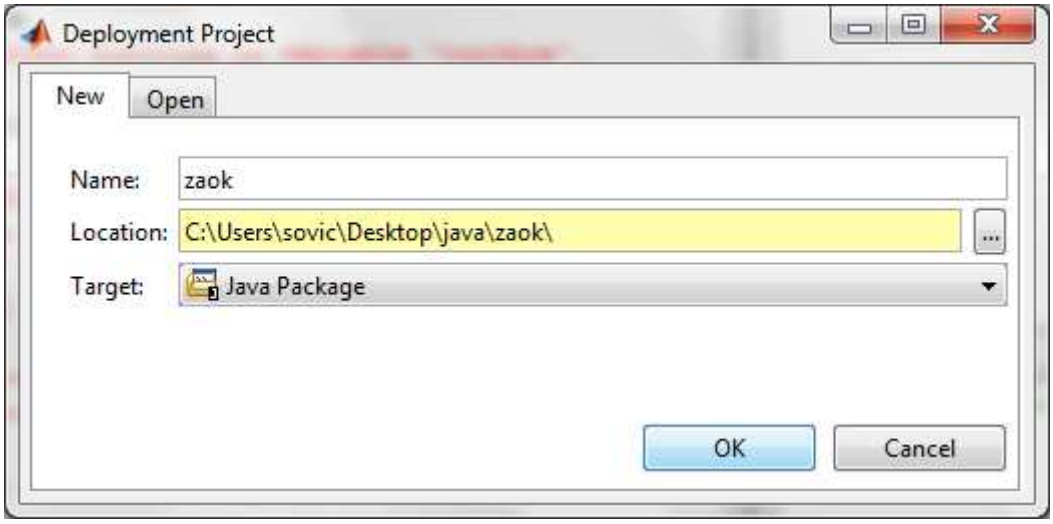
```
function vysledek=zaok(vstup)
vysledek = round(vstup);
end
```

b. Po uložení a spuštění takto vypadá výsledek (obr. 3.1-1):

```
>> zaok(4.65)

ans =

     5

>>
```

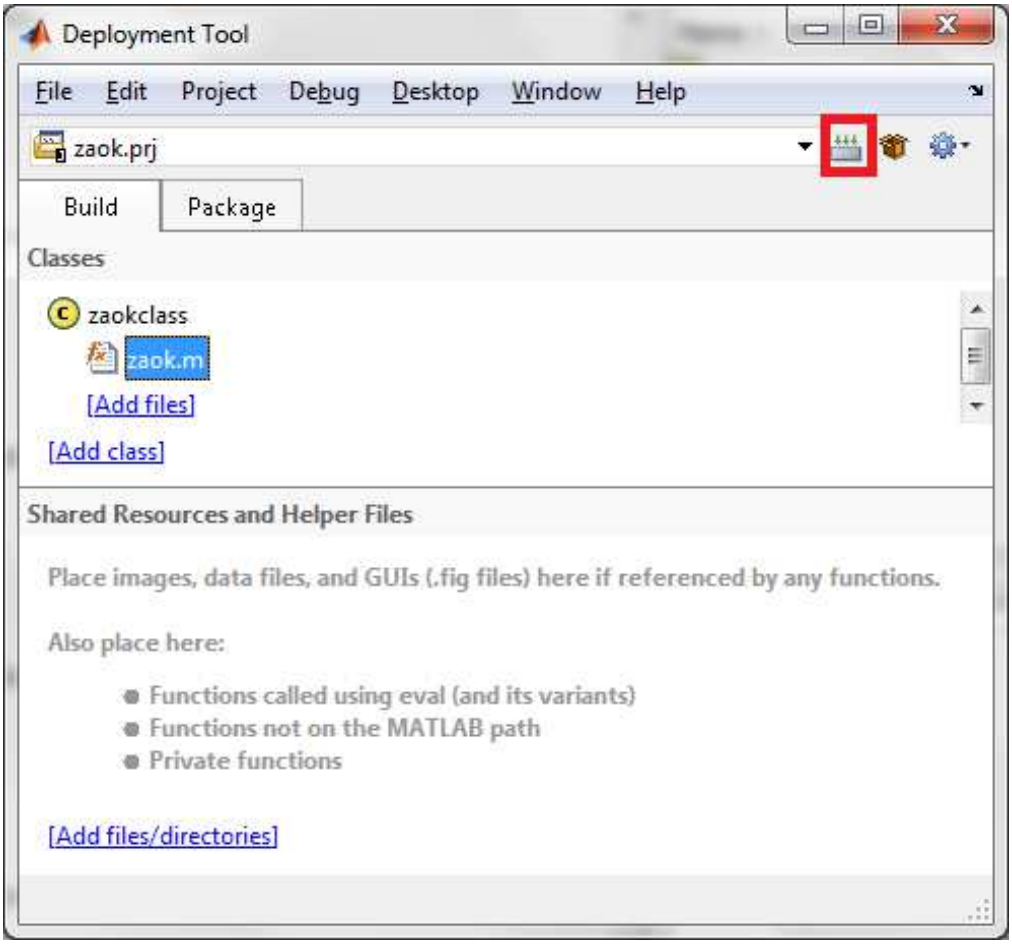
Obr. 3.1-1 Výsledek zaokrouhlení

c. V Matlabovském Command Windows se zadá příkaz deploytool, pojmenuje a určí se cesta ukládaného souboru, dále se z kolonky Target vybere položka Java Package a potvrdí se tlačítkem OK (obr. 3.1-2):



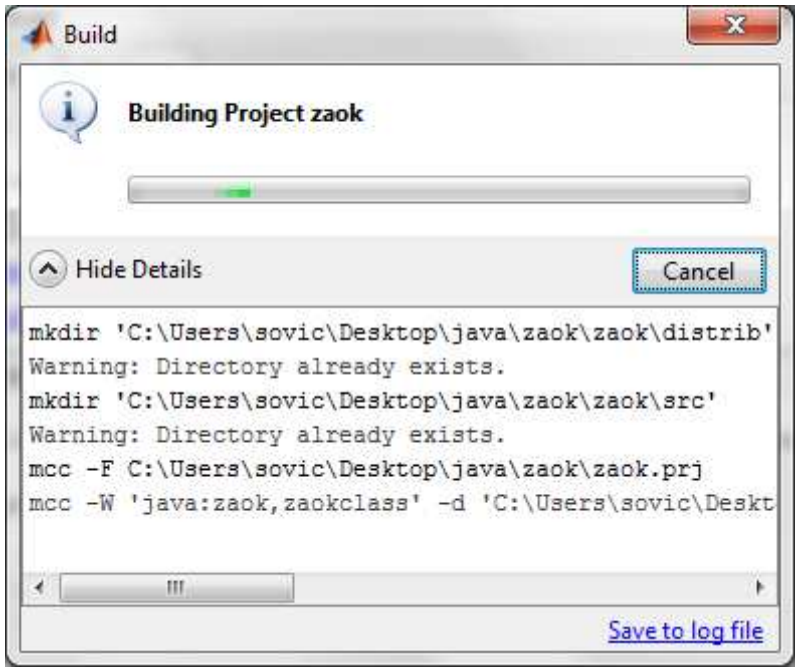
Obr. 3.1-2: Deployment Project

d. Zobrazí se nové okno Deployment Tool kde se v záložce Build po kliku na Add Class vloží jméno třídy zaokclass a po potvrzení se klikne na Add File a vloží se vytvořený m-file s názvem zaok.m. Dále se už jen klikne na tlačítko Build a počká se na dokončení projektu (obr. 3.1-3):



Obr. 3.1-3: Založení nového projektu

e. Budování projektu (obr. 3.1-4):



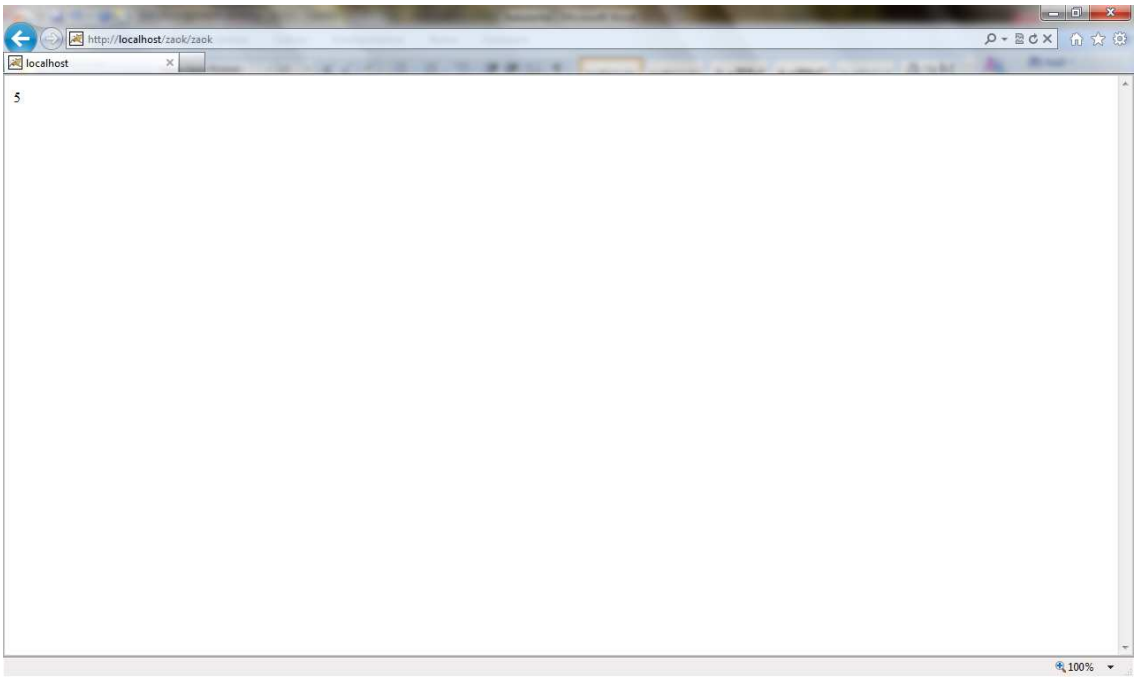
Obr. 3.1-4: Budování projektu

Po dokončení projektu se ve složce webapps (v instalaci Apache Tomcat) vytvoří složka s názvem zaok a v ní musí být dříve vysvětlená struktura složek servletu. Navíc se ve složce WEB-INF musí vytvořit složka lib, kam se kopírují potřebné knihovny. V tomto případě je to soubor zaok.jar, který vygeneroval Matlab a je uložený ve dříve zadané cestě v adresáři distrib. Dále ve složce lib musí být nakopírovaný soubor javabuilder.jar, který se nachází ve složce (dle instalace a verze Matlabu) c:\Program Files\MATLAB\R2009b\toolbox\javabuilder\jar\. Odkazy k těmto dvěma souborům musí být rovněž zadány v Systémových proměnných, v proměnné CLASSPATH. Dále se ve složce classes vytvoří soubor s názvem zaok.java a vloží se následující kód, který je následně zkompilován:

```
import com.mathworks.toolbox.javabuilder.*;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.annotation.MultipartConfig;
import zaok.*;
@WebServlet(name="zaok", urlPatterns={"/zaok"}) //specifikace Servlet 3.0
@MultipartConfig  // specifikace Servlet 3.0
 public class zaok extends HttpServlet{
                  public void doGet(HttpServletRequest request, HttpServletResponse
response)
  throws ServletException,IOException
{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        zaokclass x = null;
        Object[] result = null;
        MWNumericArray in = null;
        MWNumericArray ven = null;
                try {
                        x = new zaokclass();
                        dovnitr = new MWNumericArray(4.65); // vlozeni hodnoty, co
//chceme zaokrouhlit
// volani komponenty. První hodnota predstavuje pocet vystupnich parametru, druha
//hodnota je pole hodnot. Komponenta vraci pole typu Object, ktere se nasledne muze
//pretypovat
```

```
                    result = x.zaok(1, dovnitr);
                    ven = (MWNumericArray)result[0];
                    out.println(ven);
                } catch (MWException e)
            //  zachyceni vyjimky
            {
                                        e.printStackTrace();
                                    }
        finally {
        // zruseni objektu MWArray a objektu komponenty
                                    x.dispose();
                                    MWArray.disposeArray(dovnitr);
                                    MWArray.disposeArray(ven);
                                    MWArray.disposeArray(result);
            }

 }
 }
```

Výsledek se zobrazí adresou http://localhost/zaok/zaok (obr. 3.1-5).



