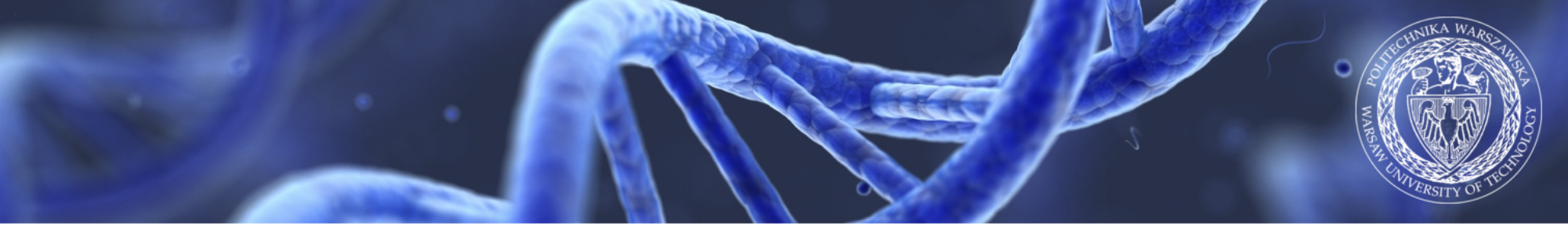
Scala(-ble) R

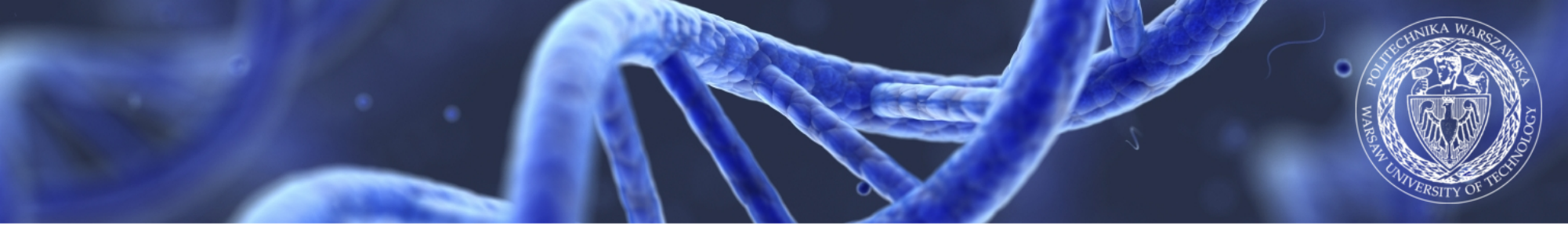
. . . czyli integracja środowiska R z językiem Scala za pomocą jvmr

