*using* FASTX

*using* BioSequences

*using* DataFrames

*using* CSV

*using* StatsBase

*using* StatsPlots

*using* CSV

*####################################################################################################################*

*# PART 1*

*#Extracting sequences with the correct 5' and 3' end priming regions and trimming to the matching regions*

*####################################################################################################################*

*# Priming regions and reference sequence*

PH\_pET23\_DA\_Biotin3 = dna"AGGGTtAATGCCAGC"

P2\_fidelity\_inestR1 = dna"CTTGCGGCcacacAG"

Fidelity\_ref = dna"AGGGTtAATGCCAGCGCTTCGTTAATACAGATGTAGGTGTTCCACAGGGTAGCCAGCAGCATATGGTGCAGGGCGCTGACTTCCGCGTTTCCAGACTTTACGAAACACGGAAACCGAAGACCATTCATGTTGTTGCTCAGGTCGCAGACGTTTTGCAGCAGCAGTCGCTTCACGTTCGCTCGCGTATCGGTGATTCATTCTGCTAACCAGTAAGGCAACCCCGCCAGCCTAGCCGGGTCCTCAACGACAGGAGCACGATCATGCGCACCCGTGGCCAGGACCCAACGCTGCCCGAGATCTCGATCCCGCGAAATTAATACGACTCACTATAGGGAGACCACAACGGTTTCCCTCTAGAAATAATTTTGTTTAACTTTAAGAAGGAGATATACCATGGATCCTCTAGAGTCGACCTGCAGGCATGCAAGCTTGCGGCcacacAG"

ref = Fidelity\_ref

*############# loading FASTA & convert to Strings of Chars ##############*

*function* load\_fasta\_file(file\_name::String)

    seqs = []

    reader = open(FASTA.Reader, file\_name)

*for* record *in* reader

        seq = FASTX.FASTA.sequence(record)

        push!(seqs, seq)

*end*

    close(reader)

*return* seqs

*end*

*############# finding fragments based on ID ##############*

*function* find\_and\_trim(data)

    seqs = []

*for* i *in* 1:length(data)

        pos1 = findfirst(ExactSearchQuery(PH\_pET23\_DA\_Biotin3), data[i])

        pos2 = findfirst(ExactSearchQuery(P2\_fidelity\_inestR1), data[i])

*if* isnothing(pos1) || isnothing(pos2)

*continue*

*end*

        trimmed\_seq = data[i][pos1.start:pos2.stop]

        push!(seqs,trimmed\_seq)

*end*

*return* seqs

*end*

*# export array as fasta file*

*function* array\_to\_fasta\_file(file\_name::String, DATA)

    reader = open(FASTA.Writer, file\_name)

*for* i *in* 1:size(DATA)[1]

            seq = DATA[i]

            record = FASTA.Record("Seq$i", seq)

            write(reader, record)

*end*

    close(reader)

*end*

*# Running functions*

EXO\_seqs =load\_fasta\_file("Galaxy250-EXO-THR.fasta")

DEL\_seqs =load\_fasta\_file("Galaxy249-DEL.fasta")

D12A\_seqs =load\_fasta\_file("Galaxy248-D12A-THR.fasta")

P2NEB\_seqs =load\_fasta\_file("Galaxy251-P2NEB.fasta")

EXO = find\_and\_trim(EXO\_seqs)

DEL = find\_and\_trim(DEL\_seqs)

D12A = find\_and\_trim(D12A\_seqs)

P2NEB = find\_and\_trim(P2NEB\_seqs)

*#Output trimmed fasta files*

array\_to\_fasta\_file("EXO.fasta", EXO)

array\_to\_fasta\_file("DEL.fasta", DEL)

array\_to\_fasta\_file("D12A.fasta", D12A)

*#adding manually the reference sequence as 'ref' to each fasta file before alignment*

*####################################################################################################################*

*#PART 2*

*#Processing multiple sequence alignments*

*####################################################################################################################*

*#Loading multiple sequence alignments*

Dic\_EXO = Dict{String, LongSequence}()

reader = FASTA.Reader(open("EXO\_MSA.fasta", "r"))

*for* record *in* reader

        Dic\_EXO[identifier(record)] = sequence(record)

