```
# TODO: CoNVex batch execution example -- ONLY FOR DOCUMENTATION
# A copy of this script is ALSO AVAILABLE from inst/Rbatch folder within CoNVex package installation!
# Author: Parthiban Vijayarangakannan (pv1)
# Date (last update): 17.04.2013
#
# This example explains:
# * Running CoNVex in a batch environment (can also be customised for non-batch environment)
# * For non-batch environment, you use the same code below. Run the Unix commands in .sh files with a 'wait' command in between.
#   This waits for the previous command to finish execution before running the next one.
# * Each step must **FINISH** before running the next step
# * WTSI specific example bsub commands for running (batch) unix commands in the farm
#   (optional: may or may not be suitable non-WTSI environment)
# * You can customise this for your project, copy/paste them in an interactive R session,
#   or make it more professional using a sophisticated batch execution pipeline (e.g., VRpipe at WTSI)
# * Don't try running the same commands in this example in the Sanger farm, these may be part of a real project (some harmless commands are
allowed!)
# * This original R script should be available from your CoNVex installation here:
#   CoNVex_Pipeline_Doc = paste(getLibPath(),"/CoNVex/inst/Rbatch/CoNVex_Pipeline_Doc.R",sep=""); # - Useful, IF formatting is lost in the
PDF file of this example
# * This example is under constant improvement. Please copy this locally for a secure copy!!
#
# CoNVex REQUIRES the following from you:
# * BAM files (full path)
# * Gender of the samples (if you include chr X)
# * Sample IDs (something like MYPROJ_12345 for each sample)
##############################################################################

# Install required external packages first
# If your network/proxy works fine, you should be able to install them easily
install.packages("mgcv") # mgcv package
install.packages("R2HTML") # R2HTML package

# Install CoNVex package - current version (August 2012) is v0.3. Look for the latest version in the folder below!
install.packages("CoNVex_0.4.tar.gz") # locally copied file [OR]
install.packages("/nfs/users/nfs_p/pv1/ConvexPackage/CoNVex_0.4.tar.gz") # directly from my NFS folder

# Load CoNVex
require(CoNVex)
# Load basic sample info - an example file is bundled with CoNVex in the inst/extdata folder
baminfoRD = read.table("/path/to/BAMfileLocations_SampleIDs_Gender_Example.txt", header=TRUE)
```

```
# STEP 1a: Preparation - Required input/output files
# INPUT files bundled with package / For regions other than Agilent V3, check the folder and/or the list of files below.
# Agilent V3 regions / bundled with CoNVex package / other exomes (human, mouse) - are listed at the end of this file
regions_file = paste(getLibPath(),"/CoNVex/extdata/SureSelect_50Mb.txt",sep="");
centromere_regions_file = paste(getLibPath(),"/CoNVex/extdata/gaps_table_hg19.txt",sep="") # bundled with package; use gaps_table_hg18.txt
for Agilent V1 exomes

# INPUT variables (R vectors) that are given as parameter options to SampleInfoPrep() function in Step 1b and other functions
sample_ids = as.character(baminfoRD[,1]); # sample_ids - vector
bamfiles = as.character(baminfoRD[,2]);  # Input bam files - vector
# Must have 'M' or 'F' if you include chr X (cX=1) for analysis in Step 3. 'M', 'F' or even 'U' (unknown) for all if you exclude!
