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

## Seth Jarvis, Mahin Khadjavi, Abdoulie Kanteh, and 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 because, 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[[1]](#footnote-1). 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 to ensure their code is neat and well commented.

## 1.4 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

## 1.5 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.

## 1.6 OVERVIEW

Section 1 is the introduction to the program and is document

Section 2 provides a system overview

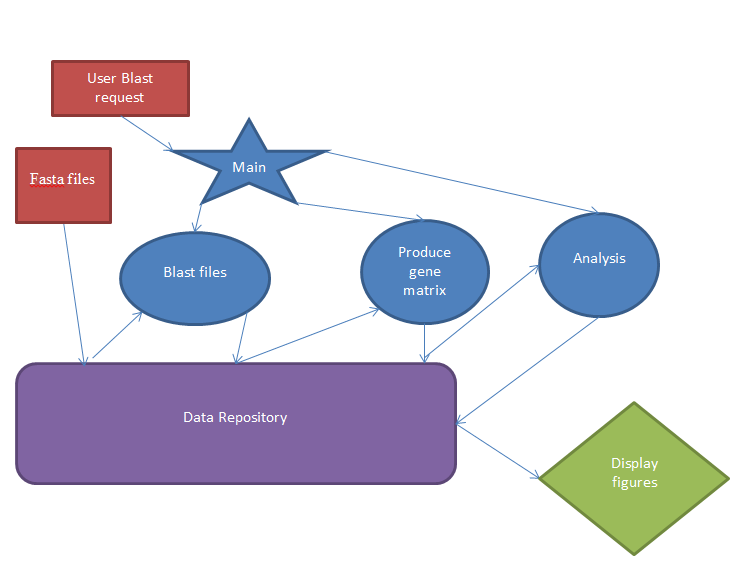
Section 3 contains the design of specific

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



***Figure 1****, diagram of the system archetectures: the flow of data or instructions is represented by arrows, red squares represent user inputs, blue shapes represent things which happen behind the scenes (star is the main, circles are functions), the purple rounded rectangle represents all the data stores and green shapes representing things displayed to the user*

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.

