EMASE Documentation

Release 0.10.16

Kwangbom "KB" Choi, Ph. D.

Jul 24, 2017

Contents

Contents:

CHAPTER 1

Overview

EMASE: Expectation-Maximization algorithm for Allele Specific Expression

Narayanan Raghupathy, Kwangbom Choi, Steve Munger, and Gary Churchill

- Free software: GNU General Public License v3 (GPLv3)
- Documentation: https://emase.readthedocs.org.

Note: The documentation for EMASE is still under work.

What is EMASE?

EMASE is a software program written in Python to quantify allele-specific expression and gene expression simultaneously from RNA-seq data. EMASE takes in the diploid transcriptome alignment BAM file and GTF file as inputs and estimates expression abundance for each isoforms and each alleles using Expectation Maxmization algorithm.

Why Use EMASE?

Current RNA-seq analysis pipeline employ two steps to quantify gene expression and allele-specific expression (ASE); gene expression is estimated from all read alignments, while ASE is assessed separately by using only reads that overlap known SNP locations.

Large-scale genome sequencing efforts have characterized millions of genetic variants across in human and model organisms. However development of tools that can effectively utilize this individual/strain-specific variation to inform quantitation of gene expression abundance have lagged behind.

EMASE, together with g2gtools (https://github.com/churchill-lab/g2gtools), offers an integrated solution to utilize known genetic variations in quantifying expression abundances at allele and gene/isoform level.

In F1 hybrids from model organisms, EMASE allows us to utilize parental strain-specific genetic variation in RNA-seq analysis to quantify gene expression and allele-specific expression (ASE) simultaneously

In humans, EMASE allows us to utilize the individual's genetic variation in doing personalized RNA-seq analysis and quantify gene expression and allele-specific expression (ASE) simultaneously

Briefly, EMASE: EM for allele-specific expression, uses individualized diploid genomes/transcriptomes adjusted for known genetic variations and quantifies allele-specific gene expression and total gene expressionsimultaneously. The EM algorithm employed in EMASE models multi-reads at the level of gene, isoform, and allele and apportions them probabilistically.

One can use g2gtools to create personalized diploid transcritpme and align RNA-seq reads simultaneously to the diploid transcriptome and get alignment file in BAM format. This diploid BAM file can be used as input to EMASE.

Applications

Allele-specific gene expression in F1 Hybrids from model organisms

If we have F1 hybrids with parental genetic variants information, one can use g2gtools to build strain specific genomes and extract diploid transcriptome. RNA-seq alignment bam file obtained by aligning RNA-seq reads to the diploid transcriptome is used as input for EMASE.

Personalized ASE analysis in Human

EMASE can be used to do personalized RNA-seq analysis in human. For this, we use the phased genetic variation (SNP and Indel) information to construct personalized diploid genome and align reads to the diploid transcriptome..

Allele-specific Binding using Chip-Seq in F1 Hybrids

Although we explained the use of EMASE to quantify Allele-Specific Expression from RNA-seq data, the tool can be used with other types of sequencing data. We have successfully used EMASE to quantify allele-specific binding from ChIP-seq data. While useing ChIP-seq, one needs to use diploid binding target sequences instead of diploid transcriptome for alignment target sequences.

Mining Diploid alignments and alignment probabilities

EMASE can be used to glean more information from the alignment, in addition to running EMASE and obtaining effective read counts for each allele and isoform. For example, we can use EMASE's count-alignment program to obtain unique reads at allele-level for every gene, gene unique reads but allele-level multireads, and the total number of reads aligned to every gene. Having these alignment statistics at for every isoform and gene can be useful in interpreting expression estimates from EMASE.

