Fig 1 for paper; S6 for Supp.
Chr1-qfiltered

July 21, 2017

Contents

1 Intro

Initially, this was a simple driver script to build fig 1 (née, fig 1b, later 3b, and now 1 again) for the paper: scatter-smooth of R-values for 2 strains, with marginal histograms, and the analogous Fig S6 for supp. There now is a fair bit of additional exploration of various options for it, and for strengthening our interpretation of it, primarily in response to Julie's concerns that "same nonref fraction" didn't equate to "same fraction of the *same* nonref nucleotide."

2 Packages

Perhaps package compactr needs to be loaded before wlr.R, so do it here; see note in section 5. (I'm still not sure what's going on with this; seems to be working now without obvious changes from me. My latest grasped-at straw is that knitr cache is contributing to apparent flakiness.)

```
# install.packages('compactr') # <-- this generally needs to be run only after upgrading R

# Getting weird errors loading/using compactr package, so this is meant to chatter at me as to
# whether it's installed/loaded/on search path.  R.utils is not used for anything else (& masks
# some other function names), so change default param to F once this is working. Perhaps also
# searchme to reduce chatter.
pack.debug <-function(rutils=FALSE,searchme=FALSE){
  if(rutils){
    cat('Print(library(R.utils)):\n')
    print(library(R.utils))
    cat('isPackageLoaded("compactr"):',isPackageLoaded('compactr'),'\n')
    cat('isPackageInstalled("compactr"):', isPackageInstalled('compactr'), '\n')
  }
  if(searchme){
    cat('search():\n')
    print(search())
  }
}
pack.debug()
print(library(compactr)) # prints silently returned package list

## [1] "compactr"  "knitr"     "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "methods"   "base"

pack.debug()
print(library(compactr)) # prints silently returned package list

## [1] "compactr"  "knitr"     "stats"     "graphics"  "grDevices" "utils"
## [7] "datasets"  "methods"   "base"

pack.debug()
```

3 Preliminaries

Load utility R code; do setup:

```
source('../../../R/wlr.R') # load util code; path relative this folder or sibling in scripts/larrys

## Running as: ruzzo @ recycle.cs.washington.edu; SVN Id, I miss you.  $Id: wlr.R  2017-07-21 or later $

setup.my.wd('paperfigs') # set working dir; UPDATE if this file moves, or if COPY/PASTE to new file
figdir <- 'Fig1-mscat-figs/'
generic.setup(figdir)
setup.my.knitr(figdir)
```

```
# frequently need to add figpath to file name
fpath <- function(base, suffix='.pdf', dir=figdir){
  return(paste(dir, base, suffix, sep=''))
}
```

4 Major Analysis/Performance Parameters.

Patterned after similar setup in shared-snps.rnw, choices set here alter how this file is processed, what data is analyzed, how fast it runs, etc. Set them carefully before running "make." Major choices are:

1. WHICH SNP TABLES ARE LOADED??? Flip T/F below to load the desired combination of qfiltered, unqfiltered, full genome and/or just Chr1. Primary analysis is only performed on one of them, but we can retain others, with separate names, in case we want more than one around for comparison and/or debugging. This is controlled by `load.tb`, a vector of 4 Booleans, in the order `full.unfiltered, chr1.unfiltered, full.qfiltered, chr1.qfiltered`. E.g., (T, F, T, F) loads *full* tables for *both* q- and un-qfiltered data.

2. WHICH MAIN ANALYSIS??? If multiple tables are loaded, which is used for the main analysis (generating scatter-smooth plots)? Parameter `pri` is a permutation of 1:4, corresponding to `load.tb`; the first loaded table in that order becomes the analysis focus. The default `pri=c(3,4,1,2)` looks at q-filtered data in preference to un-q-filtered, and full tables in preference to Chr1 within each group. (See `tset.picker` for for details.)

3. CLEAR CACHE??? Set `clear.cache` below to T/F to force Knitr cache removal (well, actually a rename) at the start of the run. This is especially important if you've changed either of the previous parameters since the last run.

The following code chunk sets all these parameters based on where it's run. I typically prototype and debug on my laptop, so faster is better—running on Chr1 is sufficient; when run on the department servers, I typically want to do full genomes. Just override them otherwise.

```
# for Makefile, params can be command line args, else base on system; see wlr.r for details.
# load.tb order: full.un, chr1.un, full.qfil,  chr1.qfil
# I initially set this to run Chr1 on laptop, whole genome on server, but decided the Chr1
# plots are preferable, so now defaults to same in either case.

params <- pick.params(
  mac   = list(load.tb=c(F,F,F,T), pri=c(3,4,1,2), clear.cache=T),  # quick on lap
  linux = list(load.tb=c(F,F,F,T), pri=c(3,4,1,2), clear.cache=T)   # same on server
# linux = list(load.tb=c(F,F,T,F), pri=c(3,4,1,2), clear.cache=T, trunc.tables=T) # full on server
)

# Alternatively, edit/uncomment the following to override the above as needed
#params <- pick.params(default=list(load.tb = c(T,T,T,T), pri=1:4, clear.cache = T, trunc.tables=T))
print(params)

# $load.tb
# full.unf chr1.unf  full.qf  chr1.qf
#    FALSE    FALSE    FALSE     TRUE
#
# $pri
# [1] 3 4 1 2
#
# $clear.cache
# [1] TRUE
```

NOTE 2: A few code chunks use the knitr cache. I do NOT check for consistency of cached data with code changes and I do NOT know to what extent/whether knitr does, either. If in doubt, delete directories "cache" (knitr's) and "00common/mycache" (mine) to force rebuild.

CLEAR CACHE!!! T/F set in params above will/won't force knitr cache removal (well, actually a rename):

```
decache(params$clear.cache)

# No cache to remove.
```