*end*

close(reader)

Dic\_D12A = Dict{String, LongSequence}()

reader = FASTA.Reader(open("D12A\_MSA.fasta", "r"))

*for* record *in* reader

        Dic\_D12A[identifier(record)] = sequence(record)

*end*

close(reader)

Dic\_DEL = Dict{String, LongSequence}()

reader = FASTA.Reader(open("DEL\_MSA.fasta", "r"))

*for* record *in* reader

        Dic\_DEL[identifier(record)] = sequence(record)

*end*

close(reader)

*# extracting reference sequence from alignment*

EXO\_template = Dic\_EXO["ref"]

D12A\_template = Dic\_D12A["ref"]

DEL\_template = Dic\_DEL["ref"]

*# Calculating entropy per position in the MSA as a measure of diversity*

*function* fasta\_char(file\_name::String)

    fasta\_file = []

    reader = open(FASTA.Reader, file\_name)

*for* record *in* reader

        seq = FASTX.FASTA.sequence(record)

        push!(fasta\_file, seq)

*end*

    close(reader)

    seqs\_char = Array{Char, 2}(undef, length(fasta\_file),length(fasta\_file[1]))

*for* i = 1:length(fasta\_file)

*for* j = 1:length(fasta\_file[i])

               seqs\_char[i,j] = fasta\_file[i][j]

*end*

*end*

*return* seqs\_char

*end*

*function* probabilities(X)

    counts = countmap(collect(eachrow(X)))

    probs = values(counts)./sum(values(counts))

*return* probs

*end*

*function* entropy\_system(X)

    H = []

*for* i *in* 1:size(X)[2]

        prob = probabilities(X[:,i])

        entropy = (-1.0).\*sum(log2.(prob).\*prob)

        push!(H, entropy)

*end*

*return* H

*end*

*# Running functions and exporting output as CSV*

EXO\_Hx = entropy\_system(fasta\_char("EXO\_MSA.fasta"))

EXO\_Hx\_df = DataFrame(Reference\_seq = collect(EXO\_template), Exo\_H= EXO\_Hx)

CSV.write("EXO\_Entropy.csv", EXO\_Hx\_df)

D12A\_Hx = entropy\_system(fasta\_char("D12A\_MSA.fasta"))

D12A\_Hx\_df = DataFrame(Reference\_seq = collect(D12A\_template), D12A\_H= D12A\_Hx)

CSV.write("D12A\_Entropy.csv", D12A\_Hx\_df)

DEL\_Hx = entropy\_system(fasta\_char("DEL\_MSA.fasta"))

DEL\_Hx\_df = DataFrame(Reference\_seq = collect(DEL\_template), DEL\_H= DEL\_Hx)

CSV.write("DEL\_Entropy.csv", DEL\_Hx\_df)

*# Entropy spike analysis*

exo\_spike = countmap(fasta\_char("EXO\_MSA.fasta")[:,219])

d12a\_spike = countmap(fasta\_char("D12A\_MSA.fasta")[:,231])

del\_spike = countmap(fasta\_char("DEL\_MSA.fasta")[:,245])

*# Counting Insertion, deletion and substitution errors*

*function* comparison(counts::Dict{String, Int64}, template::LongSequence, sequence::LongSequence)

*for* i *in* 1:length(template)

*if* template[i] == DNA\_Gap && sequence[i] != DNA\_Gap

*if* i > 1

*if* template[i-1] != DNA\_Gap

                    counts["insertion"] += 1

*end*

*else*

                counts["insertion"] += 1

*end*

*elseif* template[i] != DNA\_Gap && sequence[i] == DNA\_Gap

*if* i > 1

*if* sequence[i-1] != DNA\_Gap

                    counts["deletion"] += 1

*end*

*else*

                counts["deletion"] += 1

*end*

*elseif* template[i] != sequence[i]

            counts["mutation"] += 1

*end*

*end*

*return* counts

*end*

*function* pol\_fidelity(pol\_data::Dict{String,Dict{String,Int64}})

    insertions = []

    deletions = []

    mutations = []

    seq\_length = length(ref)

