Informatika pro moderní fyziky (3)
ladění programů a zpracování textu

František HAVLŮJ

e-mail: haf@ujv.cz

ÚJV Řež
oddělení Reaktorové fyziky a podpory palivového cyklu

akademický rok 2014/2015
13. října 2015

Obsah

- 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 - ještě musíme dodělat)

Obsah

Úlohy

Z dat v souboru `data/data_two_1.csv` (sloupce t, x1, x3):

- vyberte pouze druhý sloupec (t, x2)
- sečtěte oba sloupce do jednoho (t, x1 + x2)
- vypočtěte součet obou sloupců (suma x1, suma x2)
- vypočtěte průměr a RMS druhého sloupce (průměr x2, RMS x2)

S hvězdičkou:

- použijte soubory `*multi*` (t, x1 + x2 + ... + xn)
- proveďte pro všechny čtyři CSV soubory

Úlohy

CSV(1) / řešení

```
File.open("druhy_sloupec.csv", 'w') do |f|
  IO.foreach("../data/data_two_1.csv") do |line|
    data = line.strip.split
    f.puts "#{data[0]} #{data[2]}"
  end
end
```

Úlohy

CSV(2) / řešení

```
File.open("sectene_sloupce.csv", 'w') do |f|
  IO.foreach("../data/data_two_1.csv") do |line|
    data = line.strip.split
    f.puts "#{data[0]} #{data[1].to_f + data[2].to_f}"
  end
end
```

Úlohy

CSV(3) / řešení

```
x1, x2 = 0.0, 0.0
n = 0
IO.foreach("../data/data_two_1.csv") do |line|
  data = line.strip.split
  x1 += data[1].to_f
  x2 += data[2].to_f
  n += 1
end
puts "Prvni sloupec: soucet #{x0}"
puts "Druhy sloupec: soucet #{x1}"
```

Úlohy

CSV(4) / řešení

```
...
a1 = x1 / n
a2 = x2 / n

rms1, rms2 = 0.0, 0.0
IO.foreach("../data/data_two_1.csv") do |line|
  data = line.strip.split
  rms1 += (data[1].to_f - a1) ** 2
  rms2 += (data[2].to_f - a2) ** 2
  n += 1
end
rms1 = (rms1 / n) ** 0.5
rms2 = (rms2 / n) ** 0.5
puts "Prvni sloupec: RMS #{rms1}"
puts "Druhy sloupec: RMS #{rms2}"
```

Obsah

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 |i|
    puts "#{i} * #{j} = #{i*j} "
  end
  puts
end
```

Úloha 2

```
# vypsat soucet vsech cisel z prvniho sloupce souboru 'data.csv'

