3.T Technické výpočty v Octave/Matlabu – zpracování a zobrazení dat

Ať už se vydáš na jakýkoliv technický či přírodovědný obor, neminou tě algebraické nebo analytické výpočty. Tento tutoriál tě provede základy práce s technickým výpočetním jazykem Matlab, který ti ušetří mnoho času. Jazyk Matlab byl vyvinut společností Mathworks, jejíž vývojové prostředí Matlab je však velmi drahé. My se proto zaměříme na svobodnou alternativu, na program Octave, který využívá stejného jazyka. Vyzkoušíme si v něm výpočty hodnot funkcí jedné a dvou proměnných, zobrazení výsledků v grafech, maticové a vektorové operace, členění zdrojového kódu na skripty a funkce, symbolické výpočty funkcí a základní práci se soubory.

Tento text je pouze krátkou exkurzí do tajů Octavu. Odpovědi na další otázky hledej například v dokumentaci jazyka[1]. Nejběžnější funkce jsou stejné jako v Matlabu, proto můžeš využít i propracovanější dokumentaci Mathworks[2]. Také je možné použít příkaz `help` s jedním parametrem, který do příkazového okna neboli *Command window* vypíše popis požadované funkce. Samozřejmě je i mnoho jiných volně dostupných zdrojů.

Vývojové prostředí – instalace a nastavení Octave

Pro instalaci Octave na Windows stáhni a rozbal archiv, který jsme umístili na naše stránky[3]. Jsou v něm čtyři instalační soubory s klasickým průvodcem instalací. Je potřeba je instalovat v následujícím pořadí:

1. `octave-3.6.2-vs2010-setup.exe` – v průvodci nic neměnit, jen odklikat.
2. `octave-3.6.2-symbolic-1.1.0-vs2010-setup.exe`.
3. `Octave-UPM-R8.2-setup.exe` – změnit *Destination Folder* na `C:\Octave\Octave-3.6.2\` (tj. složka, kam se instaloval i předchozí soubor).
4. `vcredist_x86.exe` – instalovat pouze pokud se program nebude chtít spustit.

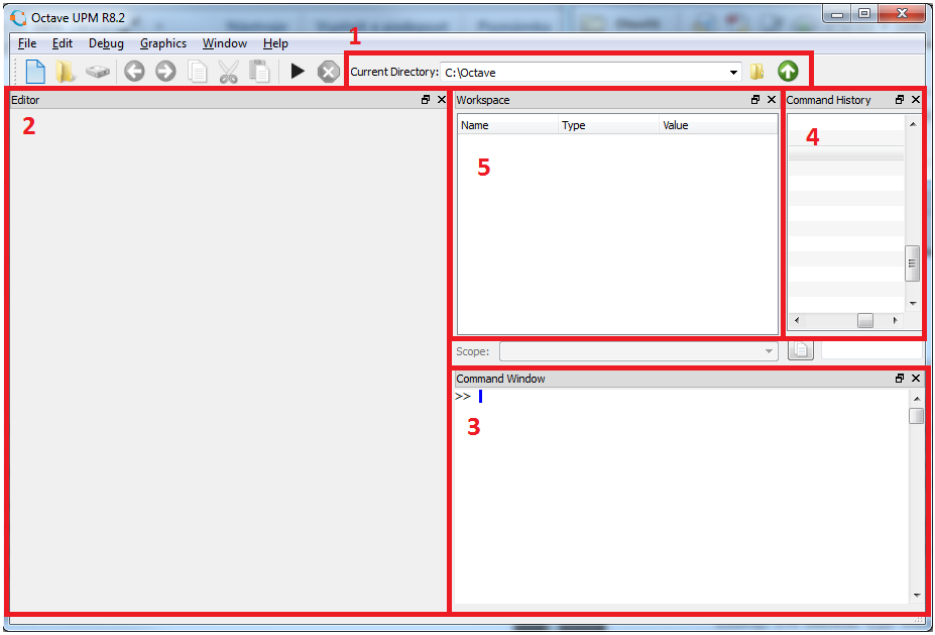
Po instalaci najdeš v nabídce Start nebo na ploše ikonu *Octave UPM*, kterou spustíš grafické rozhraní (případně spustíš souborem `bin\octave-upm.exe`, který je ve složce, kam se program nainstaloval).

