```
# Michael E. Sparks, 17 Feb 2016
# Identify and report potential quorum-sensing motifs in genomic DNA

# Time-critical routines have been ported to C, which is strongly preferred
# to using the native R implementation. Assume by default that C's available.
#Cavail = FALSE
Cavail = TRUE

# Position Weight Matrix (i.e., 0th-order Markov chain) -
# probabilities were approximated by eyeballing the logo plot
# in Figure 3 of Stauff and Bassler 2011
# (http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3147534/)
wordsize ← 18 # length of motif
smoothconst ← 0.020 # permits limited ambiguity
pwm ← matrix(
  c(0.355,0.245,0.190,0.190,smoothconst,
    0.100,0.600,0.080,0.200,smoothconst,
    0.180,0.210,0.040,0.550,smoothconst,
    0.180,0.025,0.755,0.020,smoothconst,
    0.290,0.190,0.200,0.300,smoothconst,
    0.020,0.410,0.160,0.390,smoothconst,
    0.300,0.360,0.100,0.220,smoothconst,
    0.240,0.300,0.200,0.240,smoothconst,
    0.350,0.140,0.140,0.350,smoothconst,
    0.300,0.220,0.110,0.350,smoothconst,
    0.220,0.250,0.300,0.210,smoothconst,
    0.100,0.080,0.400,0.400,smoothconst,
    0.220,0.190,0.360,0.210,smoothconst,
    0.300,0.240,0.240,0.200,smoothconst,
    0.020,0.600,0.150,0.210,smoothconst,
    0.500,0.090,0.190,0.200,smoothconst,
    0.190,0.110,0.600,0.080,smoothconst,
    0.200,0.220,0.230,0.330,smoothconst),
  nrow=5,ncol=wordsize,byrow=FALSE)
# The "Any" catchall allows for consideration of candidate
# motifs harboring ambiguous nucleotides (given a 2% likelihood).
row.names(pwm) ← c("Ade","Cyt","Gua","Thy","Any")
# columns denote position in motif and each constitutes a PMF
# sanity check proceeds with silence:
# for(i in 1:word) if(sum(pwm[,i]) != 1.0) print(i)

# threshold for reporting candidate quorum-sensing motifs (arbitrary)
#minscore <- log(0.25**wordsize)
# UPDATE: Empirically, -20.0 seems like a reasonable floor, so...
# > exp(-20)**(1/18)
# [1] 0.329193
# > log( (exp(-20)**(1/18)) ** 18)
# [1] -20
minscore ← -20.0

# highest-scoring chain possible in matrix:
#> sum(log(apply(pwm,2,max)))
#[1] -16.01553

# R function : quorumCandidates
# Reports candidate quorum-sensing motifs present in objects returned by
# the "read.fasta" function of the "seqinr" package. In particular, these
# objects should result from calls to that function with the following
# parameter settings: seqtype="DNA",as.string=TRUE,forceDNAtolower=TRUE
```

```
#
# This function depends on three "global" vars, defined supra:
#   1) "wordsize" (length of motif)
#   2) "pwm" (probability weight matrix of motif)
#   3) "minscore" (min score to merit reporting)
# It also relies on its stablemate C function, scoreQuorumCandidates,
#   if a C system interface is available.
#
# Forward & reverse strands of each sequence are processed - mutations may
# disrupt otherwise perfectly palindromic motifs, resulting in differential
# scoring of the element on each strand of the DNA duplex. When a motif is
# positioned between two proximal genes, this may help in resolving which
# of the flanking genes is most likely to be under the regulatory
# element's control.
quorumCandidates ← function(seqobj) {
  # seqinr doesn't ignore whitespace (why?!!), so strip it out
  seq ← gsub("\\s","",seqobj)[[1]]

  # build score vectors - note that joint probabilities are expressed
  # in log space, to mitigate risk of buffer underflows
  veclen ← nchar(seq)-wordsize+1
  scoresF ← vector(mode="numeric",length=veclen) # Watson strand
  scoresR ← vector(mode="numeric",length=veclen) # Crick strand

  # score candidate quorum-sensing motifs
  if(!Cavail) { # use native R code when C unavailable
    seq ← strsplit(seq,"")[[1]]

    # recode nucleotides as integers
    for(i in 1:length(seq))
      seq[i] ← switch(seq[i],'a'='1','c'='2','g'='3','t'='4','5')

    # score forward strand
    for(i in 1:veclen) {
      scoresF[i] ← 0.0
      for(j in 1:wordsize)
        scoresF[i] ← scoresF[i] + log(pwm[as.integer(seq[i+j-1]),j])
    }

    # take reverse complement
    seq ← rev(seq)
    for(i in 1:length(seq))
      seq[i] ← switch(seq[i],'1'='4','2'='3','3'='2','4'='1','5')

    # score reverse strand
    for(i in 1:veclen) {
      scoresR[i] ← 0.0
      for(j in 1:wordsize)
        scoresR[i] ← scoresR[i] + log(pwm[as.integer(seq[i+j-1]),j])
    }
  }
  else { # C is strongly preferred when available!
