Supplementary Appendix 1

Scripts

1. **FASTQ_maker.py**

```
# this script generates a FASTQ file from raw sequencing read files

import string

forwardRead = open('./filename.reads', 'r')
reverseRead = open('./filename.reads', 'r')
qualityRead = open('./filename.qvals', 'r')

forwardOutput = open('./ForwardOutput', 'w')
reverseOutput = open('./ReverseOutput', 'w')

def writeToFile(inputFile, qualityInput, outputFile):
        #defines the function; does not run until commanded to do so
        isThereInputLeft = inputFile.readline()
        if len(isThereInputLeft) == 0:
                return False
        outputFile.write('@' + isThereInputLeft[1:])
        outputFile.write(inputFile.readline())
        outputFile.write('+\n')
        qualityInput.readline()
        outputFile.write(qualityInput.readline())
        return True

while True:
#this commands function to run; goes back to definition of function
        if writeToFile(forwardRead, qualityRead, forwardOutput) == False:
                break
        if writeToFile(reverseRead, qualityRead, reverseOutput) == False:
                break
```

2. **FASTQ_Ftrimmer.py**

```
# this script trims the transposon information off of the sequencing reads to improve
mapping

import string
import sys

fastqfile = open('ForwardOutput.fastq','r')
#change the name of the FASTQ file to your respective file

trimmedfastq = open('filename_trim.fastq','w')
#change the name of the output to your desired name

def trimfastq(inputfile,outputfile):
   # Read in the header and strip any white space from the right
   header = inputfile.readline().rstrip()
```

```
    # Terminate if there is no header
    if len(header) == 0:
        return False

    # Read in the rest of the information and strip any white space on the right
    seq = inputfile.readline().rstrip()
    extra = inputfile.readline().rstrip()
    quality = inputfile.readline().rstrip()

    # Find the position of the transposon
    transposon = seq.find('TGTTA') # this identifies the position of the transposon

    # Trim the sequence and quality information
    # Take (transposon + 3) up to (transposon + 50) the 50 value can be modified to
whatever length you desire
    trimtn = seq[transposon + 3 : transposon + 50] # this takes 50 nt following the 'TA' of
the transposon
    trimqual = quality[transposon + 3 : transposon + 50]

    if len(trimtn) > 0:
        # Write to the output file
        outputfile.write(header + '\n')
        outputfile.write(trimtn + '\n')
        outputfile.write(extra + '\n') #this writes the '+' for the FASTQ file
        outputfile.write(trimqual + '\n')

    return True



while True:
    if trimfastq(fastqfile,trimmedfastq) == False:
        break
```

3. FASTQ_Rtrimmer.py

```
# this script trims the adaptor information off of the sequencing reads to improve mapping

import string
import sys

fastqfile = open('ReverseOutput.fastq','r')
#change the name of the FASTQ file to your respective file

trimmedfastq = open('ReverseOutput_trim.fastq','w')
#change the name of the output to your desired name

def trimfastq(inputfile,outputfile):
    # Read in the header and strip any white space from the right
    header = inputfile.readline().rstrip()

    # Terminate if there is no header
    if len(header) == 0:
        return False
```

```
    # Read in the rest of the information and strip any white space on the right
    seq = inputfile.readline().rstrip()
    extra = inputfile.readline().rstrip()
    quality = inputfile.readline().rstrip()

    # Find the position of the adaptor
    adaptor = seq.find('TTGTG') # this identifies the position of the adaptor

    # Trim the sequence and quality information
    trimadapt = seq[adaptor + 5:] # this takes all nt following the end of the adaptor (which
ends with TTGTG)
    trimqual = quality[adaptor + 5:]

    if len(trimadapt) >= 0:
        # Write to the output file even lines that don't contain TTGTG to get equivalent # of
lines for alignment (if the alignment program requires equal numbers of lines for F and R
reads)
        outputfile.write(header + '\n')
        outputfile.write(trimadapt + '\n')
        outputfile.write(extra + '\n') #this writes the '+' for the FASTQ file
        outputfile.write(trimqual + '\n')

    return True



while True:
    if trimfastq(fastqfile,trimmedfastq) == False:
        break
```