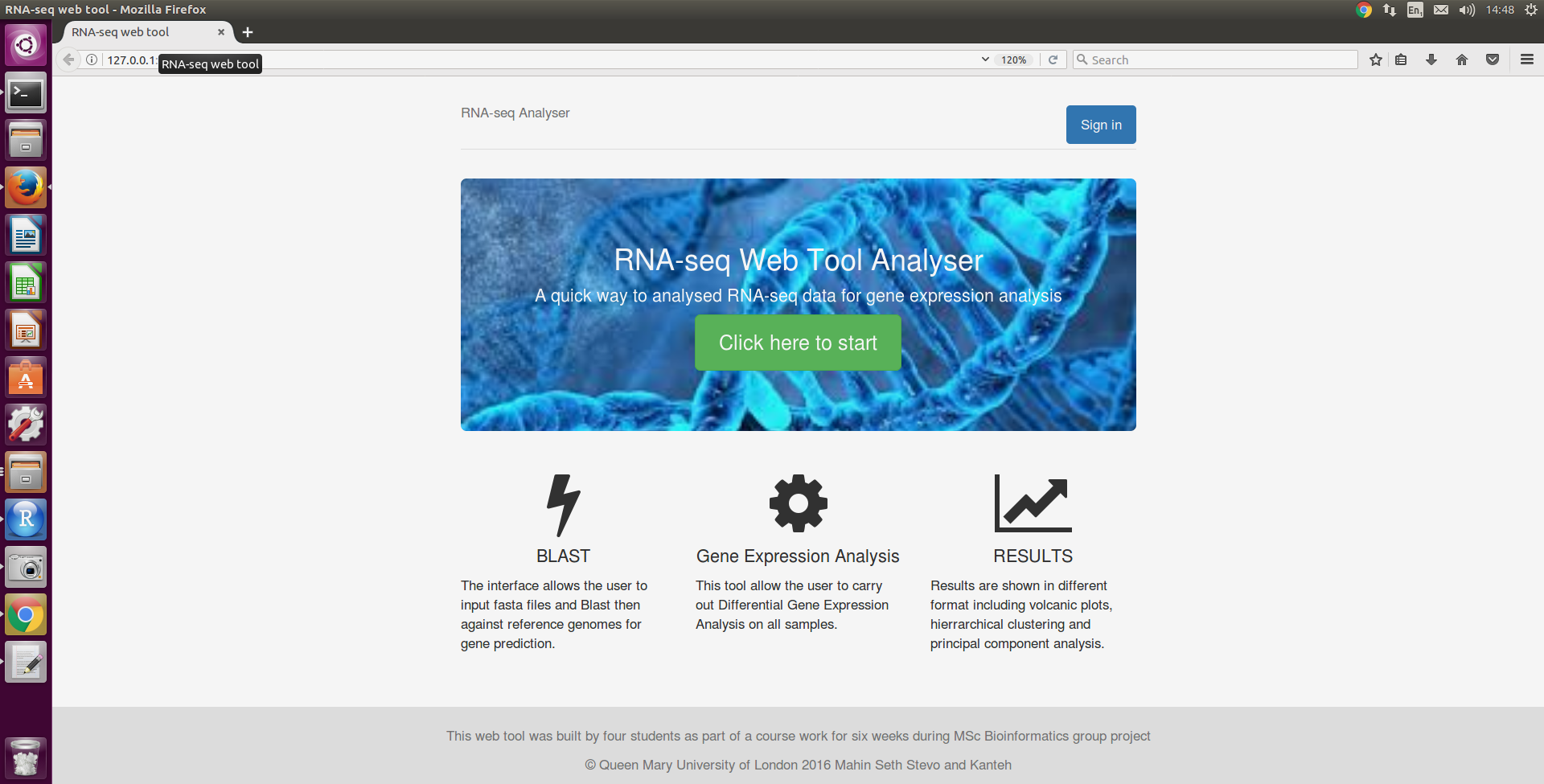
Local BLAST is a python script which reads all files within the folder Upload, and fixes the header of each sequence in each file. The fixed files are then read and Blasted sequence by sequence against the local database. The BLAST extracts the FPKM score and Accession number of each sequence and outputs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