Obr. 3.1-5: Fungující spojení

3.2 Předávání dat mezi JAVA a Matlab

Protože Matlab používá jiné datové typy než Java, musí se využít nástroj pro převod dat. Například, pokud je potřeba volat funkce Matlabu z Java komponenty, musí se předávané parametry převést na kompatibilní datové typy Matlabu. Nástroj pro převod se nazývá MWArray a je to abstraktní třída, která slouží pro ruční převod datových typů z Matlabu do Javy a zpět. Toto API poskytuje Matlab Builder JA, které abstraktní třídu MWArray implementuje jako com.mathworks.toolbox.javabuilder.MWArray balík. Abstraktní třída MWArray poskytuje několik podtříd. Každá podtřída reprezentuje jeden datový typ Matlabu [5],[11].

Podtřídy MWArray:

a. MWCharArray
b. MWCellArray
c. MWLogicalArray
d. MWNumericArray
e. MWStructArray

Třída MWArray a třídy odvozené poskytují také konstruktory pro vytvoření a zrušení MATLAB polí, metody get a set pro čtení a nastavování dat v poli, metody k určení vlastností polí, srovnání metod pro testování rovnosti nebo pořadí polí a metody pro převod do jiných datových typů [5],[11].

Metody MWArray:

a. **ClassID()** – vrátí datový typ MWArray
b. **ColumnIndex()** – vrátí index sloupce elementu v poli
c. **Dispose()** – uvolní nativní zdroje MWArray
d. **DisposeArray(Object)** – uvolní nativní zdroje pro všechny MWArray
e. **Get(int)** – vrátí prvek pole
f. **Get(int[])** – vrátí prvky pole
g. **IsEmpty()** – otestuje, zda je pole prázdné
h. **MWArray()** – vytvoří prázdný Array

i. **Set(int, Object)** – na pozici indexu Object nahradí hodnotu
j. **Set(int[], Object)** – na pozici indexu Object nahradí hodnoty
k. **Setdata(Object)** – hodnoty v MWArray nastaví na požadované hodnoty.

Konstruktory a datové typy MWNumericArray:

a. MWNumericArray() – prázdné double Array
b. MWNumericArray(MWClassID) – prázdné pole typu MWClassID
c. MWNumericArray(type, MWClassID) – Real Array typu MWClassID
d. MWNumericArray(type) – z pravidel defaultní konverze se může nastavit Real Array
e. MWNumericArray(type, type, MWClassID) – komplexní pole typu MWClassID
f. MWNumericArray(type, type) – komplexní array pole určený z pravidel konverze

Předání argumentů z Javy do Matlabu [5], [11]:

```
MWNumericArray in = null;  //nove numericke pole
x = new zaokclass();
in = new MWNumericArray(4.65);  //prevod na datovy typ
                                //matlab
result = x.zaok(1, in);  //predani a prijem
```

Vracení argumentů z Matlabu do Javy [5], [11]:

```
result = x.zaok(1, in);  //predani a prijem (jako predchozi pripad)
ven = (MWNumericArray)result[0];  //nacteni vysledku jako
                                  //numericke pole pocinaje
                                  //první hodnotou
out.println(ven);
```

4 APROXIMACE PŘECHODOVÝCH CHARAKTERISTIK

Úplně přesné určení dynamických vlastností skutečných (reálných) systémů je zcela nemožné, proto se skutečné-reálné dynamické soustavy obvykle určují aproximací skutečných vlastností dynamického systému vlastnostmi náhradního systému s předem zvolenou strukturou. Struktura se zvolí například posouzením přechodové charakteristiky *h(t)* využitím právě aproximace – přiblížení (je to jen jedna z možností, lze určovat též z impulzní, frekvenční charakteristiky, nebo na základě PRBS na vstupu a na výstupu ARX, ARMAX model a jiné). Chování takto získané soustavy bude blízké chování původní soustavy, podle toho, jak se podaří aproximovat původní soustavu. Do určité míry ji lze nahradit [7].

Dynamická charakteristika *y(t)* – je odezva systému na změnu vstupního signálu. Vstupní funkce může být ve tvaru skokové funkce nebo může být naměřenou odezvou soustavy pro **libovolně velkou změnu vstupní veličiny soustavy** [9].

Přechodová charakteristika *h(t)* – je odezva soustavy na buzení soustavy funkcí *η(t)* (platí za nulových počátečních podmínek). Přechodová charakteristika je dynamická charakteristika pro **velikost skokové změny vstupní veličiny právě rovné jedné** [9].

Obrazový přenos - je Laplaceův obraz výstupní veličiny *y(t)* k Laplaceova obrazu vstupní veličiny *u(t)* při nulových počátečních podmínkách. Laplaceovy obrazy se získají z originálů Laplaceovou transformací. Další definice uvádí obrazový přenos jako Laplaceův obraz impulsní přechodové (váhové) funkce. Obrazový přenos lze obecně napsat podle vzorce [6],[10]:

$$F(s) = \frac{b_m s^m + \cdots + b_1 s + b_0}{s^r(a_n s^n + \cdots + a_1 s + a_0)} = \frac{Y(s)}{U(s)} \rightarrow Y(s) = F(s) \cdot U(s) \cdot e^{-Tds},$$

kde - *F(s)* je obrazový přenos
- *r* je řád astatismu
- *n* a *m* jsou stupně polynomu
- *Y(s)* je Laplaceův obraz výstupní veličiny *y(t)*
- *U(s)* je Laplaceův obraz vstupní veličiny *u(t)*

Vlastnosti obrazového přenosu lze tedy bez důkazů shrnout do následujících vět. Obrazový přenos nezávisí na budící funkci ani na počátečních podmínkách, které ovšem dle definice musí být nulové. Obrazový přenos je také racionální lomenou funkcí komplexní proměnné s reálnými koeficienty a také popisuje dynamické vlastnosti pouze těch systémů, které nemění svoje parametry v čase [10].

4.2 Metoda prof. Strejce

Jednou z nejjednodušších a osvědčených metod je aproximační metoda prof. Strejce, která využívá tečny v inflexním bodě přechodové charakteristiky *h(t)*. Metoda patří mezi deterministické metody, tj. takové metody, které na vstupu soustavy předpokládají signál s přesně definovaným a matematicky snadno popsatelným průběhem, proti němuž jsou náhodné vlivy, jako je šum či poruchy, zanedbatelné. Takový signál může být např. jednotkový skok [7],[9].

Metodu prof. Strejce lze použít pouze pro soustavy:

- **stabilní statické**
- **druhého a vyšších řádů bez kmitavé složky**
- **s minimální fází** (jen jeden inflexní bod)

Jsou to nejčastější případy regulovaných soustav, u nichž všechny kořeny charakteristické rovnice jsou **reálné, záporné** a čitatelem přenosu je konstanta. Tuto metoda lze také použít i na aproximaci dynamických systémů s dopravním zpožděním. Pro účely aproximace je možné pracovat s odezvou zkoumaného systému *y(t)* vybuzeného z klidu skokovou změnou budícího signálu libovolné velikosti Δu, potom $y(t) = h(t)\Delta u$ [7],[9].

Podstatou aproximační metody prof. Strejce je, že se skutečná soustava nahradí jednodušší soustavou n-tého řádu se stejnými časovými konstantami nebo soustavou druhého řádu se dvěma různými časovými konstantami. Pro každý z těchto způsobů existuje samostatný postup stanovení konstant. O způsobu nahrazení rozhoduje velikost úseku $\tau_u = T_u / T_n$ (viz. Obr. 4.1-1). Metoda vychází ze skutečnosti, že v okolí inflexního

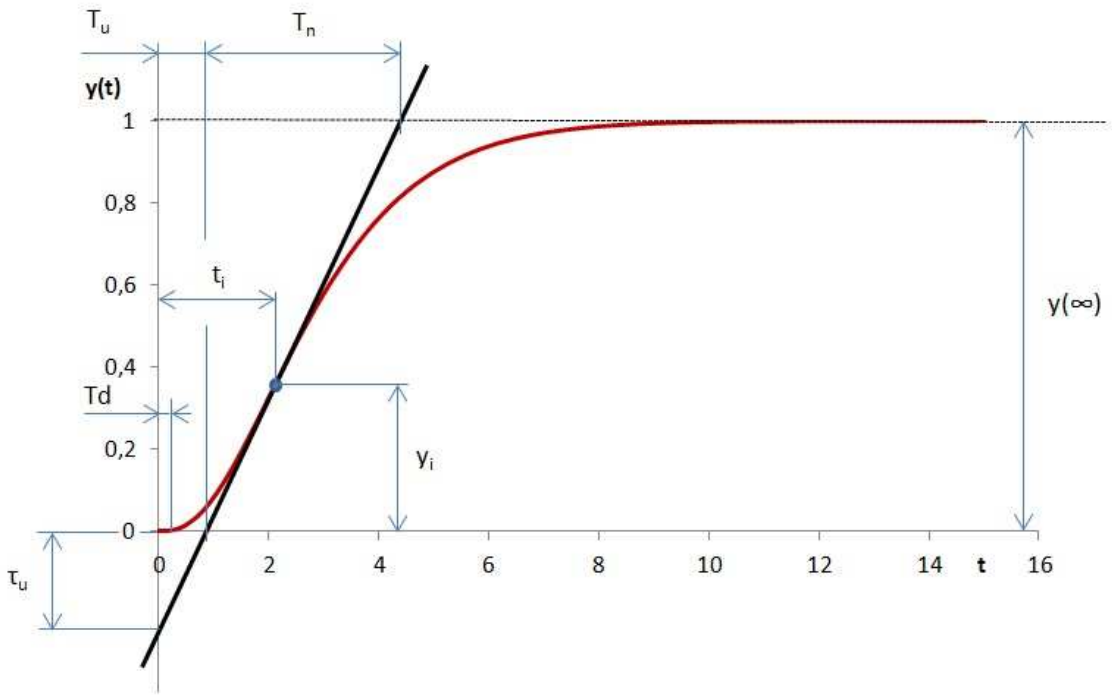
bodu je přechodová charakteristika takřka přímková, tudíž směrnici tečny není obtížné stanovit.

