

Demostración de Proyecto Final

Diego Puente y Fernanda Pacheco

2025-11-19

Código para A.M.P.L.I.F.Y.

Simulador in silico de una PCR Amplification of proteins for MicroBaby's

Por: Pacheco Estrella María Fernanda y Puente Rivera Diego

```
#Cargar paqueterías necesarias#
library(Biostrings)

## Cargando paquete requerido: BiocGenerics

## Cargando paquete requerido: generics

##
## Adjuntando el paquete: 'generics'

## The following objects are masked from 'package:base':
##
##      as.difftime, as.factor, as.ordered, intersect, is.element, setdiff,
##      setequal, union

##
## Adjuntando el paquete: 'BiocGenerics'

## The following objects are masked from 'package:stats':
##
##      IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##      anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##      colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##      get, grep, grepl, is.unsorted, lapply, Map, mapply, match, mget,
##      order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##      rbind, Reduce, rownames, sapply, saveRDS, table, tapply, unique,
##      unsplit, which.max, which.min
```

```

## Cargando paquete requerido: S4Vectors

## Cargando paquete requerido: stats4

##
## Adjuntando el paquete: 'S4Vectors'

## The following object is masked from 'package:utils':
## findMatches

## The following objects are masked from 'package:base':
## expand.grid, I, uname

## Cargando paquete requerido: IRanges

##
## Adjuntando el paquete: 'IRanges'

## The following object is masked from 'package:grDevices':
## windows

## Cargando paquete requerido: XVector

## Cargando paquete requerido: GenomeInfoDb

##
## Adjuntando el paquete: 'Biostrings'

## The following object is masked from 'package:base':
## strsplit

```

Demostración

```

#####
# VIABILIDAD DE PCR #####
#####

## función que normaliza la secuencia (porque a veces es DNAStringSet)
normalizar <- function(sec){
  if (class(sec) == "DNAStringSet") sec <- sec [[1]]
  if (class(sec) == "character") sec <- DNAString(sec)
  return(sec)
}

#####
## Análisis de los primers ##
#####

```

```

#### Reverso complementario ####
revcomp <- function(x) {
  as.character(reverseComplement(DNAString(x)))
}

#### Porcentaje de GC ####
gc_porcent <- function(primer){
  bases <- strsplit(primer, "")[[1]]
  gc <- sum(bases %in% c("G", "C"))
  return(round(100 * gc / length(bases), 1))
}

#### Homodimero ####
homodimero <- function(primer){
  seq1 <- strsplit(primer, "")[[1]]
  seq2 <- strsplit(revcomp(primer), "")[[1]]
  n1 <- length(seq1)
  n2 <- length(seq2)

  max_match <- 0

  for(i in 1:n1){
    for(j in 1:n2){
      k <- 0
      while(i+k <= n1 && j+k <= n2 && seq1[i+k] == seq2[j+k]){
        k <- k + 1
      }
      if(k > max_match) max_match <- k
    }
  }

  return(max_match)
}

#### Heterodimero ####
heterodimero <- function(fw, rv){
  seq1 <- strsplit(fw, "")[[1]]
  seq2 <- strsplit(revcomp(rv), "")[[1]]
  n1 <- length(seq1)
  n2 <- length(seq2)

  max_match <- 0

  for(i in 1:n1){
    for(j in 1:n2){
      k <- 0
      while(i+k <= n1 && j+k <= n2 && seq1[i+k] == seq2[j+k]){
        k <- k + 1
      }
      if(k > max_match) max_match <- k
    }
  }

}

```

```

    return(max_match)
}

### Temperatura (Tm) ####
tm <- function(primer) {
  bases <- strsplit(primer, "")[[1]]
  A <- sum(bases=="A")
  T <- sum(bases=="T")
  G <- sum(bases=="G")
  C <- sum(bases=="C")
  Tm <- 2*(A+T)+ 4*(G+C)
  return(round(Tm,1))
}

### GC al final del primer ####
gc_final <- function(primer){
  bases <- strsplit(primer, "")[[1]]
  n <- length(bases)
  end <- paste0(bases[(n-1):n], collapse = "")
  sum(end == "GC")
}

### Homopolímeros (AAA, CCC, GGG, TTT) ####
hopol<- function(primer, k=3){
  bases <- strsplit(primer, "")[[1]]
  count <- 1
  for (i in 2:length(bases)) {
    if(bases[i]==bases[i-1]){
      count <- count + 1
      if (count>=k) return(T)
    } else {
      count <- 1
    }
  }
  return(F)
}

### Detectar horquillas ####
horquillas <- function(primer, min_match=4){
  bases <- strsplit(primer, "")[[1]]
  nucl <- length(bases)
  if (nucl < min_match) return(F)

  for (i in 1: (nucl-min_match+1)) {
    fragmento <- paste0(bases[i:(i+min_match-1)], collapse = "")

    frag_rvcm <- as.character(revcomp(fragmento))

    for (j in 1: (nucl-min_match+1)) {
      horq <- paste0(bases[j:(j+min_match-1)], collapse = "")

      if(horq==frag_rvcm) return(T)
    }
  }
}