References

- [Hierarchical Analysis of Multi-mapping RNA-Seq Reads Improves the Accuracy of Allele-specific Expression](http://www.biorxiv.org/content/early/2017/07/22/166900). Narayanan Raghupathy, Kwangbom Choi, Matthew J Vincent, Glen L Beane, Keith Sheppard, Steven C Munger, Ron Korstanje, Fernando Pardo Manuel de Villena, Gary A Churchill, doi: https://doi.org/10.1101/166900
- [RNA-Seq Alignment to Individualized Genomes Improves Transcript Abundance Estimates in Multiparent Populations](http://www.genetics.org/content/198/1/59.short) Steven C. Munger, Narayanan Raghupathy,Kwangbom Choi, Allen K. Simons, Daniel M. Gatti, Douglas A. Hinerfeld, Karen L. Svenson, Mark P. Keller, Alan D. Attie, Matthew A. Hibbs, Joel H. Graber, Elissa J. Chesler and Gary A. Churchill. Genetics. 2014 Sep;198(1):59-73. doi: 10.1534/genetics.114.165886.

- [PRDM9 Drives Evolutionary Erosion of Hotspots in Mus musculus through Haplotype-Specific Initiation of Meiotic Recombination](http://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.1004916) Christopher L. Baker, Shimpei Kajita, Michael Walker, Ruth L. Saxl, Narayanan Raghupathy, Kwangbom Choi, Petko M. Petkov, Kenneth Paigen PLOS Genetics: published 08 Jan 2015 | info:doi/10.1371/journal.pgen.1004916

CHAPTER 2

Installation

Although EMASE is available at PyPI (https://pypi.python.org/pypi/emase/) for 'pip install' or 'easy_install', we highly recommend using Anaconda distribution of python (https://www.continuum.io/downloads) to install all the dependencies without issues. EMASE is also available on Anaconda Cloud, so add the following channels:

```
$ conda config --add channels r
$ conda config --add channels bioconda
```

To avoid version confliction among dependencies, it is desirable to create a virtual environment for EMASE:

```
$ conda create -n emase jupyter
$ source activate emase
```

Once EMASE virtual environment is activated, your shell prompt will show '(emase)' at the beginning to specify what virtual environment you are currently in. Now please type the following and install EMASE:

```
(emase) $ conda install -c kbchoi emase
```

That's all! We note that you can go out from EMASE virtual environment anytime once you are done using EMASE:

```
(emase) $ source deactivate
```

CHAPTER 3

Usage

To use EMASE in a new project

In python scripts, we can load 'emase' as a module:

```
import emase
```

Or:

```
from emase import AlignmentMatrixFactory as AMF
from emase import AlignmentPropertyMatrix as APM
from emase import EMfactory
```

To run EMASE on command line

Note: We will assume you installed EMASE in its own conda virtual environment. First of all, you have to "activate" the virtual environment by doing the following:

```
source activate emase
```

The first step of the pipeline is to process reference genome:

```
prepare-emase -G ${REF_GENOME} -g ${REF_GTF} -o ${REF_DIR} -m --no-bowtie-index
```

'prepare-emase' generates the following files for the reference genome:

```
${REF_DIR}/emase.transcriptome.fa
${REF_DIR}/emase.transcriptome.info         <== Used as ${TID_FILE} in the following␣
↪steps
${REF_DIR}/emase.gene2transcripts.tsv       <== Used as ${GROUP_FILE} in the␣
↪following steps
```

Then build a pooled transcriptome and prepare required files for EMASE:

```
create-hybrid -G ${GENOME1},${GENOME2} -g ${GTF1},${GTF2} \
              -s ${SUFFIX1},${SUFFIX2} -o ${EMASE_DIR}
```

Now the following files will be available:

```
${EMASE_DIR}/emase.pooled.transcriptome.fa
${EMASE_DIR}/emase.pooled.transcriptome.info  <== Used as ${TINFO_FILE} in the next␣
↪steps
${EMASE_DIR}/bowtie.transcriptome.1.ebwt
${EMASE_DIR}/bowtie.transcriptome.2.ebwt
${EMASE_DIR}/bowtie.transcriptome.3.ebwt
${EMASE_DIR}/bowtie.transcriptome.4.ebwt
${EMASE_DIR}/bowtie.transcriptome.rev.1.ebwt
${EMASE_DIR}/bowtie.transcriptome.rev.2.ebwt
```

