Informatika pro moderní fyziky (5) zpracování výstupů a tvorba textových dokumentů

František HAVLŮJ

e-mail: haf@ujv.cz

ÚJV Řež
oddělení Reaktorové fyziky a podpory palivového cyklu

akademický rok 2017/2018
8. listopadu 2017

Obsah

- používání Rakefile
- propojení Ruby a Gnuplotu
- rozšiřování možností jazyka Ruby (definice vlastních funkcí)
- načítání výstupů, zpracování do tabulky

Navážeme na výsledek z minula

Výsledkem je perfektní tabulka (`simple_table.csv`):

```
outputs/c_0_0o 0.94800
outputs/c_0_10o 0.99800
outputs/c_0_1o 0.94850
outputs/c_0_2o 0.95000
outputs/c_0_3o 0.95250
outputs/c_0_4o 0.95600
...
```

Hloupé je, že nikde nemáme tu polohu tyčí.

Víc by se nám hodilo

něco jak toto:

```
0  0.0  0.94800
0 64.0  0.99800
0  6.4  0.94850
0 12.8  0.95000
0 19.2  0.95250
0 25.6  0.95600
...
```

(rozsah je 0 – 640, my máme kroky 0 – 10)

A protože přehlednost je nade vše

něco jak toto:

```
keff          0.0          6.4         12.8    ...
 0.0      0.94800      0.94900      0.95000    ...
 6.4      0.94850      0.94950      0.95050    ...
12.8      0.95000      0.95100      0.95200    ...
19.2      0.95250      0.95350      0.95450    ...
25.6      0.95600      0.95700      0.95800    ...
32.0      0.96050      0.96150      0.96250    ...
38.4      0.96600      0.96700      0.96800    ...
44.8      0.97250      0.97350      0.97450    ...
51.2      0.98000      0.98100      0.98200    ...
57.6      0.98850      0.98950      0.99050    ...
64.0      0.99800      0.99900      1.00000    ...
...
```

Navážeme na úspěchy z minulých týdnů

- vykreslit graf! pro každou z 11 poloh R1 jedna čára (závislost keff na R2)
- (= csv soubor, gnuplot, znáte to)
- najít automaticky kritickou polohu R2 pro každou z 11 poloh R1
- a zase graf... (kritická poloha R2 v závislosti na R1)

Obsah

HELIOS

Tabulka výstupů:

```
List name        : list
List Title(s)  1) This is a table
               2) of some data
               3) in many columns
               4) and has a long title!
```

	bup	kinf	ab	ab	u235	
0001	0.00E+00	1.16949	9.7053E-03	7.6469E-02	1.8806E-04	7.
0002	0.00E+00	1.13213	9.7478E-03	7.9058E-02	1.8806E-04	7.
0003	1.00E+01	1.13149	9.7488E-03	7.9070E-02	1.8797E-04	7.
0004	5.00E+01	1.13004	9.7521E-03	7.9093E-02	1.8760E-04	7.
0005	1.00E+02	1.12826	9.7559E-03	7.9218E-02	1.8714E-04	7.
0006	1.50E+02	1.12664	9.7594E-03	7.9407E-02	1.8668E-04	7.
0007	2.50E+02	1.12399	9.7657E-03	7.9869E-02	1.8577E-04	7.
0008	5.00E+02	1.12007	9.7812E-03	8.1065E-02	1.8351E-04	7.
0009	1.00E+03	1.11561	9.8203E-03	8.3169E-02	1.7914E-04	7.
0010	2.00E+03	1.10542	9.9329E-03	8.6731E-02	1.7088E-04	7.
0011	3.00E+03	1.09354	1.0067E-02	8.9717E-02	1.6316E-04	7.

Co bychom chtěli

- mít načtené jednotlivé tabulky (zatím jen jednu, ale bude jich víc)
- asi po jednotlivých sloupcích, sloupec = pole (hodnot po řádcích)
- sloupce se jmenují, tedy použijeme `Hash`
- `table['kinf']`
- pozor na `ab`, asi budeme muset vyrobit něco jako `ab1`, `ab2` (ale to až za chvíli)

Nástrahy, chytáky a podobně

- tabulka skládající se z více bloků
- více tabulek
- tabulky mají jméno `list name` a popisek `list title(s)`

Jak uspořádat data?

- pole s tabulkami + pole s názvy + pole s titulky?

Jak uspořádat data?

- pole s tabulkami + pole s názvy + pole s titulky?
- co hashe tabulky[název] a titulky[název]?

Jak uspořádat data?