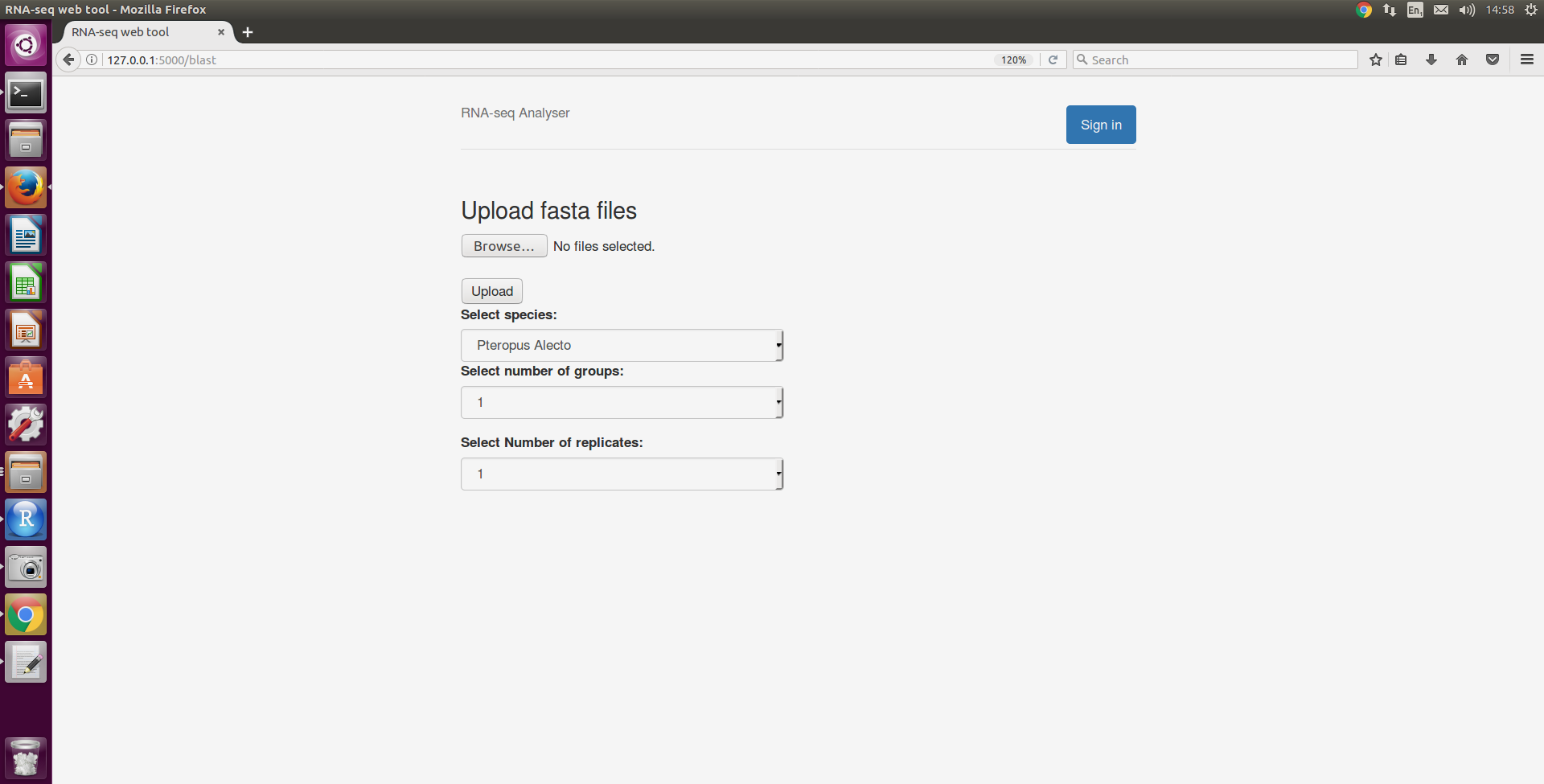


***Figure 2,*** *screenshot of the homepage*

### 3.1.1 HOW?

The User interface is written using Flask, 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.



***Figure 3****, screenshot of file upload page*

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.