```

```

    }
    return(F)
}

### Evaluación del primer individual ###

eval_primer <- function(primer, tipo) {
  long <- nchar(primer)
  gc_percent <- gc_porcent(primer)
  homod <- homodímero(primer)
  tmel <- tm(primer)
  clamp <- gc_final(primer)
  polim <- hopol(primer)
  hqlia<- horquillas(primer)

  cat(tipo, "-", primer, "\n")
  cat("Longitud:", long, "pb. ",
      ifelse(long %in% 18:24, "OK", "No óptimo"), "\n")
  cat("%GC:", gc_percent, "%",
       ifelse(gc_percent >= 40 & gc_percent<=60, "OK", "No óptimo"), "\n")
  cat("Tm:", tmel, "°C.",
      ifelse(tmel>=55 & tmel<=65, "OK", "No óptimo"), "\n")
  cat("Homodímeros máximos:", homod, "pb. ",
      ifelse(homod >= 3, "No óptimo", "OK"), "\n")
  cat("Horquillas:",
      ifelse(hqlia, "Sí, no óptimo", "No"), "\n\n")
}

### Viabilidad de ambos ###

eval_pair <- function(fw, rv) {
  dif <- abs(tm(fw) - tm(rv))
  cross <- heterodímero(fw, rv)

  cat("Compatibilidad del par:\n")
  cat("Diferencia de Tm: ", dif, " °C",
      ifelse(dif<=2, "OK (ideal <=2°C",
             ifelse(dif<=5, "OK (<=5°C)", "No óptimo")), "\n")

  cat("Heterodímero: ", cross, "pb. ",
      ifelse(cross >= 3, "Demasiados heterodímeros, no funcional", "Es aceptable"), "\n\n")

  return(list(difTm=dif, cross=cross))
}

## Búsqueda de primers en la secuencia ##
# Se unen o no a la secuencia
union_sec <- function(fw, rv, sec) {
  rc <- revcomp(rv)

  uni_fw <- start(matchPattern(fw, sec))
  uni_rv <- start(matchPattern(rc, sec))
}

```

```

if (length(uni_fw) && length(uni_rv)) {
  ini <- min(uni_fw)
  fin <- max(uni_rv) + nchar(rc) - 1
  size <- fin - ini + 1

  cat("Ambos primers se unen.\n")
  cat("Forward en:", ini, "\n")
  cat("Reverse en:", fin - nchar(rc) + 1, "\n")
  cat("Amplicón:", size, "pb\n\n")

  return(list(ok=TRUE, size=size))
}

cat("Problemas de unión:\n")
if (!length(uni_fw)) cat("Forward no se une.\n")
if (!length(uni_rv)) cat("Reverse no se une.\n")
cat("\n")
return(list(ok=FALSE, size=0))
}

#####
# PCR #####
sim_pcr <- function(E, cycles = cycles, NO = 1) {
  copias <- numeric(cycles + 1)
  copias[1] <- NO

  for (i in 2:(cycles + 1)) {
    copias[i] <- copias[i - 1] * (1 + E)
  }

  return(copias)
}

### VIABILIDAD COMPLETA (RESUMEN) ###
evaluar_primers <- function(fw, rv, sec, cycles=30, NO=1) {

  sec <- normalizar(sec)

  cat("[Evaluación de primers]\n")
  cat("-----\n")
  cat("1) Individual\n")
  eval_primer(fw, "Forward")
  eval_primer(rv, "Reverse")
  cat("-----\n")

  cat("2) Compatibilidad de ambos\n")
  eval_pair(fw, rv)
  cat("-----\n")

  cat("3) Unión a la secuencia\n")
  res <- union_sec(fw, rv, sec)
  cat("-----\n")
}

