Informatika pro moderní fyziky (4) rozšíření RubyGems, tvorba excelovské tabulky a zpracování textu

František HAVLŮJ

e-mail: haf@ujv.cz

ÚJV Řež
oddělení Reaktorové fyziky a podpory palivového cyklu

akademický rok 2020/2021, 26. října 2020

Obsah

- práci s datovými soubory
- zpracování většího množství dat

Obsah

Knihovny (gemy) jsou základ

- existují mnohá rozšíření, tzv. knihovny – v ruby se jim říká **rubygems**
- aktuálně nás zajímá něco na práci s excelovskými soubory
- gemy jdou sice instalovat na systémové úrovni, ale z toho je pak zase jenom neštěstí
- použijeme radši **bundler**, správce gemů pro každého: vyřeší za nás závislosti a postará se o snadnou instalaci

Máme bundler?

- otestujeme rubygems: `gem -v`
- pokud není, zapláčeme, protože jsme asi špatně nainstalovali Ruby
- otestujeme bundler: `bundle -v`
- pokud bundler není, doinstalujeme `gem install bundler`

Jak na to

- najdu si, která knihovna mě zajímá (třeba na rubygems.org nebo kdekoli jinde): my bychom rádi **rubyXL** `https://github.com/weshatheleopard/rubyXL`
- vytvořím si prázdný `Gemfile` – tam se specifikuje, které gemy chci používat: `bundle init`
- do gemfilu – je to normální Ruby skript! – dopíšu `gem "rubyXL"`
- nainstaluju: `bundle install --path vendor/bundle` (ten parametr stačí poprvé, bundler si to pak pamatuje v konfiguráku `bundle/.config`)

Jak použít?

- na začátku svého skriptu pak musím nahrát bundler:
- `require "bundler/setup"`
- a teď už můžu nahrát jakýkoli gem:
- `require "rubyXL"`

RTFM, RTFM, RTFM

- na stránkách `rubyXL` se nachází spousta příkladů a návodů – https://github.com/weshatheleopard/rubyXL
- kromě toho má i slušnou dokumentaci (GIYF / "rubyxl docs") – http://www.rubydoc.info/gems/rubyXL/3.3.15
- napoprvé navedu do začátku:

```
workbook = RubyXL::Workbook.new
worksheet = workbook[0]
worksheet.add_cell(0, 0, "A1")
workbook.write("data.xlsx")
```

Jednoduché cvičení

- použijte soubor `data_two_1.csv`
- vytvořte excelovský soubor se dvěma listy, na obou bude sloupec 1, sloupec 2 (data) a třetí sloupec součet
- na jednom listu součet bude jako číslo (sečte to váš skript)
- na druhém listu bude součet jako excelovský vzorec

Obsah

Problém č. 2: mnoho výpočtů, inženýrova smrt

Zadání

Při přípravě základního kritického experimentu je pomocí MCNP potřeba najít kritickou polohu regulační tyče R2.
Jak se tato poloha změní při změně polohy tyče R1?

Co máme k dispozici?

MCNP

Pokud připravíme vstupní soubor (v netriviální formě obsahující polohy regulačních tyčí R1 a R2), spočítá nám keff.

Potřebovali bychom ale něco na:

1. vytvoření velkého množství vstupních souborů
2. extrakci keff z výstupních souborů
3. popřípadě na vyhodnocení získaných poloh tyčí a keff

Pracovní postup

1. načíst keff z výstupního souboru MCNP

Pracovní postup

1. načíst keff z výstupního souboru MCNP
2. vygenerovat potřebné vstupní soubory

Pracovní postup

1. načíst keff z výstupního souboru MCNP
2. vygenerovat potřebné vstupní soubory
3. vyrobit BAT soubor na spuštění výpočtů

Pracovní postup

1. načíst keff z výstupního souboru MCNP
2. vygenerovat potřebné vstupní soubory
3. vyrobit BAT soubor na spuštění výpočtů
4. načíst výsledky ze všech výstupních souborů do jedné tabulky