- pole s tabulkami + pole s názvy + pole s titulky?
- co hashe tabulky[název] a titulky[název]?
- nejchytřeji: `{'a' => {:title => 'Table title', data => {'kinf' => ...}}}`

Jak uspořádat data?

- pole s tabulkami + pole s názvy + pole s titulky?
- co hashe tabulky[název] a titulky[název]?
- nejchytřeji: `{'a' => {:title => 'Table title', data => {'kinf' => ...}}}`
- 'nová' syntaxe: `{'a' => {title: 'Table title', data: {'kinf' => ...}}}`

Z příkazové řádky

- a co takhle z toho udělat skript, který lze pustit s argumentem = univerzální
- `ruby read_helios.rb helios1.out`
- vypíše seznam všech tabulek, seznam jejich sloupců, počet řádků
- pole `ARGV`
- vylepšení – provede pro všechny zadané soubory: `ruby read_helios.rb helios1.out helios2.out`

Obsah

Klávesnice a myš

- myš je dobrá na grafiku a jako alternativa k tabletu
- taky se hodí tam, kde se potřebuju přesouvat mezi položkami, které nemají jednoznačné pořadí nebo prostorový vztah (neseřazené ikony na ploše, rozhraní s mrakem oken atd.)
- případně ještě na použití menu pro úkony, které dělám jednou za uherský rok
- naopak na programování je nejlepší na myš vůbec nešahat a používat skoro jenom klávesnici
- extrémní školy dokonce brojí proti kurzorovým šipkám, protože (na velké klávesnici) nutí měnit polohu rukou, což je pomalé

Přepínání jazyků

- je dobré se mu vyhnout, protože to opravdu trochu otravuje (i když se s tím dá docela dobře žít, pokud máte dobrou klávesovou zkratku)
- rozhodně stojí za to zjistit – například pro psaní v LaTeXu – kde na české klávesnici máte potřebné speciální znaky (v tomto případě zejména backslash a složené závorky)
- chytré editory mají různé pochystávky a makra, která vám umožní se těmto speciálním znakům defacto vyhnout
- pokud můžu, pracuju celou dobu s anglickou (tj. pokud výjimečně nepíšu český dokument)

Klávesové zkratky

- jako s programováním – musím se něco naučit / zapamatovat, ale pak mi to ušetří hromadu času
- minimálně základní sadu stojí za to se naučit
- často jdou ručně editovat, ale většinou to není nutné (a je to stejně na houby, pokud zrovna nesedíte u svého počítače)
- jako s hudebním nástrojem – za čas už neznáte ty zkratky, ale prostě je umíte zmáčknout bez přemýšlení
- hodně jich je sdílených napříč programy a editory

Klávesové zkratky - MS Windows

- copy-paste `Ctrl+C/V`
- undo `Ctrl+Z`
- přepínání aplikací `Alt+Tab`
- přepínání oken v rámci aplikace `Ctrl+Tab`

Klávesové zkratky – Notepad++

- pohyb v textu po slovech a stránkách `Ctrl`+šipky, `PgUp/PgDn`
- uložení, otevření, zavření `Ctrl+S, O, W`
- změna odsazení bloku `Tab / Shift+Tab`
- přepínání mezi soubory
- zakomentovat/odkomentovat `Ctrl+Q`

Obsah

Jak vyrobit zprávu?

- potřebujeme udělat hezké PDF shrnující výsledky našich výpočtů
- takže úvod, popis toho co jsme dělali a pak přehled výsledků
- tabulka s hodnotami, 11+1 graf
- co by znamenalo to dělat ve Wordu ..... ?
- hodilo by se to zautomatizovat!

Jak vygenerovat text?

- zase potřebujeme lepší nástroj na text, než jsou WYSIWYG (What You See Is What You Get) editory
- ideálně něco, co bude mít plain-text vstup (který můžeme s úspěchem generovat v Ruby) a co se pak
- odpověď zní LaTeX
- nejvíc nejlepší text-processor na světě

Koncepce oddělení obsahu a formy

- když píšu, nechci říct, že je text tučně a o dva body větší, ale že je to nadpis kapitoly
- ideálně chci popsat někde, jak bude dokument vypadat a nemíchat vzhled s obsahem
- styly ve Wordu se tomu vzdáleně blíží
- v LaTeXu vlastně píšu jen obsah a o formu se musím starat jen hodně málo
- je samozřejmostí zadarmo obsah, rejstřík atd.
- kdo píše diplomku v něčem jiném, kazí si život

Příklad jednoduchého dokumentu

- viz `document.tex`
- není potřeba úplně všemu rozumět, zatím to jen budeme upravovat v mezích zákona
- všechny příkazy začínají zpětným lomítkem, parametry jsou ve složených závorkách
- napíšu `pdflatex document.tex` a dostanu pdfko!

