

# Informatika pro moderní fyziky (3) rozšíření RubyGems, ladění programů a zpracování textu

František HAVLŮJ

*e-mail: haf@ujv.cz*

ÚJV Řež

oddělení Reaktorové fyziky a podpory palivového cyklu

akademický rok 2016/2017, 19. října 2016

- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler
  - Vytvoříme excelovskou tabulku
- 3 Kde je chyba?
- 4 Dodělavka z minula: jehla v kupce sena
- 5 Zpracování textu
  - Obecný rozbor
  - Načítání výstupního souboru
  - Zápis všech výsledků do tabulky
- 6 Automatizace tvorby vstupů
  - Co dál?

Co jsme se naučili minule

Rozšíření Ruby: RubyGems a Bundler

Kde je chyba?

Dodělavka z minula: jehla v kupce sena

Zpracování textu

Automatizace tvorby vstupů

# Obsah

- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler
- 3 Kde je chyba?
- 4 Dodělavka z minula: jehla v kupce sena
- 5 Zpracování textu
- 6 Automatizace tvorby vstupů

- základy jazyka Ruby na všechny způsoby
- vstup a výstup na terminál i do souboru
- první kroky ve zpracování dat (čtení ze souboru, zpracování CSV tabulky)

# Obsah

- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler**
  - Vytvoříme excelovskou tabulku
- 3 Kde je chyba?
- 4 Dodělavka z minula: jehla v kupce sena
- 5 Zpracování textu
- 6 Automatizace tvorby vstupů

## 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](https://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 a součet
- na jednom součet bude jako číslo (sečte to váš skript)
- na druhém bude součet jako excelovský vzorec

# Obsah

- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler
- 3 Kde je chyba?**
- 4 Dodělovka z minula: jehla v kupce sena
- 5 Zpracování textu
- 6 Automatizace tvorby vstupů

## Ladění programů

- v každém programu je aspoň jedna chyba
- není důležité nedělat chyby, ale je nutné je umět najít
- když si program/Ruby na něco stěžuje, tak si to přečtěte, jinak se nic nedozvíte
- pokud nepoznám, v čem je chyba, jsem bezezbytku ztracen
- následují tři úlohy, kde je úkolem najít všechny chyby
- z didaktických důvodů postupujte metodou tupého spouštění a postupného opravování

## Úloha 1

```
# vypsat nasobilku (na kazdem radku deset soucinu)

(1..10).each do |i|
  (1..10).each do |j|
    puts "#{i} * #{j} = #{i*j} "
  end
  puts
end
```

## Úloha 2

```
# vypsat soucet vseh cisel z prvnioho sloupce souboru "data.csv"

File.foreach("data.csv") do |row|
  ary = line.strip.split
  y = ary[1]
  x += y
end
puts x
```

## Úloha 3

```
# v souboru "data.txt" najde nejmensi a nejvetsi cislo

File.readlines("data.txt").each do |line|
  ary = line.strip.split
  ary.each do |x|
    if x > max
      max = x
    end
    if x < min
      min = x
    end
  end
end
puts "Minimum = #{min}"
puts "Maximum = #{max}"
```



# Obsah

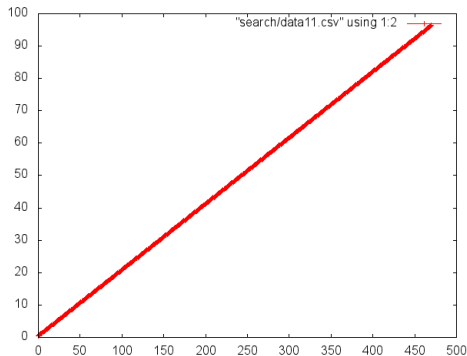
- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler
- 3 Kde je chyba?
- 4 Dodělovka z minula: jehla v kupce sena**
- 5 Zpracování textu
- 6 Automatizace tvorby vstupů

## Zadání

### # 1

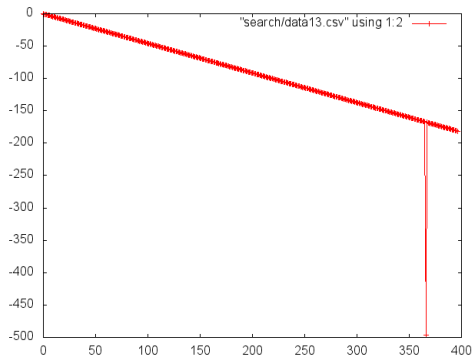
Adresář plný CSV souborů (stovky souborů) obsahuje data, která jsou záznamy signálů s lineární závislostí. V pěti z nich jsou ale poruchy - data ležící zcela mimo přímku. Kde?