Aproximační přenosy:

$$\boldsymbol{F}(s) = \frac{\boldsymbol{K}}{(\boldsymbol{T}s+1)^n} \; pro \; \tau_u > 0{,}104 \qquad \boldsymbol{F}(s) = \frac{\boldsymbol{K}}{(\boldsymbol{T_1}s+1)\cdot(\boldsymbol{T_2}s+1)} \; pro \; \tau_u < 0{,}104$$

Postup určení aproximačního přenosu *F(s)* [7],[9].

1. Přechodová charakteristika (odezva *y(t)*), se překreslí v novém měřítku tak, aby se ustálená hodnota rovnala jedné.
2. Odhadne se inflexní bod přechodové charakteristiky a v něm se nakreslí tečna a určí se doba průtahu T_u ,doba náběhu T_n a jejich poměr $\tau_u = T_u / T_n$. Podle velikosti poměru se zvolí další postup aproximace.
3. Pokud je $\tau_u > 0{,}104$ volí se pro aproximaci systém n-tého řádu s vesměs stejnými časovými konstantami T. Pokud ale tato podmínka není splněna, tak se pokračuje bodem 8.
4. Podle hodnoty τ_u se z první tabulky (tab. 4.2-1), určí nejbližší řád n aproximační soustavy a také příslušná pořadnice y_i inflexního bodu.



Obr. 4.2-1: Graf

Tab. 4.2-1: Hodnoty pro jednu časovou konstantu

n	1	2	3	4	5	6	7	8	9	10
τ_u	0	0.104	0.218	0.319	0.410	0.493	0.570	0.642	0.709	0.771
yi	0	0.264	0.323	0.353	0.371	0.384	0.394	0.401	0.407	0.413
Tu/T	0	0.282	0.805	1.425	2.100	2.811	3.549	4.307	5.081	5.861
Tn/T	1.000	2.718	3.695	4.463	5.119	5.699	6.226	6.711	7.144	7.590

5. Souřadnici y_i se přiřadí na přechodové charakteristice inflexní bod a odečte se jeho časová souřadnice t_i. Ze vzorce $t_i = (n-1)T$ vypočteme časovou konstantu T. Další způsob určení konstanty je určením poměrů pro příslušný řád soustavy. Tedy určí se T_u/T, resp. T_n/T a z těchto hodnot je možné také určit časová konstanta T.

6. Statické zesílení K se zjistí jako podíl ustálených hodnot vybuzeného a budícího signálu.

   Tedy $K = \frac{[y(\infty)-y(0)]}{[u(\infty)-u(0)]} = \frac{\Delta y}{\Delta u}$

   Δy – je skutečný rozdíl fyzikální veličiny na výstupu dynamického systému $\Delta y = y(\infty) - y(0)$.

   Δu – je velikost realizovaného skoku skutečné fyzikální veličiny na vstupu systému, při kterém byla reakce měřena $\Delta u = u(\infty) - u(0)$.

7. Tento předešlý postup lze aplikovat i pro systémy s dopravním zpožděním T_d. Hodnota T_d se nejsnáze určí odečtením úseku Tu^*. K němu z první tabulky (tab. 4.2-1) v řádku τ_u určíme nejbližší nižší hodnotu τ_u, kterou vyneseme do grafu a k ní odečteme úsek T_d. Další postup je naprosto stejný jako v systémech bez dopravního zpoždění. Ovšem s tím, že výsledný přenos bude ve tvaru:

   $F_a(s) = \frac{\mathbf{K}}{(\mathbf{T}s+1)^n} e^{-T_d s}$ kde T_d je dopravní zpoždění.

8. Je-li $\tau_u < 0{,}104$ pak se zvolí pro aproximaci systém druhého řádu s různě velikými časovými konstantami T_1, T_2. Vycházíme se z toho, že pro hodnotu

pořadnice přechodové charakteristiky $y(t_1) = 0{,}720$ je čas t_1 určen pouze součtem časových konstant systému $T_1+T_2 = t_1/1{,}2564$. Z grafu přechodové charakteristiky odečteme časový úsek t_1 a součet (T_1+T_2) vypočteme. Konstanta 0,720 je jednou z konstant užitých v postupu, stejně jako hodnoty v první tabulce (tab. 4.2-1), které prof. Strejc stanovil po zkušenostech při aproximování soustav. Proto se jedná o metodu empirickou.

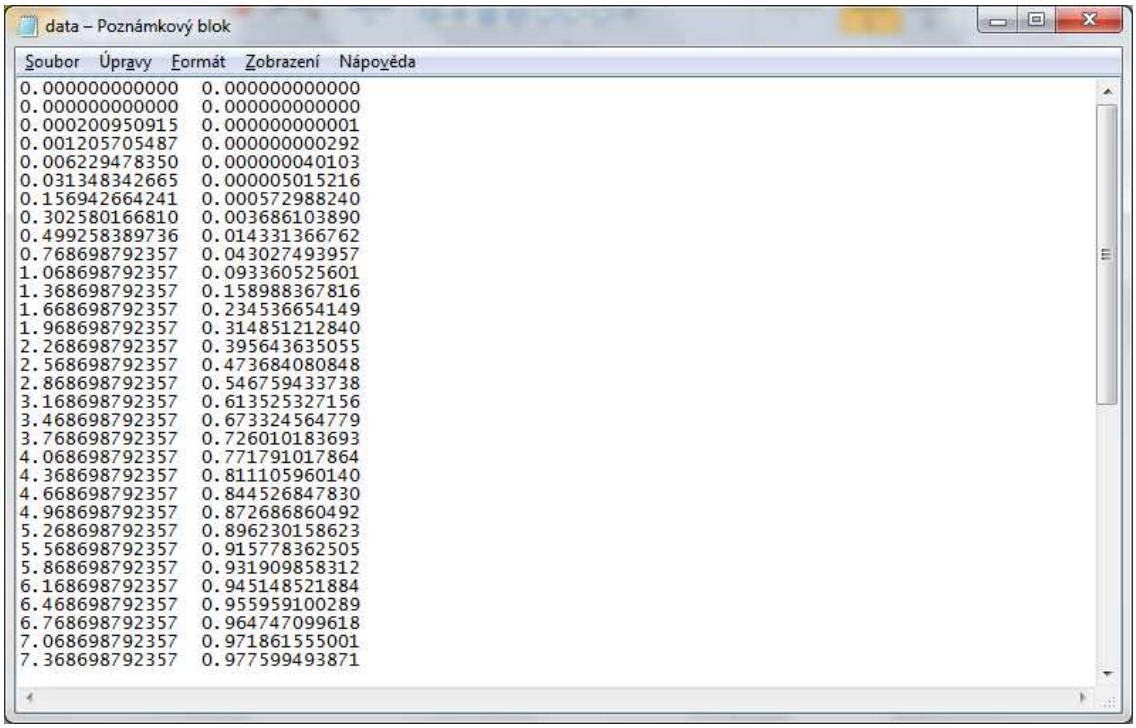
9. Pro čas $t_2 = 0{,}3574(T_1 + T_2)$ jsou pořadnice přechodové charakteristiky závislé na poměru časových konstant. Tedy $\tau_2 = T_2/T_1$. Poměr $\tau_2 = T_2/T_1$ lze nejsnáze určit způsobem, že v případě aproximačního přenosu uvažovaného typu nezávisejí $\tau_u = T_u/T_n$ a y_i na velikosti časových konstant T_1 a T_2, ale pouze na jejich poměru $\tau_2 = T_2/T1$. Tyto závislosti jsou uvedeny v druhé tabulce (tab. 4.2-1). Z grafu přechodové charakteristiky se odečte T_u, T_n a vypočte se $\tau_u = T_u/T_n$, a z tabulky 2 se určí $\tau_2 = T_2/T_1$. Protože je už známý součet i poměr obou hledaných časových konstant, lze jejich hodnoty vypočíst. Přenos takto identifikované soustavy bude pak: $\boldsymbol{F}_a(s) = \frac{\boldsymbol{K}}{(\boldsymbol{T_1}s+1)\cdot(\boldsymbol{T_2}s+1)}$ a velikost statického zesílení K se určí stejně jako v bodě 4. Dále lze tento postup aplikovat i na soustavy s dopravním zpožděním, analogicky se použije postup jako v bodě 7. A následný přenos bude vypadat takto: $\boldsymbol{F}_a(s) = \frac{\boldsymbol{K}}{(\boldsymbol{T_1}s+1)\cdot(\boldsymbol{T_2}s+1)} e^{-T_d s}$ [7],[9].

Tab. 4.2-2: Hodnoty pro dvě časové konstanty

$\tau_u=T_u/T_n$	0,016	0,03	0,05	0,062	0,072	0,084	0,092
$\tau_2=T_2/T_1$	0,02	0,05	0,1	0,15	0,2	0,3	0,4
y_i	0,058	0,104	0,148	0,177	0,197	0,224	0,24
$\tau_u=T_u/T_n$	0,097	0,1	0,102	0,103	0,103	0,104	
$\tau_2=T_2/T_1$	0,5	0,6	0,7	0,8	0,9	1	
y_i	0,25	0,256	0,26	0,263	0,264	0,264	

5 METODA PROFESORA STREJCE V MATLABU

Naprogramovat čistě „papírovou“ metodu nemusí být pro začátečníka vůbec jednoduché. Programátor by měl mít trochu větší znalosti z matematiky a geometrie. Prvním problém, který by se měl vyřešit, je import dat. Nejjednodušší je import z klasického textového souboru, kde budou data uchována v ASCII znacích a s každou naměřenou hodnotou na novém řádku ve dvou sloupcích (obr. 5-1), kde časové hodnoty jsou v levém sloupci. Tabulkové editory nejsou moc vhodné, protože ne každý má například Excel nainstalovaný. Navíc Matlab požaduje mít soubor uložený v Excelu otevřený.



Obr. 5-1 : Vzor dat

Pro takový způsob uložení je v Matlabu ideální funkce textread, která bez problému dokáže načíst data přesně tak jak chceme. Tedy:

```
[t, y1] = textread( sav, 'data.dat'),'%f %f');
```

Kde [t, y1] označují dvě proměnné, do kterých se budou načítat data. '%f %f' říká příkazu, jaké typy dat má načítat. Další hodnoty jsou popsané v Product Help ve vývojovém prostředí Matlab.

Dalším krokem se musí znormovat data tak, aby se ustálená hodnota rovnala jedné:

```
if y1(end) > 1  % porovnani, zda muzeme aproximovat, vetsi nez 1
   y = y1 / y1(end); % znormovani dat
else
   if y1(end) < 1  % porovnani, zda muzeme aproximovat, mensi nez 1
      y = y1 * (1/y1(end)); % znormovani dat
   else
      y = y1; % nemusime normovat
   end
end
```

Zde lze vidět jednoduchou podmínku if – else, kde nejprve program zjistí, zda hodnoty vůbec překračují 1 a jestli ano, tak všechny hodnoty vydělí posledním číslem z dat. Pokud jsou hodnoty menší než 1, tak data vynásobí obrácenou hodnotou posledního čísla z dat. Pokud je ustálená hodnota rovna 1, tak program nic nenormuje.