gender = as.character(baminfoRD[,3]); gender[gender=='Male'] = 'M'; gender[gender=='Female'] = 'F';
output_folder = "/path/to/my/proj/";
RDfiles = paste(output_folder,"/ProbeRD_",sample_ids,".dat",sep="") # Read depth file names - vector - to be created in STEP 2
L2Rfiles = paste(output_folder,"/L2R/L2R_",sample_ids,".dat",sep="") # Log2 ratio file names - vector - to be created in STEP 3
GAMfiles = paste(output_folder,"/L2R/GAM_",sample_ids,".dat",sep="") # ADM scores file names - vector - to be created in STEP 5

# INPUT - SW array parameters - folders and output CNV calls
p=3; swt_del=5; swt_dup=5; dv=0.5;
sw_exec = paste(getLibPath(),"/CoNVex/exec/swa_lin64",sep=""); # SW-Array execution binary
swa_folder = paste(output_folder,"/CNVcalls",sep=""); # CNV calls will be stored in this folder; create it if it's not there already
# system(paste("mkdir",swa_folder)); # Use this command within R or CREATE swa_folder FOLDER in the Unix prompt separately
CNVfiles = paste(swa_folder,"/CoNVex_",sample_ids,"_p",p,"_tdel",swt_del,"_tdup",swt_dup,"_dv",dv,"_.txt",sep="") # one file per sample

# OUTPUT files (for consequtive steps these may act as input) - define the file names here
sample_info_file = "/path/to/my/proj/MYPROJ_SampleInfo.txt" # Output file name -- this will be created in STEP 1b
features_file = "/path/to/my/proj/MYPROJ_Features.txt" # Output file name -- this will be created in STEP 2
BPfile =  "/path/to/my/proj/MYPROJ_Breakpoints.txt" # # Output file name -- this will be created in STEP 4

# STEP 1b: Evaluates input data above - automatically saves them in a tab-delimited file (file name as in 'sample_info_file' above)
# Save all vectors in a fixed format - SampleInfoPrep() automatically saves them in the specified sample_info_file
baminfoALL = SampleInfoPrepInteractive(sample_id=sample_ids, gender=gender, bamfiles=bamfiles, RDfiles=RDfiles, L2Rfiles=L2Rfiles,
GAMfiles=GAMfiles, CNVfiles=CNVfiles, sample_info_file=sample_info_file, output_folder=output_folder, overwrite = FALSE);
# baminfoALL object is _not_ required for further analysis, but useful to have in memory

# STEP 2: Calculate READ DEPTH (java commands) -- Create, Save and Execute these commands in the farm using bsub (use of perl wrapper
recommended)
# ReadDepth java program requires CLASSPATH; Use getClassPath() function without arguments - this gives you the list of jars for CLASSPATH
config
```

```
rdc_file = "/path/to/my/proj/Depth_04052011_missing.sh" # List of unix commands
rdc =
ReadDepthCommands(regions_file=regions_file,sample_ids=sample_ids,bamfiles=bamfiles,RDfiles=RDfiles,chr_prefix="",output_folder=output_folde
r,max_memory=2);
write.table(rdc,file=rdc_file,row.names=FALSE, col.names=FALSE, sep="\t", quote=FALSE)
# Run bsub job array:
bsub_command = paste("bsub -q normal -R'select[mem>2200] rusage[mem=2200]' -M2200000 -o ",rdc_file,".farm.out -J'PRD[1-",length(rdc),"]
%",length(rdc),"' /software/cnpoly/bin/submit_job_array.pl ",rdc_file, sep="")
print(rdc[1]) # print and test the unix command - just the first one as an example
print(bsub_command) # print and test the bsub command
system(bsub_command) # run the bsub command