*for* (k, v) *in* pol\_data

            ins = pol\_data[k]["insertion"]

            dels = pol\_data[k]["deletion"]

            muts = pol\_data[k]["mutation"]

            push!(insertions, ins)

            push!(deletions, dels)

            push!(mutations, muts)

*end*

    total\_bases = seq\_length\*length(collect(keys(pol\_data)))

    freq\_insertions = sum(insertions)/total\_bases

    freq\_deletions = sum(deletions)/total\_bases

    freq\_mutations = sum(mutations)/total\_bases

    total = freq\_insertions+freq\_deletions+freq\_mutations

    dict = Dict("insertions/base" => freq\_insertions, "deletions/base" => freq\_deletions,

                "substitution/base" => freq\_mutations, "total errors/base" => total, "bases sequenced" =>total\_bases)

*return* dict

*end*

*# Running functions*

EXO\_counts\_per\_seq = Dict{String, Dict{String, Int64}}()

*for* (k, v) *in* Dic\_EXO

    counts = Dict("insertion" => 0, "deletion" => 0, "mutation" => 0)

    comparison(counts, EXO\_template, v)

    EXO\_counts\_per\_seq[k] = counts

*end*

D12A\_counts\_per\_seq = Dict{String, Dict{String, Int64}}()

*for* (k, v) *in* Dic\_D12A

    counts = Dict("insertion" => 0, "deletion" => 0, "mutation" => 0)

    comparison(counts, D12A\_template, v)

    D12A\_counts\_per\_seq[k] = counts

*end*

DEL\_counts\_per\_seq = Dict{String, Dict{String, Int64}}()

*for* (k, v) *in* Dic\_DEL

    counts = Dict("insertion" => 0, "deletion" => 0, "mutation" => 0)

    comparison(counts, DEL\_template, v)

    DEL\_counts\_per\_seq[k] = counts

*end*

*# Output*

pol\_fidelity(EXO\_counts\_per\_seq)

pol\_fidelity(D12A\_counts\_per\_seq)

pol\_fidelity(DEL\_counts\_per\_seq)

*# Counting substitution error types (transitions and transversions)*

*function* errortype(error\_type::Dict{String, Int64}, template::LongSequence, sequence::LongSequence)

*for* i *in* 1:length(template)

*if* template[i] == DNA\_A && sequence[i] == DNA\_G

            error\_type["A->G"] += 1

*elseif* template[i] == DNA\_G && sequence[i] == DNA\_A

            error\_type["G->A"] += 1

*elseif* template[i] == DNA\_T && sequence[i] == DNA\_C

            error\_type["T->C"] += 1

*elseif* template[i] == DNA\_C && sequence[i] == DNA\_T

            error\_type["C->T"] += 1

*elseif* template[i] == DNA\_A && sequence[i] == DNA\_T

            error\_type["A->T"] += 1

*elseif* template[i] == DNA\_T && sequence[i] == DNA\_A

            error\_type["T->A"] += 1

*elseif* template[i] == DNA\_A && sequence[i] == DNA\_C

            error\_type["A->C"] += 1

*elseif* template[i] == DNA\_C && sequence[i] == DNA\_A

            error\_type["C->A"] += 1

*elseif* template[i] == DNA\_T && sequence[i] == DNA\_G

            error\_type["T->G"] += 1

*elseif* template[i] == DNA\_G && sequence[i] == DNA\_T

            error\_type["G->T"] += 1

*elseif* template[i] == DNA\_C && sequence[i] == DNA\_G

            error\_type["C->G"] += 1

*elseif* template[i] == DNA\_G && sequence[i] == DNA\_C

            error\_type["G->C"] += 1

*end*

*end*

*return* error\_type

*end*

*function* error\_type\_counts(pol\_data::Dict{String,Dict{String,Int64}})

    a = []

    b = []

    c = []

    d = []

    e = []

    f = []

    g = []

    h = []

    i = []

    j = []

    k = []

    l = []

*for* (ks, vs) *in* pol\_data

            AGs = pol\_data[ks]["A->G"]

            GAs = pol\_data[ks]["G->A"]

            TCs = pol\_data[ks]["T->C"]

            CTs = pol\_data[ks]["C->T"]