5 Compactr Patches

eplot in compactr v 0.1 has a bug: it ignores "box", but I fixed it below (lines marked "wlr"). (Source from https://github.com/carlislerainey/compactr/blob/master/R/eplot.R). Also some problem with "yaxislabels" sometimes undefined? And apparently a conflict between the eplot code below and the compactr library code for addxaxis relating to `.compactrEnv$plotPar$tick.length`, so I'm importing that from github as well. Later two issues may just relate to library vs this code. I think order matters; do "library(compactr)" first, then function defs for eplot and addxaxis, *then* load wlr.r (which also contains "library(compactr)"). Hmmm. The call in wlr.R is inside a conditional inside a function (nrf.6plus1), so I don't think load order for wlr.R should matter...

```
pack.debug()
eplot <-
  function(xlim, ylim, xlab = NULL, ylab = NULL,
           main = NULL, text.size = 1, tick.length = 0.02,
           xpos = -.7, ypos = -.5, xat = NULL, yat = NULL,
           xticklab = NULL, yticklab = NULL,
           xlabpos = 1.5, ylabpos = NULL,
           annx = TRUE, anny = TRUE,
           box = TRUE, log = "") {

    #cat('[MYEPLOT... ') ##WLR
    # create an empty plot
    plot(NULL, xlim = xlim, ylim = ylim, axes = F, xlab = NA, ylab = NA, log = log)

    # add a box
    ### was: box() #        ---wlr
    if(box){box()} # my fix---wlr

    # calculate adjustment factor for axis labels if the plot is a matrix
    deflate <- 1
    if (par("mfg")[3] == 2 &
          par("mfg")[4] == 2) {
      deflate <- 0.83
    }
    if (par("mfg")[3] > 2 |
          par("mfg")[4] > 2) {
      deflate <- 0.66
    }

    # Calculate the position of axis labels.
    if (length(xat) == 0) {
      ifelse (log == "x" | log == "xy" | log == "yx",
        logxpar <- TRUE,
        logxpar <- FALSE)
      xat <- axTicks(side = 1, log = logxpar)
    }
    if (length(yat) == 0) {
      ifelse (log == "y" | log == "xy" | log == "yx",
              logypar <- TRUE,
              logypar <- FALSE)
      yat <- axTicks(side = 2, log = logypar)
    }

    # calculate the x axis tick locations
    if (is.null(xat)) {
     xat <- axis(side = 1)
    }

    # calculate the y axis tick locations
    if (is.null(yat)) {
      yat <- axis(side = 2)
    }

    # add the x axis
    if (par("mfg")[1] == par("mfg")[3] & annx == TRUE) {
      axis(side = 1, at = xat, labels = NA, tck = -tick.length, lwd = 0, lwd.ticks = 1)
      if (!is.null(xticklab)) {
        if (is.character(xticklab) & length(xticklab) == 1) {
          if (xticklab == "sci_notation") {
            axis(side = 1, at = xat, tick = FALSE, line = xpos, cex.axis =  .9*text.size,
                 labels = sci_notation(xat))
          }
        }
        if (is.character(xticklab)) {
          axis(side = 1, at = xat, tick = FALSE, line = xpos, cex.axis =  .9*text.size,
               labels = xticklab)
        }
      } else {
        axis(side = 1, at = xat, tick = FALSE, line = xpos, cex.axis =  .9*text.size,
             labels = xticklab)
      }
      mtext(side = 1, xlab, line = xlabpos, cex = 1*text.size*deflate)
    }

    # add the y axis
    if (par("mfg")[2] == 1 & anny == TRUE) {
      axis(side = 2, at = yat, las = 1, labels = NA, tck = -tick.length, lwd = 0, lwd.ticks = 1)
      if (!is.null(yticklab)) {
        if (is.character(yticklab) & length(yticklab) == 1) {
          if (yticklab == "sci_notation") {
            yaxislabels <- axis(side = 2, at = yat, las = 1, tick = FALSE, line = ypos, cex.axis =  .9*text.size,
                                labels = sci_notation(yat))
          }
        }
        if (is.character(yticklab)) {
          axis(side = 2, at = yat, tick = FALSE, line = ypos, cex.axis =  .9*text.size,
               labels = yticklab)
        }
      } else {
        yaxislabels <- axis(side = 2, at = yat, las = 1, tick = FALSE, line = ypos, cex.axis =  .9*text.size,
                            labels = yticklab)
      }
      if (is.null(ylabpos)) {
        ### ylabpos <- 0.5 + 0.5*max(nchar(yaxislabels))  ### was this ---wlr
        if(exists('yaxislabels')){                        ### my fix
          ylabpos <- 0.5 + 0.5*max(nchar(yaxislabels))    ### my fix
        } else{                                           ### my fix
          ylabpos <- 0.5                                  ### my fix
        }                                                 ### my fix
      }
      if (!is.null(yticklab)) {
        if (is.character(yticklab) & length(yticklab )== 1) {
          if (yticklab == "sci_notation") {
            ylabpos = 3.2
          }
        }
      }
      mtext(side = 2, ylab, line = ylabpos, cex = 1*text.size*deflate)
    }

    # add the plot label
    mtext(side = 3, main, line = .1, cex = 1*text.size*deflate)

    #   plotPar <<- list(xlim = xlim, ylim = ylim,
    #                   xlab = xlab, ylab = ylab,
    #                   main = main, text.size = text.size,
    #                   tick.length = tick.length,
    #                   xpos = xpos, ypos = ypos,
    #                   xat = xat, yat = yat,
    #                   xlabpos = xlabpos, ylabpos = ylabpos,
    #                   annx = annx, anny = anny,
    #                   box = box)
    .compactrEnv$plotPar <- list(xlim = xlim, ylim = ylim,
                                 xlab = xlab, ylab = ylab,
                                 main = main, text.size = text.size,
                                 tick.length = tick.length,
                                 xpos = xpos, ypos = ypos,
                                 xat = xat, yat = yat,
                                 xticklab = xticklab, yticklab = yticklab,
                                 xlabpos = xlabpos, ylabpos = ylabpos,
                                 annx = annx, anny = anny,
                                 box = box, log = log)

    #cat('t.l=', .compactrEnv$plotPar$tick.length, '...MYEPLOT]\n') ##WLR
  }


.compactrEnv <- new.env()
```

```
addxaxis <- function() {
  # calculate adjustment factor for axis labels if the plot is a matrix
  #cat('[MYaddxaxis...') ###WLR
  deflate <- 1
  if (par("mfg")[3] == 2 &
        par("mfg")[4] == 2) {
    deflate <- 0.83
  }
  if (par("mfg")[3] > 2 |
        par("mfg")[4] > 2) {
    deflate <- 0.66
  }
  # add the axis
  axis(side = 1, at = .compactrEnv$plotPar$xat, labels = NA, tck = -.compactrEnv$plotPar$tick.length,
       lwd = 0, lwd.ticks = 1)
  axis(side = 1, at = .compactrEnv$plotPar$xat, tick = FALSE, line = .compactrEnv$plotPar$xpos,
       cex.axis =  .9*.compactrEnv$plotPar$text.size)
  mtext(side = 1, text = .compactrEnv$plotPar$xlab, line = .compactrEnv$plotPar$xlabpos,
        cex = 1*.compactrEnv$plotPar$text.size*deflate)
  #cat('..., t.l=',.compactrEnv$plotPar$tick.length,'...MYaddxaxis]\n') ###WLR
}
```

6 Load Tables

Load the main SNP data file(s) based on the parameters set in section 4, and possibly prune to just Chromosome 1. (In the later case, the result is cached by `load.snp.tables` into `00common/mycache`, so we can reload it more quickly.)

```
# short names to keep the following chunk compact
tb <- params$load.tb
tset <- list(NULL, NULL, NULL, NULL) # tset = 'table set'
```

```
if(tb[1]){tset[[1]] <- load.snp.tables(use.chr1.tables = FALSE, data.name='full.tables.01.26.14')} # see wlr.R fo
if(tb[2]){tset[[2]] <- load.snp.tables(use.chr1.tables = TRUE , data.name='full.tables.01.26.14')}
if(tb[3]){tset[[3]] <- load.snp.tables(use.chr1.tables = FALSE, data.name='full.tables.02.25.15')}
if(tb[4]){tset[[4]] <- load.snp.tables(use.chr1.tables = TRUE , data.name='full.tables.02.25.15')}

# Loading ../00common/mycache/snp.tables.chr1.qfiltered.rda ...Loaded.
# Bandaiding qfiltered tables...
```

The tersely-named `tset` list is sometimes convenient, but give them more descriptive names, too.

```
snp.tables.full.unfiltered <- tset[[1]]
snp.tables.chr1.unfiltered <- tset[[2]]
snp.tables.full.qfiltered  <- tset[[3]]
snp.tables.chr1.qfiltered  <- tset[[4]]
```

The main analysis just uses one of the potentially 4 table sets, using the shorter name `snp.tables` for this default choice. Pick it according to the priority specified in section 4.

```
snp.tables <- tset.picker(priority=params$pri, table.set=tset)
```

Which tables have we got?:

```
cat('This analysis uses: (', paste(unlist(lapply(tset,which.snp.tables)),collapse=', '), ') SNP tables.\n')

# This analysis uses: ( NULL, NULL, NULL, Chr1-qfiltered ) SNP tables.

cat('Main analysis focuses on', which.snp.tables(snp.tables), '\n')

# Main analysis focuses on Chr1-qfiltered
```

A LaTeX hack: I want which.snp.tables info in doc title/page headers, but it is unknown until now, so the following writes a command definition `\whichsnptables` into the .aux file, which is read during the *next* LaTeX run, when `\begin{document}` is processed:

```
\makeatletter
\immediate\write\@auxout{\noexpand\gdef\noexpand\whichsnptables{Chr1-qfiltered}}
\makeatother
```

Subsequent analysis was initially all directed at Chr1, so following built/cached/loaded the Chr1 subset. Possibly some of the code will break if given bigger tables; we'll see... In general, I have *not* updated the discussion to reflect genome-wide analysis.

```
chr1.len   <-  genome.length.constants()$chr1.length  ## 3042585
if(exists('snp.tables.chr1.qfiltered') && exists('snp.tables.chr1.unqfiltered')){
  # If have both, where is new unequal to old?
  uneq <- snp.tables.chr1.qfiltered[[1]]$Ref[1:chr1.len] != snp.tables.chr1.unqfiltered[[1]]$Ref[1:chr1.len]
  cat('Sum uneq:', sum(uneq,na.rm=T), '\n')
  cat('Sum NA:  ', sum(is.na(uneq)),  '\n')
  print(which(is.na(uneq))[1:10])
  seecounts(which(is.na(uneq))[1:4],snp.tables=snp.tables.qfiltered,debug=F)
}
```

Also load the desert tables:

```
# from svn+ssh://ceg1.ocean.washington.edu/var/svn/7_strains/trunk/code/snpNB/data
load('../../../data/des.rda')
```

7 The Fig

7.1 Filtering

Code below makes several different versions, with different filtering. (Not sure which I like best. Last is probably "cleanest," but has the most complex processing chain to explain. See end of section 9.2 for more on this.)