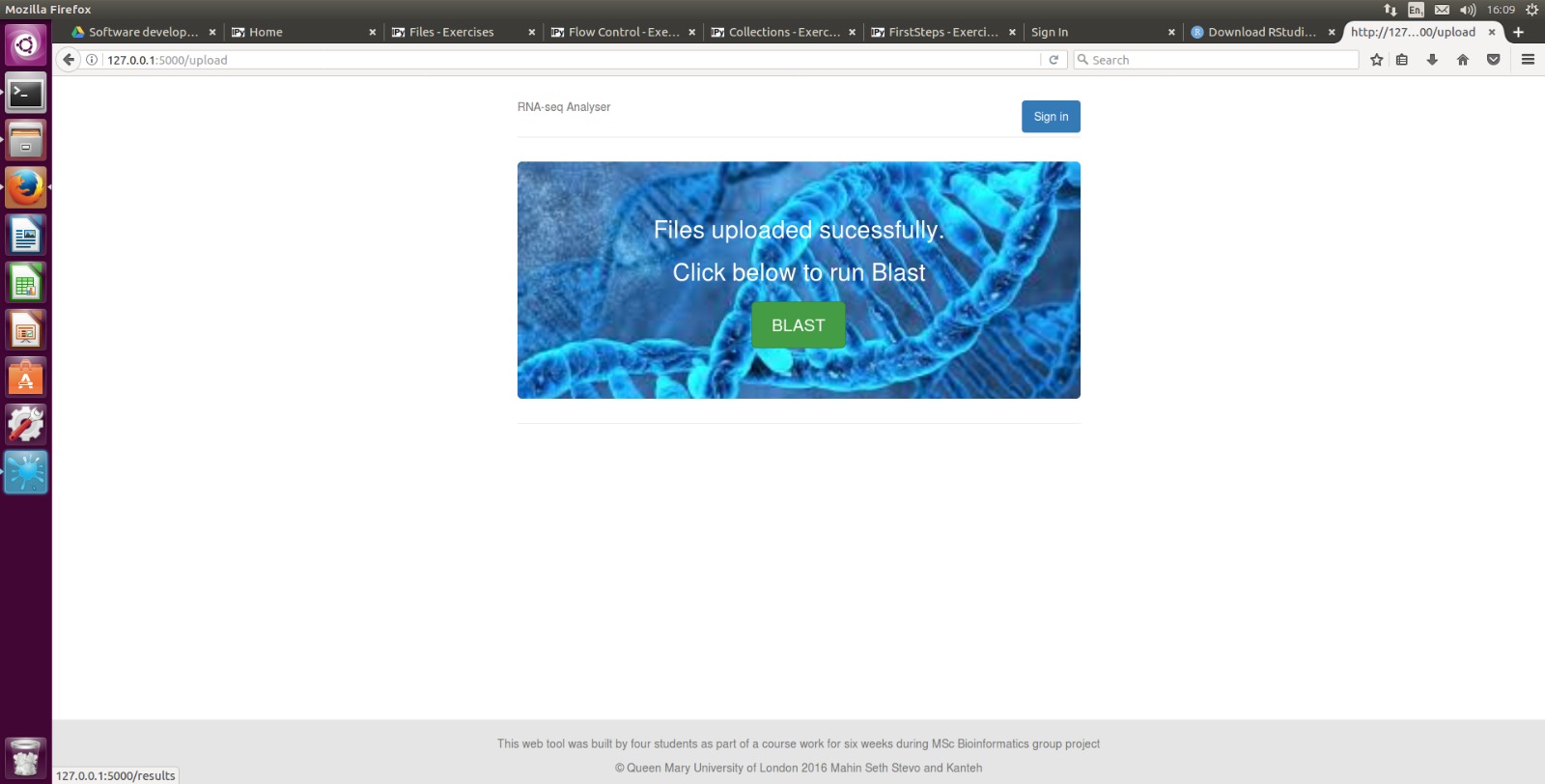
### 3.1.2 WHY?

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 that 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



***Figure 4,*** *Screenshot of the BLAST page*

### 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 [[2]](#footnote-2) 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 as there is little functional difference and 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