# STEP 3: Generate and save log2 ratio
# FOR LARGE SAMPLE SIZES (e.g., 100+ samples) USE ALTERNATIVE STEP 3 FROM BELOW!
# SampleLogRatio Command - This is a SINGLE command line call to R
slr_command =
SampleLogRatioCallCommands(sample_info_file=sample_info_file,regions_file,features_file=features_file,cX=1,Rbatch_folder="",version=1) # if
Rbatch_file's "" (or unused), the default lib location will be used
# Execute *_command using bsub or VRpipe or <your-method-for-running-parallel-jobs-in-cluster>
bsub_command = paste("bsub -q normal -R'select[mem>2500] rusage[mem=2500]' -M2500000 -o SLR.farm.out -J'SLR' '",slr_command,"'", sep="")
print(slr_command)
print(bsub_command)
system(bsub_command)

# STEP 4: Generate and save Breakpoints file - This is a SINGLE command line call to R
bp_command = BreakpointsCallCommands(RDfile_RepSample=RDfiles[1],BPfile=BPfile,Rbatch_folder="")
# Execute *_command using bsub or VRpipe or <your-method-for-running-parallel-jobs-in-cluster>
bsub_command = paste("bsub -q normal -R'select[mem>500] rusage[mem=500]' -M500000 -o BP.farm.out -J'BP' '",bp_command,"'", sep="")
print(bp_command)
print(bsub_command)
system(bsub_command)

# STEP 5: GAM correction commands -- one per sample
gam_commands = GAMCorrectionCommands(features_file,L2Rfiles,GAMfiles,RDfiles,BPfile,output_folder,sample_ids,Rbatch_folder="");
gam_file = "/path/to/my/proj/GAM_04052011.sh"
write.table(gam_commands,file=gam_file,row.names=FALSE, col.names=FALSE, sep="\t", quote=FALSE)
# Execute *_commands using bsub (perl wrapper) or VRpipe
bsub_command = paste("bsub -q normal -R'select[mem>1000] rusage[mem=1000]' -M1000000 -o ",gam_file,".farm.out -
J'GAM[1-",length(gam_commands),"]%",length(gam_commands),"' /software/cnpoly/bin/submit_job_array.pl ",gam_file, sep="")
print(gam_commands[1])
```

```
print(bsub_command)
system(bsub_command)

# STEP 6: CNV detection using Smith-Waterman algorithm -- one command per sample
# Input calling parameters
# SW-Array execution parameters
# Recommended p values (p=3 for Agilent V3, p=2, DDD exome+, p>3 (3.5,4) for Agilent V1 (not tested extensively)
# p values decide how far the CNV boundaries can be extended, looking at the multipanel plots (each CNV separately) later is mandatory

sw_commands =
SWCNVCallCommands(p,swt_del,swt_dup,dv,GAMfiles,CNVfiles,sample_ids,centromere_regions_file,swa_folder,sw_exec,Rbatch_folder="");
sw_file = "/path/to/my/proj/SW_04052011.sh"
write.table(sw_commands,file=sw_file,row.names=FALSE, col.names=FALSE, sep="\t", quote=FALSE)

# Execute *_commands using bsub (perl wrapper) or VRpipe or <your-method-for-running-parallel-jobs-in-cluster>
bsub_command = paste("bsub -q normal -R'select[mem>1000] rusage[mem=1000]' -M1000000 -o ",sw_file,".farm.out -
J'SWCNV[1-",length(sw_commands),"]%",length(sw_commands),"' /software/cnpoly/bin/submit_job_array.pl ",sw_file, sep="")
print(sw_commands)
print(bsub_command)
system(bsub_command)

######### ######### ######### ######### ######### #########

######### REGIONS FILES -- bundled with CoNVex / use ONE of these depending on your project
regions_file = paste(getLibPath(),"/CoNVex/extdata/SureSelect_V1.txt",sep=""); # Agilent V1 regions - ** hg18 build **
regions_file = paste(getLibPath(),"/CoNVex/extdata/SureSelect_50Mb.txt",sep=""); # Agilent V3 regions
regions_file = paste(getLibPath(),"/CoNVex/extdata/SureSelect_50Mb_DDD.txt",sep=""); # Agilent V3 - DDD exome+ regions
regions_file = paste(getLibPath(),"/CoNVex/extdata/SureSelect_MouseV1_S0276129.txt",sep=""); # Agilent - Mouse V1 exome regions - NCBI37
build

######### PLOTTING AND VISUALISATION #########
# OPTIONAL: If 'CNVfiles' is not in memory, you can load it from sample_info_file:
sample_info_file = "/path/to/my/proj/MYPROJ_SampleInfo.txt" # Copied/pasted from above -- this was created in STEP 1a/1b
d = read.table(sample_info_file); CNVfiles = as.character(d[,7]);

# Get all CNV calls of all samples (from each sample's CNV file)
CNVcalls = GetCNVCalls(CNVfiles)

# Plot CNV stats
setwd(output_folder);
```

```
PlotCNVStats(CNVcallsAll=CNVcalls);
