Problem Set 6

Cameron Adams

November 1, 2017

1 For this question you will read a journal article ...

1.1 What are the goals of their simulation study and what are the metrics that they consider in assessing their method?

The authors were understand distribution of likelihood ratio for gaussian mixture models. They draw data from gaussian mixture distributions (iid draws) and get a 2LR statistical test. They are interested in power to detect deviations from the null hypothesis for the 2LR statistical test.

The goals of the simulation study are to investigate the finite sample properties of the test distinguish distributions in mixture distribution. The authors used differences in significance levels for differing values of D (difference in mean of normal distributions in mixture distribution), while varying sample size and mixture proportion.

1.2 What choices did the authors have to make in designing their simulation study? What are the key aspects of the data generating mechanism that likely affect the statistical power of the test? Are there data-generating scenarios that the authors did not consider that would be useful to consider?

As stated above, the authors, make choices about mixture coefficient, sample size, and nominal significance, different in means (D) between gaussian distributions, k=the number of component gaussian distributions, number of replications, and proportion of p and q. The mixture proportion and difference in means likely affects the statistical power of the test. The authors did not consider scenarios with normal distributions with different variances, only different norms. Seems likely there would be scenarios where there is a mixture of distributions

Each simulation is a draw of the sample (data generating process), on which the test statistic is calculated.

1.3 Do their tables do a good job of presenting the simulation results and do you have any alternative suggestions for how to do this?

The tables are confusing, I would like to see plots rather than tables. It would have been interesting to see how simulations perform across a distribution of mixture proportions, rather than a few discrete values (e.g, 0.5, 0.7, etc).

1.4 Interpret their tables on power (Tables 2 and 4) - do the results make sense in terms of how the power varies as a function of the data generating mechanism?

As sample size increases, 2LR should also be increasing, and you can reject a greater proportion of null hypothesis. However, as D increased, there was more power to reject the null. Power should increase and difference in means increases, and results indicate that is true. Table 4 is similar, but shows results for k=3 component distributions. Results

1.5 How do you think the authors decided to use 1000 simulations. Would 10 simulations be enough? How might we decide if 1000 simulations is enough?

They decided to do 1000 simulations because it would give good resolution for p-values (1/1000 = 0.001). Ten simulations would give us a resolution of 1/10=0.1. Convergence rate also plays a role in determining number of simulations. If computation is cheap, they doing very large amounts of simulations are trivial (e.g, n=10,000, n=1e6, etc.), and the opposite is true, if computation is "expensive", one wants to do smallest number of simulations that provide an adequate distribution of test statistics.