RNA-seq reads should be aligned against the pooled transcriptome:

```
bowtie -q -a --best --strata --sam -v 3 ${EMASE_DIR}/bowtie.transcriptome ${FASTQ} \
    | samtools view -bS - > ${BAM_FILE}
```

Before running EMASE, we need to convert the bam file into the emase format:

```
bam-to-emase -a ${BAM_FILE} -i ${TID_FILE} -s ${SUFFICE1},${SUFFIX2} -o ${EMASE_FILE}
```

For paired-end data, perform upto this step with R1 and R2 end independently, and get their common alignments:

```
get-common-alignments -i ${EMASE_FILE_R1},${EMASE_FILE_R2} -o ${EMASE_FILE}
```

Finally, to run EMASE:

```
run-emase -i ${EMASE_FILE} -g ${GROUP_FILE} -L ${TINFO_FILE} -M ${MODEL} -o ${OUTBASE}
↪ \
          -r ${READLEN} -p ${PSEUDOCOUNT} -m ${MAX_ITERS} -t ${TOLERANCE}
```

'run-emase' outputs the following files:

```
${OUTBASE}.isoforms.expected_read_counts
${OUTBASE}.isoforms.tpm
${OUTBASE}.genes.expected_read_counts
${OUTBASE}.genes.tpm
```

CHAPTER 4

Real Use Cases

Estimating allele-specific expression from human RNA-seq data

1. Process reference data

We need to first extract transcript information from the reference genome and gene annotation. Most importantly, EMASE requires the list of transcript ID's and which gene each transcript belong to.:

```
prepare-emase -G ${REF_FASTA} -g ${REF_GTF} -o ${REF_DIR} -m --no-bowtie-index
```

'prepare-emase' generates the following files for the reference genome:

```
${REF_DIR}/emase.transcriptome.fa
${REF_DIR}/emase.transcriptome.info
${REF_DIR}/emase.gene2transcripts.tsv
```

2. Build an individualized genome

We assume there is a vcf file that contains phased variant information for every sample of your population. Unless we know which allele is M(aternal) or P(aternal), we are going to distinguish two alleles with suffices, L(eft) and R(ight). We also recommend to use different ${SAMPLE_DIR} for each sample:

```
python build_new_sequence_from_vcfs.py -r ${REFERENCE_FASTA} \
                                       -i ${INDEL_VCF} \
                                       -s ${SNP_VCF} \
                                       -d ${SAMPLE_DIR} \
                                       -o L.fa \
                                       ${SAMPLE_HAP1_ID_IN_VCF}
python build_new_sequence_from_vcfs.py -r ${REFERENCE_FASTA} \
                                       -i ${INDEL_VCF} \
                                       -s ${SNP_VCF} \
                                       -d ${SAMPLE_DIR} \
                                       -o R.fa \
                                       ${SAMPLE_HAP2_ID_IN_VCF}
```

3. Individualize gene annotation

We want to incorporate individual variation into the gene annotation too so we can build individualized transcriptome in the following step:

```
python adjust_annotations.py -s 4 -e 5 -c 1 -t 9 \
                             -d ${SAMPLE_DIR} \
                             -o L.gtf \
                             -C ${SAMPLE_HAP1_ID_IN_VCF}_comments.txt
                             ${REFERENCE_GTF} \
                             ${SAMPLE_HAP1_ID_IN_VCF}
python adjust_annotations.py -s 4 -e 5 -c 1 -t 9 \
                             -d ${SAMPLE_DIR} \
                             -o R.gtf \
                             -C ${SAMPLE_HAP1_ID_IN_VCF}_comments.txt
                             ${REFERENCE_GTF} \
                             ${SAMPLE_HAP2_ID_IN_VCF}
```