Uživatelské rozhraní, jak je vidět na obrázku 1, má několik podoken:

1. aktuální složka – pokud se volá skript nebo funkce ze souboru, musí být tento soubor umístěn právě v aktuální složce,
2. editor souboru se zdrojovým kódem – okno, kde se píše zdrojový kód funkce nebo skriptu,

[1] www.gnu.org/software/octave/doc/interpreter/
[2] http://www.mathworks.com/help/matlab
[3] svat.fjfi.cvut.cz/files/Octave.zip

3. příkazové okno – místo, kam se píší příkazy (po stisknutí klávesy `Enter` se příkaz provede),
4. historie volaných příkazů – seznam minulých příkazů, které byly zavolány v příkazovém okně,
5. proměnné v paměti – aktuální proměnné a jejich hodnoty v paměti programu.



Obrázek 1 Okno grafického uživatelského prostření Octave UPM po prvním spuštění

Navíc na záložce *Window* je možné zaškrtnout *Current Directory* pro zobrazení stromu adresářů a souborů současné složky. Tvar a umístění oken je možné měnit tahem myši.

Uživatelům většiny linuxových systémů stačí nainstalovat balíček `octave` obsahující jádro programu a například `qtoctave` přidávající GUI nastavbu. Okna vypadají téměř identicky jako na obrázku 1. V případě problémů se podívej na oficiální stránky stejnojmenného projektu.

Základní funkce a matice

Začněme naším prvním výpočtem. Do příkazového okna napiš příkaz `sin(pi/2)` a potvrď `Enter`. Ano samozřejmě, $\sin \pi/2$ je jedna, proto `ans = 1`.

Co se ale stane, když funkci sin zadáme dva argumenty? Odpověď vyzkoušej zjistit například příkazem `help sin`. Další funkce viz dokumentace Octave nebo Matlabu.

Úkol a *(1 b.)*: *V dokumentaci najdi, jak se počítá hodnota exponenciální funkce v zadaném bodě, a vypočítej ji pro* π. *Tedy zajímá nás výsledek* e^{π}.

Tímto způsobem se dá Octave použít jako rychlá kalkulačka. Pro komplexnější práci se ale hodí zavést si proměnné.

Proměnné mohou být číselnými hodnotami, vektory či maticemi.[4] Numerické hodnoty, resp. vektory jsou považovány za matice typu 1×1, resp. $1 \times n$ či $n \times 1$, kde n je délka řádkového, resp. sloupcového vektoru. Matici můžeme definovat výčtem jejích prvků v hranatých závorkách, přičemž jednotlivé prvky na řádku oddělujeme mezerou nebo čárkou; jednotlivé řádky matice oddělujeme středníkem. Matice a vektory lze vytvořit mnoha způsoby:

```
u = [1 2 3]  %řádkový vektor
v = [1; 2; 3]  %sloupcový vektor
w = [1 2 3]'  %sloupc. vektor vytvoř. operátorem transpozice
A = [1 2 3; 4 5 6; 7 8 9] %matice rozměrů 3 x 3
O = [] %prázdná matice

B = [v v v] %matice 3 x 3 vytvoř. složením sloupcových vektorů
C = [u; u; u]  %matice 3 x 3 vytvoř. slož. řádkových vektorů
D = [u v] %chyba -- nevyhovující rozměry matice
E = ones(3,4) %matice 3 x 4 složená ze samých jedniček
F = zeros(3,4) %matice 3 x 4 složená ze samých nul
G = eye(3,3) %matice 3 x 3 s jedničkami na diagonále
                                                a nulami jinde
```