# This will create two plots in output_folder currently; (1) Number of calls vs Samples; (2) #Dels/#Duls ratio in all samples

# Plot Known CNVs (%)
# Options:
# -- mark_outliers=0 (default) does not highlight outliers; highlighted otherwise
# -- outlier_from_median = (default=5) marks the outliers that have known (%) lower than median(known%)-outlier_from_median
setwd(output_folder);
PlotKnownCNVStats(CallsAll=CNVcalls,mark_outliers=1,outlier_from_median=5);

# Plot Known CNVs (%) - Version 2 [ #Calls vs. Plot Known CNVs (%) ] - since CoNVex v0.4
PlotKnownCNVStatsV2(CallsAll=CNVcalls);

# Genome-wide plots of ADM scores
GAMfiles = as.character(d[,6]);
for(i in 1:length(sample_ids)) { GenomePlot(GAMfiles[i],sample_id=sample_ids[i],output_folder=output_folder); }
# This will create a plot from the scores; and use sample_id in the plot title; if sample_id is "", it uses GAMfile name in plot title

## Sample QC metrics - are there any dodgy outlier samples?
## Sample Means and MADs - [Method 1] ## - REQUIRES UNIX - uses large amount of memory - helpful to identify noisy samples in the batch
## SampleMeans can be calculated using Step 3 by speciying a file name to 'sample_means_file' option
## SampleMads, SampleMediansAuto, and SampleMediansX can be calculated using the following Java command (fast and efficient on 1000s os
samples)
smm_command = "java -Xmx500m SampleMediansMads -sample_info_file /path/to/MYPROJ_SampleInfo.txt -num_probes_auto <N> -num_probes_X <M> -
output_file /path/to/SampleMediansMadsOutput.txt"
bsub_command = paste("bsub -q normal -R'select[mem>520] rusage[mem=520]' -M520 -o smm.out '",smm_command,"'",sep="")
print(bsub_command)
system(bsub_command)

## Sample Means and MADs - [Optional Method 2] ## - REQUIRES UNIX - uses large amount of memory - helpful to identify noisy samples in the
batch
## PlotKnownCNVStatsV2 - already available from DDD_03102013.pptx slides ##
smm_command = SMMCallCommand(sample_info_file = sample_info_file, regions_file=regions_file, output_file="/path/to/output/
SampleMeansMads.txt")
bsub_command = paste("bsub -q normal -R'select[mem>15000] rusage[mem=15000]' -M15000 -o ",output_folder,"/SMMCC.farm.out -J'SMM'
'",smm_command,"'", sep="")
print(smm_command)
print(bsub_command)
system(bsub_command)
```

```
## Call QC - this is optimised for V3 and V3+ libraries, you may like to fine tune this to V4 and V5
# Call QC is implemented using two functions:
# - CallQCChrX() - applied only on ChrX (?CallQCChrX for documentation)
# - CallQCAll() - applied on all chromosomes including X (?CallQCAll for documentation)
#
# The following QC metrics are needed:
# - SampleMediansX vs. SampleMediansAuto (already calculated during SampleQC - available above)
# - CoNVex score (convex_score) - available from Step 6
# - #Probes (num_probes) - available from Step 6
# - Mean log2 ratio (mean_l2r)  - available from MeansMadsCommands() below
# - MAD of log2 ratio (mad_l2r) - available from MeansMadsCommands() below
# - Internal frequency proportion at ANY and 50% reciprocal overlap (internal_freq and rc50_internal_freq) using GetFrequency() function
# - common_forward and common_backward overlap proportion (output of recipeB package's toverlap() function OR CoNVex's ChrOverlap()
function)

## Multi panel plots [for visualising each CNV]
setwd(output_folder)
sample_info_file = "/path/to/my/proj/MYPROJ_SampleInfo.txt" # Copied/pasted from above -- this was created in STEP 1a/1b
features_file = "/path/to/my/proj/MYPROJ_Features.txt" # Copied/pasted from above -- this was created in STEP 3
# cnv_file contains all CNVs that need to be plotted from all samples - this should match the output of GetCNVCalls()
cnv_file = "path/to/cnv_file.txt"
# Optional file - probe regions in this format: chr, start, end, gene_symbol
gene_file = paste(getLibPath(),"/CoNVex/extdata/SureSelect_50Mb_GENES.txt",sep="");

# Correlation matrix file is required if you use median reference from a subset of samples - created in Step 3
cor_matrix = paste(output_folder,"/AllSamples_CorMatrix.txt",sep="") ## All other options as in STEP 3 on top