All start by finding positions having a certain minimum coverage in all 7 strains (21 at time of writing) and a certain minimum fraction of nonreference reads in at least one strain (10% at time of writing; note that 10% of 21 is greater than 2, so a minimum of 3 nonreference reads must be seen in some strain). Exact values for these parameters are printed by the code below.

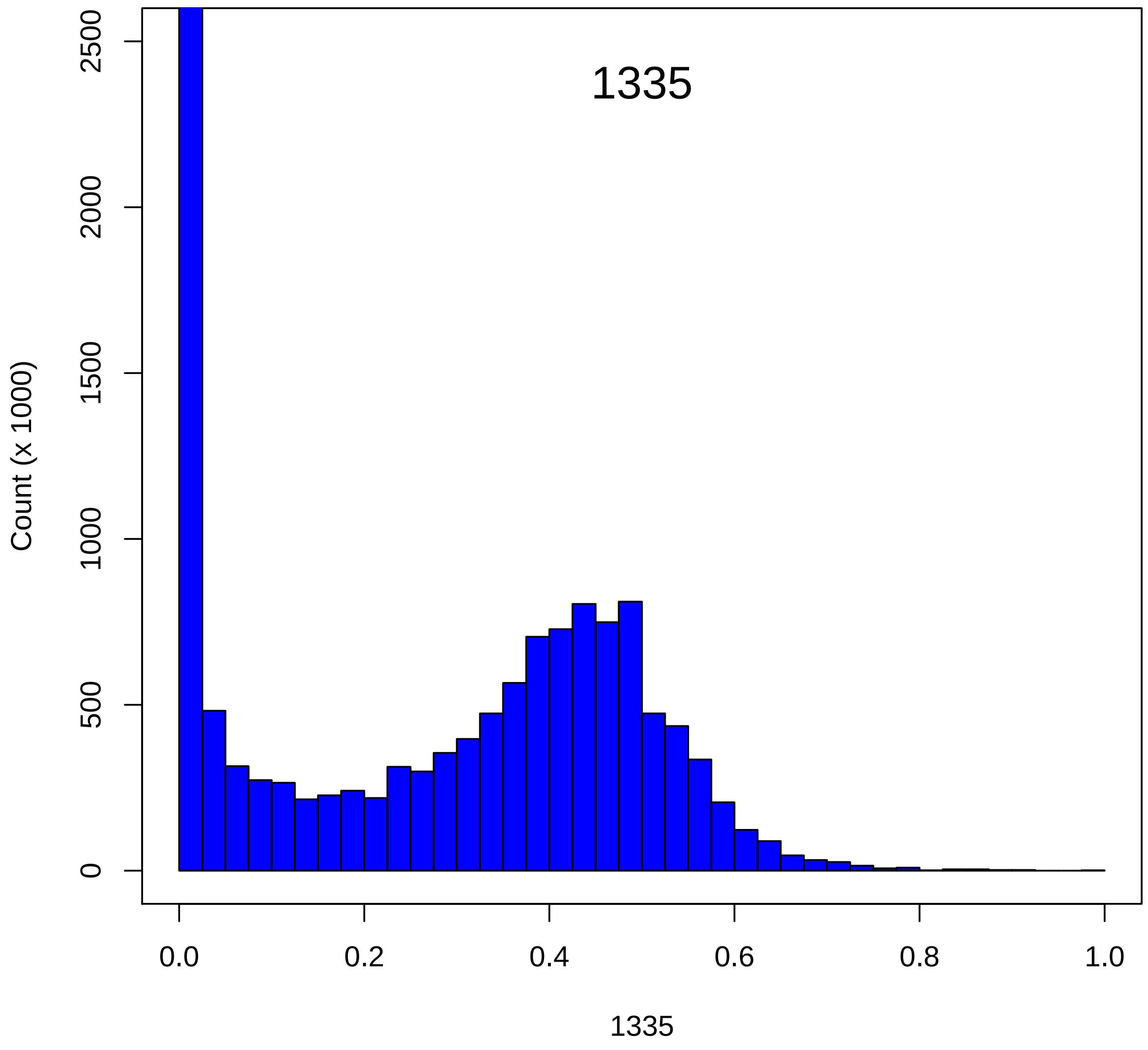
The "Julie Filters" are a late addition trying to address Julie's concern that our plots highlight "agreement in the *amount* of nonreference," but do not measure agreement in the *nature* of the nonreference; e.g. 50% nonref in two strains might both be "A," but could also be "A" in one strain but "G" in another. In short, filter 1 removes low read counts, e.g., filter1=0 does nothing, filter1=1 removes singletons, etc.; on top of that, filter2=T removes all but the max nonref count (including all but one copy of ties, if any). In any case, the value plotted is the ratio of total remaining nonref reads to that plus matches. See section 9 for more on this.

```
# make a piecewise smooth transform function for smoothScatter.
#
#   xformer returns an anonymous function. This function is called by ScatterSmooth with a vector of
#   values, and returns an equal-length vector of transformed values.
#
#   As a side effect, the input vector may be reported out through 'logger', which was useful to see
#   the scale and range of values ScatterSmooth generated.
#
#   I tried several things for the transform itself, e.g. 'x^a below threshold, then x^b', but
#   eventually settled on '(asinh(x))^c'.
#
logger <- NULL
xformer <- function(th = 1.8, a = 0.2, b = 0.1, c=.33){
  return(function(x){
    if(is.null(logger)){logger <<- x}
    return(asinh(x)^c)
    #return(min(asinh(x),th))
    #return(ifelse(x<th, x^a, th^a + asinh(x-th)))
  }
  )
}
```