2 Using the Stack Overflow database ...

```
library(RSQLite)

###############
# load SQL database

#set SQL db driver
drv <- dbDriver("SQLite")

getwd()

## [1] "/Users/CamAdams/repos/STAT243/ps6"

#connec5 to db

dbFilename <- 'stackoverflow-2016.db'
db <- dbConnect(drv, dbname = file.path(dbFilename))
#dbDisconnect(db)



#drop tables not in original set
tbls <- c("answers", "maxRepByQuestion", "questions",
          "questionsAugment", "questions_tags", "users")
tbls_toDrop <- dbListTables(db)[!dbListTables(db) %in% tbls]

if (length(tbls_toDrop) > 0) {
    sapply(tbls_toDrop, function(X) {
        sql_cmd <- paste0("DROP VIEW if exists ", X)
        dbSendQuery(db, sql_cmd) #code to drop tables/views
        })
} else {cat("No user created tables")}

## Warning:  Closing open result set, pending rows
## Warning:  Closing open result set, pending rows

## $query_py
## <SQLiteResult>
## EXPIRED
##
## $query_r
## <SQLiteResult>
```

```
## EXPIRED
##
## $query_r_py
## <SQLiteResult>
##   SQL  DROP VIEW if exists query_r_py
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
#look at fields of needed tables table
dbListFields(db, "questions")
```

```
## Warning:  Closing open result set, pending rows
```

```
## [1] "questionid"   "creationdate" "score"        "viewcount"
## [5] "title"        "ownerid"
```

```
dbListFields(db, "questions_tags")
```

```
## [1] "questionid" "tag"
```

```
###############
# Find users who only asked R not python qestions

#create view tables for py
dbSendQuery(db, "CREATE VIEW query_r AS SELECT questions.questionid, ownerid, tag as tagR from questions
           on questions.questionid = questions_tags.questionid
           where tag = 'r'")
```

```
## <SQLiteResult>
##   SQL  CREATE VIEW query_r AS SELECT questions.questionid, ownerid, tag as tagR from questions join q
##              on questions.questionid = questions_tags.questionid
##              where tag = 'r'
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
# and r tag
dbSendQuery(db, "CREATE VIEW query_py AS SELECT questions.questionid, ownerid, tag as tagPy from questi
           on questions.questionid = questions_tags.questionid
           where tag = 'python'")
```

```
## Warning:  Closing open result set, pending rows
```

```
## <SQLiteResult>
##   SQL  CREATE VIEW query_py AS SELECT questions.questionid, ownerid, tag as tagPy from questions joi
##              on questions.questionid = questions_tags.questionid
##              where tag = 'python'
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
#check views
dbGetQuery(db, "select * from query_r limit 5")
```

```
## Warning:  Closing open result set, pending rows
```

```
##   questionid ownerid tagR
## 1   34553225  575952    r
## 2   34565336 5492392    r
## 3   34570389 5738949    r
## 4   34574110 4802680    r
## 5   34579747 3507767    r
```

```
dbGetQuery(db, "select COUNT(*) from query_r")
```

```
##   COUNT(*)
## 1    48079
```

```
dbGetQuery(db, "select * from query_py limit 5")
```

```
##   questionid ownerid  tagPy
## 1   34553559  845642 python
## 2   34556493 4458602 python
## 3   34557898 2927983 python
## 4   34560088 5736692 python
## 5   34560213 5636400 python
```

```
dbGetQuery(db, "select COUNT(*) from query_py")
```

```
##   COUNT(*)
## 1   171745
```

```
#left outer join r and py view tables
dbSendQuery(db, "CREATE VIEW query_r_py AS SELECT * from query_r left outer join query_py
            on query_r.ownerid = query_py.ownerid")
```

```
## <SQLiteResult>
##   SQL  CREATE VIEW query_r_py AS SELECT * from query_r left outer join query_py
##              on query_r.ownerid = query_py.ownerid
##   ROWS Fetched: 0 [complete]
##        Changed: 0
```

```
#dbGetQuery(db, "select * from query_r_py limit 5")
#dbGetQuery(db, "select COUNT(*) from query_r_py")

#remove tagPy = <NA> to get answer to question
r_not_py_user_count <- dbGetQuery(db, "SELECT COUNT (DISTINCT ownerid) from query_r_py
                WHERE tagPy IS NULL")
```

```
## Warning:  Closing open result set, pending rows
```

```
r_not_py_user_count
```

```
##   COUNT (DISTINCT ownerid)
## 1                    18611
```

```
#18611

#check answer in R
r <- dbGetQuery(db, "select * from query_r")
py <- dbGetQuery(db, "select * from query_py")

sum(r$ownerid %in% py$ownerid) #8884 users asked both matches
```

```
## [1] 8884
```

```
r_not_py_user_count == sum(!unique(r$ownerid) %in% py$ownerid)
```

```
##      COUNT (DISTINCT ownerid)
## [1,]                     TRUE
```

```
#THEY MATCH!!! :)
```

The answer is there are 18,611 stackoverflow users who asked R questions and no python questions.

3 With the full Wikipedia traffic data for October-December 2008 ...

Code to load pyspark from bash

```
#load pyspark
srun -A ic_stat243 -p savio2 --nodes=1 -t 1:00:00 --pty bash
module load java spark
source /global/home/groups/allhands/bin/spark_helper.sh
spark-start
## note the environment variables created
env | grep SPARK


# load PySpark using Python 2.7.8 (more packages available)
module load python/2.7.8 numpy
pyspark --master $SPARK_URL --executor-memory 60G \
        --conf "spark.executorEnv.PATH=${PATH}" \
        --conf "spark.executorEnv.LD_LIBRARY_PATH=${LD_LIBRARY_PATH}" \
        --conf "spark.executorEnv.PYTHONPATH=${PYTHONPATH}"
```