Inflexní bod odpovídá maximu derivace přechodové funkce *y(t)*. Matematicky je to přechod mezi konvexností a konkávností funkce. Je to místo, kde je druhá derivace funkce *f(x)* nulová. Proto se odhad inflexního bodu nejjednodušeji provede numericky díky předpokladu, že druhá diference hodnot *y* představuje druhou derivaci. Následně se najde hodnota, která je rovna 0. To jde z naměřených dat bohužel těžko, tudíž je výhodnější použít hodnotu, kdy se v datech mění znaménko. Pro demonstrační účely jsou podmínkou nezašuměná data, proto je použita tato velice jednoduchá metoda.

```
plusd2y = find(diff(y,2)>0); % nalezeni kladnych cisel do pomocne promenne
inflexnibod = plusd2y(end); % nacteni pozice inflexniho bodu
```

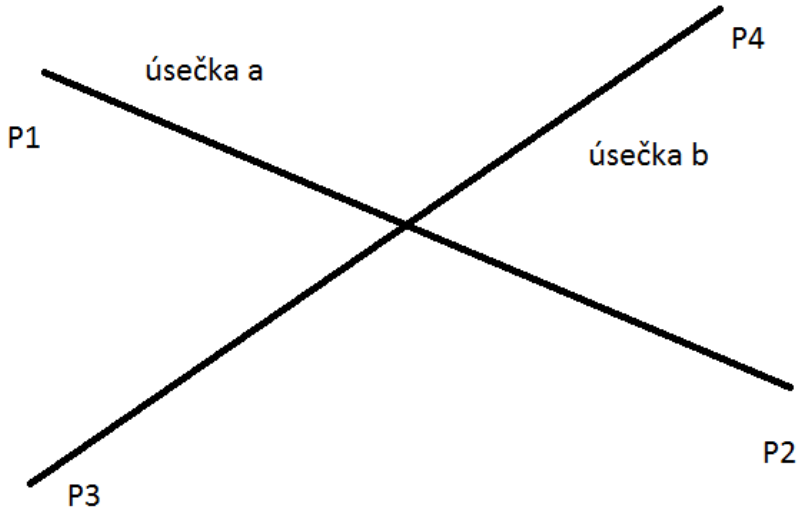
Zde se s výhodou používá příkaz diff(), který vypočítá druhou diferenci hodnot, a poté se ihned do pomocné proměnné příkazem find() načtou pozice všech kladných čísel. V následujícím řádku se pouze načte poslední hodnota což je výsledná pozice inflexního bodu. Tuto pozici se je možno použít i k vyhledání souřadnice na časové ose. Tedy například ti = t(inflexnibod); % urceni ti.

Při vyhledání tečny je dobré vycházet ze směrnicové rovnice pro přímku, tedy $t: y = kx + q$, kde k je směrnice přímky (tečny) a q je tzv. úsek (vytnutý přímkou) na ose y, což je druhá souřadnice průsečíku přímky s osou y. Tečna se ovšem numericky hledá pro každý bod této přímky, tedy:

```
d1t = diff(t); % prvni derivace t
d1y = diff(y);
k = d1y(inflexnibod)/d1t(inflexnibod); % podíl derivace y a t (smernice)
tecna = k*(t-ti)+y(inflexnibod); % vypocet tecny
```

Program vypočítá numerické derivace a spočítá jejich podíl, což je směrnice tečny. Poslední řádek v kódu je již samotná funkce pro výpočet tečny.

Pro hledání T_u a T_n lze použít i čistě numerické řešení s využitím příkazu find(), ale při malém vzorku hodnot je to dost nepřesné. Proto je lepší použít analytickou geometrii, kdy v Matlabu najdeme průsečík dvou přímek. Ve 2D prostoru lze s úspěchem použít algoritmus, který pracuje s počáteční a konečnou souřadnicí přímek (obr. 5-2) [8]:



Obr. 5-2 : Přímky

Rovnice přímek tedy jsou $P_a = P_1 + u_a (P_2 - P_1)$ a $P_b = P_3 + u_b (P_4 - P_3)$. Řešení bodů, kde $P_a = P_b$, dá ve 2D prostoru v kartézských souřadnicích dvě rovnice o dvou neznámých směrnic u_a a u_b.

Tedy:

$x1 + u_a (x2 - x1) = x3 + u_b (x4 - x3)$

a

$y1 + u_a (y2 - y1) = y3 + u_b (y4 - y3)$.

Z těchto rovnic se vyjádří požadované neznámé, které vypadají takto:

$$u_a = \frac{(x4 - x3)(y1 - y3) - (y4 - y3)(x1 - x3)}{(y4 - y3)(x2 - x1) - (x4 - x3)(y2 - y1)}$$

$$u_b = \frac{(x2 - x1)(y1 - y3) - (y2 - y1)(x1 - x3)}{(y4 - y3)(x2 - x1) - (x4 - x3)(y2 - y1)}$$

Dosazením jedné z těchto dvou neznámých do příslušné rovnice se získají souřadnice hledaného průsečíku přímek.

$x_i = x1 + u_a (x2 - x1)$

$y_i = y1 + u_a (y2 - y1)$

Ve vývojovém prostředí Matlab je naprogramování podle tohoto postupu ovšem o dost jednodušší [8]:

```
tu_x=[0 0;t(end) t(end)]; % pocatecni body jsou v prvnim radku, koncove body ve
%druhem
tu_y=[0 tecna(1);0 tecna(end)];
dx = diff(X);  %Vypocita se diference z kazde matice
dy = diff(Y);
jmen = dx(1)*dy(2)-dy(1)*dx(2);  %Vypocet jmenovatele
ua = (dx(2)*(y(1)-y(3))-dy(2)*(x(1)-x(3)))/jmen;
xi = x(1)+ua*dx(1); %ziskame souradnice pruseciku
yi = y(1)+ua*dy(1);
Tu = tu_x(1)+tu_ua*tu_dx(1); %vypocet pruseciku pro dobu prutahu, souradnice osy y
%je 0
```

Tento stejný výpočet je aplikován i na výpočet doby náběhu T_n. S tím že doba náběhu je vektor Tn* zkrácený o T_u.

Poměr těchto dvou hodnot dává již hledané $\tau_u = T_u / T_n$.

V dalším kroku je nutné vytvořit tabulku a algoritmus pro výběr správných hodnot. Matlab na to má opět jednoduché řešení v podobě maticově zapsané tabulky a jednoho cyklu pro selekci, kde se vybere celý sloupec hodnot a ten se zapíše do proměnné. Takto může vypadat výběr z pole:

```
i = 1;
while pole_T(1,i) <= tau_u %vyber z pole
   i = i + 1;
end
i=i-1;
jednakonst=pole_T(:,i); %nacteni do konstanty
```

Když je již načten požadovaný sloupec, vyberou se jen potřebné hodnoty, provede se několik výpočtů a výsledkem bude požadovaná aproximace.

```
T = (Tn/jednakonst(6));
K = y1(end); % staticke zesileni
Td1 = tecna(1) * (-1); %hledame dopravni zpozdeni, nacteme do konstanty Td1 a
nasledne priradime nejblizsi nizsi hodnotu
i = 1; %zjistujeme dopravni zpozdeni
while pole_T(3,i) < Td1
   i = i + 1;
end
if i==0
   i=1;
else
   i = i-1;
end
Td = (Td1 - pole_T(3,i)) * (Tu/Td1); %vypocet dopravniho zpozdeni
if Td==0  % soustava muze byt i bez dopravniho zpozdeni
   s=tf('s');
   H = K/((T*s+1)^n);
   [ynov, tnov] = step(H); %nacteni dat charakteristiky do pole
   T1=0; %osetreni pro vystup
   T2=0;
```

```
    Td=0;
else
    s=tf('s');
    H = (K/((T*s+1)^n))*exp(-Td*s);
    [ynov, tnov] = step(H);
    T1=0;
    T2=0;
end
```

Takovýto kód se upraví i do podoby pro $\tau_u < 0{,}104$. Princip je naprosto stejný, ale program se samozřejmě liší jak v tabulce, tak v rovnici pro aproximaci.

Tato práce zobrazuje grafické výsledky kódu, takže vykreslení grafů je řešeno uložením grafu do obrázku na disku. Existuje i jiné řešení, například přes HTML kód, který generuje třída WebFigures z balíčku javabuilder.jar. Řešení s uložením grafu jako obrázek na disku bylo vybráno proto, že je nejjednodušší na naprogramování a velice snadno jsou nalezeny chyby v programu. Uložení grafu do souboru je řešeno přes příkaz saveas(h,'jmeno_souboru','format') kde h představuje figuru. Příkaz figure vytvoří grafický objekt. Další položkou je jméno objektu, který se ukládá. Před tímto jménem může být také cesta, kam přesně má být objekt uložen. Matlab podporuje ukládání do více datových formátů, proto musí být určen i formát ukládaného objektu. Jen ve zkratce Matlab umí ukládat do formátů jako *.jpg, *.bmp či *.pdf. Celý popis formátů je k nalezení v Product Help Matlabu. Takto může vypadat vykreslení originálních dat a následné uložení:

```
h1=figure; %vytvori novy graficky objekt
plot(t,y1,'LineWidth',1); % vykresleni originalnich dat
legend('Originální data'); %popis grafu
ylim([-0.5 (y1(end)+0.5)]); % limit osy y
xlim([0 t(end)]); % limit osy x
grid on; %zapnuti mrizkovani
xlabel('Čas (t)'); %popisky os
ylabel('h(t)');
title('Graf - Originální data'); % nazev grafu
saveas(h1, fullfile(sav, 'my1.jpg'), 'jpeg'); % samotne ulozeni do obrazku
```

Příkazy pro vykreslení grafu a jeho uložení byly použity úspěšně. Z příkazů je vidět, že graf se podle dat sám upraví podle posledních hodnot naměřených dat. Pevně danou velikost os není možné použít.

Zajímavé je ovšem vykreslení přechodové charakteristiky. V Matlabu je na výpočet a vykreslení přechodové charakteristiky příkaz step(SYS), kde SYS představuje přenos systému. Z neznámých důvodů ale vykreslení pomocí step(SYS) v servletu nefunguje. Matlab i JAVA samotná pracují správně. Server se chová tak, že první spuštění servletu proběhne správně, servlet data uloží a vygeneruje HTML kód. Při druhém spuštění servletu, ale z neznámých příčin vykáže server chybu a servlet není dále přístupný a je potřeba server restartovat. Proto je lepší uložit příkazem step(SYS) hodnoty přechodové charakteristiky do proměnných a samotný graf vykreslit příkazem plot().