# If you use median reference from 'all' samples, cor_matrix is optional.
# Warning: If you use a 'all' samples each command in the shell script will read all samples' log2 ratio files.
# This is a highly parallel command, please make sure this is not too IO intensive

mpp_commands = MultiPanelCommands(CNVcallsfile=cnv_file, cor_matrix=cor_matrix_file, sample_info_file=sample_info_file,
features_file=features_file, gene_file=gene_file)
mpp_commands_file = paste(output_folder,"/MPP_execute.sh",sep=""); # List of Unix commands to run - one for each 'sample' in the cnv_file
write.table(mpp_commands,file=mpp_commands_file,row.names=FALSE, col.names=FALSE, sep="\t", quote=FALSE)
# Execute *_commands using bsub (perl wrapper) or VRpipe or with 'wait' command
bsub_command = paste("bsub -P ddd -q normal -R'select[mem>2000] rusage[mem=2000]' -M2000000 -q normal -o ",mpp_commands_file,".farm.out -
J'MPP[1-",length(mpp_commands),"]%10' /software/cnpoly/bin/submit_job_array.pl ",mpp_commands_file, sep="")
print(mpp_commands[1])
print(bsub_command)
system(bsub_command)
```

```
## How to calculate Mean log2 ratio and MAD of the mean log2 ratio for each CNV?
## INPUT requirements similar to above (multi panel plots) - you also need the regions file
mpp_cmm = MeansMadsCommands(CNVfiles=CNVfiles, sample_ids=sample_ids, cor_matrix=CMfile, sample_info_file=sample_info_file,
regions_file=regions_file, features_file=features_file)
mpp_cmm_file=paste(output_folder,"/CNVMeansMeads_ALlsamples.sh",sep="");
write.table(mpp_cmm,file=mpp_cmm_file,row.names=FALSE, col.names=FALSE, sep="\t", quote=FALSE)
# Execute *_commands using bsub (perl wrapper) or VRpipe
bsub_command = paste("bsub -q normal -R'select[mem>1000] rusage[mem=1000]' -M1000000 -q normal -o ",mpp_cmm_file,".farm.out -
J'MMC[1-",length(mpp_cmm),"]%100' /software/cnpoly/bin/submit_job_array.pl ",mpp_cmm_file, sep="")
print(mpp_cmm[1])
print(bsub_command)
system(bsub_command)