Python code for pyspark/mapReduce

```
#####
# python code to get world series by time

dir='/global/scratch/paciorek/wikistats_full/dated_for_R/'
lines=sc.textFile(dir)

### filter to sites containing "Christmas"
import re
from operator import add

def find(line, regex = "Christmas", language = None):
    vals = line.split(' ')
    if len(vals) < 6:
        return(False)
    tmp = re.search(regex, vals[3])
    if tmp is None or (language != None and vals[2] != language):
        return(False)
    else:
        return(True)

xmas=lines.filter(find).repartition(960)

xmas.count()

### map-reduce step to sum hits across date-time-language triplets

def stratify(line):
    # create key-value pairs where:
```

```
    #   key: date-time-language
    #   value: number of website hits
    vals=line.split(' ')
    return(vals[0]+'-'+vals[1]+'-'+vals[2],int(vals[4]))

## sum number of hits for each date-time-language value
counts=xmas.map(stratify).reduceByKey(add)

### map step to prepare output
def transform(vals):
    # split key info back into separate fields
    key=vals[0].split('-')
    return(",".join((key[0],key[1],key[2],str(vals[1]))))

### output to file
output=counts.map(transform).repartition(1).collect()
with open('/global/home/users/camadams/Christmas.txt', 'w') as txtFile:
  txtFile.write('\n'.join(output))
```

Bash code to get world sereis data onto my local computer

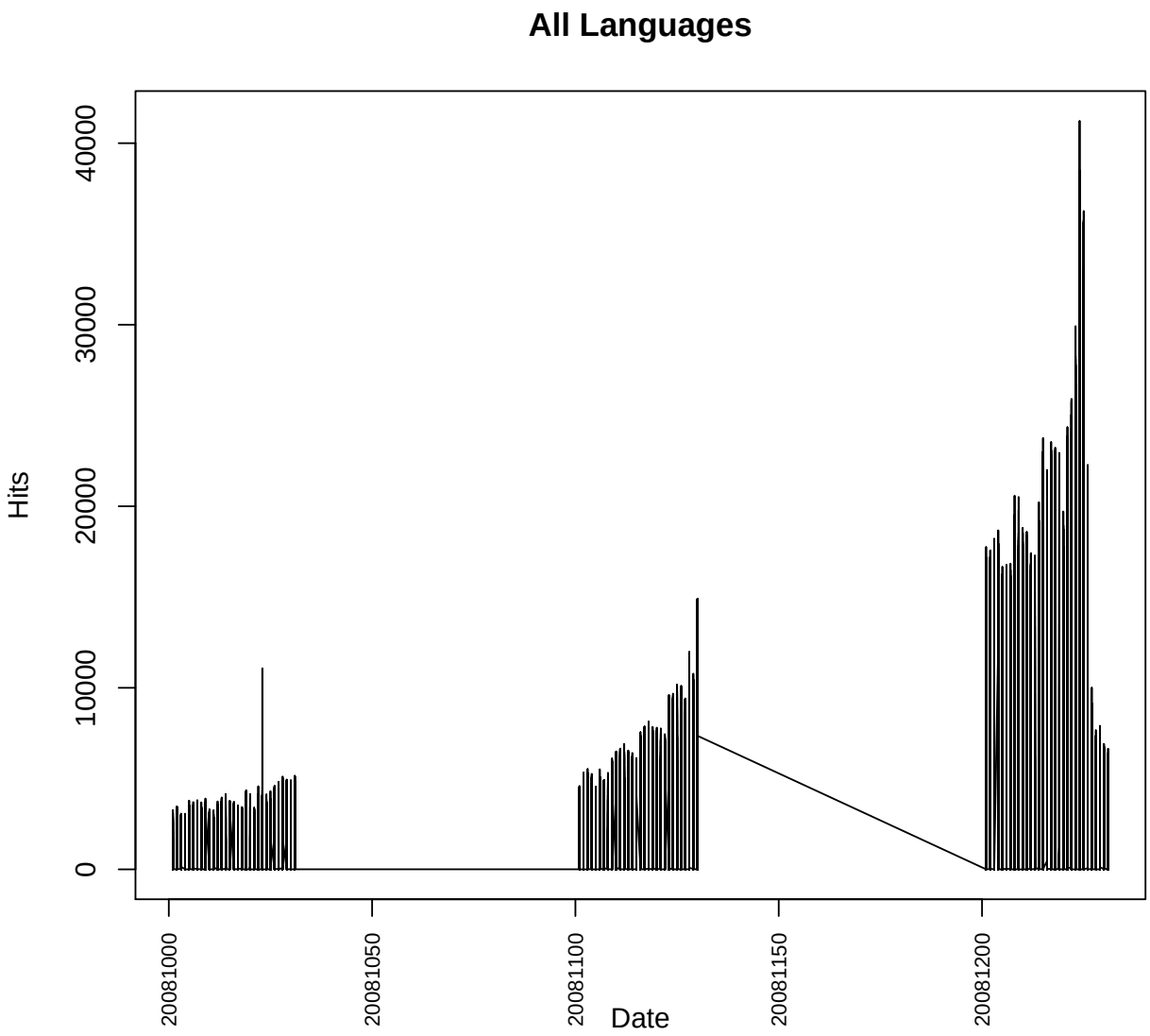
```
#bash code to get world sereis data onto my local computer
scp camadams@dtn.brc.berkeley.edu:/global/home/users/camadams/Christmas.txt \
    ~/Downloads/.
```