Text

- konce řádku nejsou důležité
- nový odstavec se dělá prázdným řádkem
- nové (pod)kapitoly pomocí příkazů `section`, `subsection`

Tabulka

- prostředí (= `begin` ... `end`)
- sloupce se

```
\begin{tabular}{ll}
a & b \\
c & d \\
\end{tabular}
```

Úkol

- vyrobit PDF s výsledky
- 11 grafů keff
- 1 graf závislosti kritické polohy R2 na poloze R1
- tabulka keff
- tabulka kritických poloh
- (a my to dělat nebudeme)

Pozor na backslash

- v Ruby se v řetězci backslash `\` používá jako escape character
- např konec řádky je `\n`
- pokud chci vytiskout zpětné lomítko (což bude asi pro LaTeX potřeba), musím ho zdvojit: `\\`

Složitější práce

- LaTeX je ideální pro rozsáhlé texty
- výzkumák, diplomka, disertace se naformátuje sama a všechno funguje bez trápení a snažení
- odkazy, reference, citace .. všechno bez starostí

Prezentace

- balíček `beamer`
- bez nutnosti se uklikat to samo od sebe vypadá slušně
- viz tahle prezentace
- nevýhoda(?): obtížnost přizpůsobit rozmístění textu apod., ale na druhou stranu to aspoň drží jednotný styl

Obsah

Úkol na dnešek

- pro jeden blok JE mám provozní data - v určitých dnech hodnotu koncentrace kyseliny borité a axiálního ofsetu - pro několik kampaní (blíže neurčený počet)
- chci vyrobit přehledové PDF, které bude hezky prezentovat grafy obou veličin pro každou kampaň a k tomu i tabulky
- data pro jednotlivé kampaně mám v CSV souborech, každý má tři sloupce (datum, cB, AO)

Rozbor

- načíst tabulky a vykreslit grafy umíme
- převést tabulky v CSV na tabulky v LaTeXu se záhy naučíme
- vložit obrázek do latexu taky umíme
- předem neznámý počet souborů nás netrápí (`Dir["*.csv"]`)

Tak nejdřív ty grafy

- to už je vážně obehraná písnička, ale tady je aspoň trochu změna
- potřebujeme vybrat, které dva sloupce použít - parametr `using`
- `plot "data.csv" using 1:2`
- datum na vodorovné ose – potřeba načíst ve správném formátu atd.
- `set xdata time`
- `set timefmt "%m/%d/%Y"`
- s hvězdičkou: nastavit nadpis a popisky os

Ne tak úplně chytře

```
Dir["*.csv"].each do |fn|
  base = fn.split(".").first

  File.open("#{base}_bc.gp", "w") do |f|
    f.puts "set terminal png"
    f.puts "set xdata time"
    f.puts "set timefmt \"%m/%d/%Y\""
    f.puts "set output \"#{base}_bc.png\""
    f.puts "plot \"#{base}.csv\" using 1:2"
  end
  `gnuplot #{base}_bc.gp`

  File.open("#{base}_ao.gp", "w") do |f|
    f.puts "set terminal png"
    f.puts "set output \"#{base}_ao.png\""
    f.puts "set xdata time"
    f.puts "set timefmt \"%m/%d/%Y\""
    f.puts "plot \"#{base}.csv\" using 1:3"
  end
  `gnuplot #{base}_ao.gp`
end
```

DRY

- základní paradigma: DRY = don't repeat yourself
- nemá cenu psát dvě věci stejně
- použití `copy and paste` při programování je varovný signál
- pokud nejsou stejně, ale skoro stejně, je potřeba trochu chytrosti
- připomeňme si hash: `{"a" => 1, "b" => 2}`
- přes hash se dá iterovat:

```
{"a" => 1, "b" => 2}.each do |key, value|
```

Grafy – DRY

```
Dir["*.csv"].each do |fn|
  base = fn.split(".").first
  {"bc" => 2, "ao" => 3}.each do |var, col|
    File.open("#{base}_#{var}.gp", "w") do |f|
      f.puts "set terminal png"
      f.puts "set xdata time"
      f.puts "set timefmt \"%m/%d/%Y\""
      f.puts "set output \"#{base}_#{var}.png\""
      f.puts "plot \"#{base}.csv\" using 1:#{col}"
    end
    `gnuplot #{base}_#{var}.gp`
  end
end
```