IO.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

IO.read_lines("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}"
```

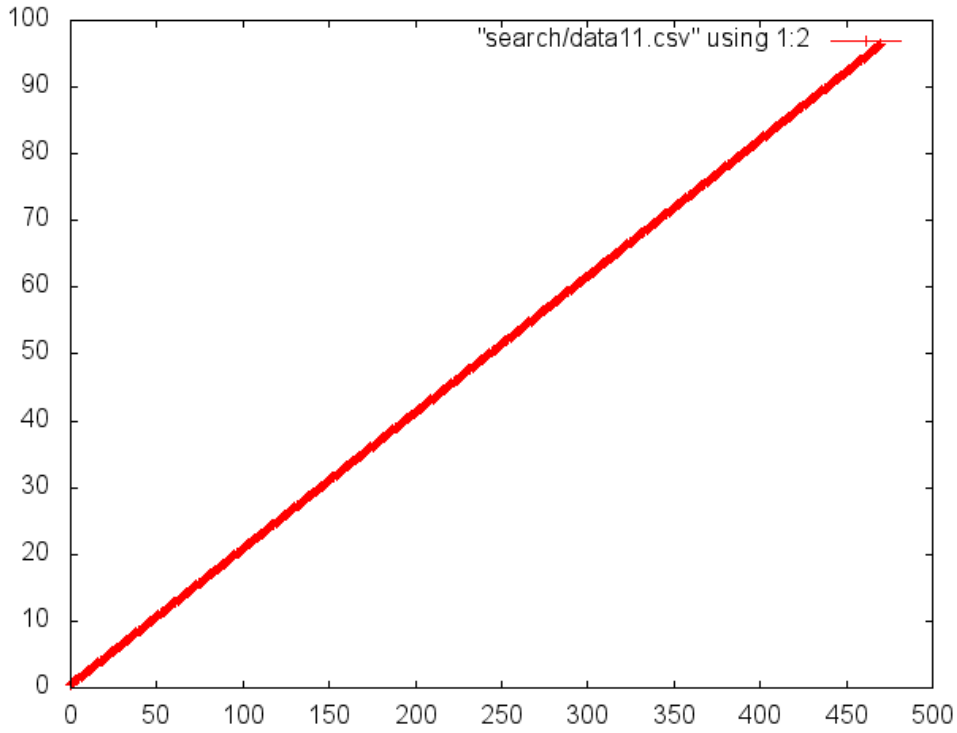
Obsah

Zadání

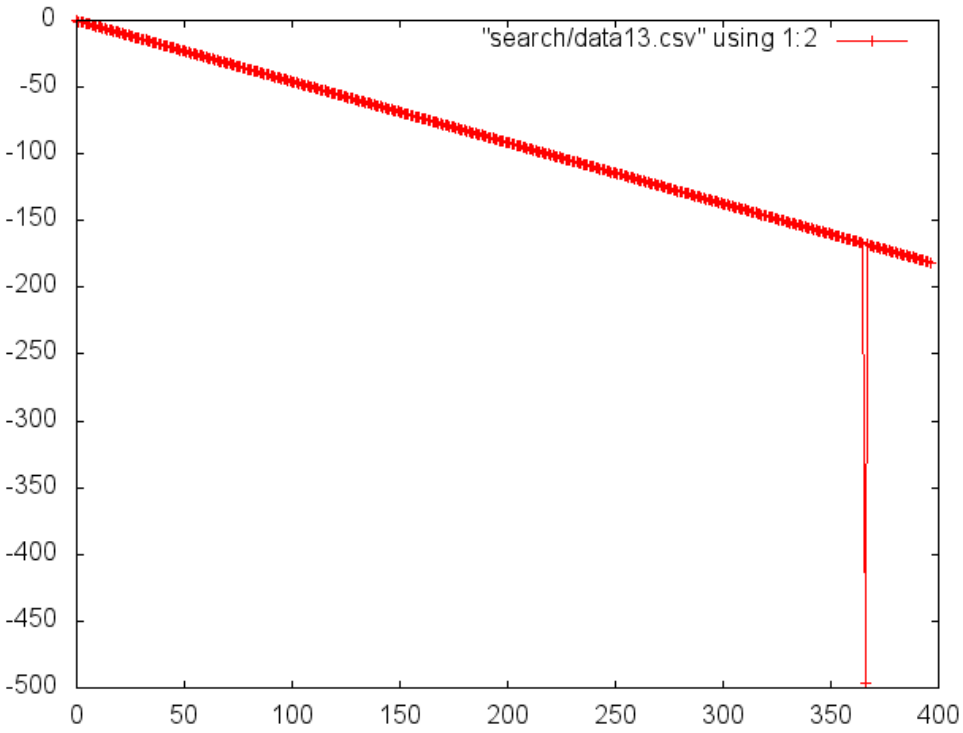
1

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



Příklad - špatný signál



Řešení

- stačí vykreslit grafy pro všechny
- `Dir` pro najití 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 sgn(dx) změní, tak je jasno

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

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
5. buď zpracovat ručně (Excel), nebo být Myšpulín a vyrobit skript (úkol s hvězdičkou)

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

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

end

puts keff
```

Realizace (2/5)

```
keff = nil

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

  end
end

puts keff
```

Realizace (3/5)

```
keff = nil

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

  end
end

puts keff
```

Realizace (4/5)

```
keff = nil

IO.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

IO.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
```

Obsah

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? Vyrobíme šablonu, tzn nahradíme

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

Výroba šablon

Jak dostat polohy tyčí do vstupního souboru? Vyrobíme 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 `IO.read` načítající celý soubor do řetězce (na což nelze v Pascalu ani pomyslet)
- možno ovšem použít i `IO.readlines` (v čem je to lepší?)

Realizace

```
DELTA = 44.8000 - 37.6980

template = IO.read('template')
(0..10).each do |i1|
  (0..10).each do |i2|
    r1 = i1 * 50
    r2 = i2 * 50
    File.open("inputs/c_#{i1}_#{i2}", "w") do |f|
      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)
      f.puts template
    end
  end
end
```

Jak na to

Máme všechno, co potřebujeme:

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

Takže už to stačí jen vhodným způsobem spojit dohromady!

Realizace

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

  IO.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

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

  s = filename.split("_")
  puts "#{s[1].to_i} #{s[2].to_i} #{keff}"
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)

A to je vše, přátelé!