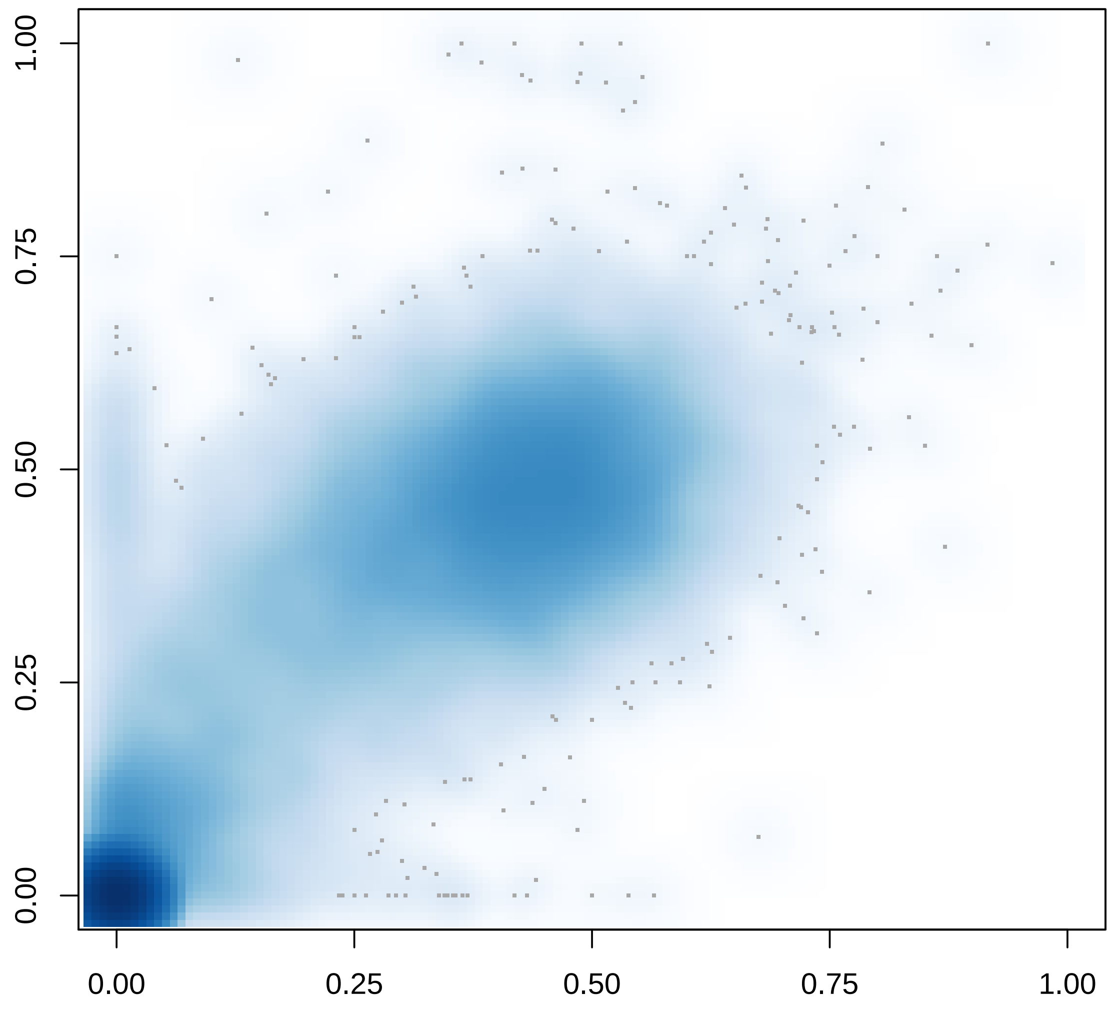
```
# following call filters by various params, generates smooth-scatter .pdf's, and returns a blob
# containing the plotted data, for further analysis.
set.seed(1)
filt99 <- nrf.6plus1smooth(snp.tables=snp.tables,sample=3e6,pch='.',export=T,
                           min.cover=10, max.cover=120,
                           julie.filter1=0,julie.filter2=T,xform=xformer(),smooth=T, cex=.1,
                           fig.path=figdir)

# null mask 3042585 positions.
# nrf.6plus1: From a region of length: 3042585 we identified all positions satisfying:
# 10 <= coverage <= 120 in *all* 7 isolates,
# and having 0.1 <= nr.frac <= 1.0 in *at least 1* of them.
# In these positions, counts <= 0 were forced to zero.
# Nonref fraction includes only the max nonref count.
# From these 35291 positions, we sampled 35291 to plot.

nrfall <- filt99$nrfall
samp <- filt99$sample
st <- 5
```

```r
# function to plot marginal histograms (upright or rotated), by default based on
# return value from previous chunk.  Probably needs tweaking, but it's a start.
# For filt99 above on chr1, default ymax clips only bin 0, except in 1014,
# where 3rd or 4th bin has count 4625.
# "Clip" code/params puts a white diagonal line  or arrow across bin 0 at ymax to
# visually flag clipping, but in the end I didn't think it looked very good.
gamma.hist <- function(st=7,nrfall=filt99$nrfall, samp=filt99$sample, bins=40,
                       ymax=2500*ifelse(which.snp.tables(string.val=FALSE)[1]=='full',10,1),
                       lcex=1.6, rotate=F, compact=F,
                       show.count.axis=T,
                       panel.label='',
                       clip=FALSE, clip.lwd=3, clip.col='white', clip.pct=1){
  breaks <- (0:bins)/bins      # break point for histogram bins
  xleft <- breaks[-(bins+1)]   # left edges of plotted rectangles
  xright <- breaks[-1]         # right edges
  ybot <- rep(0,bins)          # rectangle bottoms
  ytop <- hist(nrfall[samp,st],breaks=breaks,plot=F)$counts # rectangle tops
  cat('Counts clipped at', ymax, '; Top 5:', sort(ytop)[bins:(bins-4)], '\n')
  xl <- colnames(nrfall)[st] # paste(colnames(nrfall)[st], 'R Distribution')  # axis labels
  yl <- 'Count (x 1000)'
  yl.pos <- 2.5
  if(!rotate){
    # normal histogram orientation
    if(!compact){
      plot(0,0, xlab=xl, ylab=yl, xlim=c(0,1), ylim=c(0,ymax), type='n')
    } else {
      eplot(xlim=c(0,1), ylim=c(0,ymax), annx=F, anny=F, box=FALSE)
      if(show.count.axis){
        axis(2, at=c(0,ymax/2,ymax), labels=c(0,ymax/2,ymax)/1000, tick=TRUE)
        mtext(yl, side=2, line=yl.pos, cex=.9)
      }
    }
    rect(xleft,ybot,xright,ytop,border='black',col='blue')
    if(clip){
      # flag clip @ ymax in 1st bin
      #lines(c(0,1/bins),ymax*(1+clip.pct/100*c(-1,1)),col=clip.col,lwd=clip.lwd)
      polygon(1/2/bins*c(-0.5,1,2.5,2.5,-0.5), ymax*c(1,1.02,1,2,2), border=NA, col='white')
    }
    text(0.5, 0.95*ymax, xl, cex=lcex)            # identify strain
    text(0.1, 0.95*ymax, panel.label, cex=lcex)   # Panel A, B, ...
  } else {
    # rotate to put it against y axis on right;
    # i.e. rotate 270 deg clockwise, then flip about vertical axis.
    # [I think putting it on the left, i.e. just 270 rotation, just requires
    # changing plot to xlim=c(max,0).]
    if(!compact){
      plot(0,0, ylab=xl, xlab=yl, ylim=c(0,1), xlim=c(0,ymax), type='n')
    } else{
      eplot(ylim=c(0,1), xlim=c(0,ymax), annx=F, anny=F, box=FALSE)
      if(show.count.axis){
        axis(1, at=c(0,ymax/2,ymax), labels=c(0,ymax/2,ymax)/1000, tick=TRUE)
        mtext(yl, side=1, line=yl.pos, cex=.9)
      }
    }
    rect(ybot,xleft,ytop,xright,border='black',col='blue')
    if(clip){
      # flag clip @ ymax in 1st bin
      #lines(ymax*(1+clip.pct/100*c(-1,1)), c(0,1/bins), col=clip.col, lwd=clip.lwd)
      #lines(ymax*(1+clip.pct/100*c(-1,1)), c(1/bins,0), col=clip.col, lwd=clip.lwd)
      #cat('polygon:',ymax*c(1,1.05,1,2,2),'*', 1/2/bins*c(0,1,2,2,0),'\n')
      polygon(ymax*c(1,1.02,1,2,2), 1/2/bins*c(-0.5,1,2.5,2.5,-0.5), border=NA, col='white')
    }
    text(0.95*ymax, 0.5, xl, srt=90, cex=lcex)    # identify strain
    text(0.2*ymax, 0.95, panel.label, cex=lcex)   # Panel A, B, ...
  }
}
# margin.scat(6,2,7) #debug
```

```
gamma.hist() # default marginal histo

# Counts clipped at 2500 ; Top 5: 25041 811 804 749 728
```



```
# sample scatter-smooth for debugging
#pdf('scatter15-35-j2T',5,5)
smoothScatter(nrfall[samp,7], nrfall[samp,st], pch='.', cex=2, col='gray66', nrpoints=200,
              transformation=xformer(20,.2,.03,0.33),
              xlab='', xlim=0:1, xaxp=c(0,1,4),
              ylab='', ylim=0:1, yaxp=c(0,1,4))
```



```
#dev.off()
```

```
# Main fig 1 & supp Fig S6: smoothScatter of joint R, with marginal histograms.
# Mostly circa 10/2015.
# TO DO, maybe:
#   5 ticks on x,y (but still 3 labels)?
#   y-ticks (no lables) on right scatter
#   y ticks on histos?
#   some marker for clipped y in histos?
#   X: (tried this, doesn't help; change filtering so gyre looks better, maybe get 100 k points or so?)
#
margin.scat <- function(stxl=6, stxr=2, sty=7, label.panels=TRUE){
  # plot main fig with 2 side-by-side smooth-scatters and marginal histograms
  # if stxl is NULL, omit left scatter/margin
  opar <- par(mar=c(0.6,0.3,0.0,0.4),oma=c(3,4,0,0),tck=-.02); on.exit(par(opar))
  # layout in 2 x 4 grid; '0's provide some spacing
  if(!is.null(stxl)){
    layout.mat <- matrix(c(1,0,2,0,4,0,5,3),nrow=2,byrow=T)
  } else {
    layout.mat <- matrix(c(0,0,1,0,0,0,3,2),nrow=2,byrow=T)
  }
  layout(layout.mat, widths=c(2,0.2,2,1),heights=c(1,2),
         respect=matrix(rep(1,8),nrow=2,byrow=T))

  # 1: upper left histo
  if(!is.null(stxl)){
    gamma.hist(stxl,compact=T, show.count.axis=T, panel.label=ifelse(label.panels,'A',''))
  }

  # 2: upper mid histo
  gamma.hist(stxr,compact=T, show.count.axis=is.null(stxl), panel.label=ifelse(label.panels,'B',''))

  # 3: lower right histo
  gamma.hist(sty,rotate=T,compact=T, show.count.axis=T, panel.label=ifelse(label.panels,'C',''))

  # axis labels
  xl1 <- 'R' # paste('R (',colnames(nrfall)[stxl], ')',sep='')
  xl2 <- 'R' # paste('R (',colnames(nrfall)[stxr], ')',sep='')
  yl  <- 'R' # paste('R (',colnames(nrfall)[sty],  ')',sep='')

  # 4: lower left scatter
  if(!is.null(stxl)){
    smoothScatter(nrfall[samp,stxl], nrfall[samp,sty],
                  pch='.', cex=2, col='gray66', nrpoints=200,
                  transformation=xformer(20,.2,.03,0.33),
                  xlab=xl1, xlim=0:1, xaxp=c(0,1,2),
                  ylab=yl,  ylim=0:1, yaxp=c(0,1,2),xaxt='s',yaxt='s')
    mtext(xl1,side=1, line=2.5, cex=.9)
    mtext(yl, side=2, line=2.5, cex=.9)
    if(label.panels){text(.10, .95, 'D', cex=1.5)}
  }

  # 5: lower mid scatter
  #eplot(0:1,0:1)
  smoothScatter(nrfall[samp,stxr], nrfall[samp,sty],
                pch='.', cex=2, col='gray66', nrpoints=200,
                transformation=xformer(20,.2,.03,0.33),
                xlab=xl2, xlim=0:1, xaxp=c(0,1,2),
                ylab='',  ylim=0:1, yaxp=c(0,1,2),xaxt='s',yaxt=ifelse(is.null(stxl),'s','n'))
  mtext(xl2, side=1, line=2.5, cex=.9)
  if(is.null(stxl)){
    mtext(yl, side=2, line=2.5, cex=.9)
  }
  if(label.panels){text(.10, .95, 'E', cex=1.5)}
}
```