            ATs = pol\_data[ks]["A->T"]

            TAs = pol\_data[ks]["T->A"]

            ACs = pol\_data[ks]["A->C"]

            CAs = pol\_data[ks]["C->A"]

            TGs = pol\_data[ks]["T->G"]

            GTs = pol\_data[ks]["G->T"]

            CGs = pol\_data[ks]["C->G"]

            GCs = pol\_data[ks]["G->C"]

            push!(a, AGs)

            push!(b, GAs)

            push!(c, TCs)

            push!(d, CTs)

            push!(e, ATs)

            push!(f, TAs)

            push!(g, ACs)

            push!(h, CAs)

            push!(i, TGs)

            push!(j, GTs)

            push!(k, CGs)

            push!(l, GCs)

*end*

    total\_base\_substitutions = sum(vcat(a,b,c,d,e,f,g,h,i,j,k,l))

    avg\_a = 100\*sum(a)/total\_base\_substitutions

    avg\_b = 100\*sum(b)/total\_base\_substitutions

    avg\_c = 100\*sum(c)/total\_base\_substitutions

    avg\_d = 100\*sum(d)/total\_base\_substitutions

    avg\_e = 100\*sum(e)/total\_base\_substitutions

    avg\_f = 100\*sum(f)/total\_base\_substitutions

    avg\_g = 100\*sum(g)/total\_base\_substitutions

    avg\_h = 100\*sum(h)/total\_base\_substitutions

    avg\_i = 100\*sum(i)/total\_base\_substitutions

    avg\_j = 100\*sum(j)/total\_base\_substitutions

    avg\_k = 100\*sum(k)/total\_base\_substitutions

    avg\_l = 100\*sum(l)/total\_base\_substitutions

    dict = Dict("total"=>total\_base\_substitutions,"A->G"=>avg\_a,"G->A"=>avg\_b,"T->C"=>avg\_c,"C->T"=>avg\_d,"A->T"=>avg\_e,"T->A"=>avg\_f,"A->C"=>avg\_g,"C->A"=>avg\_h,"T->G"=>avg\_i,"G->T"=>avg\_j,"C->G"=>avg\_k,"G->C"=>avg\_l)

*return* dict

*end*

*# Running functions*

EXO\_errortype = Dict{String, Dict{String, Int64}}()

*for* (k, v) *in* Dic\_EXO

    error\_type = Dict("A->G"=>0,"G->A"=>0,"T->C"=>0,"C->T"=>0,"A->T"=>0,"T->A"=>0,"A->C"=>0,"C->A"=>0,"T->G"=>0,"G->T"=>0,"C->G"=>0,"G->C"=>0)

    errortype(error\_type, EXO\_template, v)

    EXO\_errortype[k] = error\_type

*end*

D12A\_errortype = Dict{String, Dict{String, Int64}}()

*for* (k, v) *in* Dic\_D12A

    error\_type = Dict("A->G"=>0,"G->A"=>0,"T->C"=>0,"C->T"=>0,"A->T"=>0,"T->A"=>0,"A->C"=>0,"C->A"=>0,"T->G"=>0,"G->T"=>0,"C->G"=>0,"G->C"=>0)

    errortype(error\_type, D12A\_template, v)

    D12A\_errortype[k] = error\_type

*end*

DEL\_errortype = Dict{String, Dict{String, Int64}}()

*for* (k, v) *in* Dic\_DEL

    error\_type = Dict("A->G"=>0,"G->A"=>0,"T->C"=>0,"C->T"=>0,"A->T"=>0,"T->A"=>0,"A->C"=>0,"C->A"=>0,"T->G"=>0,"G->T"=>0,"C->G"=>0,"G->C"=>0)

    errortype(error\_type, DEL\_template, v)

    DEL\_errortype[k] = error\_type

*end*

*# Output*

EXO\_errortype\_per = error\_type\_counts(EXO\_errortype)

D12A\_errortype\_per = error\_type\_counts(D12A\_errortype)

DEL\_errortype\_per = error\_type\_counts(DEL\_errortype)

*# Error type distribution per position*

*function* dist\_errortype(file\_name::String, template)

    fasta = fasta\_char(file\_name)

    fasta\_dic = Dict()