4. Create a personalized diploid transcriptome

From L.fa, R.fa, and the corresponding gtf files, we are going to create diploid transcriptome and other information that EMASE requires. We assume bowtie v1.0.0 or newer is available.:

```
prepare-emase -G ${SAMPLE_DIR}/L.fa,${SAMPLE_DIR}/R.fa -s L,R -o ${SAMPLE_DIR}
```

This will generate the following files:

```
${SAMPLE_DIR}/emase.pooled.transcriptome.fa
${SAMPLE_DIR}/emase.pooled.transcriptome.info
${SAMPLE_DIR}/bowtie.transcriptome.1.ebwt
${SAMPLE_DIR}/bowtie.transcriptome.2.ebwt
${SAMPLE_DIR}/bowtie.transcriptome.3.ebwt
${SAMPLE_DIR}/bowtie.transcriptome.4.ebwt
${SAMPLE_DIR}/bowtie.transcriptome.rev.1.ebwt
${SAMPLE_DIR}/bowtie.transcriptome.rev.2.ebwt
```

5. Align RNA-seq reads against the diploid transcriptome

Although EMASE is a flexible framework for many other alignment strategies, the current version of EMASE was most intensely tested with bowtie1 transcriptome alignments with the following parameters:

```
bowtie -q -a --best --strata --sam -v 3 ${SAMPLE_DIR}/bowtie.transcriptome ${FASTQ_
↪FILE} \
    | samtools view -bS - > ${SAMPLE_DIR}/bowtie.transcriptome.bam
```

6. Convert bam file into the emase format

EMASE runs on an alignment profile of three-dimensional incidence matrix. We convert an alignment file (bam) to the EMASE format using the following script:

```
bam-to-emase -a ${SAMPLE_DIR}/bowtie.transcriptome.bam \
             -i ${REF_DIR}/emase.transcriptome.info \
             -s L,R \
             -o ${SAMPLE_DIR}/bowtie.transcriptome.h5
```

7. Run EMASE

Now we are ready to run EMASE:

```
run-emase -i ${SAMPLE_DIR}/bowtie.transcriptome.h5 \
          -g ${REF_DIR}/emase.gene2transcripts.tsv \
          -L ${SAMPLE_DIR}/emase.pooled.transcriptome.info \
          -M ${MODEL} \
          -o ${SAMPLE_DIR}/emase \
          -r ${READ_LENGTH} \
          -c
```

'run-emase' outputs the following files as a result:

```
${SAMPLE_DIR}/emase.isoforms.effective_read_counts
${SAMPLE_DIR}/emase.isoforms.tpm
${SAMPLE_DIR}/emase.isoforms.alignment_counts
${SAMPLE_DIR}/emase.genes.effective_read_counts
${SAMPLE_DIR}/emase.genes.tpm
${SAMPLE_DIR}/emase.genes.alignment_counts
```

Estimating allele-specific binding from ChIP-seq data

We assume you have a set of individualized genome and annotation files, in this example, S1 and S2, created by Seqnature package. We also assume you have a bed file that specifies genomic regions of your interest. First, you need to convert your bed file into a simple gtf format:

```
bed-to-gtf -i targets.bed -o targets.gtf
```

The targets.gtf files should be modified according to the strains of our interest:

```
python adjust_annotations.py -s 4 -e 5 -c 1 -t 9 -o S1.gtf -C S1_comments.txt targets.
↪gtf S1
python adjust_annotations.py -s 4 -e 5 -c 1 -t 9 -o S2.gtf -C S2_comments.txt targets.
↪gtf S2
```

Finally, run:

```
prepare-emase -G S1.fa,S2.fa -g S1.gtf,S2.gtf -s S1,S2 -o S1xS2
```