4. TA_finder.py

```
# This script finds and reports the position of all TA motifs in a genome
# input:  python TAfinder.py argv[1] > output_file.txt
# argv[1] is the fasta sequence of the organism downloaded form NCBI
# The output file has 2 columns:  1) misc_feature; 2) genome position of the TA site on
the Forward (+) strand

import sys

fasta={}
fasta[1]=[]

for line in open(sys.argv[1]):
    split=line.split('\t')
    if split[0][0]!=">":
        read=split[0].rstrip()
        fasta[1].append(read)

genome="".join(fasta[1])

TAs={}
TAposition=-1
length=len(genome)
newposition=0
```

```
for x in range(0,length):
    position=genome[newposition:].find("TA")
    TAposition = position + TAposition + 1
    TAs[int(TAposition)+1]=[]
    newposition = TAposition + 1

TA_keys=TAs.keys()
TA_keys.sort()

for i in TA_keys:
    print "misc_feature\t%d" %(i)
```

5. Readcounter_SAMparse.py

```
# this script parses a SAM file (from Bowtie mapping) to tally read counts at TA sites
# argv[1] is TAgb file for Chr1
# arvg[2] is mapped reads SAM format (single end mapping)
# argv[3] is gene list (no header)

import sys

chrITA={}

for line in open(sys.argv[1]):  # opens files line by line
    split= line.split()         #splits string into columns--treates consecutive spaces as 1
single tab
    TA = int(split[0])
    chrITA[TA]=[0,0]
#counts reads at each TA site from SAM output

for line in open(sys.argv[2]):
    split=line.split('\t')
    if len(split) > 8: #ignores headers (analyzes rows that have more than 8 columns only)
        if split[1]=='0':  #if mapping is to the plus strand
            site=int(split[3])
            if site in chrITA:
                chrITA[site][0] += 1
        if split[1]=="16": #if mapping is on the minus strand
            site=int(split[3])-2 + len(split[9])
            if site in chrITA:
                chrITA[site][1] += 1  #if read has been found before, tally 1 more

# assign gene names

gene_dict = {}; prev_end = 0; IG = 'IG_1'; prev_chrom = 2

for line in open(sys.argv[3]):
    split = line.split('\t')
    gene = split[0]; start = int(split[4]); end = int(split[5])
    gene_dict[gene]=[start, end]
    IG = 'IG_' + gene
    if start > prev_end+1:
        gene_dict[IG]=[prev_end+1,start-1]
    prev_end = end
```

```
gene_dict['IG_chrmEnd'] = [4410930,4411532] #this is the intergenic region from the last gene to the end of the organism's chromosome. one must adjust last value for organism of interest.

# store gene names into TA site dictionary

for k,v in gene_dict.iteritems():
    start = v[0]; end = v[1];
    for i in range(start,end+1):
        if i in chrITA: chrITA[i].append(k)


#print

CI_sites = chrITA.keys()
CI_sites.sort()

for i in CI_sites:
    print 'H37Rv', '\t',  i, '\t',  i+1, '\t', chrITA[i][2], '\t', chrITA[i][0], '\t', chrITA[i][1]
```

6. TA_percent.py

```
#argv[1] = readcounter input file from after mapping (e.g. from Readcounter_SAMparse.py)

import sys

TA = 0; Reads = 0; HitSites = 0
readcounts = []

for line in open(sys.argv[1]):
    split=line.split()
    if split[4].isdigit() == True:
        readcounts.append(int(split[4])+int(split[5])) # record read counts in list
        TA += 1
        Reads += int(split[4])+int(split[5])
        if (int(split[4])+int(split[5])) > 0: HitSites += 1

Percent = int(((float(HitSites)/int(TA)) * 100) + 0.5)

print 'TA = %d' % (TA)
print 'Reads = %d' % (Reads)
print 'Sites Hit = %d' % (HitSites)
print 'Percent TAs hit = %d' % (Percent)
print 'Average Read Count = %d' % (float(sum(readcounts))/float(len(readcounts)))
```