Repeating, the chatter from the call building `filt99` defines the data being summarized in these plots:

```
cat(filt99$chatter)

# null mask 3042585 positions.
# nrf.6plus1: From a region of length: 3042585 we identified all positions satisfying:
# 10 <= coverage <= 120 in *all* 7 isolates,
# and having 0.1 <= nr.frac <= 1.0 in *at least 1* of them.
# In these positions, counts <= 0 were forced to zero.
# Nonref fraction includes only the max nonref count.
# From these 35291 positions, we sampled 35291 to plot.
```

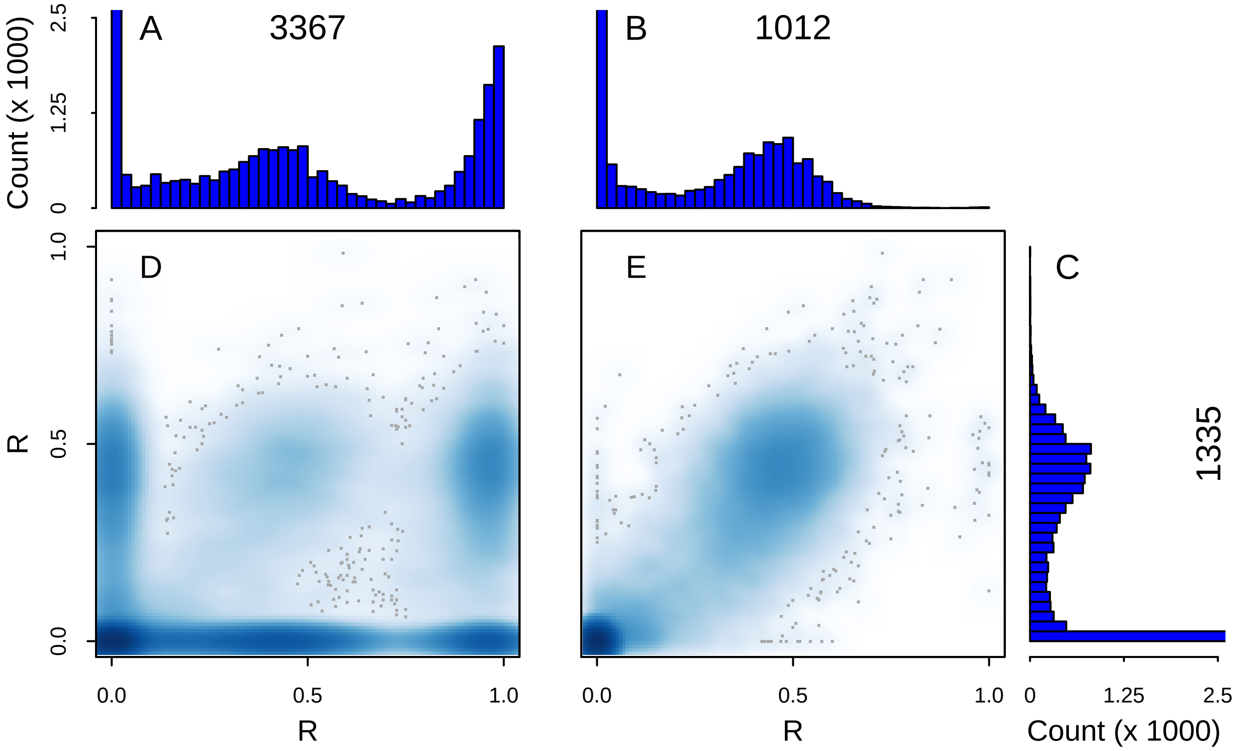
```
pdf(fpath('mscat-6-2-7'), width=6.5, height=4)
margin.scat(6,2,7,label.panels=TRUE)
```

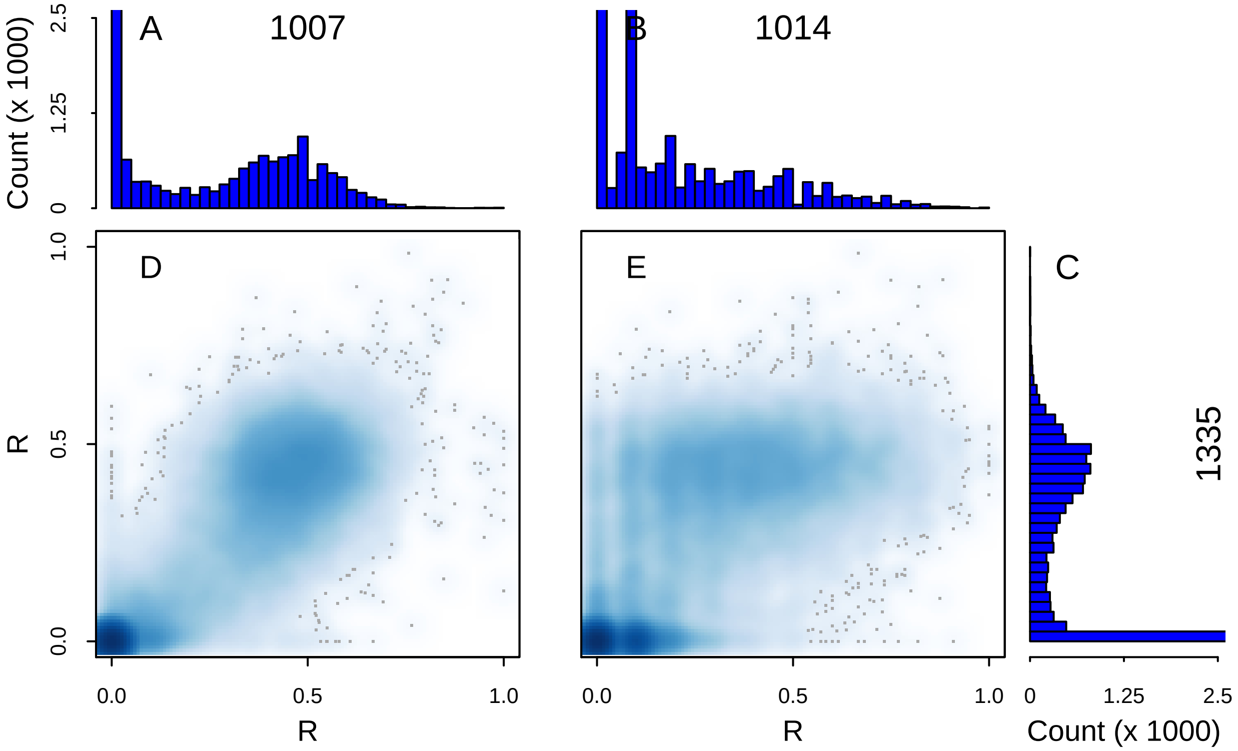
```
# Counts clipped at 2500 ; Top 5: 16283 2126 1619 1160 813
# Counts clipped at 2500 ; Top 5: 24452 925 865 841 719
# Counts clipped at 2500 ; Top 5: 25041 811 804 749 728

dev.off()

# pdf
#   2
```