```

```

cat("4) Viabilidad final y simulación PCR\n")

## Si los primers no se unen bien ##
if (!res$ok) {
  cat("Los primers no funcionan porque no se pudieron alinear bien con la
      secuencia.\n")
  cat("Por esta razón no se hace la simulación PCR.\n")
  return(list(union_sec=res, copias = NULL))
}

## Si el tamaño del amplicón no es útil ##
if (res$size < 100 || res$size > 2000) {
  cat("Los primers si se unen, sin embargo el amplicón no queda en un tamaño
      que sea adecuado.\n")
  cat("Tamaño del amplicón: ", res$size, " pb\n\n")
  cat("No se hace simulación PCR .\n")
  return(list(union_sec = res, copias = NULL))
}

## Si llegó hasta acá, todo está bien ##
cat("Los primers son viables .\n")
cat("Tamaño del amplicón: ", res$size, " pb\n\n")

## Función para estimar eficiencia segun la diferencia de la Tm ##
eficiencia <- function(fw, rv){
  dif_tm <- abs(tm(fw) - tm(rv))
  if (dif_tm <=2) {
    return(0.95)
  } else if (dif_tm <=5) {
    return(0.85)
  } else {
    return(0.60)
  }
}

### Calcular eficiencia y simular PCR ###
efi <- eficiencia(fw, rv)
copias <- sim_pcr(efi, cycles = cycles, NO = NO)
cat("Eficiencia estimada: ", round(efi, 3), "\n")
cat("Copias después de ", cycles, " ciclos:", format(copias[length(copias)],
                                                       scientific = T), "\n")
return(list(union_sec = res, eficiencia = efi, copias = copias))
}

### Cargar secuencia
bla0xy <- Biostrings::readDNAStringSet("bla0XY.fna")

```

Resultados

Primers funcionales

```

blaOxy <- Biostrings::readDNAStringSet("blaOXY.fna")

resultado <- evaluar_primers(
  fw = "AGAGCGGAATGACGCTGGCT",
  rv = "CGATCGAGACGAAAGGTGGC",
  sec = blaOxy,
  cycles = 45
)

## [Evaluación de primers]
## -----
## 1) Individual
## Forward - AGAGCGGAATGACGCTGGCT
## Longitud: 20 pb. OK
## %GC: 60 %. OK
## Tm: 64 °C. OK
## Homodímeros máximos: 4 pb. No óptimo
## Horquillas: Sí, no óptimo
##
## Reverse - CGATCGAGACGAAAGGTGGC
## Longitud: 20 pb. OK
## %GC: 60 %. OK
## Tm: 64 °C. OK
## Homodímeros máximos: 6 pb. No óptimo
## Horquillas: Sí, no óptimo
##
## -----
## 2) Compatibilidad de ambos
## Compatibilidad del par:
## Diferencia de Tm: 0 °C OK (ideal <=2°C
## Heterodímero: 2 pb. Es aceptable
##
## -----
## 3) Unión a la secuencia
## Ambos primers se unen.
## Forward en: 447
## Reverse en: 578
## Amplicón: 151 pb
##
## -----
## 4) Viabilidad final y simulación PCR
## Los primers son viables .
## Tamaño del amplicón: 151 pb
##
## Eficiencia estimada: 0.95
## Copias después de 45 ciclos: 1.12605e+13

```

Primers no funcionales

```

resultado <- evaluar_primers(
  fw = "AATTGATGATGGAATTCCAT",
  rv = "GGTCCCGCAGACGGCATGAA",

```

```

sec = bla0xy,
cycles = 45
)

## [Evaluación de primers]
## -----
## 1) Individual
## Forward - AATTGATGATGGAATTCCAT
## Longitud: 20 pb. OK
## %GC: 30 %. No óptimo
## Tm: 52 °C. No óptimo
## Homodímeros máximos: 12 pb. No óptimo
## Horquillas: Sí, no óptimo
##
## Reverse - GGTCCGCAGACGGCATGAA
## Longitud: 19 pb. OK
## %GC: 63.2 %. No óptimo
## Tm: 62 °C. OK
## Homodímeros máximos: 4 pb. No óptimo
## Horquillas: Sí, no óptimo
##
## -----
## 2) Compatibilidad de ambos
## Compatibilidad del par:
## Diferencia de Tm: 10 °C No óptimo
## Heterodímero: 3 pb. Demasiados heterodímeros, no funcional
##
## -----
## 3) Unión a la secuencia
## Problemas de unión:
## Reverse no se une.
##
## -----
## 4) Viabilidad final y simulación PCR
## Los primers no funcionan porque no se pudieron alinear bien con la
## secuencia.
## Por esta razón no se hace la simulación PCR.

```