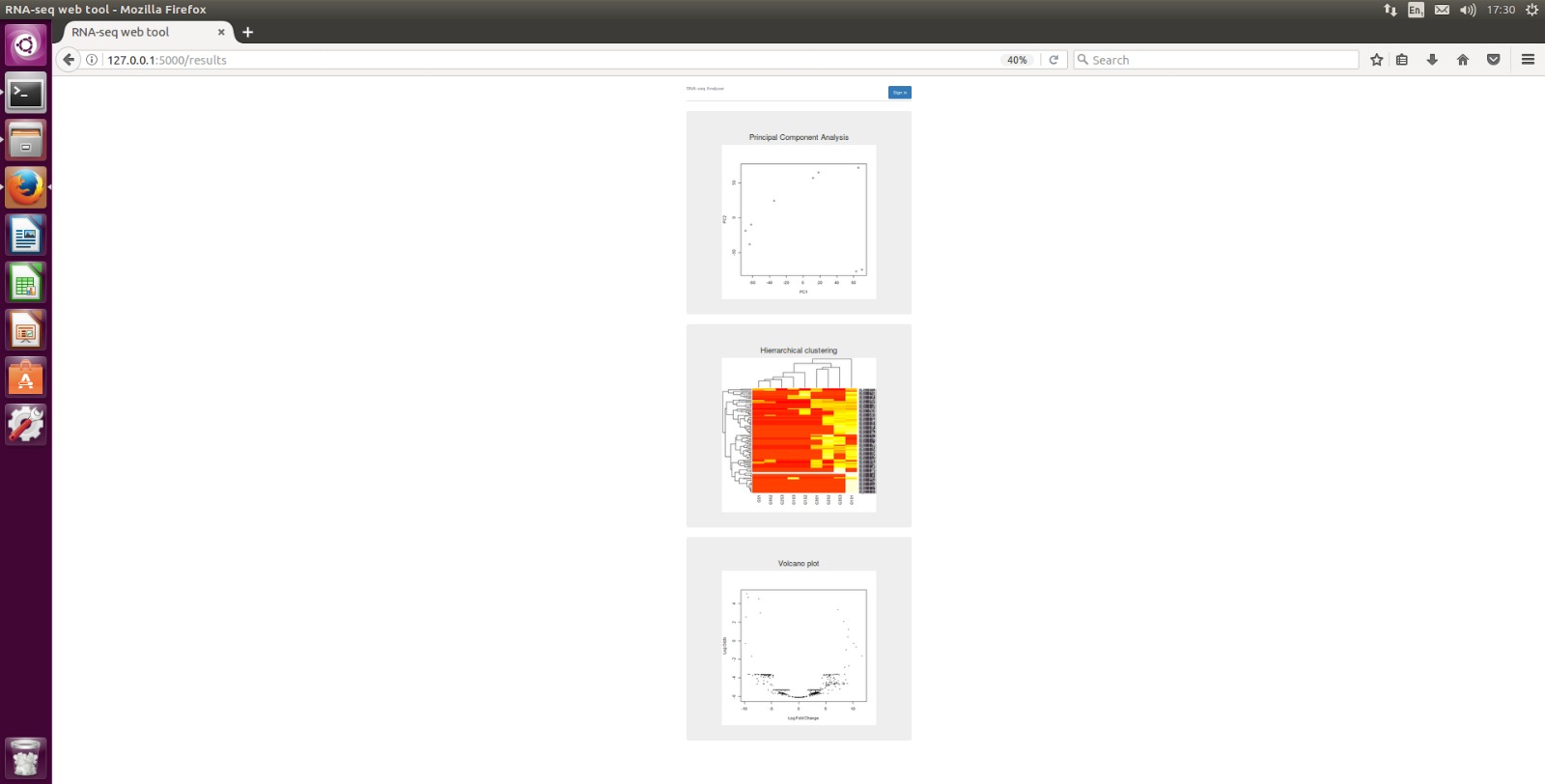
### 3.3.1 HOW?

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.

***Figure 5****, screenshot of our the plots displayed within the webpage*

### 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 that 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 counts, 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 counts. This proved substantially more challenging to do. This was because most packages that we found seemed to do their own normalisation, if they tried to run normalisation on already normalised reads you would get effectively garbage out.

According to most advice online, our best bet was to reverse engineer the raw counts 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

## 4.1 CURRENT LIMITATIONS

Our program is currently very much in a proof of concept stage. This means it’s sadly very restricted in its capabilities. We have had to make a lot of compromises from our initial vision of the program due to bugs and time constraints.

Our biggest current issue is that samples aren’t necessarily sorted or labelled correctly. When the files are imported they are renamed to a certain format, this re-naming doesn’t always seem to name files in the correct order. The files are also sent through the pipeline and, once again, their order doesn’t seem to be preserved when merging the files into a single matrix post BLAST. This means that in their current state, we can’t be sure that the sample names in the group are correct.