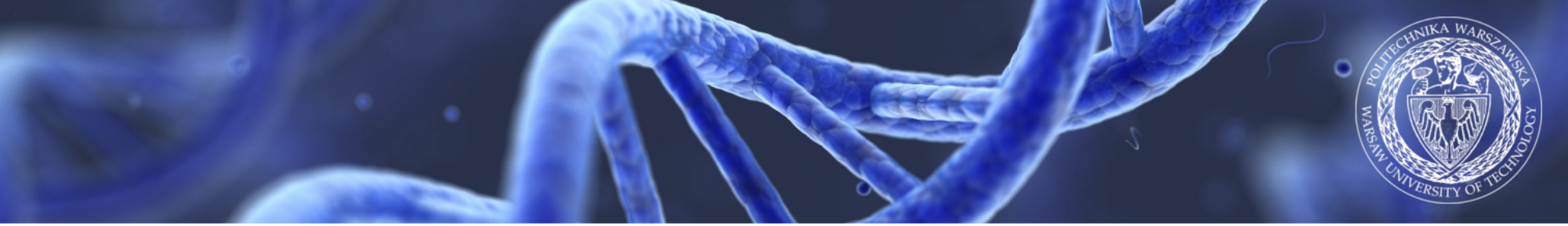
Marek Wiewiórka

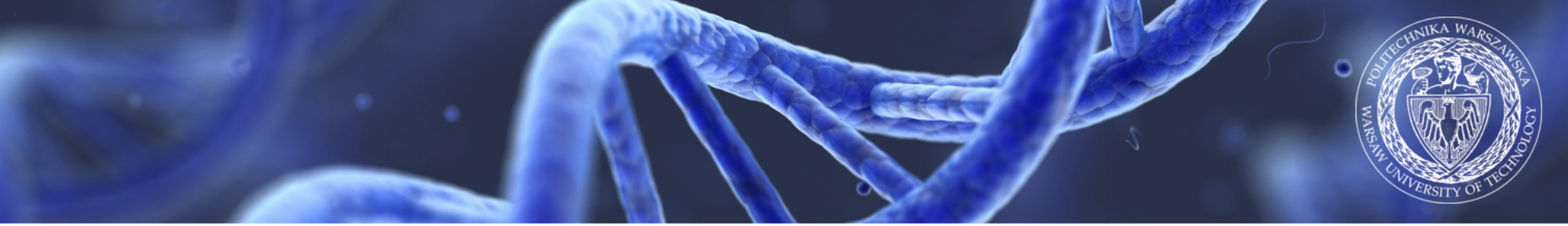
Instytut Informatyki,
Politechnika Warszawska

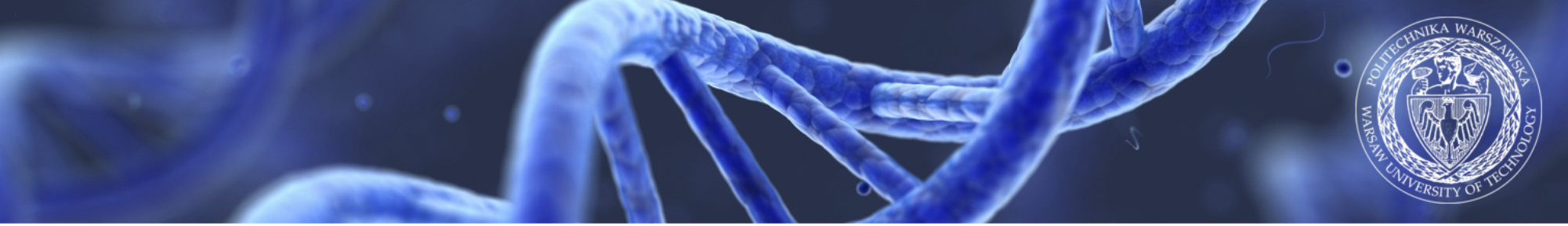
17 października 2014

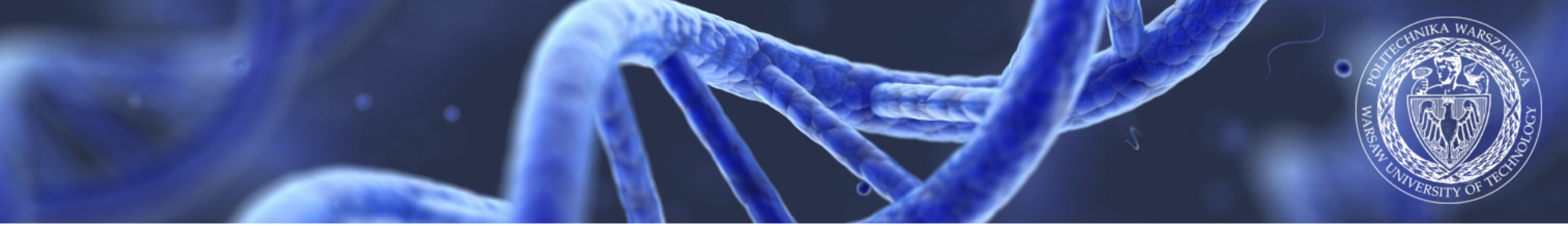












Plan prezentacji

Plan prezentacji

Scala

Scala

Język programowania łączący cechy języków funkcyjnych i obiektowych działający na maszynie wirtualnej Javy.