Na ukázku, kdy graf porovná aproximovanou charakteristiku, originální data a znormovaná data, se může stát, že řešený systém má statické zesílení K rovno 1. V tomto případě se použije podmínka, kdy program ověří, zda se ustálená hodnota rovná jedné nebo ne. V případě že ano, zapíše se do pomocné proměnné, například s názvem p, hodnota, podle které se bude vykreslovat na konci správný graf. Kód pak může vypadat takto:

```
if y1(end)>0.99 && y1(end)<1.01 %osetreni pro vykreslovani spravnych grafu
   p=1;
else
   p=0;
end
```

A přímo vykreslení:

```
if p==1;
   h7=figure;
   plot(t,y,tnov, ynov,'LineWidth',1); %vykresleni pouze aproximace a znormovanych
dat
   hold on;
   legend('y', 'Aproximace');
   axis([0 t(end) -0.5 K]);
```

```
   grid on; % limit os
   xlabel('Čas ');
   ylabel('h(t)');
   title('Graf - Porovnání všech charakteristik ');
   saveas(h7, fullfile( sav, 'my7.jpg'), 'jpeg');

else
   h7=figure;
   plot(t,y,t,y1,tnov, ynov,'LineWidth',1); %vykresleni aproximace, znormovanych a
%originálních dat
   hold on;
   xlabel('Čas ');
   ylabel('h(t)');
   legend('y','Originální data', 'Aproximace');
   axis([0 t(end) -0.5 K]);
   grid on; % limit os
   title('Graf - Porovnání všech charakteristik ');
   saveas(h7, fullfile( sav, 'my7.jpg'), 'jpeg');
  end
```

Aby ale program mohl být převedený do JAVA komponenty, musí program být napsán jako funkce.

```
function [tau_u, Tu, Tn, K, T, n, Td, T1, T2] = strejda(sav)
.
.
%kod programu
.
.
end
```

Na výstupu jsou zapsány všechny konstanty, které mohou být servletem ukázány. Všechny konstanty se ale vůbec nepoužijí, takže se načtou jako hodnoty nulové. Servlet si už požadované hodnoty vybere sám. Celý kód MATLAB programu je v příloze na CD.

Po napsání a ověření funkčnosti programu může být kód přes funkci deploytool převeden na JAVA komponentu a poté se přejde na programování servletu.

6 PROGRAMOVÁNÍ VÝUKOVÉHO SERVLETU

Servlet, který bude spouštět komponentu vytvořenou v Matlabu, musí umět soubor nahrát, uložit a zobrazit HTML kód, který metodu prof. Strejce demonstruje. K tomu se použijí metody pro použití MWArray, JSP stránek a JAVA kód.

6.1 JSP Stránka

Jako první věc by se měla uživateli zobrazit vstupní JSP stránka, která nastíní lehký úvod, o jakou stránku se jedná a nabídne nahrání dat na server, se kterými bude metoda pracovat. Pro demonstraci také nabídne stáhnutí vzorových dat, které posléze může uživatel nahrát na server a prohlédnout si tvorbu aproximace. Důležitou částí JSP stránky jsou HTML značky a skriptovací značky JSP, které vytvoří dynamickou stránku [2]. V použitém kódu není skriptovacích značek vůbec potřeba, ale důležitou součástí je správné nastavení zobrazovaných znaků (jazyka):

```
<%@ page contentType="text/html; charset=windows-1250" %>
```

Tento kód předá prohlížeči informaci, do jaké znakové sady má stránku přeložit.

K odeslání dat se použije formulář HTML a v něm následující kód:

```
  <form action="strejc" method="post" enctype="multipart/form-data">
<input  name="file" type="file"/>
<input type="submit" value="Nahrat"/>
  </form>
```

a. Atribut action je povinný atribut, který servletu specifikuje adresu, na kterou budou data odesílána. V tomto případě je to jméno servletu.
b. Atribut method specifikuje, jak budou data přenášena do serveru. V tomto případě se používá metoda POST, která data odešle na samostatné řádce oproti metodě GET, která je implicitní a také se použije, když si prohlížeč vyžádá běžnou URL [2].

c. Atribut enctype specifikuje způsob, kterým se data před přenosem zakódují. V této práci je vhodné použít atribut ve tvaru multipart/form-data. Toto zakódování přenese všechna pole jako samostatné části dokumentu.

Pro nahrání souboru se zvolil klasický způsob s použitím řídícího prvku pro přenos souborů.

```
<input  name="file" type="file"/>.
```

Formulář se odešle tlačítkem pro předání formuláře, kde se po klinutí na tlačítko odešle formulář označeném parametrem ACTION značky FORM.:

```
<input type="submit" value="Nahrat"/>
```

6.2 Sezení a nahrání souboru

V této práci byly zmíněny tzv. anotace, které jsou s výhodou použity pro tento servlet. Tudíž nemusí používán specifikační soubor web.xml.:

```
@WebServlet(name="strejc", urlPatterns={"/strejc"})     // specifikace url pro servlet
@MultipartConfig          // urci servletu pouziti multipart/form-data
```

Tento servlet využívá SESSION (sezení) k pojmenování adresáře do kterého budou nahrávána vstupní data a ukládány obrázky, které vygeneruje JAVA komponenta získaná z Matlabu.:

```
public class strejc extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();

            HttpSession session = request.getSession(true); //vytvoreni session
            String abv = session.getId(); //ziskani jmena session
```

```
                String mee = this.getServletContext().getRealPath(abv); //ziska cestu
//serveru
   File d1 = new File(mee);    //vytvoreni slozky
   if (d1.exists() == false)
         {   //podminka zda existuje
    d1.mkdir();
   }
      try {
        // ziskani pristupu k souboru, ktery je poslan z klienta
        Part p1 = request.getPart("file");
        InputStream is = p1.getInputStream();
                        String filename = "data.dat";  // pojmenovani souboru
        String outputfile = d1 + "\\" +filename ;  // ziskani cesty souboru
        FileOutputStream os = new FileOutputStream (outputfile); //definovani
//datoveho proudu
        // zapise byty z nahraneho souboru do ciloveho souboru
        int ch = is.read();
        while (ch != -1) {
            os.write(ch);
            ch = is.read();
        }
        os.close();
                        stryc(response, request, abv);        //spusteni metody stryc a
//predani parametru
      }
      catch(Exception ex) {
        out.println("Exception -->" + ex.getMessage());
      }
      finally {
        out.close();
      }
                        if(session!=null)        {  //zavreni session
                                session.invalidate();


                } // konec doPost()

public void stryc(HttpServletResponse response, HttpServletRequest request, String
abv)
         throws ServletException, IOException
         {
…
//kod metody stryc
…
} //konec metody stryc
                                        } // konec classstrejc
```

Jak je z kódu vidět, nahrání souboru na server, vytvoření SESSION a volání metody stryc je celé v metodě doPost.

Nahrání souboru na server využívá třídy pro zpracování datových proudů. S výhodou byl v tomto servletu použit binární datový tok, jehož zastáncem je v tomto případě třída FileOutputStream. Pro samotné zapsání souboru je použit jednoduchý algoritmus, který zapíše byty do souboru.

Pokud chceme v jednom prohlížeči postupně vyzkoušet několik naměřených dat, musí být na konci servletu metoda pro ukončení SESSION, protože servlet pro jeden prohlížeč zachovává již jednou vytvořené sezení. Tato vlastnost je s výhodou použita například při programování elektronického obchodu, ale v tomto případě je nežádoucí.

6.3 Volání komponenty

Pro v Matlabu vytvořenou komponentu se vytvoří nová metoda stryc, která přijme z metody doPost název sezení a vrátí HTML kód.

```
public void stryc(HttpServletResponse response, HttpServletRequest request, String abv)
        throws ServletException, IOException
        {
                String mee = this.getServletContext().getRealPath(abv); //ziskani cesty z //nazvu session

                PrintWriter print = response.getWriter();
                strejdaclass x = null; //definovani tridy strejdaclass
                Object[] result = null; //vytvoreni objektu
                MWCharArray in = null; //vytovreni vstupniho pole MWArray
                        try {

                        x = new strejdaclass();
                        in = new MWCharArray(mee); //zapsani cesty do konstanty in
                        result = x.strejda(10, in); //samotne volani strejda komponenty, //preda se cesta a ziska se 10 vystupnich hodnot

                MWNumericArray tau_u4 = new MWNumericArray(result[9]); //nacteni, //pretypovani a zaokrouhleni ziskanych hodnot
                        double tau_u3 = tau_u4.getDouble(1)*10000;
```

```
                    int tau_upr1 = (int) tau_u3;
                    double tau_upr = (double) tau_upr1/10000;
                    MWNumericArray tau_u1 = new MWNumericArray(result[0]);
                    double tau_u2 = tau_u1.getDouble(1)*10000;
                    int tau = (int) tau_u2;
                    double tau_u = (double) tau/10000;
.
// Dalsi pretypovani
.
print.println("<html>"); //vytisknuti HTML kodu
                    print.println("<body>");
                    print.println("<h3><p>Metoda prof. Strejce - řešení</p></h3>");
                    print.println("<br>");
                    print.println("<p>Tento postup zjednodušeně ukazuje jak
aproximovat přechodové charakteristiky podle prof. Strejce</p>");
                    print.println("<br>");

if (tau_u >= 0.10401) { //podmínka pro vykresleni spravnych grafu

        MWNumericArray T1 = new MWNumericArray(result[4]); //pretypovani
                            double T2 = T1.getDouble(1)*10000;
                            int t11 = (int) T2;
                            double T = (double) t11/10000;
                            MWNumericArray n1 = new MWNumericArray(result[5]);
                            double n2 = n1.getDouble(1);
                            int n = (int) n2;
        print.println("<p>") //HTML kod
        print.println("Takto vypadají originální data");
        print.println("<br>");
        print.println("<br>");
        print.println("<img src=\"" +abv+"\\my1.jpg\" width=\"800\" height=\"500\"
>");

…
//Další HTML kod
…


}
else
{
//Stejne jako v predeslem pripade, ale pro druhy zpusob aproximace
}

        print.println("</p>");
                    print.println("</body>");
                    print.println("<html>");
```

```
} catch (MWException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
}finally {
        x.dispose(); //uvolneni tridy
        MWArray.disposeArray(result);
        MWArray.disposeArray(in);
}
```

Metoda stryc použije získaný název sezení, získá cestu na server, ta je převedena na řetězec a následně předána komponentě s pracovním názvem strejda. Ještě před voláním Matlab komponenty byly vytvořeny objekty a třídy pro převod datových typů, což je potřeba pro práci se vstupem a výstupem z Matlab komponenty. V dalších krocích se získávají a rovnou přetypovávají vypočítané hodnoty. Důležitá podmínka je pro vygenerování správných grafů. Rozhodující hodnota je v tomto případě 0.10401.

