# Mango group’s RNA-seq Web Tool Analyser Documentation

## Seth Jarvis, Mahin Khadjavi and Abdoulie Kaneth, Stephen Olowe

## Version release 0.1

# 1 INTRODUCTION

## PURPOSE

This is the technical documentation to accompany Mango group’s RNA-seq Web Tool Analyser. It is intended to describe the basic workings and architecture of our program along with the rationale behind our decisions in order to assist with potential future development.

## 1.2 SCOPE

The aim of our project was to develop a piece of software which allows a user to input a set of assembled transcripts with FPKM scores and output various forms of analysis such as PCA plots, volcano plots and heatmaps as well as tables of top 10 differentially expressed genes between each group of samples.

This version of the program is an early prototype and is currently limited in flexibility. It is only able to accept sets of samples which contain 3 different groups each containing 3 replicates. It can also currently only run if the samples are from Pteropus Alecto infected with Hendra virus.

## 1.3 DESIGN PHILOSOPHY

The core of our design philosophy was to use modular design. In designing the project we almost completely separated the 4 sections: GUI, sequence matching, matrix production, and analysis. This meant that, for the most part, progress of any one section was not reliant on progress in other sections. In taking a modular approach we also ensured that it is relatively easy to modify one segment without disrupting the whole pipeline as, as long as the points where methods interact aren’t disrupted, no section is necessarily reliant on any other, at least during development.

Ideally we would have liked to take the idea of modular design to its logical conclusion, and used the S.O.L.I.D. methodology. This would have involved using more objects and methods so that almost nothing is integrated too heavily. We however found that this was too time consuming given the short timeframe, and were forced to compromise.

We also tried to ensure that the architecture was kept reasonably simple and uncluttered which will help with future development. While it should be an idea which is more common sense than design philosophy, far too many people don’t take the effort do ensure their code is neat and well commented.

## 1.3 PACKAGES USED

BLAST+1: A program used to find the gene within a database which most closely matches the input sequence

Flask: a python microframework used for web development

Pandas2: a python package for handling data

Limma3: an R package for differential expression analysis

Biobase4: an R package for handling basic biological packages, used to create an expression set

gridExtra: an R package which allows easy output of tables to PDF

## 1.4 REQUIREMENTS TO RUN

The application requires python to be installed in order to run at all. It requires Flask in order to run the user interface. NCBI BLAST+ is required to run the BLAST script and pandas to produce the matrix. R along with the R packages limma, biobase, and gridExtra must be installed in order to run the analysis. As UNIX command line arguments are used at various points the program can only run in a UNIX environment.

## 1.5 OVERVIEW

Section 1 is the introduction to the program and is document

Section 2 provides a system overview

Section 3 contains the design of specific

Section 4 gives an overview of the limitations of our program and intentions for future development

Section 5 is the bibliography

# 2 SYSTEM OVERVIEW

## 2.1 SYSTEM ARCHETECTURE

Our system is built using python as the backend. The GUI is built using Flask to create a web interface, Flask then allows the user to submit a set of files. These files are stored in the folder Upload. The user is directed to the next page and when they press BLAST the main pipeline starts. The main pipeline consists of 3 scripts, ‘localBLAST.py’, ‘MatrixProduction.py’ and ‘Analysis.R’. These scripts are run sequentially from within the main by using subprocess.call.

Local BLAST is a python script which reads all files within the folder Upload, and fixes the header of each sequence in each file. The fixed files are then read and BLASTed sequence by sequence against the local database. The BLAST extracts the FPKM score and Accession number of each sequence and outputs them to a CSV file.

Matrix Production is a python script which copies all .csv files into the folder CSV, these are then sorted, fixed, and duplicates are removed. The resulting dataframes are combined using pandas into a single list and the resulting matrix is then output to a CSV file.

Analysis is an R script which imports the matrix produced by matrix production. The matrix is processed into a dataframe which is logged and then converted into an expression set. The expression set is run through limma to create contrasts between samples and find levels of differential expression. Toptable is used to find the top 10 differentially expressed genes between each two groups which are then output to the folder static. PCA and Volcano plots as well as heatmaps are also created and output to static.

The resulting plots and tables are then read by Flask and displayed to the user.

# 3 SECTION DESIGN

## 3.1 USER INTERFACE AND WEB DESIGN

### 3.1.1 HOW?

The User interface is written using Flask, is a microframework of python which uses Werkzeug and Jinja2 to render templates.

The web interface was created by using bootsraps which allows creation of CSS templates and HTML pages without much experience in manual creation. The required functions are imported from bootstrap into our templates.