Jak na tabulky

- tabulky budou dost rozsáhlé a montovat je přímo nějak do latexových vstupů je asi spíš nepraktické, naštěstí to jde i jinak
- naštěstí má LaTex příkaz `\input`, kterým můžeme prostě vložit do dokumentu nějaký externí soubor
- takže si nejdřív přichystáme soubory s tabulkami a pak se na ně budeme už jenom odkazovat

Jak na tabulky v LaTeXu (1)

Základem tabulky je prostředí `tabular` s definicí počtu a zarovnání sloupců:

```
\begin{tabular}{lrr}
  ...
\end{tabular}
```

Jak na tabulky v LaTeXu (2)

Uvnitř tabulky se sloupce oddělují ampersandem a řádky dvojitým backslashem:

```
\begin{tabular}{lrr}
  Data 1 & a & 1.0 \\
  Data 2 & b & 2.0 \\
  Data 3 & c & 3.0 \\
\end{tabular}
```

Jak na tabulky v LaTeXu (3)

Přidání mřížky je nesnadné, leč proveditelné a vlastně docela dobře vymyšlené - přidáváme jednotlivé čáry po sloupcích a řádcích:

```
\begin{tabular}{|l|r|r|}
  \hline
  Data 1 & a & 1.0 \\
  \hline
  Data 2 & b & 2.0 \\
  \hline
  Data 3 & c & 3.0 \\
  \hline
\end{tabular}
```

Úkol na teď: výroba tabulek

- vyrobit z CSV souboru (tři sloupce) dvě LaTeX tabulky (po dvou sloupcích)
- postarat se, aby byly hezké
- chytré je vyrobit tabulku třeba o šesti sloupcích (jakože tři dvousloupce), pak už se to na stránku v klidu vejde

Obsah

Úskalí šablon

- snadno umíme nahradit jeden řetězec druhým
- trochu méně pohodlné pro větší bloky textu
- navíc by se hodila nějaká logika (cyklus) přímo v šabloně
- naštěstí jsou na to postupy

ERb (Embedded Ruby)

- lepší šablona - “aktivní text”
- používá se například ve webových aplikacích
- hodí se ale i na generování latexových dokumentů, resp. všude, kde nám nesejde na whitespace
- poměrně jednoduchá syntax, zvládne skoro všechno (viz předmět MAA3)

Základní syntaxe ERb (1)

Jakýkoli Ruby příkaz, přiřazení, výpočet ...

```
<% a = b + 5 %>
<% list = ary * ", " %>
```

Základní syntaxe ERb (2)

Pokud chci něco vložit, stačí přidat rovnítko

```
<%= a %>
<%= ary[1] %>
<%= b + 5 %>
```

Základní syntaxe ERb (3)

Radost je možnost použít bloky a tedy i iterátory apod. v propojení s vkládaným textem:

```
<% (1..5).each do |i| %>
Number <%= i %>
<% end %>
<% ary.each do |x| %>
Array contains <%= x %>
<% end %>
```

ERb – shrnutí

- dobrý sluha, ale špatný pán
- můžu s tím vyrobit hromadu užitečných věcí na malém prostoru
- daň je velké riziko zamotaného kódu a nízké přehlednosti (struktura naprosto není patrná na první pohled, proto je namístě ji držet maximálně jednoduchou)

Důležité upozornění

- oddělení modelu a view
- přestože lze provádět zpracování dat a výpočty přímo v ERb, je to nejvíc nejhorší nápad
- je chytré si všechno připravit v modelu (tj. v Ruby skriptu, kterým data chystáme)
- a kód ve view (tj. v ERb šabloně) omezit na naprosté minimum

Jak ze šablony udělat výsledek

Příklad překladu ERb

```
require "erb_compiler"

erb(template, filename, {:x => 1, :y => 2})
```

Příklad – kreslení grafů z minula

template.gp

```
set terminal png
set output "plot_<%=n%>.png"
plot "data_<%=n%>.csv"
```

```
(1..10).each do |i|
  erb("template.gp", "plot_#{i}.gp", {:n => i})
end
```

Takže v latexu třeba

```
\subsection{Koncentrace kyseliny borité}

<% files.each do |f| %>

\subsubsection{Kampaň <%= f.split("_").last %>}
\begin{center}
\includegraphics[width=0.8\textwidth]{<%= f %>_bc.eps}
\end{center}
<% end %>
```

A teď už to jenom dejte dohromady...

1. připravit si základní kostru dokumentu v latexu
2. převést na šablonu: mít seznam souborů, správně generovat kapitoly
3. vyrobit grafy
4. vložit grafy do šablony
5. vyrobit tabulky
6. vložit tabulky do šablony
7. A JE TO!

A to je vše, přátelé!