Scala

Scala

Język programowania łączący cechy języków funkcyjnych i obiektowych działający na maszynie wirtualnej Javy.

Scala

Stworzony przez Martin Odersky'go i upubliczniony w 2004 roku.

Scala

Scala

Język programowania łączący cechy języków funkcyjnych i obiektowych działający na maszynie wirtualnej Javy.

Scala

Stworzony przez Martin Odersky'go i upubliczniony w 2004 roku.

Scala

Często wykorzystywany jako tzw. język dziedzinowy (ang. *domain-specific language*).

Dlaczego Scala . . .

Dlaczego Scala . . .

Dlaczego Scala . . .

Dlaczego Scala . . .

Inne frameworki . . .

- BIDMat - bardzo szybka biblioteka macierzowa wykorzystująca Intel Matrix Kernel Library oraz JCUDA(`https://github.com/BIDData/BIDMat`);
- BIDMach - biblioteka do eksploracji danych korzystająca z BIDMat (`https://github.com/BIDData/BIDMach`);
- SparkSeq - biblioteka do analiz danych z sekwencjonowania nowej generacji (`https://bitbucket.org/mwiewiorka/sparkseq/`).

Plan prezentacji

Instalacja jvmr

- pakiet jvmr[1] instalujemy standardowo z repozytorium CRAN za pomocą komendy: `install.packages("jvmr")`
- pakiet jvmr wymaga poprawnie zainstalowanego i skonfigurowanego pakietu rJava;
- w szczególności trzeba zwrócić uwagę na poprawnie ustawione zmienne:

```
export JAVA="/usr/lib/jvm/java-7-oracle/jre/bin/java"
export JAVAC="/usr/lib/jvm/java-7-oracle/bin/javac"
export JAVAH="/usr/lib/jvm/java-7-oracle/bin/javah"
export JAVA_CPPFLAGS="-I/usr/lib/jvm/java-7-oracle/include -I/usr/lib/jvm/java-7-oracle/include/linux"
export JAVA_HOME="/usr/lib/jvm/java-7-oracle"
export JAVA_LD_LIBRARY_PATH="/usr/lib/jvm/java-7-oracle/jre/lib/amd64/server"
export JAVA_LIBS="-L/usr/lib/jvm/java-7-oracle/jre/lib/amd64/server -ljvm"
export LD_LIBRARY_PATH=$JAVA_LD_LIBRARY_PATH
```

- w celu ustawienia zmiennych należy wykonać komendę `R CMD javareconf -e`

[1] `http://cran.r-project.org/web/packages/jvmr/index.html`

Przykładowa sesja z jvmr 1/2

```
library(jvmr)
#utworzenie dowiazania do interpretera Scali
scala<-scalaInterpreter(unlist(strsplit(Sys.getenv("PAZUR_CLASSPATH"),":")))

interpret(scala, '
import cern.jet.random.engine.MersenneTwister
import cern.jet.random.Normal
val mt = new MersenneTwister()
val randN = new Normal(0,1,mt)',eval.only=TRUE)
system.time(interpret(scala,'for(i<-1 to 100000000) randN.nextDouble()',eval.only=TRUE))
  user  system elapsed
 4.591   0.017   4.320
system.time(rnorm(100000000,0, 1))
  user  system elapsed
 7.359   0.031   7.386
```

```
marek@marekse:~/Phd/pazur2014_code$ echo $PAZUR_CLASSPATH | sed -e 's/:/\n/g'
/home/marek/Phd/pazur2014_code/jars/concurrent.jar
/home/marek/Phd/pazur2014_code/jars/commons-math3-3.3.jar
/home/marek/Phd/pazur2014_code/jars/colt.jar
/home/marek/Phd/pazur2014_code/jars/Thyme.jar
/home/marek/Phd/pazur2014_code/jars/SNVerPool.jar
```