R code to plot christmas data from wikipedia.

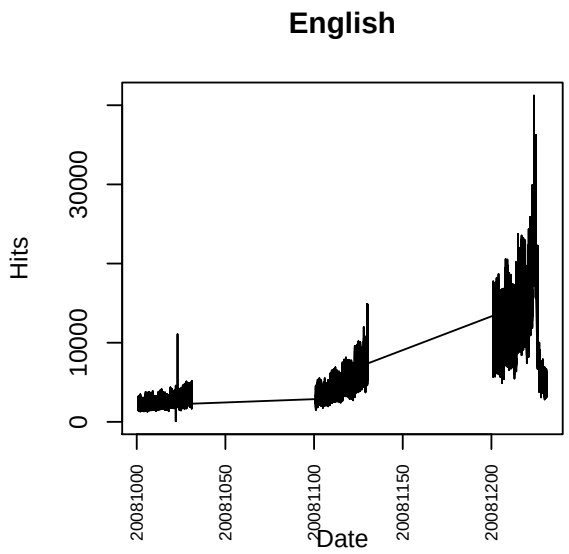
```
#R code to analyze world series data from wikipedia

#read in data
ws <- read.csv("./Christmas.txt", header = F)
dim(ws)

## [1] 88625     4

head(ws)

##         V1     V2   V3 V4
## 1 20081212  40000 th.s  9
## 2 20081120 210000   nl 33
## 3 20081125 120001   cs  3
## 4 20081008  80000   pt  6
## 5 20081112  10000   ru  3
## 6 20081026 120001   fr 21

#order data by date
ws <- ws[order(ws$V1), ]
```

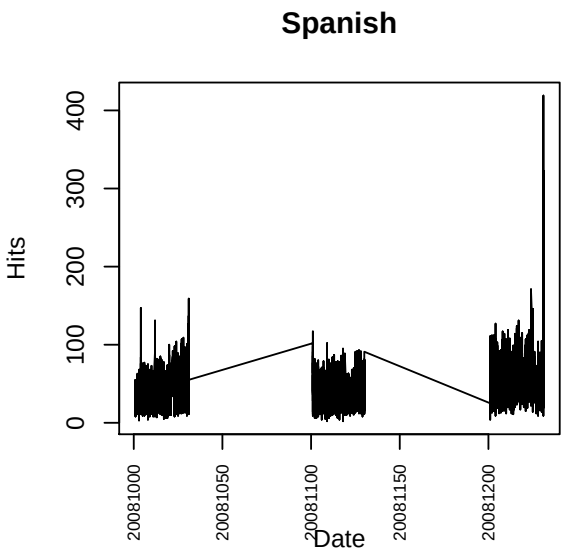
I am missing some of the data, not sure why, but I downloaded all hits for "Christmas" for the wikipedia data located in `/global/scratch/paciorek/wikistats_full/dated_for_R/`. In the plots you can see that hits for "Chrimstas" grow as time approaches Dec 25, and then sharply drop off towards the end of the year. This is broadly true for all languages and for English and Spanish (who both use daylight savings). Russia doesn't have daylight savings, and there are virtually no hits during the time period.