Our main Flask script calls the CSS and HTML files which are rendered in a web browser; it then uses POST to allow a user to upload files. The files are then re-named in a way which is currently hardcoded. The user is moved onto the next page and Flask calls the function blast\_file when the button blast sequences is clicked. This function runs the other scripts sequentially.

After blast\_file has finished, the results of the R analysis which have been stored in the static folder and can be loaded into the HTML page results and displayed to the user.

### 3.1.2 WHY?

We were writing the app in python primarily so we looked for a web interface which was python friendly. We chose Flask because it was both reasonably user and is well used enough that support would be available if we needed it.

We chose to develop our web interface in bootstraps because, as none of us have any experience in web development, we wanted to go for the simplest way to create a good looking webpage.

Finally we used subprocess.call to integrate our other scripts for two reasons. First it was in keeping with our modular design philosophy, by not directly copying code into the main we could easily change out the modules if we design a version which works better. Second, Flask refused to recognise the files were present in the folder after upload until the page had been refreshed when the code was directly present, this issue did not occur when calling via a subprocess.

## 3.2 SEQUENCE MATCHING

### 3.2.1 HOW?

Our process of sequence matching has several distinct stages, it could probably be subdivided into 2 modules but we combined them for brevity. First we have written a script to fix the heads of the fasta files, ensuring that they all have ‘>’ and semi colons in the correct places and saving the fixed files.

Next, the main section is run. It relies on a combination of nested for loops, subprocess.call and NCBI BLAST+. For each file named ‘\*\_fixed.fasta’ in the folder ‘Upload’, the loop BLASTs each sequence against the custom database[[1]](#footnote-1) using ‘blast\_sequence’; a method which uses subprocess.call to run BLAST+ on the command line and returns the Accession number and FPKM score if a match is found and a comma if there is no match. This is then output to a CSV

### 3.2.2 WHY?

Database sequence similarity searching is very important to obtain information about unknown gene, protein, in order to predict their functionality. There are different methods, but time and accuracy are factors that determine which method to use, especially when working with big data set like ours. We decided that BLAST+ was the best choice for our dataset because we felt it provides the best balance between accuracy, and computational efficiency. It is also the most well documented and supported which was useful due to the time frame in which we were completing our project.

In our tests, on a moderately powerful computer, our program blasted sequences locally in approximately 0.18s. In the original files provided there are 1,564,678 lines containing 782339 sequences across 9 files. This means that, all things being equal, it would take a little over 39 hours of uninterrupted computer time. This may seem like a lot but remote blast is significantly slower.

When running blast remotely, we have to consider; consistency of Internet connection, network traffic, and process caps set by the NCBI, and so on. It therefore seemed sensible to create a local database to blast against and run blast in that way. While it is disadvantageous that it is difficult to blast exclusively against a particular species without creating custom databases, the advantages gained, particularly in speed, the ability to scale the process – if we were to use a server or web services – and increasing the amount of control we have over how the blast runs seemed to more than make up for it.

We chose to implement local blast by using subprocess.call rather than other methods such as the wrapper which biopython provides there is little functional difference and it requires fewer extra packages to be installed.

When running local blast there are 2 main options: running against a full database or running against a custom database. We decided to use a custom database because it is both much faster than, and takes up far less space than the full nucleotide database. It also ensures that genes from the wrong organisms aren’t found.

## 3.3 MATRIX PRODUCTION

### 3.3.1 HOW?

Matrix production is a relatively simple script. It first moves the CSV files from Pandas to the folder CSV. The script then imports each CSV file as a pandas data frame, removes all invalid lines, sorts the dataframe by ascession number then by FPKM, and removes any duplicate Accession numbers, keeping the highest FPKM score –it keeps the first item and the lists are sorted in descending order – the fixed lists are then added to the overall matrix by using pandas merge. Pandas is then used to insert 0.01 into any space which has nothing in it and to replace any 0s with 0.01 so they don’t break the log used in the analysis. The final dataframe is output to a CSV.

### 3.3.2 WHY?

Pandas is a very powerful tool for handling data within python. It made both the process of importing the CSV files easy and the process of fixing them and merging them into a single data frame far easier. It made handling the data far easier than using vanilla python and we couldn’t find another method which could to do what was needed which was any easier or better than Pandas.

## 3.4 DIFFERIENTIAL EXPRESSION ANALYSIS

### 3.4.1 HOW?

The combined matrix created by the previous stage is imported into R. It is then converted into a usable form, setting the ascession codes as the row names, renaming the columns and logging the data.

The model matrix is created; this is a way of grouping the samples together and naming the groups. It is then attached to the data matrix using the function ExpressionSet from biobase.

A linear model is produced from the expression set. The contrasts between samples are created and the contrast matrix is used to fit the contrasts and analyse the levels of differential expression between all groups. Emperical Bayes methods are run on the contrasts matrix (using eBayes) which creates an object with differences between every group in every gene.