Przykładowa sesja z jvmr 2/2

Narzut uruchomieniowy oszacowany za pomocą pakietu rbenchmark[2](R) oraz biblioteki Thyme[3](Scala):

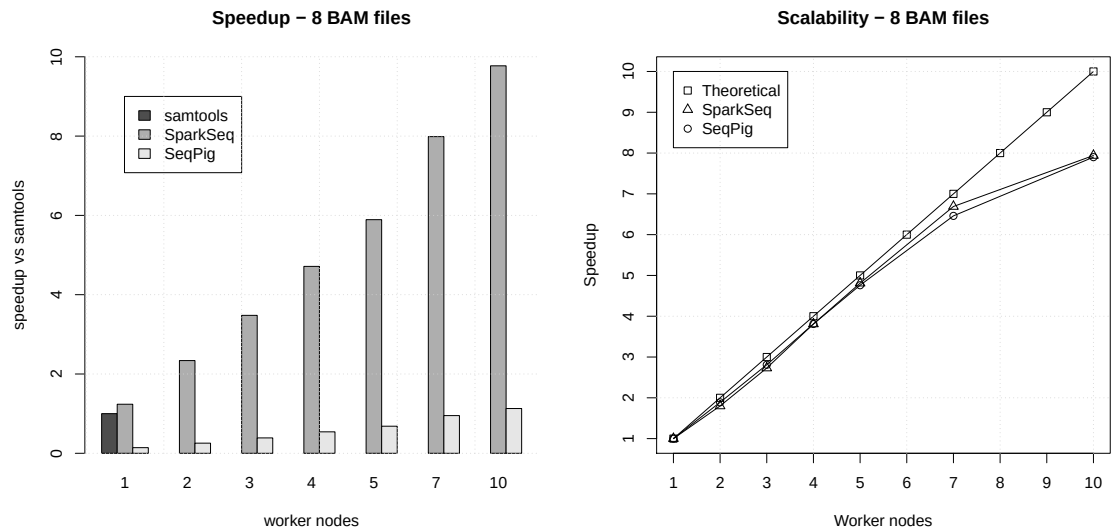
```
interpret(scala,'val th = new ichi.bench.Thyme;
  th.pbench(for(i<-1 to 100000000) randN.nextDouble())',echo.output=TRUE)
Benchmark (20 calls in 84.78 s)
  Time:    4.179 s   95% CI 4.163 s - 4.194 s   (n=20)
  Garbage: 60.20 ms   (n=474 sweeps measured)


library(rbenchmark)
benchmark(interpret(scala,'for(i<-1 to 100000000) randN.nextDouble()',eval.only=TRUE,echo.output=TRUE)
,replications=20)
  replications elapsed relative user.self sys.self user.child sys.child
  20           86.934        1     92.33    0.341          0         0

> 4.3467-4.179
[1] 0.1677
```

[2] http://cran.r-project.org/web/packages/rbenchmark/index.html

[3] https://github.com/Ichoran/thyme

R i SparkSeq = RSparkSeq[4] 1/2

SparkSeq to narzędzie do analiz danych z sekwencjonowania nowej generacji DNA/RNA z nukleotydową precyzją w chmurach obliczeniowych rozwijane przez grupę ZSI-Bio przy wykorzystaniu frameworku obliczeniowego Apache Spark.