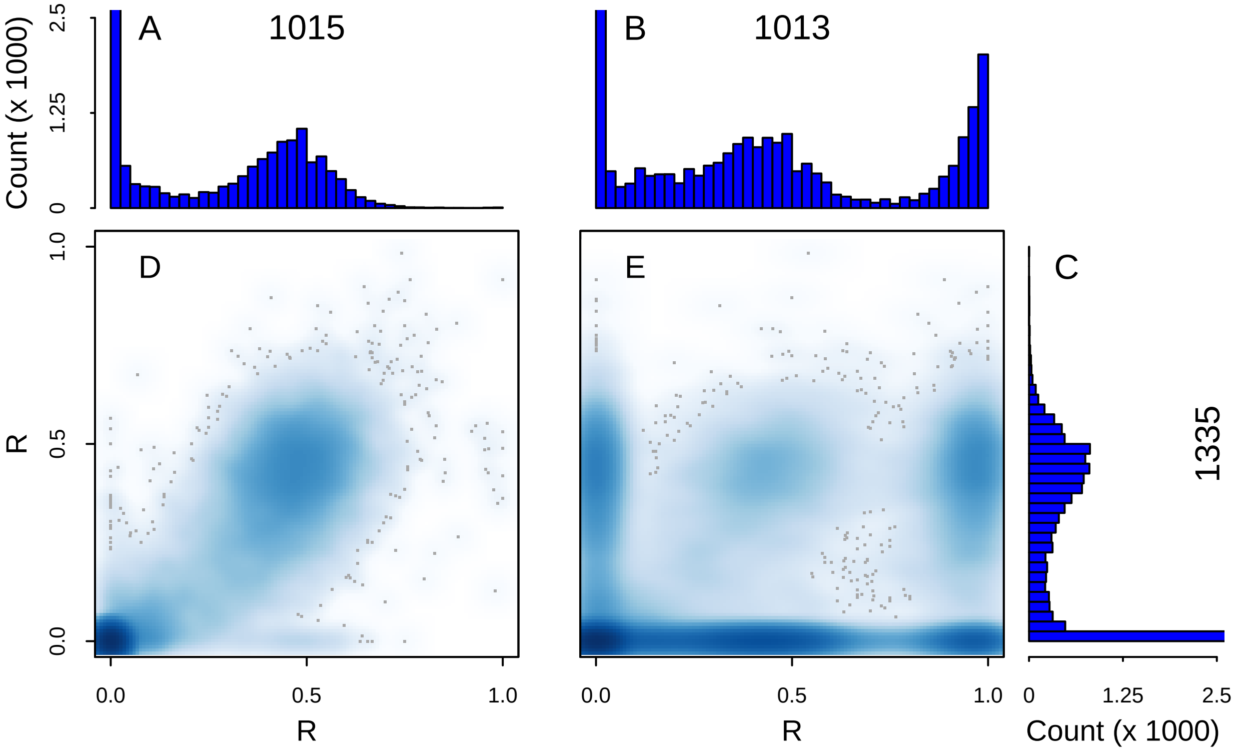
```
pdf(fpath('mscat-1-4-7'), width=6.5, height=4)
margin.scat(1,4,7)

# Counts clipped at 2500 ; Top 5: 24172 942 697 689 670
# Counts clipped at 2500 ; Top 5: 20258 4625 950 731 587
# Counts clipped at 2500 ; Top 5: 25041 811 804 749 728

dev.off()

# pdf
#   2
```

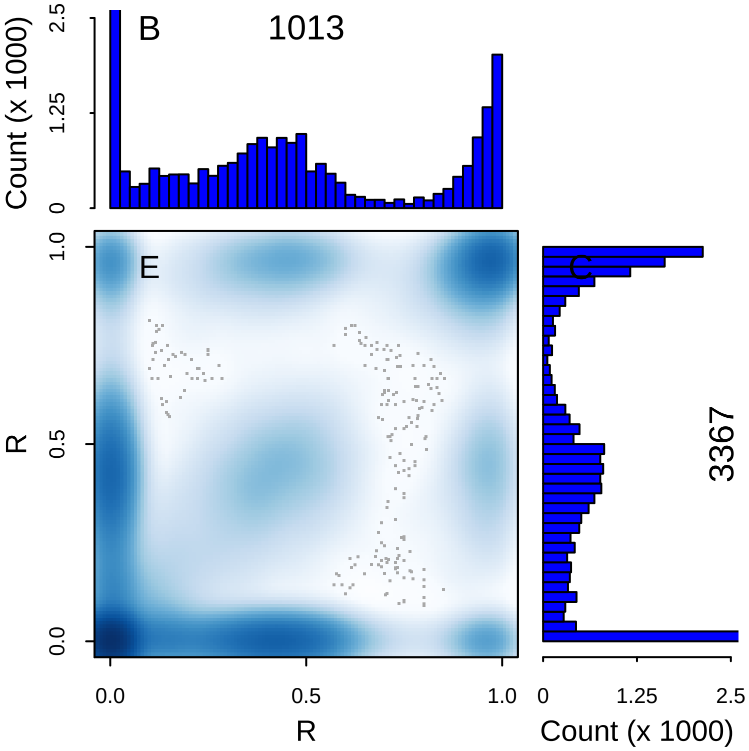
```
pdf(fpath('mscat-5-3-7'), width=6.5, height=4)
margin.scat(5,3,7)

# Counts clipped at 2500 ; Top 5: 24248 1043 890 871 730
# Counts clipped at 2500 ; Top 5: 15320 2019 1327 975 931
# Counts clipped at 2500 ; Top 5: 25041 811 804 749 728

dev.off()

# pdf
#   2
```

```
pdf(fpath('mscat-null-3-6'), width=6.5, height=4)
margin.scat(NULL,3,6) # need 1 x 1 version here, rather than 1 x 2

# Counts clipped at 2500 ; Top 5: 15320 2019 1327 975 931
# Counts clipped at 2500 ; Top 5: 16283 2126 1619 1160 813

dev.off()

# pdf
#   2
```


A
3367
B
1012
C
1335
D
E
Count (x 1000)
0
1.25
2.5
R
0.0
0.5
1.0
R
0.0
0.5
1.0
R
0.0
0.5
1.0
0
1.25
2.5
Count (x 1000)


A
1007
B
1014
C
1335
D
E
Count (x 1000)
0
1.25
2.5
R
0.0
0.5
1.0
R
0.0
0.5
1.0
R
0.0
0.5
1.0
0
1.25
2.5
Count (x 1000)





8 "6+1" with dots

Generate versions of the "6+1" plots (dots, not smoothed), with various filtering parameters. This was an earlier idea for displaying the data. I like the smooth-scatters better; keeping this "just in case."

```
pack.debug()
yaxislabels <- '' ### some bug in compactr; this var sometimes used but undefined
make.dots <- TRUE
if(make.dots){
  pdf(fpath('6+1julie0F'), width=1.75, height=8.25)
  set.seed(1)
  filt1 <- nrf.6plus1(snp.tables=snp.tables,sample=10000,pch='.',export=T,julie.filter1=0,julie.filter2=F)
  dev.off()
}

# null mask 3042585 positions.
# nrf.6plus1: From a region of length: 3042585 we identified all positions satisfying:
#  21 <= coverage <= 150 in *all* 7 isolates,
# and having 0.1 <= nr.frac <= 1.0 in *at least 1* of them.
# From these 5213 positions, we sampled 10000 to plot.
# pdf
#   2
```

```
if(make.dots){
  pdf(fpath('6+1julie1F'), width=1.75, height=8.25)
  set.seed(1)
  filt2 <- nrf.6plus1(snp.tables=snp.tables,sample=10000,pch='.',export=T,julie.filter1=1,julie.filter2=F)
  dev.off()
}

# null mask 3042585 positions.
# nrf.6plus1: From a region of length: 3042585 we identified all positions satisfying:
#  21 <= coverage <= 150 in *all* 7 isolates,
# and having 0.1 <= nr.frac <= 1.0 in *at least 1* of them.
# From these 5213 positions, we sampled 10000 to plot.
# pdf
#   2
```