The model created using eBayes is hugely flexible and is used for the rest of our analysis. From it we create the list of the top 10 differentially expressed genes between each group by using top table as well as creating our PCA plot, volcano plot and heatmap of the top 100 differentially expressed genes. All of these results are output to the static folder and are then called by the GUI to be output to the user.

### 3.4.2 WHY?

When analysing differential expression of samples, almost all methods used for visualising the differential expression and a large number for performing the analysis are within R. Given as R is the analysis language with which we are the most confident, using R to perform the analysis was a no brainer.

What was substantially more difficult to decide was how to do the analysis. When given raw reads, there are a large number of methods which can be used such as DESeq and the voom function of limma. However, our program required that we run the analysis using FPKM scores rather than raw reads. This proved substantially more challenging to do.

According to most advice online, our best bet was to reverse engineer the raw reads from the library size. However, this would have been both time consuming to figure out and we felt went against the spirit of the project brief.

One of the members of our group had previous used limma when analysing FPKM scores and, although the results were from cuffdiff this seemed like a promising start. After some time we found a post5 by the creator limma saying that if you absolutely have to use FPKM scores you should log the samples and then go on to create a fit and eBayes for the analysis. We didn’t manage to find a better suggestion for producing the results so this was the method we used.

# 4. LIMITATIONS AND DEVELOPMENT

Both sections in this part are in rough descending order of the urgency/importance of the point.

## 4.1 CURRENT LIMITATIONS

1. The files currently may get mixed up at various points in the process: the file renaming currently brute forces the file names rather than allowing the user to specify which group each sample belongs to, matrix production also doesn’t save the file names as column headers meaning that the files may be given the wrong names in analysis
2. As the folders are hardcoded, samples are renamed to a very specific format, and the BLAST and analysis take all files in a folder, running multiple samples concurrently is not currently possible
3. The format of samples is not currently flexible, 3 groups each containing 3 samples
4. We can currently only BLAST against the custom database containing sequences from *Pteropus Alecto* and Hendra Virus
5. The software as a whole is exceptionally slow, the bottleneck is the BLAST function
6. All samples must have their expression levels as FPKM rather than raw reads
7. The program currently chooses the top FPKM for each accession number if there are multiple transcripts corresponding to the same gene which is a suboptimal way of dealing with the problem

## 4.2 OPPORTUNITIES FOR FUTURE DEVELOPMENT

1. Preserve the sample names in the matrix production so they aren’t hardcoded in during the analysis
2. Group samples in a softer manner when sending them through the pipeline rather than re-naming them to allow for scalability
3. Use a better method of assigning FPKM scores to Accession numbers in the case that multiple transcripts correspond to the same Accession number
4. Allow users to submit samples in a format other than 3 groups of samples each with 3 replicates
5. Allow users to select the group each sample belongs to and name samples
6. Allow use of user submitted databases or allow users a choice of multiple databases
7. Run the software on a server or cloud platform such as Amazon web services or apocrita to speed up the BLAST
8. Test BLASTing whole files at once or multiple sequences at once and examine the effect on program speed.
9. Allow users to save analysis they’ve done previously and look at them whenever they like; be that by logging in, by using a URL they’ve been given or by entering a job number
10. Add a system to email users when their BLAST is finished so they don’t have to wait with the page open for their BLAST to run
11. Allow users to save plots easily
12. Allow for the users to view more detail about which form of a gene or which section of a gene is particularly expressed

# 5 BIBILIOGRAPHY

1. Camacho, C. *et al.* BLAST+: architecture and applications. *BMC Bioinformatics* **10,** 421 (2009).

2. McKinney, W. Data Structures for Statistical Computing in Python. *Proc. 9th Python Sci. Conf.* **1697900,** 51–56 (2010).

3. Ritchie, M. E. *et al.* limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Res.* **43,** e47 (2015).

4. Huber, W. *et al.* Orchestrating high-throughput genomic analysis with Bioconductor. *Nat Methods* **12,** 115–121 (2015).

5. Smyth, G. Question: Differential expression of RNA-seq data using limma and voom(). *Bioconductor* (2013). at <https://support.bioconductor.org/p/56275/>

1. Our custom database was created by extracting the relevant files from the *Pteropus Alecto* and Hendra Virus genomes and running the following commands:

   Cat GCF\_000852685.1\_ViralProj14911\_cds\_from\_genomic.fna Gnomon\_mRNA.fsa rna.fa > BatVirus.fna

   makeblastdb -in BatVirus.fna -parse\_seqids -dbtype nucl -out BtVrDb

   The files form the database were then moved into the folder Database within our main to be blasted against [↑](#footnote-ref-1)