*for* i *in* 1:size(fasta)[2]

        map = countmap(fasta[:,i])

        fasta\_dic[i] = map

*end*

    sequences = size(fasta)[1]

    A\_list = []

    T\_list = []

    G\_list = []

    C\_list = []

    DEL\_list = []

*for* (k,v) *in* sort(fasta\_dic)

*if* haskey(v, 'A') == false

                push!(A\_list, 0)

*elseif* haskey(v, 'A') == true

                push!(A\_list, v['A'])

*end*

*if* haskey(v, 'T') == false

                push!(T\_list, 0)

*elseif* haskey(v, 'T') == true

                push!(T\_list, v['T'])

*end*

*if* haskey(v, 'G') == false

                push!(G\_list, 0)

*elseif* haskey(v, 'G') == true

                push!(G\_list, v['G'])

*end*

*if* haskey(v, 'C') == false

                push!(C\_list, 0)

*elseif* haskey(v, 'C') == true

                push!(C\_list, v['C'])

*end*

*if* haskey(v, '-') == false

                push!(DEL\_list, 0)

*elseif* haskey(v, '-') == true

                push!(DEL\_list, v['-'])

*end*

*end*

    transitions = []

    transversions = []

*for* i *in* 1:length(template)

*if* template[i] == DNA\_A

            push!(transitions,G\_list[i])

            push!(transversions, (T\_list[i]+C\_list[i]))

*elseif* template[i] == DNA\_T

            push!(transitions,C\_list[i])

            push!(transversions, (A\_list[i]+G\_list[i]))

*elseif* template[i] == DNA\_G

            push!(transitions,A\_list[i])

            push!(transversions, (T\_list[i]+C\_list[i]))

*elseif* template[i] == DNA\_C

            push!(transitions,T\_list[i])

            push!(transversions, (A\_list[i]+G\_list[i]))

*end*

*end*

    freq\_transitions = []

    freq\_transversions = []

*for* i *in* 1:length(transitions)

            push!(freq\_transitions,(transitions[i]/sequences))

            push!(freq\_transversions,(transversions[i]/sequences))

*end*

*return* DataFrame(Transitions=transitions,Transversions=transversions,

                     Freq\_transitions=freq\_transitions,Freq\_transversions=freq\_transversions)

*end*

*# Running functions*

EXO\_dist = dist\_errortype("EXO\_MSA.fasta", EXO\_template)

D12A\_dist = dist\_errortype("D12A\_MSA.fasta", D12A\_template)

DEL\_dist = dist\_errortype("DEL\_MSA.fasta", DEL\_template)

error\_map = hcat(EXO\_dist,D12A\_dist,DEL\_dist, makeunique=true)

rename!(error\_map,[:Exo\_transitions,:Exo\_transversions,:Exo\_transitions\_freq, :Exo\_transversions\_Freq,

                    :D12A\_transitions,:D12A\_transversions,:D12A\_transitions\_freq, :D12A\_transversions\_Freq,

                    :DEL\_transitions,:DEL\_transversions,:DEL\_transitions\_freq, :DEL\_transversions\_Freq,

                    ])

*# Output*

CSV.write("mutants\_error\_distribution.csv", error\_map)

*# Identifying the location and abundance of deletions and insertions*

*function* concat\_range(list)

    dats =[]

*for* i *in* 1:length(list)

        dat = collect(list[i])

        append!(dats,dat)

*end*

    list = union(dats)

    final\_range = []

    vectorized\_range = []

    start = 0

    finish = 0

    del\_length = 0

*for* element = 1:length(list)

*## Starting new ranges*

*if* element == 1

            start = list[1]

            finish = list[1]

            del\_length += 1

*elseif* element > 1 && del\_length == 0

            start = list[element]

            finish = list[element]

            del\_length += 1

*## Extending new ranges*

*elseif* del\_length > 0 && list[element-1] == list[element]-1 && list[element] != list[end]

            finish = list[element]

            del\_length += 1

*## Closing the ranges*

*elseif* del\_length == 1 && list[element-1] != list[element]-1 && list[element] != list[end]