This will store the following files in the folder 'S1xS2':

```
S1xS2/emase.pooled.transcriptome.fa
S1xS2/emase.pooled.transcriptome.info
S1xS2/bowtie.transcriptome.1.ebwt
S1xS2/bowtie.transcriptome.2.ebwt
S1xS2/bowtie.transcriptome.3.ebwt
S1xS2/bowtie.transcriptome.4.ebwt
S1xS2/bowtie.transcriptome.rev.1.ebwt
S1xS2/bowtie.transcriptome.rev.2.ebwt
```

Now you can align your RNA-seq reads against the pooled bowtie index of target region:

```
bowtie -q -a --best --strata --sam -v 3 S1xS2/bowtie.transcriptome ${FASTQ_FILE} \
    | samtools view -bS - > S1xS2/bowtie.transcriptome.bam
```

Next, we convert the alignment file into a format that EMASE use for running EM algorithm:

```
bam-to-emase -a S1xS2/bowtie.transcriptome.bam \
             -i S1xS2/emase.transcriptome.info \
             -s S1,S2 \
             -o S1xS2/bowtie.transcriptome.h5
```

It is now ready to run emase. We assume the read length is 100bp:

```
run-emase -i bowtie.transcriptome.h5 -L S1xS2/emase.pooled.transcriptome.info -M 4 -c
```

Deconvolving human and mouse gene expression from Patient-Derived Xenograft (PDX) models

Coming soon!

Estimating allele-specific expression from a F1 sample

Coming soon!

CHAPTER 5

Modules

emase Package

AlignmentMatrixFactory Module

class emase.AlignmentMatrixFactory.**AlignmentMatrixFactory**(*alnfile*)

cleanup()

prepare(*haplotypes*, *loci*, *delim='_'*, *outdir=None*)

produce(*h5file*, *title='Alignments'*, *index_dtype='uint32'*, *data_dtype=<type 'float'>*, *complib='zlib'*, *incidence_only=True*)

Sparse3DMatrix Module

class emase.Sparse3DMatrix.**Sparse3DMatrix**(*other=None*, *h5file=None*, *datanode='/'*, *shape=None*, *dtype=<type 'float'>*)

3-dim sparse matrix designed for "pooled" RNA-seq alignments

add(*addend_mat*, *axis=1*)

In-place addition

Parameters

- **addend_mat** – A matrix to be added on the Sparse3DMatrix object
- **axis** – The dimension along the addend_mat is added

Returns Nothing (as it performs in-place operations)

add_value(*lid*, *hid*, *rid*, *value*)

combine(*other*)

copy()

finalize()

get_cross_section(*index*, *axis=0*)

multiply(*multiplier*, *axis=None*)
In-place multiplication

Parameters

- **multiplier** – A matrix or vector to be multiplied
- **axis** – The dim along which 'multiplier' is multiplied

Returns Nothing (as it performs in-place operations)

reset()

save(*h5file*, *title=None*, *index_dtype='uint32'*, *data_dtype=<type 'float'>*, *incidence_only=True*, *complib='zlib'*)

set_value(*lid*, *hid*, *rid*, *value*)

sum(*axis=2*)

AlignmentPropertyMatrix Module

class emase.AlignmentPropertyMatrix.**AlignmentPropertyMatrix**(*other=None*, *h5file=None*, *datanode='/'*, *metanode='/'*, *shallow=False*, *shape=None*, *dtype=<type 'float'>*, *haplotype_names=None*, *locus_names=None*, *read_names=None*, *grpfile=None*)

Bases: *emase.Sparse3DMatrix.Sparse3DMatrix*

Axis
alias of Enum

bundle(*reset=False*, *shallow=False*)
Returns AlignmentPropertyMatrix object in which loci are bundled using grouping information.