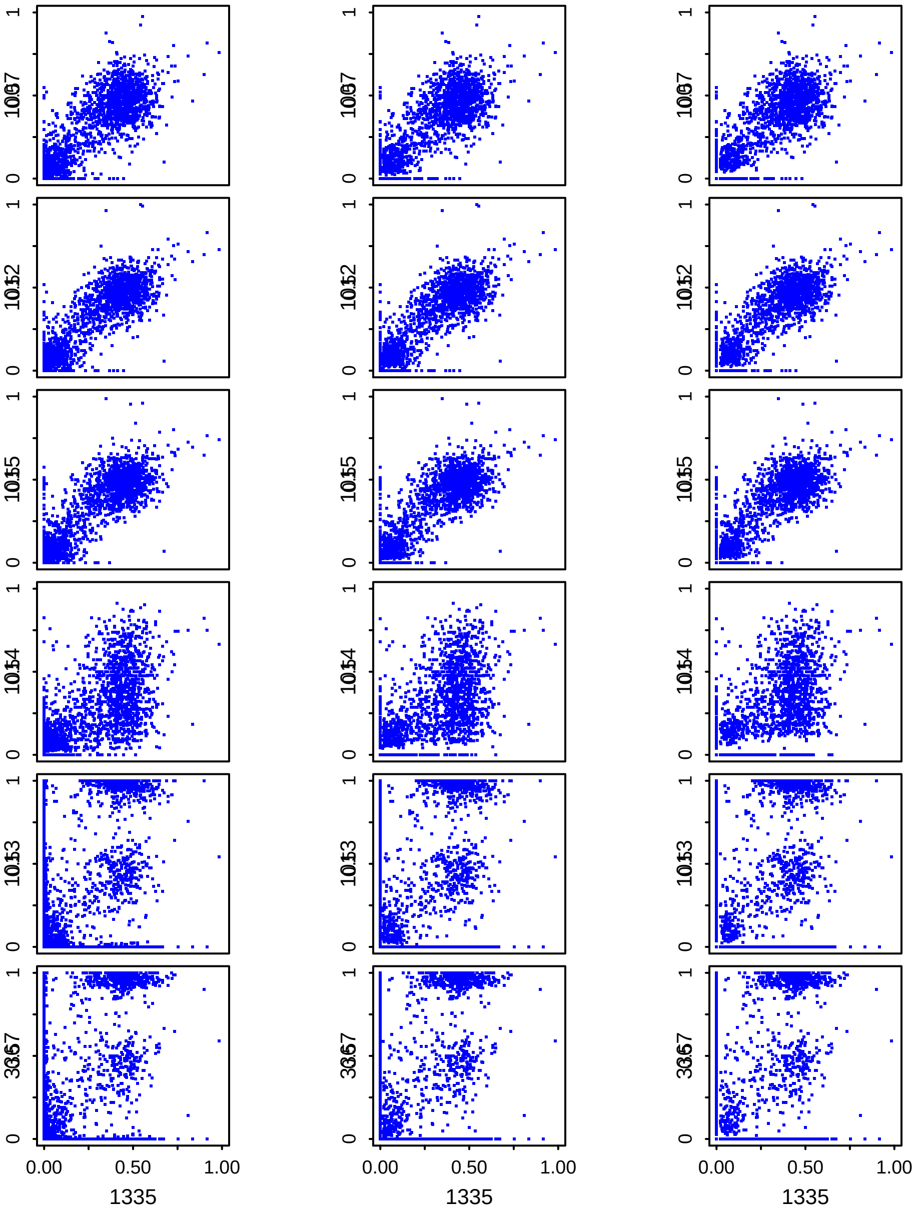
```
if(make.dots){
  pdf(fpath('6+1julie2F'), width=1.75, height=8.25)
  set.seed(1)
  filt3 <- nrf.6plus1(snp.tables=snp.tables,sample=10000,pch='.',export=T,julie.filter1=2,julie.filter2=F)
  dev.off()
}

# null mask 3042585 positions.
# nrf.6plus1: From a region of length: 3042585 we identified all positions satisfying:
#  21 <= coverage <= 150 in *all* 7 isolates,
# and having 0.1 <= nr.frac <= 1.0 in *at least 1* of them.
# From these 5213 positions, we sampled 10000 to plot.
# pdf
#   2
```

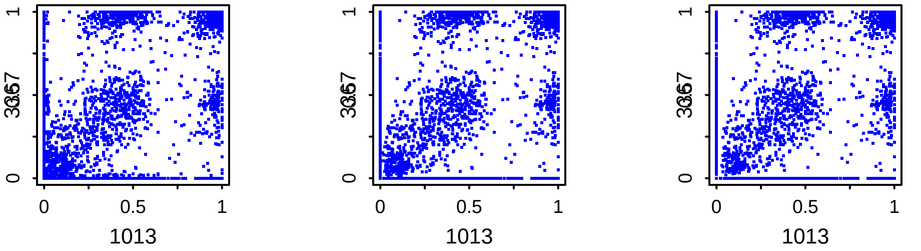
```
if(make.dots){
  pdf(fpath('6+1julie0T'), width=1.75, height=8.25)
  set.seed(1)
  filt4 <- nrf.6plus1(snp.tables=snp.tables,sample=10000,pch='.',export=T,julie.filter1=0,julie.filter2=T)
  dev.off()
}

# null mask 3042585 positions.
# nrf.6plus1: From a region of length: 3042585 we identified all positions satisfying:
#  21 <= coverage <= 150 in *all* 7 isolates,
# and having 0.1 <= nr.frac <= 1.0 in *at least 1* of them.
# From these 5213 positions, we sampled 10000 to plot.
# pdf
#   2
```