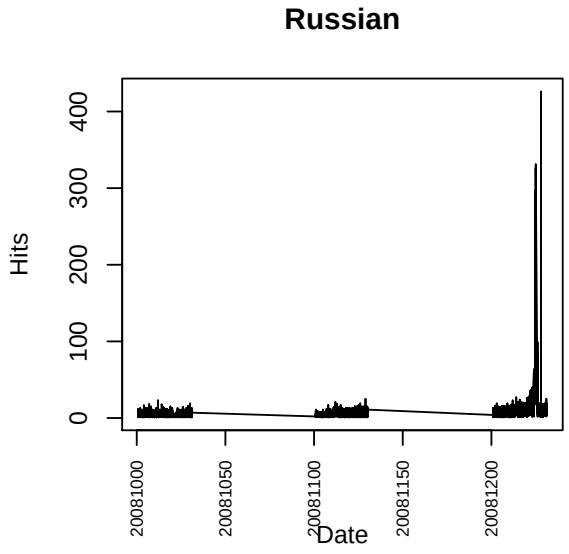
```r
#plot hits by date
#all languages
par(mfrow = c(1, 1), mar = c(7, 4, 4, 1))
plot(ws$V1, ws$V4, type = 'l',
     main = "All Languages", ylab = "Hits", xlab = "Date", xaxt = "n")
axis(1, las = 2, cex.axis = 0.75)
```



```r
#English, spanish, russian
par(mfrow = c(2, 2))
plot(ws$V1[ws$V3 == "en"], ws$V4[ws$V3 == "en"], type = 'l',
     main = "English", ylab = "Hits", xlab = "Date", xaxt = "n")
axis(1, las = 2, cex.axis = 0.75)
plot(ws$V1[ws$V3 == "es"], ws$V4[ws$V3 == "es"], type = 'l',
     main = "Spanish", ylab = "Hits", xlab = "Date", xaxt = "n")
axis(1, las = 2, cex.axis = 0.75)
plot(ws$V1[ws$V3 == "ru"], ws$V4[ws$V3 == "ru"], type = 'l',
     main = "Russian", ylab = "Hits", xlab = "Date", xaxt = "n")
axis(1, las = 2, cex.axis = 0.75)
```







4 This question asks you to complete the exercise begun in ...

4.1 Using either foreach or parSapply ...

Bash code to srun into savio and load r modules

```
# Code to login savio node and load needed R modules
srun -A ic_stat243 -p savio2 --nodes=1 -t 1:00:00 --pty bash
env | grep SLURM  ## see what environment variables are set by SLURM

module load r/3.2.5 doParallel/1.0.10 ggplot2/2.1.0 RColorBrewer/1.1-2 stringr/1.0.0 plyr/1.8.3 dplyr/0

#R CMD BATCH --no-save BO_matches.R BO_matches.Rout
    #I couldn't get R CMD BATCH To work, used R interactively
```

R script used to answer quesiton 4a) is below.

```
####
# R script

rm(list=ls())

#load packages
require(readr)
#Loading required package: readr

#get files in directory
dir <- "/global/scratch/paciorek/wikistats_full/dated_for_R/"
files <- list.files(dir)

#remove uneeded files
files <- files[grepl("part", files)]

#set file paths
filePaths <- paste0(dir, files)

head(filePaths)
#[1] "/global/scratch/paciorek/wikistats_full/dated_for_R/part-00000"
#[2] "/global/scratch/paciorek/wikistats_full/dated_for_R/part-00001"
#[3] "/global/scratch/paciorek/wikistats_full/dated_for_R/part-00002"
#[4] "/global/scratch/paciorek/wikistats_full/dated_for_R/part-00003"
#[5] "/global/scratch/paciorek/wikistats_full/dated_for_R/part-00004"
#[6] "/global/scratch/paciorek/wikistats_full/dated_for_R/part-00005"

#set read_delim progress bar options
options(readr.show_progress = F)

#get num cores
nCores <- as.integer(Sys.getenv("SLURM_CPUS_ON_NODE"))
nCores
#[1] 24

require(parallel)
```

```
require(doParallel)
require(foreach)

#initialize cores
registerDoParallel(nCores)

#set iterations
nSub <- length(filePaths)
nSub
#[1] 960

system.time(
result <- foreach(i = 1:nSub,
                  .packages = c("readr"),       # libraries to load onto each worker
                  .combine = rbind,             # how to combine results
                  .errorhandling=c("pass"),
                  .verbose = TRUE) %dopar% {

    dat <- readr::read_delim(filePaths[i], delim = " ", col_names = F)

    cat("########  iteration ", i, " is complete!! :) ########")

    dat[grep("Barack_Obama", dat$X4, fixed = T), ]
                  }
)
# user    system   elapsed
#24082.902 72954.630  4497.273


#check data
dim(result)
#[1] 430160      6

head(result)
# A tibble: 6 x 6
#        X1     X2    X3
#     <int>  <chr> <chr>
#1 20081129 210000    pt
#2 20081014 190000    en
#3 20081108 190000    no
#4 20081128 190001    en
#5 20081110 160000    et
#6 20081101 110000    fr
# ... with 3 more variables: X4 <chr>, X5 <int>, X6 <dbl>



write.csv(result, "/global/home/users/camadams/ps6_BOmatches.csv",
          row.names = F, quote = F)
```