Parameters

- **reset** – whether to reset the values at the loci
- **shallow** – whether to copy all the meta data

combine(*other*, *shallow=False*)

copy(*shallow=False*)

count_alignments()

count_unique_reads(*ignore_haplotype=False*)

get_read_data(*rid*)

get_reads_aligned_to_locus(*lid*, *hid=None*)

get_unique_reads(*ignore_haplotype=False*, *shallow=False*)
Pull out alignments of uniquely-aligning reads

Parameters

- **ignore_haplotype** – whether to regard allelic multiread as uniquely-aligning read
- **shallow** – whether to copy sparse 3D matrix only or not

Returns a new AlignmentPropertyMatrix object that particular reads are

load_groups(*grpfile*)

normalize_reads(*axis*, *grouping_mat=None*)
Read-wise normalization

Parameters

- **axis** – The dimension along which we want to normalize values
- **grouping_mat** – An incidence matrix that specifies which isoforms are from a same gene

Returns Nothing (as the method performs in-place operations)

Return type None

print_read(*rid*)
Prints nonzero rows of the read wanted

pull_alignments_from(*reads_to_use*, *shallow=False*)
Pull out alignments of certain reads

Parameters

- **reads_to_use** – numpy array of dtype=bool specifying which reads to use
- **shallow** – whether to copy sparse 3D matrix only or not

Returns a new AlignmentPropertyMatrix object that particular reads are

report_alignment_counts(*filename*)

save(*h5file*, *title=None*, *index_dtype='uint32'*, *data_dtype=<type 'float'>*, *incidence_only=True*, *complib='zlib'*, *shallow=False*)

sum(*axis*)

emase.AlignmentPropertyMatrix.**enum**(***enums*)

EMfactory Module

class emase.EMfactory.**EMfactory**(*alignments*)
A class that coordinate Expectation-Maximization

export_posterior_probability(*filename*, *title='Posterior Probability'*)
Writes the posterior probability of read origin

Parameters

- **filename** – File name for output
- **title** – The title of the posterior probability matrix

Returns Nothing but the method writes a file in EMASE format (PyTables)

get_allelic_expression (*at_group_level=False*)

prepare (*pseudocount=0.0*, *lenfile=None*, *read_length=100*)
Initializes the probability of read origin according to the alignment profile

Parameters **pseudocount** – Uniform prior for allele specificity estimation

Returns Nothing (as it performs an in-place operations)

report_depths (*filename*, *tpm=True*, *grp_wise=False*, *reorder='as-is'*, *notes=None*)
Exports expected depths

Parameters

- **filename** – File name for output
- **grp_wise** – whether the report is at isoform level or gene level
- **reorder** – whether the report should be either 'decreasing' or 'increasing' order or just 'as-is'

Returns Nothing but the method writes a file

report_read_counts (*filename*, *grp_wise=False*, *reorder='as-is'*, *notes=None*)
Exports expected read counts

Parameters

- **filename** – File name for output
- **grp_wise** – whether the report is at isoform level or gene level
- **reorder** – whether the report should be either 'decreasing' or 'increasing' order or just 'as-is'

Returns Nothing but the method writes a file

reset (*pseudocount=0.0*)
Initializes the probability of read origin according to the alignment profile

Parameters **pseudocount** – Uniform prior for allele specificity estimation

Returns Nothing (as it performs an in-place operations)

run (*model*, *tol=0.001*, *max_iters=999*, *verbose=True*)
Runs EM iterations

Parameters

- **model** – Normalization model (1: Gene->Allele->Isoform, 2: Gene->Isoform->Allele, 3: Gene->Isoform*Allele, 4: Gene*Isoform*Allele)
- **tol** – Tolerance for termination
- **max_iters** – Maximum number of iterations until termination
- **verbose** – Display information on how EM is running

Returns Nothing (as it performs in-place operations)

update_allelic_expression (*model=3*)
A single EM step: Update probability at read level and then re-estimate allelic specific expression

Parameters **model** – Normalization model (1: Gene->Allele->Isoform, 2: Gene->Isoform->Allele, 3: Gene->Isoform*Allele, 4: Gene*Isoform*Allele)