S maticemi lze provádět množství operací:

```
>> A - B %rozdíl (podobně součet) matic stejných rozměrů
ans =
      0  1  2
      2  3  4
      4  5  6

>> A + 1 %součet (podobně rozdíl) matice a čísla
ans =
      2  3  4
      5  6  7
      8  9 10

>> A * 2 %násobení (podobně dělení) matice číslem
```

[4] Pro více podrobností o tom, co je matice, viz text k úloze o Robotickém rameni.

```
ans =
      2   4   6
      8  10  12
     14  16  18

>> A * B %násobení dvou matic
ans =
     14  14  14
     32  32  32
     50  50  50
```

Přitom musejí být samozřejmě použité matice kompatibilní (při sčítání a odčítání musejí mít matice stejné rozměry a při násobení musí být počet sloupců první roven počtu sloupců druhé). Součtem matice a čísla se rozumí přičtení čísla ke každému prvku matice (v matematice se tato operace nepoužívá, ale při programování může zjednodušit kód). Rozdíl, součin a podíl matice a čísla jsou založeny na stejném principu, opět se daná operace provede s číslem a každým jednotlivým prvkem matice.

Speciální operací na maticích je pak . (tečka). Způsobí, že se operace napsané za ní provedou po jednotlivých prvcích matice. Například násobení:

```
>> A .* B %násobení (podobně dělení) prvek po prvku
    1  2  3
    8 10 12
   21 24 27
```

Už umíme matice vytvořit a aplikovat na ně základní matematické operace, ale jak se dostat k jejím konkrétním prvkům? Tzv. indexováním matice. Na jednotlivé prvky matice je možné se odkazovat pomocí kulatých závorek:

```
>> v = [4 7 2 8 1];

>> v(3);  %vybere třetí prvek
  ans = 2;

>> v([1 5]);  %vybere první a pátý prvek
  ans = 4 1

>> v(2:4) %vybere druhý až čtvrtý
  ans = 7 2 8

>> v([3:end 1:2]) %vybere poslední tři a pak první dva
  ans = 2 8 1 4 7

>> W = [8 6 7; 5 2 3; 9 4 1];
```

```
>> W(3,2) %vybere prvek ve třetím řádku a druhém sloupci
  ans = 4

>> W([1 3], [3 2]) %prvky 1. a 3. řádku, které jsou zároveň
  ans =                                        v 3. a 2. sloupci
      7  6
      1  4

>> W(:, 1:2) %vybere všechny řádky a první dva sloupce
  ans =
      8  6
      5  2
      9  4
```

Úkol b *(1 b.)*: *Napiš kód, který vygeneruje dvě náhodné matice* 5×5 *(použij* `magic`*), provede jejich maticový součin a z výsledku vypočítá součet všech řádků a součet všech sloupců. Takto vzniklé vektory spolu opět maticově vynásobí. (Konečným výsledkem by tedy mělo být číslo.)*

Členění zdrojového kódu – skripty, funkce

Při složitějších výpočtech je příliš pracné a pomalé psát všechny jednotlivé příkazy do příkazového okna. Například pro výpočet kořenů kvadratické rovnice $ax^2 + bx + c = 0$ bychom museli postupně zadat hned několik příkazů:

```
a = 1
b = -3
c = 2
D = b^2 - 4*a*c
x1 = (-b + sqrt(D))/(2*a)
x2 = (-b - sqrt(D))/(2*a)
```

Jednodušší bude vytvořit si na výpočet kvadratické rovnice skript. Vytvoř nový soubor kliknutím na první ikonu na panelu, předešlé příkazy do něj zkopíruj a soubor ulož v aktuální složce (případně změn umístění na vhodnější). Nyní v příkazovém okně napiš jméno souboru a potvrď. Měla by se provést celá sekvence příkazů naráz. Pokud nechceš, aby se vypisovaly výsledky některých nebo i všech příkazů, ukonči každý z těchto příkazů středníkem.