*# Close the previous range*

            push!(final\_range, [start:finish])

            push!(vectorized\_range, [start, del\_length])

*# open the new range*

            start = list[element]

            finish = list[element]

            del\_length = 1

*elseif* del\_length > 1 && list[element-1] != list[element]-1 && list[element] != list[end]

*# Close the previous range*

            finish = list[element-1]

            push!(final\_range, [start:finish])

            push!(vectorized\_range, [start, del\_length])

*# Open the new range*

            start = list[element]

            finish = list[element]

            del\_length = 1

*elseif* list[element] == list[end]

*# Close the range*

            finish = list[end]

            del\_length += 1

            push!(final\_range, [start:finish])

            push!(vectorized\_range, [start, del\_length])

*end*

*end*

*return* reduce(vcat, final\_range)

*end*

*function* find\_dels(Dic, reference)

    df\_dels = []

    del\_freq = Dict()

    gaps = []

    ref\_gaps = countmap(concat\_range(findall(r"-",string(reference))))

*for* (k,v) *in* Dic

        seq = string(v)

        positions\_seq = concat\_range(findall(r"-",seq))

        append!(gaps, positions\_seq)

*end*

    count\_gaps = countmap(gaps)

    unique\_ranges = setdiff(collect(keys(count\_gaps)), collect(keys(ref\_gaps)))

*for* i *in* 1:length(unique\_ranges)

        del\_freq[unique\_ranges[i]] = count\_gaps[unique\_ranges[i]]

*end*

*for* (keys,vals) *in* del\_freq

        len = length(keys)

        range = keys

        freq = vals

        start\_del = parse(Int64,split(string(range),":")[1])

        push!(df\_dels, (freq,range,start\_del,len))

*end*

    df\_dels = DataFrame([[df\_dels[k][kk] *for* k *in* 1:length(df\_dels)] *for* kk *in* 1:length(df\_dels[1])], [:Freq, :Range, :Start, :Length])

*return* sort(df\_dels, :Freq, rev=true)

*end*

*function* find\_ins(Dic, reference)

    df\_ins = []

    ins\_freq = Dict()

    gaps = []

    ref\_gaps = countmap(concat\_range(findall(r"-",string(reference))))

*for* (k,v) *in* Dic

        seq = string(v)

        positions\_seq = concat\_range(findall(r"-",seq))

        append!(gaps, positions\_seq)

*end*

    count\_gaps = countmap(gaps)

    unique\_ranges = intersect(collect(keys(count\_gaps)), collect(keys(ref\_gaps)))

*for* i *in* 1:length(unique\_ranges)

        ins\_freq[unique\_ranges[i]] = count\_gaps[unique\_ranges[i]]

*end*

*for* (keys,vals) *in* ins\_freq

        len = length(keys)

        range = keys

        freq = vals

        start\_ins = parse(Int64,split(string(range),":")[1])

        push!(df\_ins, (freq,range,start\_ins,len))

*end*

    df\_ins = DataFrame([[df\_ins[k][kk] *for* k *in* 1:length(df\_ins)] *for* kk *in* 1:length(df\_ins[1])], [:Freq, :Range, :Start, :Length])

*return* sort(df\_ins, :Freq, rev=true)

*end*

*#Running functions*

exo\_del\_map = find\_dels(Dic\_EXO, EXO\_template)

d12a\_del\_map = find\_dels(Dic\_D12A, D12A\_template)

del\_del\_map = find\_dels(Dic\_DEL, DEL\_template)

exo\_ins\_map = find\_ins(Dic\_EXO, EXO\_template)

d12a\_ins\_map = find\_ins(Dic\_D12A, D12A\_template)

del\_ins\_map = find\_ins(Dic\_DEL, DEL\_template)

*#Output as CSV*

CSV.write("exo\_del\_map.csv", exo\_del\_map)

CSV.write("d12a\_del\_map.csv", d12a\_del\_map)

CSV.write("del\_del\_map.csv", del\_del\_map)

CSV.write("exo\_ins\_map.csv", exo\_ins\_map)

CSV.write("d12a\_ins\_map.csv", d12a\_ins\_map)

CSV.write("del\_ins\_map.csv", del\_ins\_map)