Rysunek : Przyspieszenie (A) and skalowalność (B) narzędzi SparkSeq i SeqPig dla 8 plików BAM na zbiorze danych RNA-seq (w sumie 9 GB)

[4] `https://bitbucket.org/mwiewiorka/rsparkseq/wiki/Home`

R i SparkSeq = RSparkSeq 2/2

```
rcont<-RSparkContext(master="spark://spark001.net:7070",",
    sparkJar="sparkseq-core-assembly-0.1-SNAPSHOT.jar",debug=TRUE)

seqAnalysis<-RSparkSeqAnalysis(rcont,
dfs://sparkseq002.cloudapp.net:9000/test/F1_Y.bam",1,1,1,debug=TRUE)
regionHashMap(seqAnalysis) <-
    "hdfs://sparkseq002.cloudapp.net:9000/aux/Homo_sapiens.GRCh37.74_genes_chr_merged.bed"

samplesID<-c(11, 32, 3, 42, 43, 47, 53, 58)
for(i in samplesID)
  addBAMFile(seqAnalysis,c(paste("hdfs://spark002.net:9000/test/F",as.character(i) ,"_Y.bam",sep=""),i) )

genes<-geneCounts(seqAnalysis)
> genes[1:5,]
  Feature Sample_1 Sample_3 Sample_11 Sample_32 Sample_42 Sample_43 Sample_47 Sample_53 Sample_58
1 ENSG00000012817     8720     9103      8585      7375      5387      5835      7669      8175      7258
2 ENSG00000067048     6321     8647      5325      5598      3468      4140      4376      6827      4335
3 ENSG00000067646     1354     1499      1691       889       977       558       872      1128      1004
4 ENSG00000092377       75       48        31        61        29        10        31        26        42
5 ENSG00000099715      291      418       158       229        42       128       125       145       303
```

Plan prezentacji

Prosta usługa sieciowa REST z R w Play!


```
import org.ddahl.jvmr.RInScala

case class TTest(stat: Double, pvalue: Double)
def tTest(vec1:List[Double],vec2:List[Double]) = Action{
  val R = RInScala()
  R.vec1=vec1.toArray
  R.vec2=vec2.toArray
  R.eval("test<-t.test(vec1,vec2)")
  val Pvalue = R.toPrimitive[Double]("test$p.value")
  val stats = R.toPrimitive[Double]("test$statistic")
  val tTest = new TTest(stats,Pvalue)
  val json = Json.toJson(tTest)
  Ok(json)
}
```

```
GET     /tTest              controllers.Application.tTest(vec1:List[Double],vec2:List[Double])
```

```
marek@marekse:~$ curl "http://localhost:9000/tTest?vec1=1.0&vec1=2.0&vec2=1.1&vec2=0.9" -w"\n"
{"stats":0.9805806756909201,"pvalue":0.4963960631272696}
```

Plan prezentacji

Przykładowy problem - analiza danych metylacji DNA

Metylacja

to reakcja biochemiczna, która polega na przyłączaniu grup metylowych do cytozyny w DNA. Proces ten zachodzi najczęściej, gdy cytozyna występuje obok guaniny, tworząc tzw. dwunukleotyd CpG.

Problem badawczy

Wyłonienie potencjalnych markerów stopnia metylacji DNA skorelowanych z wiekiem człowieka.

Zarys metody

Rozwiązanie sprowadza się do estymacji milionów modeli regresji liniowej dla wybranych miejsc metylacyjnych oraz oszacowania przedziałów ufności dla współczynników za pomocą bootstrapu parametrycznego.

Przykładowy problem - analiza danych metylacji DNA

Metylacja

to reakcja biochemiczna, która polega na przyłączaniu grup metylowych do cytozyny w DNA. Proces ten zachodzi najczęściej, gdy cytozyna występuje obok guaniny, tworząc tzw. dwunukleotyd CpG.

Problem badawczy

Wyłonienie potencjalnych markerów stopnia metylacji DNA skorelowanych z wiekiem człowieka.

Zarys metody

Rozwiązanie sprowadza się do estymacji milionów modeli regresji liniowej dla wybranych miejsc metylacyjnych oraz oszacowania przedziałów ufności dla współczynników za pomocą bootstrapu parametrycznego.