    alien ← .C("scoreQuorumCandidates",
               gDNAseq=as.character(seq),
               forwardScores=as.numeric(scoresF),
               reverseScores=as.numeric(scoresR),
               posSpecProbs=as.vector(pwm),
               wordLen=as.integer(wordsize),
               DUP=TRUE)
    scoresF ← alien$forwardScores
    scoresR ← alien$reverseScores
  }

  # Report instances where likelihood of candidate’s being
  # a quorum−sensing motif exceeded the threshold minimum.
  # Results are printed via side effects, so we hand off the
  # return value to a dummy variable, to be ignored.
  ignore ← lapply(
    which(!scoresF < minscore),
    function(x) write(paste(scoresF[x],x,"+",
        paste(attr(seqobj,"Annot"),"(Forward sense)",sep=" "),
        sep="\t"),file=""))

  ignore ← lapply(
    which(!scoresR < minscore),
    function(x) write(paste(scoresR[x],x,"−",
        paste(attr(seqobj,"Annot"),"(Reverse sense)",sep=" "),
        sep="\t"),file=""))

} # end quorumCandidates

# "Main application" −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

# "Customizable" stuff (point to appropriate working directories, filenames)
args ← commandArgs(trailingOnly=TRUE)
#setwd("/some/path/to/motifs_PWM")
setwd(args[1])
#sourcefile <− "test.fa.txt"
sourcefile ← args[2]
#sink("session_output.txt") # tab−delimited, Excel−importable score set
sink(args[3]) # tab−delimited, Excel−importable score set

# check/ remediate critical dependencies (not terribly robust!)
if(!require(seqinr)) {
  install.packages("seqinr")
  library(seqinr)
}

# use C code if available on system
if(Cavail ≡ TRUE) {
  dyn.load("quorumScoring.so")
  if(!is.loaded("scoreQuorumCandidates")) {
    Cavail = FALSE
    dyn.unload("quorumScoring.so")
  }
}

# should generally be able to byte compile functions (moderate speedup)
if(require(compiler)) quorumCandidates ← cmpfun(quorumCandidates)

# Results are printed via side effects, so we hand off the
# return value to a dummy variable, to be ignored.
ignore ← lapply(
  read.fasta(file=sourcefile,seqtype="DNA",
             as.string=TRUE,forceDNAtolower=TRUE),
  quorumCandidates)

sink()
```

```
q("no")
```

```c
/* Michael E. Sparks, 16 Feb 2016
 *
 * Stablemate C function for the quorumCandidates
 * function I've written in R.
 *
 * Nothing profound here - systems with RTools installed
 * can benefit from this "portable assembly code" speedup.
 * Modify the Cavail variable in quorum_sense.R accordingly.
 * ``R CMD SHLIB quorumScoring.c"
 */

#include <R.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#define SCORE(VEC) \
for(i=0;i<seqlen-*wordlen+1;++i) { \
  *(VEC+i)=0.0; \
  for(j=0;j<*wordlen;++j) \
    *(VEC+i)+=log(*(probs+*(intseq+i+j)+j*5)); \
}

void scoreQuorumCandidates(
  char **seq,
  double *scoresF,
  double *scoresR,
  double *probs,
  int *wordlen
) {
  register int
      i,j,             /* iterator vars                          */
      revaux,          /* auxiliary var for reversing intseq     */
      seqlen;          /* stores length of sequence argument     */
  int *intseq=NULL;    /* storage for integer translation of seq */

  /* allocate space for sequence */
  seqlen=strlen(*seq);
  if((intseq=(int*)malloc(sizeof(int)*seqlen))==NULL) {
    Rprintf("Cannot allocate sufficient memory for sequence\n");
    exit(EXIT_FAILURE);
  }

  /* recode using integer scheme */
  for(i=0;i<seqlen;++i)
    switch(*(*seq+i)) {
      case('a') :
        *(intseq+i)=0;
        break;
      case('c') :
        *(intseq+i)=1;
        break;
      case('g') :
        *(intseq+i)=2;
        break;
      case('t') :
        *(intseq+i)=3;
        break;
      default :
        *(intseq+i)=4;
    }

  /* score forward strand */
  SCORE(scoresF)

  /* reverse strand... */
  for(i=0,j=seqlen-1;i<seqlen/2;++i,--j) {
    revaux=*(intseq+i);
    *(intseq+i)=*(intseq+j);
    *(intseq+j)=revaux;
  }

  /* ...and take its complement */
  for(i=0;i<seqlen;++i)
    switch(*(intseq+i)) {
      case(0) :
        *(intseq+i)=3;
        break;
      case(1) :
        *(intseq+i)=2;
        break;
      case(2) :
        *(intseq+i)=1;
        break;
      case(3) :
        *(intseq+i)=0;
        break;
      default :
        ;
    }

  /* score reverse strand */
  SCORE(scoresR)

  free(intseq);
  return;
}
```