Nevýhodou skriptu je fakt, že nepřijímá žádná data zvenčí, vše musí být definováno uvnitř. Pro podobné účely je tedy často lepší použít tzv. *funkce*. Funkce se od skriptu liší tím, že má na prvním řádku hlavičku s klíčovým slovem `function`, výstupními parametry, názvem funkce a vstupními parametry. Navíc soubor musí mít stejný název, jako má funkce v hlavičce. Funkce pro výpočet kvadratické rovnice by vypadala následovně:

```
function [x1, x2] = koreny_kvadr_rce(a,b,c)
```

```
  D = b^2 - 4*a*c;
  x1 = (-b + sqrt(D))/(2*a);
  x2 = (-b - sqrt(D))/(2*a);
end
```

Takovou funkci bychom zavolali třeba následovně:

```
>> [koren1, koren2] = koreny_kvadr_rce(1,-3,2)
```

Tip: Vlevo od čísel řádků v okně editoru je možné umístit kliknutím tzv. *breakpoints*. Běh programu se pak zastaví na takto označeném řádku. V tu chvíli je možné program krokovat pomocí příkazů či klávesových zkratek. Buď necháš provést jeden příkaz, nebo necháš program rozběhnout doběhnout až k dalšímu breakpointu. Více viz menu Debug.

Úkol c *(1 b.)*: *Napiš funkci na výpočet povrchu rotačního kužele na základě jeho výšky a poloměru podstavy.*

Graf funkce – výpočet hodnot a zobrazení

Octave umožňuje vykreslovat přehledné grafy funkcí jedné, ale i dvou proměnných. Je k tomu potřeba vždy vektor (libovolné délky) hodnot z definičního oboru a jim odpovídající vektor funkčních hodnot.

```
>> x = -5 : 0.1: 5;
>> y = x.*sin(x);
>> plot(x,y);
```

Dále je možné přidat titulek grafu, popsat osy a určit jejich rozsah.

```
>> title("graf funkce x*sin(x)");
>> xlabel("definiční obor");
>> ylabel("funkční hodnoty");
>> xlim([-5 5]);
```

Podobně lze postupovat i u funkce dvou proměnných. Nejprve je ale potřeba definovat síť hodnot z definičního oboru. Jsou to matice, kde hodnoty rostou vždy jen v jedené dimenzi, ve druhém směru je celý sloupec (resp. řádek) stejný.

```
>> [X Y] = meshgrid(-5 : 0.1 : 5);
>> Z = Y.*sin(X);
>> mesh(X,Y,Z);
```

Funkce `mesh` vykresluje síť bodů sestavenou z funkčních hodnot Z nad definičním oborem. Podobně funguje `surf`, která z funkčních hodnot sestaví povrch. Podobně jako ve 2D je možné zobrazit i ve 3D čárový graf pomocí `plot3`, kde vstupem jsou tři vektory stejné délky, každý pro jednu dimenzi.

```
>> x = 0: pi/20 : 10*pi;
>> y = cos(x);
>> z = sin(x);
>> plot3(x,y,z);
```

Úkol d *(2 b.)*: *Vytvoř graf, který se svým tvarem co nejvíce podobá hoře Říp.*

Řízení běhu programu – podmínky, cykly

Stejně jako ve všech programovacích jazycích, i v Octave existují klíčová slova pro cykly, podmínky, jejich přerušení a návrat z funkce. To se hodí například při počítání posloupností, při vícero opakování výpočtu k získání statistických dat nebo pro ošetřování výjimek. Ukažme si tyto elementy na příkladu počítání faktoriálu.