Przykładowy problem - analiza danych metylacji DNA

Metylacja

to reakcja biochemiczna, która polega na przyłączaniu grup metylowych do cytozyny w DNA. Proces ten zachodzi najczęściej, gdy cytozyna występuje obok guaniny, tworząc tzw. dwunukleotyd CpG.

Problem badawczy

Wyłonienie potencjalnych markerów stopnia metylacji DNA skorelowanych z wiekiem człowieka.

Zarys metody

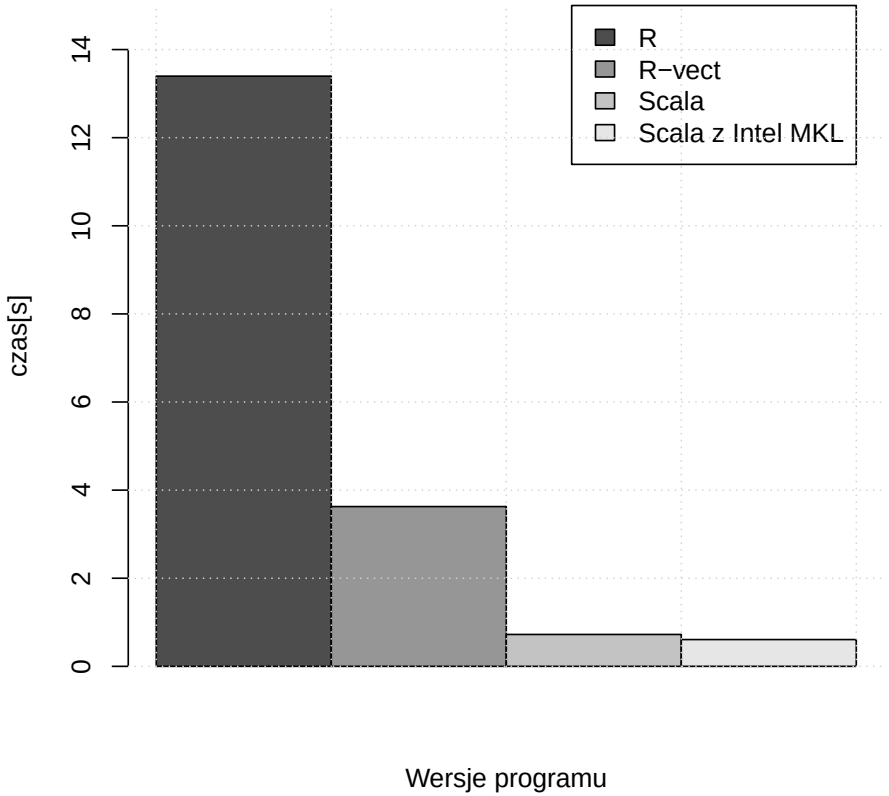
Rozwiązanie sprowadza się do estymacji milionów modeli regresji liniowej dla wybranych miejsc metylacyjnych oraz oszacowania przedziałów ufności dla współczynników za pomocą bootstrapu parametrycznego.

Czas wykonania operacji R vs Scala

funkcja	R[s]	Scala[s]	BIDMach[s]	Przyspieszenie
rbinom	0.181	0.131	0.031	5.03
rnorm	0.184	0.065	0.011	16.73
lm.fit	0.587	0.028	N/A	20.96
solve	0.209	0.004	0.012	52.25

Tabela : Czas wykonania operacji dla różnych bibliotek matematycznych ($k = 208 \times 10^4$), $n \times m = 2 \times 208$

Łączne uzyskane przyspieszenie przetwarzania...



Rysunek : Czas obliczeń dla 1 miejsca metylacyjnego

Plan prezentacji

Podsumowanie

Pakiet `jvmr` umożliwia dwustronną integrację środowiska R z językiem Scala.

. . . obliczenia w R mogą zyskać na wydajności, a język Scala dostęp do bogatych zbiorów metod statystycznych oraz wizualizacji danych.