Nejprve najdeme, kde je ve výstupu z MCNP žádané keff:

```
.....
         the k(trk length) cycle values appear normally distributed at the 95 percent confide
------------------------------------------------------------------------------------------
|
| the final estimated combined collision/absorption/track-length keff = 1.00353 with an estima
|
| the estimated 68, 95, & 99 percent keff confidence intervals are 1.00329 to 1.00377, 1.0030
|
| the final combined (col/abs/tl) prompt removal lifetime = 1.0017E-04 seconds with an estimat
.....
```

Algoritmus

1. najít řádek s keff

Algoritmus

1. najít řádek s keff
2. vytáhnout z něj keff, takže například:

Algoritmus

1. najít řádek s keff
2. vytáhnout z něj keff, takže například:
3. rozdělit podle rovnítka

Algoritmus

1. najít řádek s keff
2. vytáhnout z něj keff, takže například:
3. rozdělit podle rovnítka
4. druhou část rozdělit podle mezer

Algoritmus

1. najít řádek s keff
2. vytáhnout z něj keff, takže například:
3. rozdělit podle rovnítka
4. druhou část rozdělit podle mezer
5. vzít první prvek

Realizace (1/5)

```
keff = nil

File.foreach("c1_1o") do |line|

end

puts keff
```

Realizace (2/5)

```
keff = nil

File.foreach("c1_1o") do |line|
  if line.include?("final estimated combined")

  end
end

puts keff
```

Realizace (3/5)

```
keff = nil

File.foreach("c1_1o") do |line|
  if line.include?("final estimated combined")
    a = line.split("=")

  end
end

puts keff
```

Realizace (4/5)

```
keff = nil

File.foreach("c1_1o") do |line|
  if line.include?("final estimated combined")
    a = line.split("=")
    b = a[1].split

  end
end

puts keff
```

Realizace (5/5)

```
keff = nil

File.foreach("c1_1o") do |line|
  if line.include?("final estimated combined")
    a = line.split("=")
    b = a[1].split
    keff = b[0]
  end
end

puts keff
```

Jak na to

Máme všechno, co potřebujeme:

- načtení keff z jednoho výstupního souboru (`File.foreach`, `include` a `split`)
- procházení adresáře (`Dir.each`)
- zápis do souboru (`File.open` s parametrem `w` anebo `File.write`)

Takže už to stačí jen vhodným způsobem spojit dohromady!

Realizace

```
Dir["*o"].each do |filename|
  keff = nil

  File.foreach(filename) do |line|
    if line.include?("final estimated combined")
      a = line.split("=")
      b = a[1].split
      keff = b[0]
    end
  end

  puts "#{filename} #{keff}"
end
```

Výstup

Výsledkem je perfektní tabulka:

```
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  0.94800
0 640  0.99800
0  64  0.94850
0 128  0.95000
0 192  0.95250
0 256  0.95600
...
```

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

A protože přehlednost je nade vše

něco jak toto:

keff	0	64	128	...
0	0.94800	0.94900	0.95000	...
64	0.94850	0.94950	0.95050	...
128	0.95000	0.95100	0.95200	...
192	0.95250	0.95350	0.95450	...
256	0.95600	0.95700	0.95800	...
320	0.96050	0.96150	0.96250	...
384	0.96600	0.96700	0.96800	...
448	0.97250	0.97350	0.97450	...
512	0.98000	0.98100	0.98200	...
576	0.98850	0.98950	0.99050	...
640	0.99800	0.99900	1.00000	...

– alternativně můžete vyrobit tabulku v excelu!

Hash to the rescue

Možná se bude velmi hodit hash!

```
data = {}
data[3] = 0.99
```

no a dokonce, protože klíč může být cokoliv:

```
data = {}
data[[1,2]] = 0.99
data[[7,8]] = 1.05
puts data[[1,2]]
```

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)

A to je vše, přátelé!