Podobná situace s vygenerováním grafů nastane i pro soustavy bez dopravního zpoždění proto je v dalších řádcích kódu zavedena i podmínka, která testuje, zda se dopravní zpoždění rovná nule a podle toho vykreslí správné grafy a vypíše rovnice.

```
if (Td != 0) {
        print.println("Dále se určí dopravní zpoždění a to tak, že se zakreslí a změří se
úsek &#932;<sub>u</sub>* ");
        print.println("<br>");
        print.println("A podle něj se přiřadí z tabulky nejnižší nižší hodnota
&#932;<sub>u</sub>* ");
        print.println("<br>");
        print.println("<br>");
        print.println("<img src=\"" +abv+"\\my8.jpg\" width=\"800\" height=\"500\"
>");
        print.println("<br>");
        print.println("Výsledná hodnota dopravního zpoždění tedy je " +Td);
        print.println("<br>");
        print.println("Výsledný přenos po dosazení: <br> F<sub>&#40;s&#41;</sub> =
&#40;"+K+"// &#40;"+T+"*s + 1 &#41;&#94;"+n+"&#41;*e&#94;-"+Td+"*s ");
        print.println("<br>");
        print.println("<br>");

                }
```

```
                    else {
        print.println("Výsledek dopravního zpoždění je roven nule");
        print.println("<br>");
        print.println("Výsledný přenos je <br> F<sub>&#40;s&#41;</sub> = "+K+"//
&#40;"+T+"*s + 1 &#41;&#94;"+n+"");
        print.println("<br>");
                                        }
```

Při psaní HTML kódu v jazyce JAVA se musí dát pozor na správné psaní horních uvozovek a speciálních znaků. HTML má ovšem zvláštní kódy pro psaní speciálních znaků. Například znak „(“ se napíše kódem ([12].

6.4 Kaskádové styly a závěrečné úpravy

Aby byl servlet graficky lehce upravitelný, byl vytvořen konfigurační soubor s kaskádovými styly. V HTML kódu stačí jen uvést v hlavičce cestu ke kaskádovému stylu <link rel="stylesheet" type="text/css" href="\strejc\styly.css">. Cesta serveru je pevná, takže neobsahuje žádnou proměnou. Totéž se zavedlo i v samotném JAVA kódu [13].

```
body {background-color: #D8F8FD}
p { text-align: center }
table, th, td { border: 0px; margin: auto}
h3 {font-size: 14pt; font-weight: bold}
```

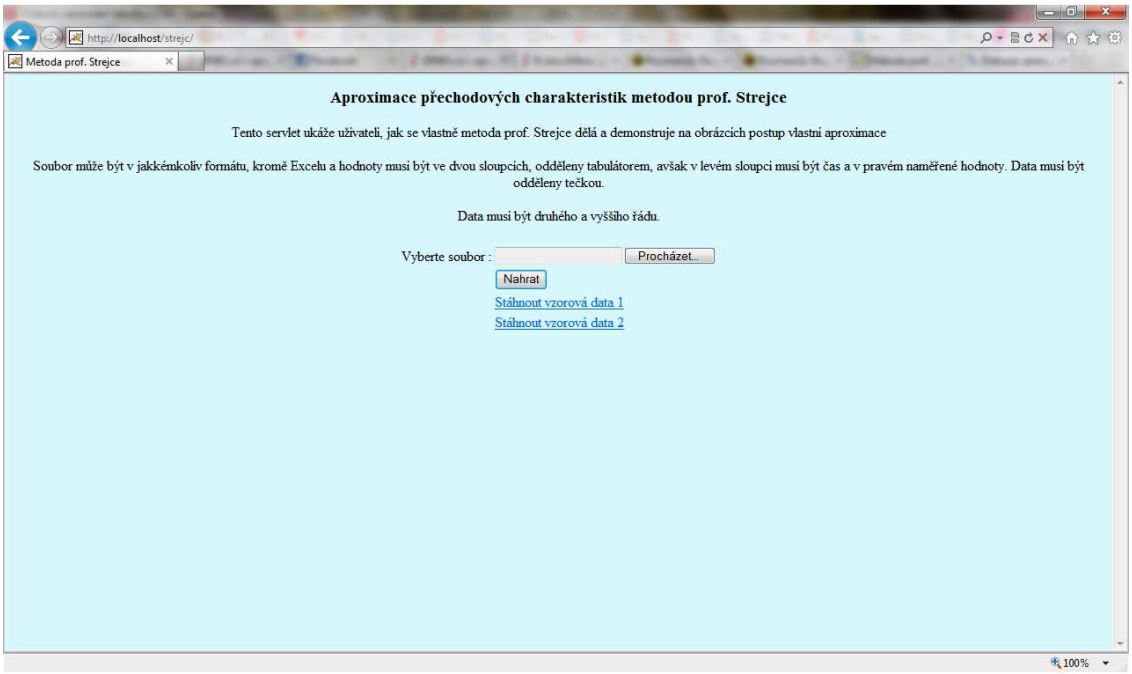
První řádek obsahuje nastavení samotného těla HTML kódu jako je například barva. Druhý řádek popisuje zarovnání textu. Ve třetím je nastavení tabulek tak, aby tabulky neměly žádný viditelný okraj a příkaz margin: auto upravuje zarovnání tabulky. V posledním řádku je nastavení nadpisů tak, aby měly potřebnou velikost a tučnost.

Pro výuku aproximační metody je také vhodné dát k dispozici ke stažení vzorová data. Díky znalosti HTML byl doplněn kód na stažení souboru.:

```
<a href="\strejc\1\data2.dat">Stáhnout vzorová data 1</a>
```

8 OVĚŘENÍ FUNKČNOSTI

Je samozřejmě nutné, aby byl program funkční. Po přeložení a nakopírování potřebných souborů do složky serveru, je potřeba ověřit funkčnost. Spustí se Apache Tomcat a do adresového řádku prohlížeče se zadá adresa localhost/strejc. Výsledek vypadá takto (obr. 8-1):

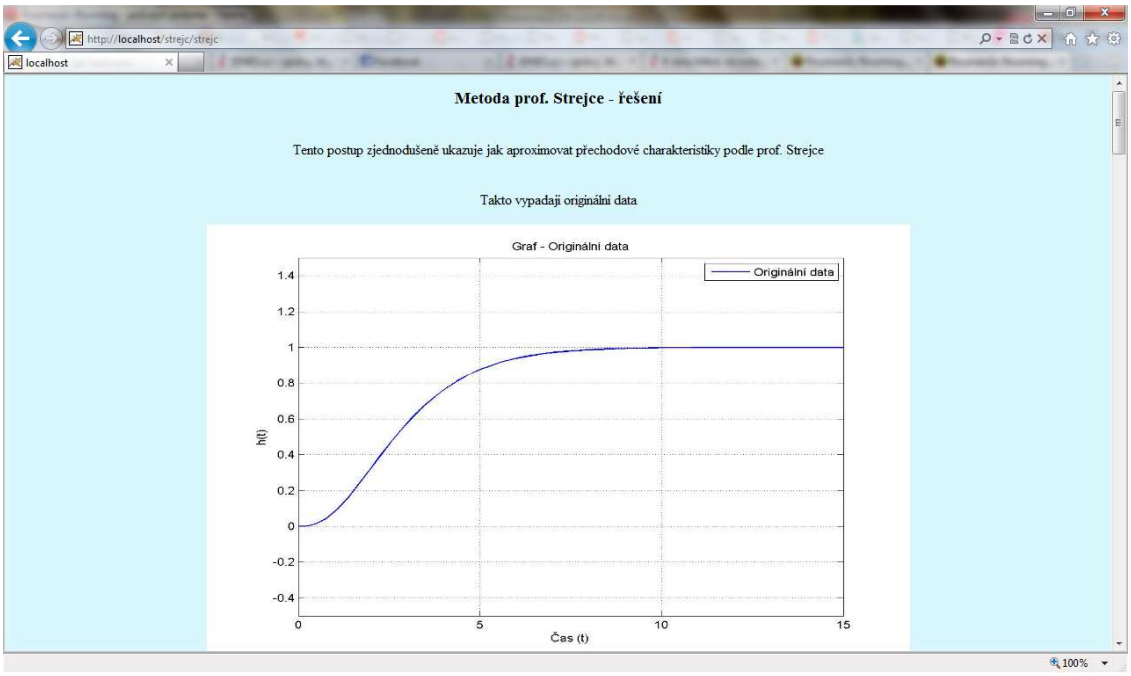


Obr. 8-1 : Hlavní stránka

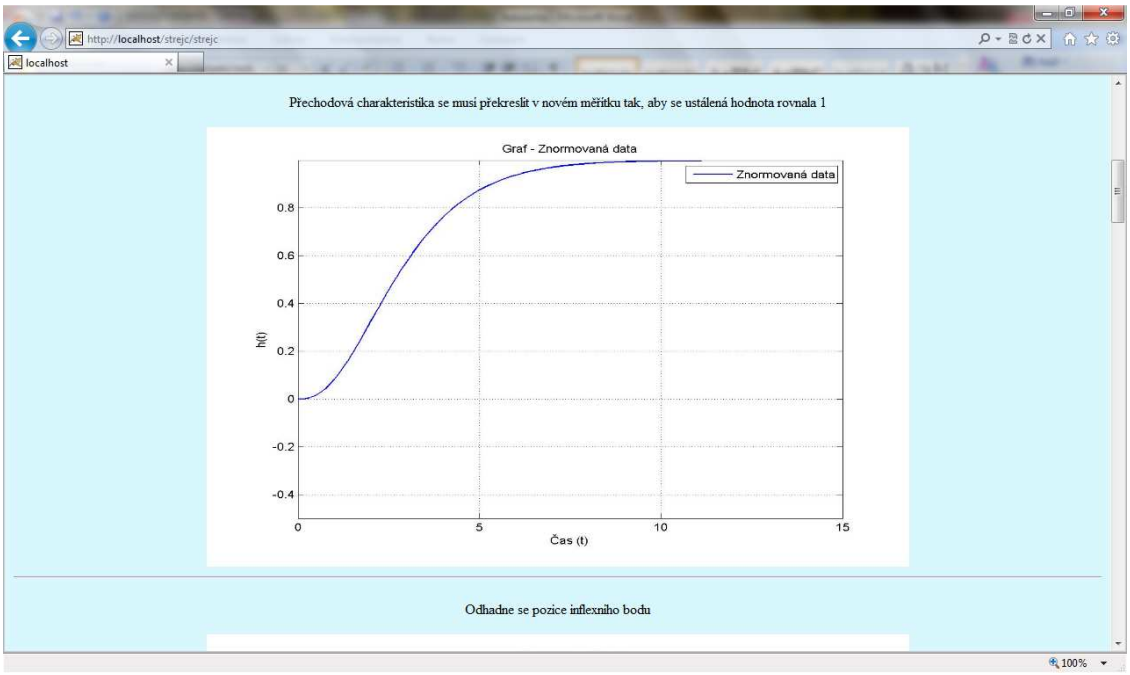
Pro demonstraci aproximace si může uživatel stáhnout vygenerovaná data. První data jsou s jednou časovou konstantou a druhý soubor obsahuje data pro metodu se dvěma časovými konstantami. Data se načtou a vyberou klikem na tlačítko Procházet. Po kliknutí na tlačítko Nahrát se spustí vlastní aproximace.