In a similar vein, due to the fact we have had to hardcode a lot of functions and variables to save time there is almost no flexibility for the user. They have to use our one custom database and have to submit samples with 3 groups of samples each containing 3 replicates. Also, because the samples are re-named we can’t currently support multiple sets of samples running at one time.

Another limitation related to lack of flexibility is that our program can only take FPKM scores and not raw counts, in most cases people will not have FPKM scores on their assembled transcript, in the cases where they do they will often have had them created as part of another pipeline such as cuffdiff. Also, when collapsing duplicate ascession numbers with different FPKM scores we currently just take the top FPKM score rather than something which is possibly more biologically valid.

Due to time constraints we couldn’t successfully output the tables of differentially expressed genes to the webpages. Finally, due to the way we have implemented BLAST, it is exceptionally slow as blasting sequences line by line is an exceptionally computationally intensive process.

## 4.2 OPPORTUNITIES FOR FUTURE DEVELOPMENT

Given the early stage of our program there are a lot of improvements we would like to make. These broadly fall into improving file handling, improving blast speed, improving analysis and improving user experience.

The way we handle files is less than ideal. As far as improvements go, first we would like to make sure that samples are correctly ordered and grouped as they should be in the analysis. This could take several forms: if we wished to keep our current method of re-naming files it’s possible that by sorting the files by name, both at the stage of re-naming and again matrix production, we could ensure that the order is correct when imported into R. If we wanted to change the method which we do it we could store the file names as a variable. Either of these methods would allow the samples to be grouped by users.

Because the files are re-named to fit a certain format we can’t currently run multiple samples at once. The obvious way to handle this is to assign job numbers. This would allow files to either be actually sorted by job no. into folders or would allow us to create an index for each job telling us which samples belong to which group in which job.

Using job numbers would also allow us to improve user experience. Instead of users having to wait on the page for their analysis to run, we would be able to assign them a job number and either create an account which they can use to look at all their past jobs or just allow them to return to their analysis by using their job number. We could also create an email system to let people know when their job is done.

We had 2 ideas for how to speed up BLAST, first we could use a server or cloud hosting service to run our program. This would both speed up our analysis and would provide better architecture for sending emails to users or logging in. Second we could improve the way in which we BLAST the files, either by blasting whole files at once or by running multiple sequences in parallel.

We would also like to improve the flexibility of our program. Each set of samples will often have different numbers of replicates and different numbers of samples; it would be ideal if we could allow users to say how many groups they had, how many replicates are in each group, which samples were part of which groups, and name each of the groups.

When it comes to improving flexibility it would also be ideal for users to be able to change the databases they are blasting against as well as the analysis they would like, imputing information such as the type of analysis they want done, and whether they have raw counts or FPKM scores. We could then change the analysis according to their requests.

In terms of analysis, the biggest flaw is that currently, in the case of multiple transcripts we select the highest FPKM score. This is not necessarily very biologically valid and, in an ideal world we would be able to look at the levels of expression of each area of a gene, using another file format such as GFF files with reads instead of our current CSV format. This would also allow us to give users more granularity in their analysis, showing them which regions of a gene are particularly over/under expressed.

Finally, to improve user experience we would like to give users access to the CSV matrix in case they want to do their own analysis and allow them to easily save both the plots and the tables of differentially expressed genes.

# 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. S.O.L.I.D. is an acronym made of acronyms. It stands for: Single responsibility principle i.e. a class should only have a single repsonsibilty, Open-Closed principle i.e. objects/classes should be open to have more things added to them but their base function shouldn’t change, Liskov substitution principle i.e. every subclass can be a substitution for their parent class, Integration segregation principle i.e. clients shouldn’t be forced to see/use things which they don’t want to, and Dependency Inversion principle i.e. high level classes should not be dependend on low level clases, they should both be based on abstractions so changing one won’t break the other. [↑](#footnote-ref-1)
2. 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-2)