It took 4497.273 sec $\sim$ 75 min of "clock" time to run the Barack Obama script on 960 files with 1 node/24cores.

4.2 When I run the Spark code provided with Unit 8, it takes 15 minutes using 9...

We can approximate the amount of real time by taking the total kernal time it took to run the barack obama object (user time + system time) and divide it by teh number of processes/cores we are using for the parallel operations. Therefore $(4082.902 + 72954.630)/(24 \times 4) = 16.85$ min. This compares favorably to spark, though it is a bit slower. 16.85 min is likely an under estiamte, and it would likely take a little longer than the estimate.

```
#Estimate speed of R processes BO data using 96 codes
    #(user + system time) / cores
((24082.902 + 72954.630 ) / (24 * 4)) / 60

## [1] 16.84679
```

4.3 Unit 7 discusses the idea of prescheduling ...

```
#... same code as in 4a


#turn off preschedule and employ dynamic allocation
mcoptions <- list(preschedule=FALSE)
nSub <- nSub/8 #takes too long to run full data
nSub
#120

system.time(
result <- foreach(i = 1:nSub,
                  .options.multicore = mcoptions,
                  .packages = c("readr"),       # libraries to load onto each worker
                  .combine = rbind,             # how to combine results
                  .errorhandling=c("pass"),
                  .verbose = TRUE) %dopar% {

    dat <- readr::read_delim(filePaths[i], delim = " ", col_names = F)

    cat("########  iteration ", i, " is complete!! :) ########")

    dat[grep("Barack_Obama", dat$X4, fixed = T), ]
                  }
)
 #user    system   elapsed
 #4159.053 14686.835   828.382
(828.382 * 8) / 60 #120 * 8 = 960 files
# 6627.056 sec ~ 110 min

dim(result)
#[1] 54186     6

head(result)
# A tibble: 6 x 6
#        X1     X2    X3
#     <int>  <chr> <chr>
```

```
#1 20081129 210000    pt
#2 20081014 190000    en
#3 20081108 190000    no
#4 20081128 190001    en
#5 20081110 160000    et
#6 20081101 110000    fr
# ... with 3 more variables: X4 <chr>, X5 <int>, X6 <int>
```

Time for static and dyanmic allocation for Barack Obama wikipedia script in R:

4a) Static: $\sim$ 75 min

4c) Dynamic: $\sim$ 110 min

Dynamic allocation is slower than static allocation. This is expected here, as each task is expected to take teh same amount of time and use similar computer resources. Dynamic approach sends tasks one at a time to a node, which will increase communication overhead compared to static allocaiton. Sending a set number of tasks to a core (static) reduces communcation time, and is more efficient when tasks and cores are similar.

5 Details of the Cholesky decomposition presented in Unit 9 ...

5.1 Work out the operation count (multiplies and divides) for the Cholesky decomposition ...

- $\alpha_{11} = \sqrt{\alpha_{11}}$ is neglibible
- $\alpha_{21} = \frac{\alpha_{21}}{\alpha_{11}} \sim (m-k-1)$ operations
- $A_{22} \sim (m-k-1)^2$ operations

Therefore,

$$\begin{aligned}\text{Operations } &= \sum_{k=0}^{m-1}(m-k-1)^2 = \sum_{k=0}^{m-1}(m-k-1) \\ &= \sum_{j=0}^{m-1} j^2 + \sum_{j=0}^{m-1} j \\ &\approx \frac{1}{3}m^3 + \frac{1}{2}m^2\end{aligned}$$

5.2 Suppose Ive written out the Cholesky calculation based on for loops ...

Yes, and one would save memory and computational time.