8.1 Vygenerování aproximace

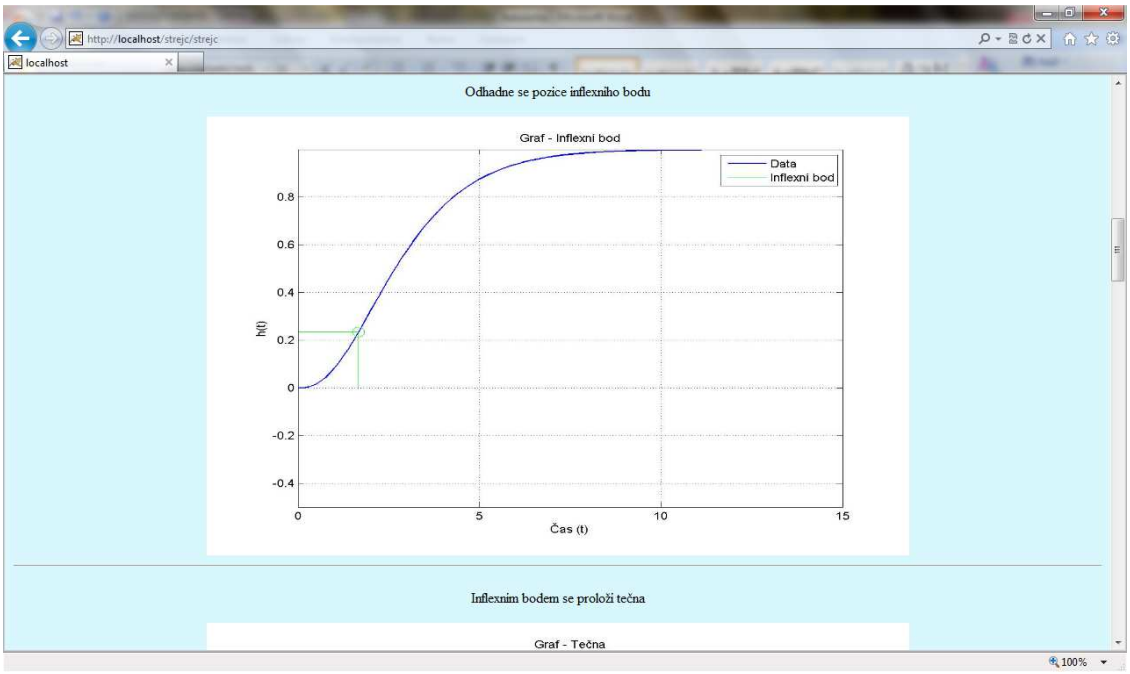
Na prvním obrázku (obr. 8.1-1) je vidět graf originálních dat a stručný úvod do výsledku. Obrázek (obr. 8.1-6) ukazuje výběr a výpočet z dat. Speciální tabulky musí být pro každou hodnotu zvlášť, jinak by program ukazoval stále tu samou tabulku.