. . . integracja R i Scali ma swój narzut uruchomieniowy, ale odpowiedni podział obliczeń może go w dużym stopniu zniwelować.

Podsumowanie

Pakiet `jvmr` umożliwia dwustronną integrację środowiska R z językiem Scala.

. . . obliczenia w R mogą zyskać na wydajności, a język Scala dostęp do bogatych zbiorów metod statystycznych oraz wizualizacji danych.

. . . integracja R i Scali ma swój narzut uruchomieniowy, ale odpowiedni podział obliczeń może go w dużym stopniu zniwelować.

Podsumowanie

Pakiet `jvmr` umożliwia dwustronną integrację środowiska R z językiem Scala.

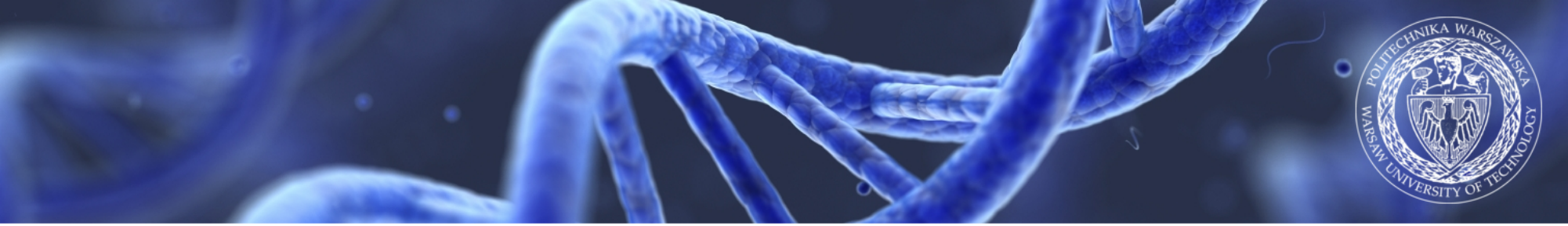
. . . obliczenia w R mogą zyskać na wydajności, a język Scala dostęp do bogatych zbiorów metod statystycznych oraz wizualizacji danych.

. . . integracja R i Scali ma swój narzut uruchomieniowy, ale odpowiedni podział obliczeń może go w dużym stopniu zniwelować.

Kody programów w R & Scali

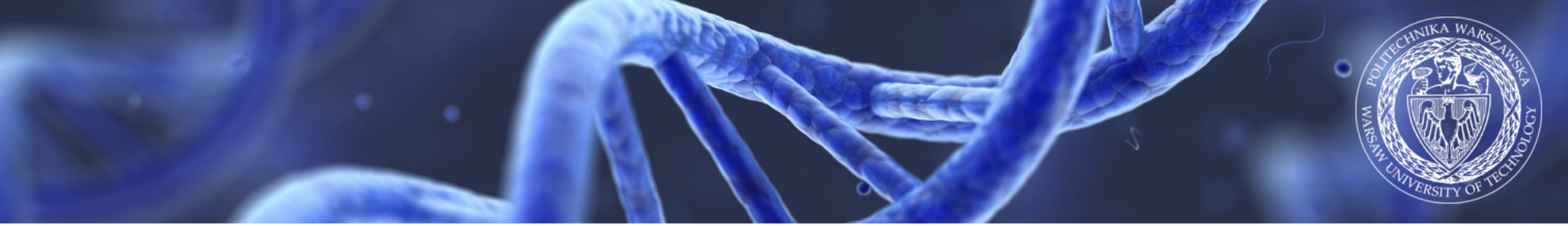
https://github.com/mwiewior/pazur2014

. . . btw join us!



Dziękuję za uwagę.
Marek Wiewiórka

marek.wiewiorka@gmail.com



Dziękuję za uwagę.
Marek Wiewiórka

marek.wiewiorka@gmail.com