Returns Nothing (as it performs in-place operations)

update_probability_at_read_level (*model=3*)
Updates the probability of read origin at read level

Parameters **model** – Normalization model (1: Gene->Allele->Isoform, 2: Gene->Isoform->Allele, 3: Gene->Isoform*Allele, 4: Gene*Isoform*Allele)

Returns Nothing (as it performs in-place operations)

CHAPTER 6

History

0.10.16 (05-10-2016)

- Modified `prepare-emase` so it can process the newest Ensembl gene annotation (Release 84)
- The script `prepare-emase` can process gzipped files
- Updated documentation

0.10.15 (05-04-2016)

- Uploaded to Anaconda.org
- Updated documentation

0.10.14 (04-25-2016)

- Added option to not having rname when loading/saving `AlignmentPropertyMatrix`
- Documentation updated to reflect recent changes (e.g., processing paired-end data etc.)

0.10.12 (04-22-2016)

- `run-emase` report file names changed (effective -> expected)
- `run-emase` report file can have notes

0.10.11 (02-15-2016)

- Minor change in documentation

0.10.9 (02-09-2016)

- Fixed readthedocs compiling fails

0.10.5 (01-20-2016)

- Added `pull_alignments_from` method in `AlignmentPropertyMatrix` class
- Added a script `pull-out-unique-reads` that unsets emase pseudo-alignments that are not uniquely aligning

0.10.3 (01-06-2016)

- Fixed a bug in `run-emase` on handling inbred (reference or one haplotype) alignments

0.10.2 (01-04-2016)

- Added `get-common-alignments`: To compute intersection between each of paired ends

0.9.10 (01-04-2016)

- `AlignmentMatrixFactory` can handle unmapped reads

0.9.8 (07-31-2015)

- Fixed a bug in `simulate-reads`: No more duplicate read ID's

0.9.7 (07-28-2015)

- Added `create-hybrid`: Build hybrid target directly using custom transcripts
- Added `simulate-reads`: Four nested models

0.9.6 (06-02-2015)

- `AlignmentPropertyMatrix` can represent an equivalence class
- Fixed a bug in length normalization
- Swapped Model ID's between 1 and 2
  - Model 1: Gene->Allele->Isoform (*)
  - Model 2: Gene->Isoform->Allele (*)
  - Model 3: Gene->Isoform*Allele
  - Model 4: Gene*Isoform*Allele (RSEM model)

0.9.5 (05-17-2015)

- Fixed length normalization: Depth = Count / (Transcript_Length - Read_Length + 1)

0.9.4 (02-23-2015)

- Fixed a bug in `prepare-emase`

0.9.3 (02-22-2015)

- Fixed a bug in Model 2 of handling multireads
- `run-emase` checks absolute sum of error (in TPM) for termination

0.9.2 (02-17-2015)

- Added three more models of handling multireads
  - Model 1: Gene->Isoform->Allele
  - Model 2: Gene->Allele->Isoform
  - Model 3: Gene->Isoform*Allele
  - Model 4: Gene*Isoform*Allele (RSEM model)

0.9.0 (01-31-2015)

- First release on PyPI
- Only implements RSEM model for handling Multiply-mapping reads (or multireads)

CHAPTER 7

Credits

Development Lead

- Kwangbom “KB” Choi <kb.choi@jax.org>

Contributors

- Narayanan Raghupathy <Narayanan.Raghupathy@jax.org>
- Glen Beane <Glen.Beane@jax.org>
- Al Simons <Al.Simons@jax.org>

CHAPTER 8

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/jax-cgd/emase/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

EMASE could always use more documentation, whether as part of the official EMASE docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/jax-cgd/emase/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *emase* for local development.

1. Fork the *emase* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/emase.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv emase
$ cd emase/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 emase tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/jax-cgd/emase/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_emase
```

CHAPTER 9

Indices and tables

- genindex
- modindex

Python Module Index

e

Index

R

S

U