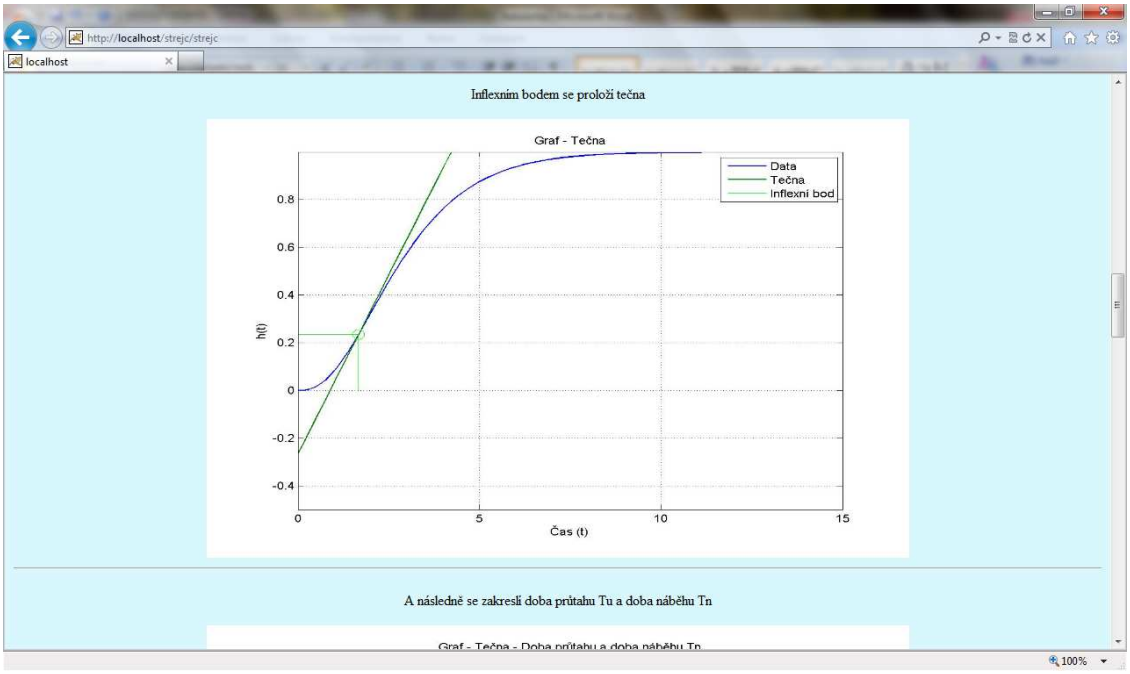
Obr. 8.1-1 : Skutečná data



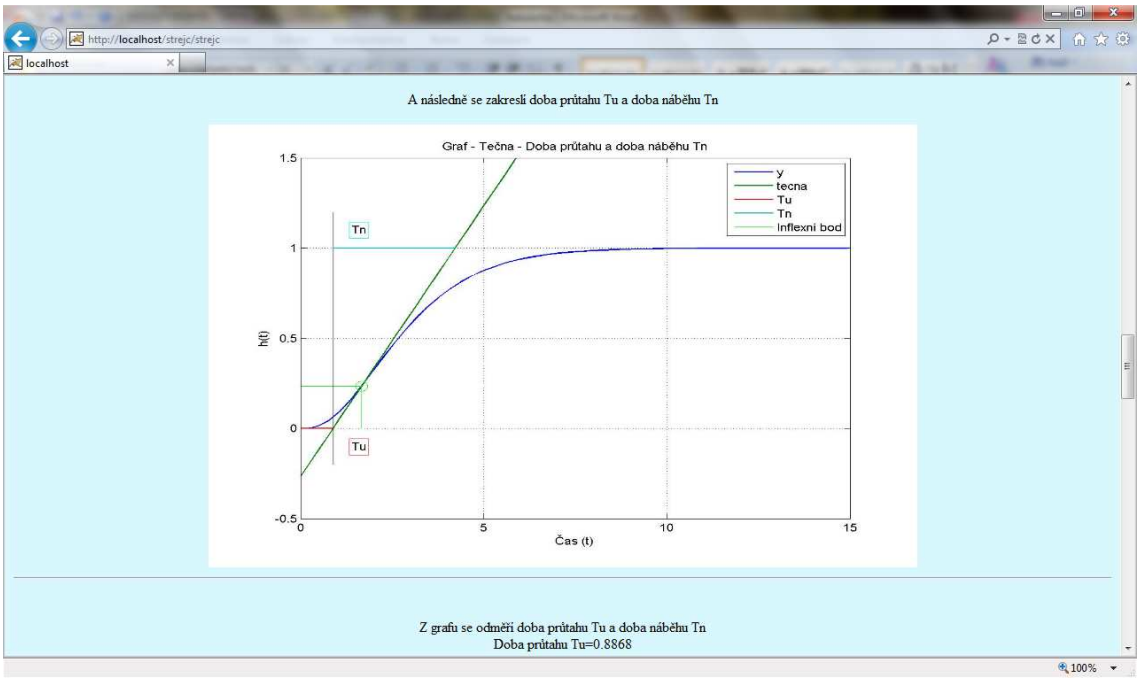
Obr. 8.1-2 : Znormovaná data



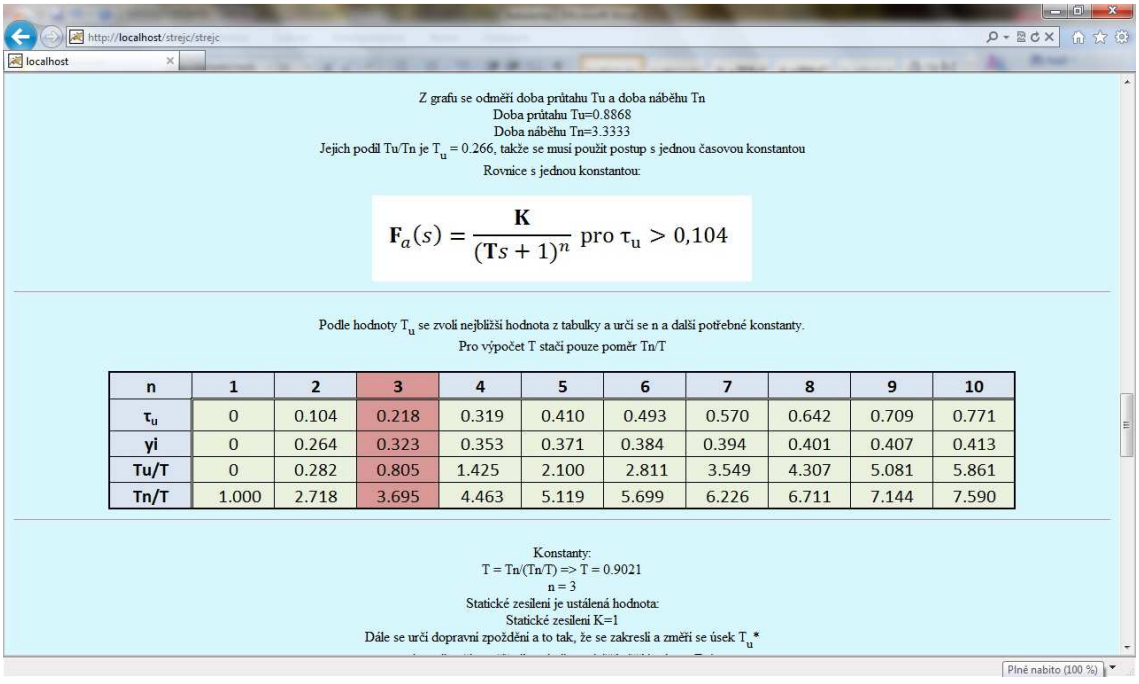
Obr. 8.1-3 : Odhad inflexního bodu



Obr. 8.1-4 : Proložení tečny

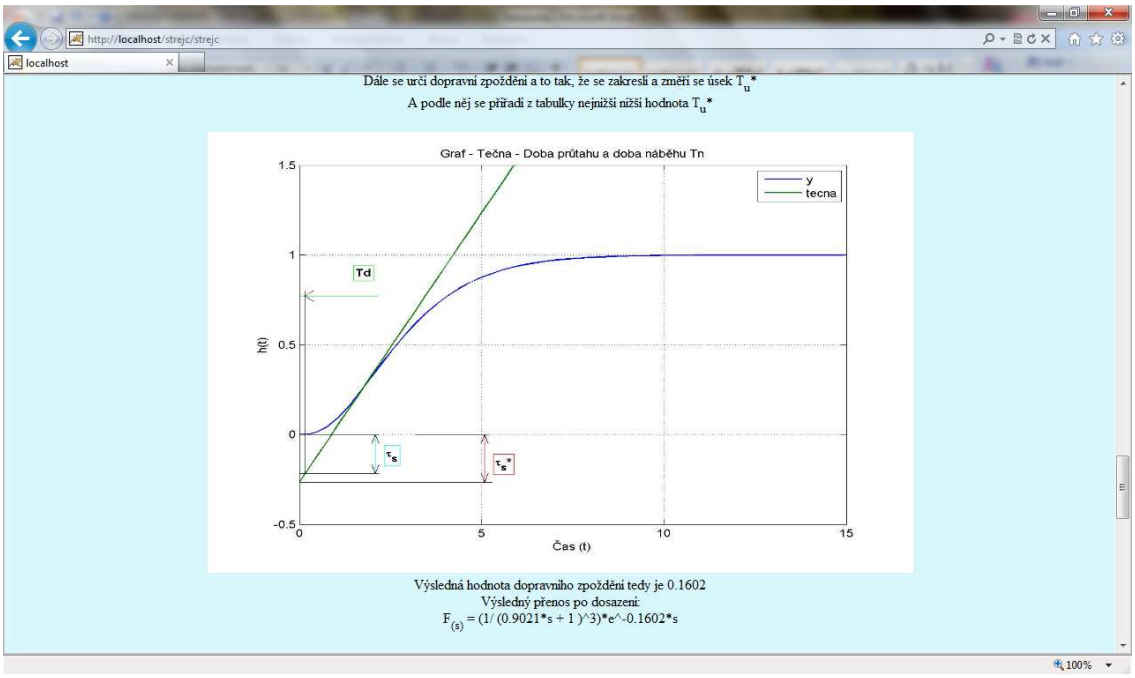


Obr. 8.1-5 : Zakreslení dob průtahu a náběhu

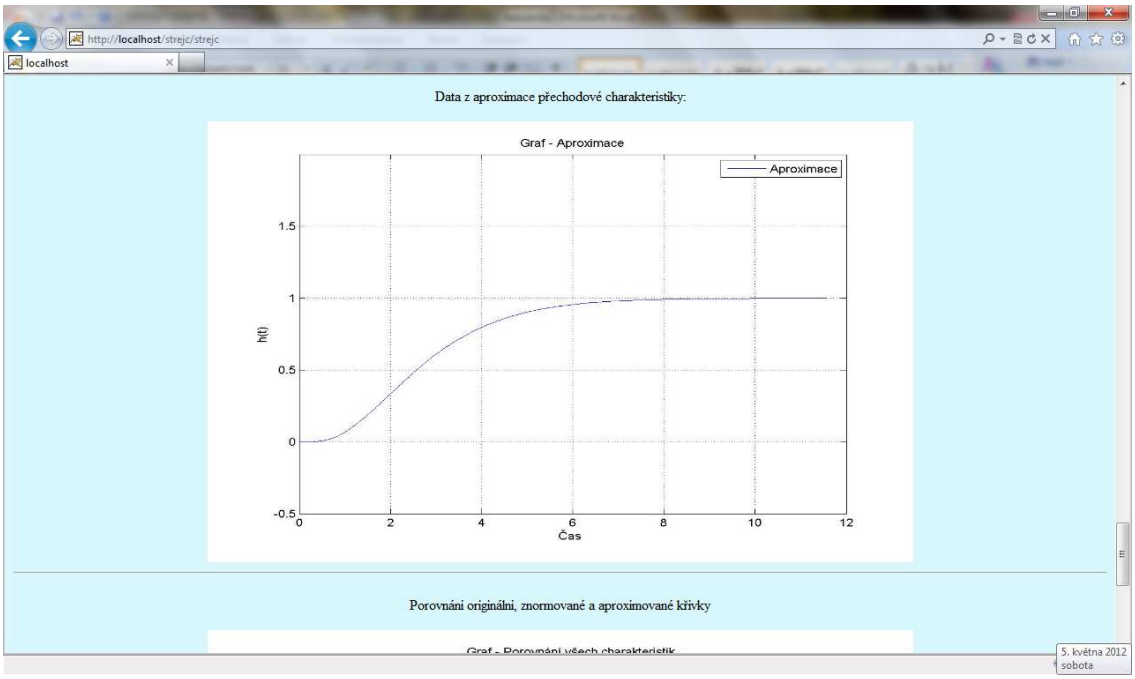


n	1	2	3	4	5	6	7	8	9	10
τ_u	0	0.104	0.218	0.319	0.410	0.493	0.570	0.642	0.709	0.771
yi	0	0.264	0.323	0.353	0.371	0.384	0.394	0.401	0.407	0.413
Tu/T	0	0.282	0.805	1.425	2.100	2.811	3.549	4.307	5.081	5.861
Tn/T	1.000	2.718	3.695	4.463	5.119	5.699	6.226	6.711	7.144	7.590

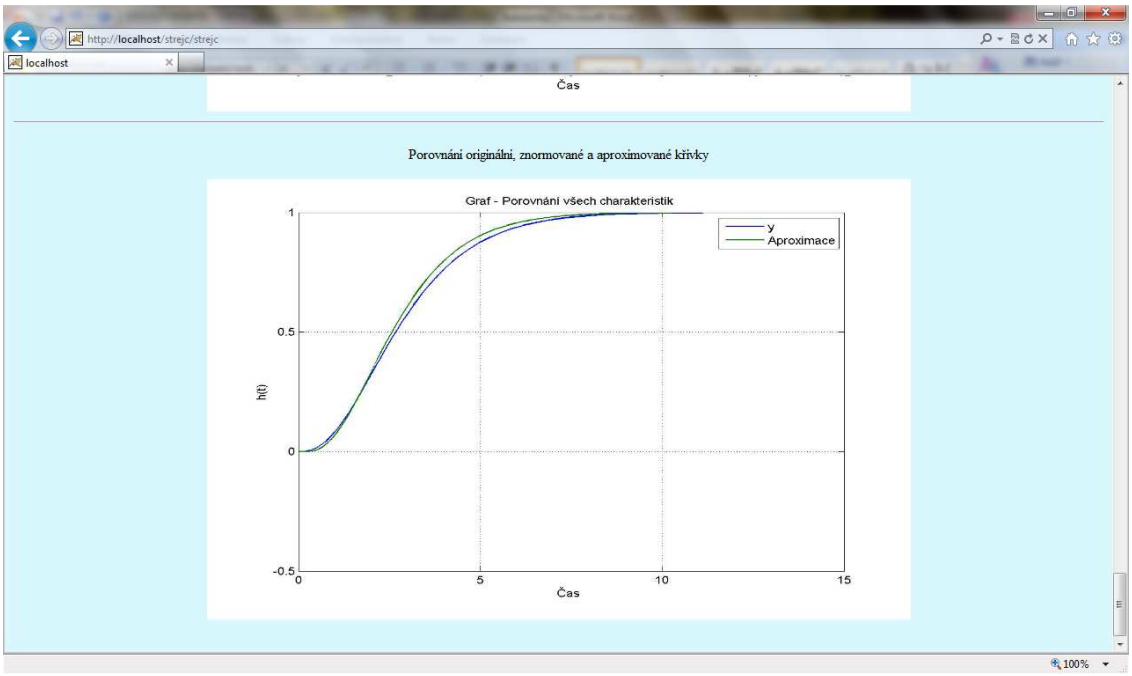
Obr. 8.1-6 : Výpočet a výběr z tabulky



Obr. 8.1-7 : Určení dopravního zpoždění



Obr. 8.1-8 : Aproximovaná data



Obr. 8.1-9 : Porovnání dat

ZÁVĚR

Tato práce popsala jednoduchou cestu jak vytvořit výukový servlet na aproximaci přechodových charakteristik pomocí metody prof. Strejce za použití nástrojů jako je JAVA, Apache Tomcat a Matlab.

Zadaný úkol byl pro mne velkou výzvou, protože výuku jazyka JAVA jsem nikdy neabsolvoval, stejně tak jako výuku aproximace přechodových charakteristik. Studium nového programovacího jazyka a aproximace bylo náročné a zdlouhavé, avšak po delší době studia jsem zjistil, že programování servletu i výpočtového programu v jazyce Matlab není pro člověka alespoň trochu znalého těchto nástrojů nijak náročné.

Bakalářskou práci jsem doplnil 24 obrázky a 2 tabulkami, které zpřehledňují vytváření úkolu.

Funkčnost řešení byla ověřena instalací na školní server, kde si může kdokoliv prohlédnout postup aproximace.

Přínosy zpracované úlohy:

a. Zpřístupnění rozsáhlých funkcí jazyka Matlab prostřednictvím internetu.
b. Využití servletu jako výukového programu aproximační metody prof. Strejce.
c. Cenová nenáročnost

Celou práci jsem koncipoval také tak, aby popsala metodu pro výuku a použití spojení prostředí Matlab a JAVA servletů, proto tento text a celá práce může sloužit i pro základní návod k tvorbě dalších servletů.

Seznam obrázků

Seznam tabulek

Seznam použité literatury

[1] HEROUT, Pavel: Učebnice Jazyka Java – rozšířené vydání zahrnuje změny od Java 5. Nakladatelství Kopp, České Budějovice, 2011.

[2] HALL, Marty: Java: servlety a stránky JSP, Neocortex, Praha, 2001

[3] ZAPLATÍLEK, Karel - DOŇAR, Bohuslav: Matlab – pro začátečníky, Nakladatelství BEN, Praha, 2005.

[4] ZAPLATÍLEK Karel – DOŇAR, Bohuslav: Matlab - tvorba uživatelských aplikací, Nakladatelství BEN, Praha, 2005.

[5] MATLAB Help

[6] OLEHLA, Miroslav – NĚMEČEK, Slavomír: Základy aplikované kybernetiky, TUL, Liberec, 2005

[7] JANEČEK, Josef. Aproximace přechodových charakteristik metodou prof. Strejce. [online]. 2000 [cit. 2012-02-20]. Dostupné z: http://www.mti.tul.cz/files/zky/Strejc.pdf

[8] BOURKE, Paul: Intersection point of two lines (2 dimensions). [online]. [cit. 2012-02-15]. Dostupné z: http://local.wasp.uwa.edu.au/~pbourke/geometry/lineline2d/

[9] KLOBOUČEK, Jan – STIANKO, Martin.: Základy aplikované kybernetiky [online]. Liberec: TUL, 2007 [cit. 2012-02-20]. Dostupné z: http://www.kky.tul.cz/ladmin/soubory/kky/File/skripta/ZAK-CVICENI-web.pdf

[10] MODRLÁK, Osvald: Analýza dynamických systémů, TUL, Liberec, 2002

[11] FAJKUS, Marcel: *Matlab Server* [online]. 2.11.2011 [cit. 2012-03-24]. Dostupné z: http://matlabserver.ic.cz/

[12] SEDLÁČEK, Vladimír. HTML - Speciální znaky. Sedlo [online]. 2005 [cit. 2012-05-02]. Dostupné z: http://sedlo.sedlo.net/html/specialniznaky.php

[13] JANOVSKÝ, Dušan. CSS: Kaskádové styly. Jak psát Web [online]. 2012 [cit. 2012-05-03]. Dostupné z: http://www.jakpsatweb.cz/css/

Obsah přiloženého CD

1. Adresář TEXT obsahující text bakalářské práce
2. Adresář PROGRAM obsahující zdrojové kódy programu
3. Adresář DATA obsahující vzorová data