7. TAcounts_gene.py

```
#argv[1] = readcounter input file from after mapping

import sys

TA = 0; Hit = 0; Ratio=0; Reads=0
readcounts = {}
```

```
readcounts['IG_1']=[0,0,0,0,0]

for line in open(sys.argv[1]): #use file with IG and Genes
    split=line.split('\t')
    if len(split)>1:
        if split[3] in readcounts:
            TA +=1
            if float(split[4]) + int(split[5]) > 0: # to get floated ratios
                Hit += 1
            Ratio = Hit/float(TA)
            Reads = Reads + int(split[4]) + int(split[5])
            Avg= Reads/float(TA)
            readcounts[split[3]] = [TA, Hit, Ratio, Reads, Avg]
        if split[3] not in readcounts:
            TA = 0
            Hit = 0
            Ratio = 0
            Reads=0
            TA +=1
            if float(split[4]) + int(split[5]) > 0:
                Hit += 1
            Ratio = Hit/float(TA)
            Reads = Reads + int(split[4]) + int(split[5])
            Avg = Reads/float(TA)
            readcounts[split[3]] = [TA, Hit, Ratio, Reads, Avg]

sort = readcounts.keys()
sort.sort()

print "Locus",'\t',"# of hit TAs",'\t',"Total TAs",'\t',"Fraction TAs hit",'\t',"Total
reads",'\t',"Avg_Reads per TA"

for i in sort:
    hit_fraction = readcounts[i][2]
    total_TA = readcounts[i][0]
    hit_TA = readcounts[i][1]
    total_reads = readcounts[i][3]
    avg_reads = readcounts[i][4]
    print "%s\t%d\t%d\t%.3f\t%d\t%.1f" %
(i,hit_TA,total_TA,hit_fraction,total_reads,avg_reads)

## columns are as follows:  1) locus, 2) #TA sites hit, 3) #total TA sites;
    ## 4) fraction of TA sites hit; 5) total reads/locus; 6) Avg reads/locus
```

8. DNAplotter.py

```
This script turns readcounter files into a DNA plotter file
# input:  python DNAplotter.py argv[1] argv[2] > output.txt
# argv[1]= the size of genome in base pairs
# argv[2]= readcounts (output of Readcounter_SAMparse.py)
# The output file in a single column in which every row is a nucleotide in the genome.
# The number of read counts for each nucletoide is reported.

import sys
```

```
from math import *

Allsites = {}
TAsites = {}
DNAplot = {}

for i in range(int(sys.argv[1])):
    Allsites[i]=[0]

for line in open(sys.argv[2]):
    split = line.split('\t')
    reads = int(split[4]) + int(split[5])
    TAsites[int(split[1])]=[reads]

for i in Allsites:
    if i in TAsites:
        DNAplot[i]=[TAsites[i][0]]
    else:
        DNAplot[i]=[Allsites[i][0]]

sites = DNAplot.keys()
sites.sort()

for i in sites:
    print DNAplot[i][0] #DNA plotter doesn't want the basepair info--it assumes each line is
a new basepair position

# this script will have the right # of bp, but the position is staggered by 1
# since range include 0, so every basepair will be one less than it should be
```

9. Artemize.py

```
# this program converts a readcounter file to a mapped Artemis plot for visualization
#argv[1] = readcounter input file from after mapping

import sys

TA={}

for line in open(sys.argv[1]):
    split=line.split()
    TAsite=int(split[2])
    F = int(split[5])
    R = int(split[6])/-1
    TA[TAsite]=[F,R]

TA_keys =TA.keys()
TA_keys.sort()

print "%s\t%s\t%s" % ("#BASE","Finsert","Rinsert")
print "%s\t%s\t%s" % ("colour","5:150:55","225:0:0")
for i in TA_keys:
    print "%d\t%d\t%d" % (i,TA[i][0],TA[i][1])
```

10. **geomean.m**

```
gene=gene+1;
gene_CtrlSims =ctrl_gene+1;
GeneGeoMeanRatio=zeros(length(uniqueindices),size(gene_CtrlSims,2));
GeneGeoMeanDiff=zeros(length(uniqueindices),size(gene_CtrlSims,2));
GeneGeoMeansCtrl=[];
GeneGeoMeansExp=[];

gene_CtrlSims =mean(gene_CtrlSims,2);

for x=1:length(uniqueindices); %goes locus by locus in 1 column
   p=uniqueindices(x,1):uniqueindices(x,2); %takes the first TAsite and last TA site in the
locus
   A=min(p);
   B=max(p);
   ctrl_geomean=geomean(gene_CtrlSims (A:B));
   GeneGeoMeansCtrl(x,1)=ctrl_geomean;
   exp_geomean=geomean(gene(A:B));
   GeneGeoMeansExp (x,1)=exp_geomean;
   GeneGeoMeanRatio (x,1)=exp_geomean/ctrl_geomean; %calculates the ratio
between the samples
   GeneGeoMeanDiff (x,1)=ctrl_geomean-exp_geomean;
end
```