Cyklus `while` běží, dokud platí podmínka následující za klíčovým slovem `while`:

```
function f = faktorial1(n)
  f = 1;
  while n > 1
    f = f*n;
    n = n-1;
  end
end
```

Na začátku se do proměnné `f` uloží jednička. Poté se, dokud je číslo `n` větší než jednička, provádějí dva příkazy – `f` se vynásobí `n` a `n` se zmenší o jedničku. Jestliže funkci spustíme pro `n=3` (tedy chceme zjistit, kolik je 3!), pak bude probíhat program takto: do `f` se uloží 1, `n` je rovno třem, takže je větší než jedna, do `f` se uloží `f=f*n=1*3=3` a `n` se zmenší o jedničku na 2. Dvojka je opět větší než jednička, tak se znovu provede totéž: `f=f*n=3*2=6` a `n` se zmenší o jednotku na jedničku. Ta už není větší než 1 a program končí. Výsledkem je 6, což se skutečně rovná faktoriálu tří.

Cyklus `for` má obdobný účel. Používá se, zejména pokud je předem jasné, kolikrát má příslušná sekvence příkazů proběhnout:

```
function f = faktorial2(n)
  f = 1;
  for i=1:n
    f = f*i;
  end
end
```

Zde `for i=1:n` znamená, že příkazy uvnitř cyklu se provedou pro všechna `i` od jedné do `n`. Začne se tedy s `i=1` a do `f` se uloží `f=f*i=1*1=1`. Pak se zvolí `i=2` a spočítá se opět `f=f*i=1*2=2`, dále `i=3` a `f=f*i=2*3=6`. Poslední cyklus proběhne pro `i=n`. Celkově se tak příkazy uvnitř cyklu provedou `n`-krát, pokaždé pro jinou hodnotu `i`.

V cyklech se navíc používají příkazy `continue` pro okamžité přeskočení k další iteraci cyklu a `break` pro ukončení cyklu.

Pro kontrolu vstupního parametru můžeme použít podmínku `if`, která provede blok podřazených příkazů pouze tehdy, pokud je výraz uvnitř kulatých závorek vyhodnocen jako pravdivý.

```
function f = faktorial3(n)
  if n < 0
    f = n;
    return;
  end
  f = 1;
  while n > 1
    f = f*n;
    n = n-1;
  end
end
```

V případě, že je vstupní parametr faktoriálu menší než nula, dojde k ukončení funkce pomocí příkazu `return`.

Úkol e *(2 b.)*: *Napiš skript (funkci se vstupem N), který vrátí všechna prvočísla z intervalu 1 až 100 (N) pomocí tzv. Eratosthenova síta*[5].

Soubory – otevření, čtení, zápis

Často je potřeba načíst data generovaná měřícími aparaturami nebo uložit hodnoty z programu pro další využití. K tomu je potřeba umět pracovat s textovými soubory. Soubory se vždy před samotným čtením/zápisem musí otevřít a poté zavřít příkazy `fopen` resp. `fclose`. Příkaz `fopen` vrátí identifikátor otevřeného souboru, se kterým se pak pracuje v dalších funkcích. Soubory se otevírají s určitým oprávněním, např. pro čtení, zápis či vytvoření nového souboru. O tom se dočteš více v helpu funkce `fopen`. Pro formátovaný zápis do souboru se používá funkce `fprintf`. Prvním parametrem je identifikace otevřeného souboru, druhý je formát ukládaného textu a poslední samotná data.

```
fileId = fopen('soubor.txt', 'w');
  % 'w' (write) specifikuje zápis do souboru
fprintf(fileId, '%f\n', 1:10);
  % '%f' (float) formát v jakém se má číslo uložit
fclose(fileId); % '\n' je znak pro konec řádku
```

Obdobně se postupuje při načítání obsahu:

```
fileId = fopen('soubor.txt', 'r');
  % 'w' (write) specifikuje zápis do souboru
v = fscanf(fileId, '%f\n', 10);
  % poslední parametr je počet načítaných čísel
fclose(fileId);  % výsledek se uloží do vektoru 'v'
```

[5] http://cs.wikipedia.org/wiki/Eratosthenovo_síto

Úkol f *(2 b.)*: *Napiš skript (funkci se vstupem N), který vygeneruje sto (N) náhodných čísel, uloží je do souboru, následně znovu načte, setřídí od nejmenšího k největšímu a uloží zpět.*