## How to retrieve Ensembl VEP consequences for each CNV?
## You should have ENSEMBL PERL API installed or accessible locally
## Link here: http://www.ensembl.org/info/docs/api/index.html
## Two Perl scripts are bundled with CoNVex (Modify the script to use your local Ensembl API / Perl location):
## (1) GetVEPAnnotation.pl [retrieves/shows all consequences and #transcripts]; (2) GetVEPAnnotation2.pl [shows only the most severe
consequence]
## these are available here:
paste(getLibPath(),"/scripts/GetVEPAnnotation.pl",sep="") # (1)
paste(getLibPath(),"/scripts/GetVEPAnnotation2.pl",sep="") # (2)
vep_command1 = "perl GetVEPAnnotation.pl /path/to/cnv_file.txt" # all CNVs from all samples (as above)
vep_command2 = "perl GetVEPAnnotation2.pl /path/to/cnv_file.txt" # all CNVs from all samples (as above)

## How to calculate the INTERNAL/EXTERNAL FREQUENCY? ##
## INTERNAL FREQUENCY
cnv_calls = read.table(file=cnv_file, header=TRUE) # Columns should match the output of GetCNVCalls()
cnv_calls_freq = GetFrequency(cnv_calls) # Internal frequency - any overlap [OR]
cnv_calls_freq = GetFrequency(cnv_calls, ro_threshold=50) # Internal frequency >50% reciprocal overlap

## EXTERNAL FREQUENCY
cnv_calls_external = read.table(file=cnv_file_external, header=TRUE) # cnv_file_external file contains the CNV calls from another dataset
cnvs_calls_ext_freq = GetFrequency(chr_list1=cnv_calls, chr_list2=cnv_calls_external) # External frequency - any overlap
cnvs_calls_ext_freq = GetFrequency(chr_list1=cnv_calls, chr_list2=cnv_calls_external, ro_threshold=50) # External frequency - 50% recip.
overlap

## How to calculate % overlap with KNOWN CNVs?
cnv_calls_knownpc = GetKnownCNVPC(CallsAll=cnv_calls)
```

```
## How to retrieve GENES/TRANSCRIPTS from Ensembl?
## You should have ENSEMBL PERL API installed or accessible locally
## Link here: http://www.ensembl.org/info/docs/api/index.html
## Two Perl scripts are bundled with CoNVex (Modify the script to use your local Ensembl API / Perl location):
## (1) GetEnsemblGenes.pl [retrieves genes only]; (2) GetEnsemblGenesTranscripts.pl [retrieves genes and transcripts]
## these are available here:
paste(getLibPath(),"/scripts/GetEnsemblGenes.pl",sep="") # (1)
paste(getLibPath(),"/scripts/GetEnsemblGenesTranscripts.pl",sep="") # (2)
command1 = "perl GetEnsemblGenes.pl /path/to/cnv_file.txt" # all CNVs from all samples (as above)
command2 = "perl GetEnsemblGenesTranscripts.pl /path/to/cnv_file.txt" # all CNVs from all samples (as above)

#### MORE PLOTS/EXAMPLES ARE COMING SOON ####
######### ######### ######### ######### ######### #########

######### ######### ######### ######### ######### #########
######## ALTERNATIVE ########## to the functions above #

# ALTERNATIVE STEP 3: Generate and save log2 ratio
# SampleLogRatio Command - This is a SINGLE command line call to R
# PLEASE NOTE THE FOLLOWING OPTIONS: version=3,RPKM=0,min_samples=50
# CORRELATION MATRIX: Correlation matrix holds the list of correlations between samples
# SAMPLE MEANS FILE: Sample mean depth can be calculated in this step itself
# VERSION: version=1 uses the all samples in the batch for estimating median reference; version=2 and version=3 use correlated subset of
samples to estimate median reference; version=3 requires Unix (cut command, etc.) - memory efficient than version=2 (generic R)
# RPKM: RPKM=0 uses read depth for correlating samples (preferred); RPKM=1 use RPKM/FPKM to correlate samples [IGNORED IF version=1 is used]
# min_samples: minimum number of correlated samples to use; ALSO required the same number of males and females if ChrX=1 [IGNORED IF
version=1 is used]
# If you use Unix, the most friendly options are given below:
cor_matrix = paste(output_folder,"/AllSamples_CorMatrix.txt",sep="") ## Correlation matrix All other options as in STEP 3 on top
sm_file = paste(output_folder,"/AllSamples_Means.txt",sep="") ## Sample means - All other options as in STEP 3 on top
slr_command =
SampleLogRatioCallCommands(sample_info_file=sample_info_file,regions_file,features_file=features_file,cX=1,Rbatch_folder="",version=3,RPKM=0
,min_samples=25, cor_matrix=cor_matrix, sample_means_file=sm_file)
```