## Příklad - dobrý signál



Co jsme se naučili minule  
Rozšíření Ruby: RubyGems a Bundler  
Kde je chyba?  
Dodělování z minula: jehla v kupce sena  
Zpracování textu  
Automatizace tvorby vstupů

## Příklad - špatný signál



## Řešení

- stačí vykreslit grafy pro všechny
- `Dir` pro najít souborů
- připravit a spustit `gnuplot`
- kouknu a vidím

## Řešení 2

- trocha matematiky po inženýrsku
- odečtu vhodnou lineární funkci
- podívám se na rozdíl mezi minimem a maximem

nebo

- sleduju rozdíl dvou po sobě jdoucích hodnot
- pokud se mi  $\text{sgn}(dx)$  změní, tak je jasno

# Obsah

- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler
- 3 Kde je chyba?
- 4 Dodělovka z minula: jehla v kupce sena
- 5 Zpracování textu**
  - Obecný rozbor
  - Načítání výstupního souboru
  - Zápis všech výsledků do tabulky

Co jsme se naučili minule  
Rozšíření Ruby: RubyGems a Bundler  
Kde je chyba?  
Dodělavka z minula: jehla v kupce sena  
**Zpracování textu**  
Automatizace tvorby vstupů

## Obecný rozbor

Načítání výstupního souboru  
Zápis všech výsledků do tabulky

# 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

Co jsme se naučili minule  
Rozšíření Ruby: RubyGems a Bundler  
Kde je chyba?  
Dodělavka z minula: jehla v kupce sena  
**Zpracování textu**  
Automatizace tvorby vstupů

## Obecný rozbor

Načítání výstupního souboru  
Zápis všech výsledků do tabulky

# Pracovní postup

- 1 načíst keff z výstupního souboru MCNP

## Pracovní postup

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

Co jsme se naučili minule  
Rozšíření Ruby: RubyGems a Bundler  
Kde je chyba?  
Dodělavka z minula: jehla v kupce sena  
**Zpracování textu**  
Automatizace tvorby vstupů

## Obecný rozbor

Načítání výstupního souboru  
Zápis všech výsledků do tabulky

# 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 confidence
-----
|
| the final estimated combined collision/absorption/track-length keff = 1.00353 with an estimate
|
| 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 estimate
.....
```

Co jsme se naučili minule  
Rozšíření Ruby: RubyGems a Bundler  
Kde je chyba?  
Dodělavka z minula: jehla v kupce sena  
**Zpracování textu**  
Automatizace tvorby vstupů

Obecný rozbor

**Načítání výstupního souboru**

Zápis všech výsledků do tabulky

# 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("cl_1o") do |line|

end

puts keff
```

## Realizace (2/5)

```
keff = nil

File.foreach("cl_1o") do |line|
  if line.include?("final estimated combined")

    end
end

puts keff
```

## Realizace (3/5)

```
keff = nil

File.foreach("cl_1o") do |line|
  if line.include?("final estimated combined")
    a = line.split("=")

    end
end

puts keff
```

## Realizace (4/5)

```
keff = nil

File.foreach("cl_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("cl_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.

## Chytrá horákyně

... by jistě vyrobila toto:

```
0 0 0.94800
0 10 0.99800
0 1 0.94850
0 2 0.95000
0 3 0.95250
0 4 0.95600
...
```

Nápovědou je funkce `split` (podle podtržítka) a funkce `to_i` (co asi dělá?)

## Realizace chytré horákyně

```
Dir["outputs/*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  
end  
  
s = filename.split("_")  
puts "#{s[1].to_i} #{s[2].to_i} #{keff}"  
end
```

# Obsah

- 1 Co jsme se naučili minule
- 2 Rozšíření Ruby: RubyGems a Bundler
- 3 Kde je chyba?
- 4 Dodělovka z minula: jehla v kupce sena
- 5 Zpracování textu
- 6 Automatizace tvorby vstupů**
  - Co dál?

## Určení poloh tyčí

Ve vstupním souboru si najdeme relevantní část:

```
c -----  
c polohy tyci (z-plochy)  
c -----  
c  
67 pz 47.6000      $ dolni hranice absoberu r1  
68 pz 40.4980      $ dolni hranice hlavice r1  
69 pz 44.8000      $ dolni hranice absoberu r2  
70 pz 37.6980      $ dolni hranice hlavice r2
```

## Výroba šablon

Jak dostat polohy tyčí do vstupního souboru? Vytvoříme šablonu, tzn nahradíme

```
67 pz 47.6000      $ dolni hranice absoberu r1
```



## Výroba šablon

Jak dostat polohy tyčí do vstupního souboru? Vytvoříme šablonu, tzn nahradíme

```
67 pz 47.6000      $ dolni hranice absoberu r1
```

nějakou značkou (*placeholder*):

```
67 pz %r1%      $ dolni hranice absoberu r1
```

## Chytáky a zádrhele

- kromě samotné plochy konce absorbéru je nutno správně umístit i z-plochu konce hlavice o 7,102 cm níže
- obecně je na místě ohlídat si, že placeholder nebude kolidovat s ničím jiným

Doporučené nástroje jsou:

- již známá funkce `sub` pro nahrazení jednoho řetězce jiným
- pro pragmatické lenochy funkce `File.read` načítající celý soubor do řetězce (na což nelze v Pascalu ani pomyslet)
- možno ovšem použít i `File.readlines` (v čem je to lepší?)

## Realizace

```
DELTA = 44.8000 - 37.6980

template = File.read("template")
(0..10).each do |i1|
  (0..10).each do |i2|
    r1 = i1 * 50
    r2 = i2 * 50
    s = template.sub("%r1%", r1.to_s)
    s = s.sub("%r1_", (r1 - DELTA).to_s)
    s = s.sub("%r2%", r2.to_s)
    s = s.sub("%r2_", (r2 - DELTA).to_s)
    File.write("inputs/c_#{i1}_#{i2}", s)
  end
end
```

## 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)

Co jsme se naučili minule  
Rozšíření Ruby: RubyGems a Bundler  
Kde je chyba?  
Dodělávká z minula: jehla v kupce sena  
Zpracování textu  
Automatizace tvorby vstupů

Co dál?

A to je vše, přátelé!