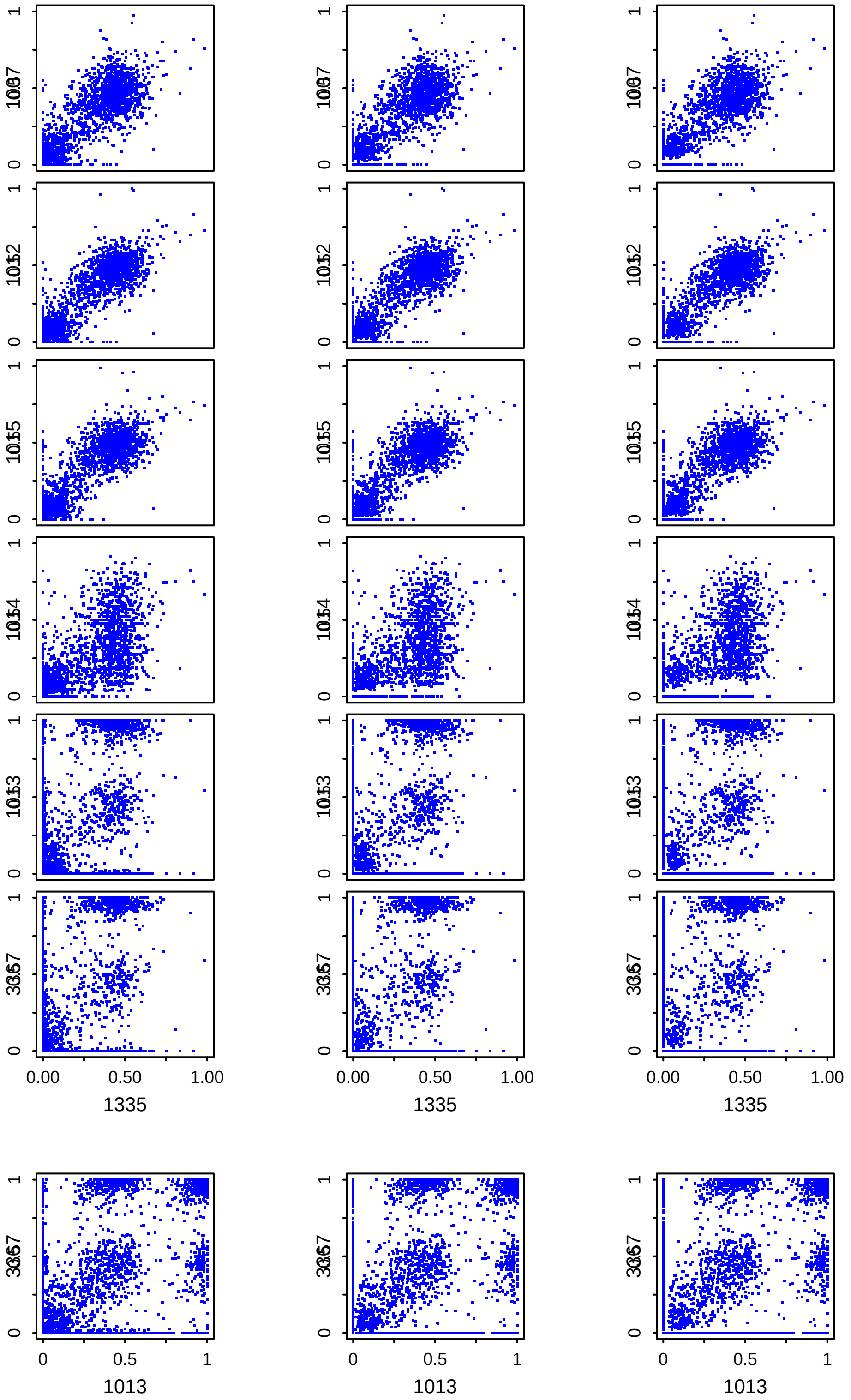
```
if(make.dots){
  pdf(fpath('6+1julie1T'), width=1.75, height=8.25)
  set.seed(1)
  filt5 <- nrf.6plus1(snp.tables=snp.tables,sample=10000,pch='.',export=T,julie.filter1=1,julie.filter2=T)
  dev.off()
}

# null mask 3042585 positions.
# nrf.6plus1: From a region of length: 3042585 we identified all positions satisfying:
#  21 <= coverage <= 150 in *all* 7 isolates,
# and having 0.1 <= nr.frac <= 1.0 in *at least 1* of them.
# From these 5213 positions, we sampled 10000 to plot.
# pdf
#   2
```

```
if(make.dots){
  pdf(fpath('6+1julie2T'), width=1.75, height=8.25)
  set.seed(1)
  filt6 <- nrf.6plus1(snp.tables=snp.tables,sample=10000,pch='.',export=T,julie.filter1=2,julie.filter2=T)
  dev.off()
}

# null mask 3042585 positions.
# nrf.6plus1: From a region of length: 3042585 we identified all positions satisfying:
#  21 <= coverage <= 150 in *all* 7 isolates,
# and having 0.1 <= nr.frac <= 1.0 in *at least 1* of them.
# From these 5213 positions, we sampled 10000 to plot.
# pdf
#   2
```

```
if(make.dots){
  # verify consistent sampling
  all(filt1$sample==filt2$sample) &&
  all(filt1$sample==filt3$sample) &&
  all(filt1$sample==filt4$sample) &&
  all(filt1$sample==filt5$sample) &&
  all(filt1$sample==filt6$sample)
}

# [1] TRUE
```


1
0.5
0
1067
1012
1015
1014
1013
3367
0.00
0.50
1.00
1335
1335
1335


1
0.5
0
3367
0
0.5
1
1013
1013
1013



Proposed caption: ...

9 Post-Analysis

The figure plots total nonref count over coverage, without regard to how nonref count is split among the three possibilities or to whether the same nonref nucleotide is dominant in different strains. Julie expressed concerned about this; should we change it? It is at least confusing, so we need to think how to explain it. (Following analysis mostly based on the un-Julie-filtered data for the 1st plot.)

9.1 Extract shared positions used in pairwise plots

Plot function returns data useful for making the plots themselves, but we need to cross-reference this to the original SNP tables.

```
str(filt1)

# List of 2
#  $ sample: int [1:5213] 1 2 3 4 5 6 7 8 9 10 ...
#  $ nrfall: num [1:5213, 1:7] 0.0741 0 0 0.0385 0 ...
#   ..- attr(*, "dimnames")=List of 2
#   .. ..$ : chr [1:5213] "Chr1:118" "Chr1:128" "Chr1:279" "Chr1:567" ...
#   .. ..$ : chr [1:7] "1007" "1012" "1013" "1014" ...
```

Re-form indices into snp.tables from `nrfall` row names, and, as a sanity check, visually spot check that these satisfy the expected filters.

```
selected <- as.integer(sub('Chr1:', '', rownames(filt1$nrfall), fixed=T))
seecounts(selected[1:5],snp.tables=snp.tables)

#     chr  pos Ref Strain   A   G   C   T SNP  exon indel nrf rat
# 1  Chr1  118   A
# 2                 1007  50   4   0   0   0 FALSE FALSE
# 3                 1012  88   5   0   0   0 FALSE FALSE
# 4                 1013 105   4   0   0   0 FALSE FALSE
# 5                 1014  29   4   0   0   0 FALSE FALSE
# 6                 1015  72   3   0   0   0 FALSE FALSE
# 7                 3367  67   5   0   0   0 FALSE FALSE
# 8                 1335 139   7   0   0   0 FALSE FALSE
# 9  Chr1  128   C
# 10                1007   0   0  46   0   0 FALSE FALSE
# 11                1012   0   0  64   0   0 FALSE FALSE
# 12                1013   0   0  53  15   0 FALSE FALSE
# 13                1014   0   0  25   0   0 FALSE FALSE
# 14                1015   0   0  59   0   0 FALSE FALSE
# 15                3367   0   0  25   6   0 FALSE FALSE
# 16                1335   0   0 136   1   0 FALSE FALSE
# 17 Chr1  279   T
# 18                1007   0   0   0  46   0 FALSE FALSE
# 19                1012   0   0   0  73   0 FALSE FALSE
# 20                1013  36   0   0  74   0 FALSE FALSE
# 21                1014   0   1   0  40   0 FALSE FALSE
# 22                1015   0   0   0  89   0 FALSE FALSE
# 23                3367  20   0   0  37   0 FALSE FALSE
# 24                1335   0   0   0 139   0 FALSE FALSE
# 25 Chr1  567   T
# 26                1007   0   0   1  25   0  TRUE FALSE
# 27                1012   0   0  14  39   1  TRUE FALSE
# 28                1013   0   0  13  87   0  TRUE FALSE
# 29                1014   0   1   0  23   0  TRUE FALSE
# 30                1015   0   0   8  40   1  TRUE FALSE
# 31                3367   0   0  16  38   1  TRUE FALSE
# 32                1335   0   0   2  99   0  TRUE FALSE
# 33 Chr1 1878   G
# 34                1007   0  26   0   0   0  TRUE FALSE
# 35                1012   0  46   0   0   0  TRUE FALSE
# 36                1013   9  47   0   0   0  TRUE FALSE
# 37                1014   0  24   0   0   0  TRUE FALSE
# 38                1015   0  57   0   0   0  TRUE FALSE
# 39                3367   0  22   0   0   0  TRUE FALSE
# 40                1335   0 135   0   0   0  TRUE FALSE

n.selected <- length(selected)
if(!exists('n.selected')){n.selected <- NA} ### bug chasing...
```

Recast the selection as a long Bool vector, instead of a short vector of indices, for convenience in masking `snp.tables`.

```
longsel <- vector('logical',nrow(snp.tables[[1]]))
longsel[selected] <- T
```

Build a set of subtables with just selected positions.

```
seltab <-NULL
for(st in 1:7){
  seltab[[st]] <- snp.tables[[st]][longsel,]
}
names(seltab) <- names(snp.tables)
```

9.2 Discordant “Nonreference Alleles”

We need to asses the degree of discordance between selected positions in different strains. E.g., in particular, how often do we see significant read counts at *different* nonreference nucleotides at selected positions?

The following text and code snippet is taken verbatim from `shared-snps.rnw`, svn 557, except that I have added `snp.tables` as an explicit parameter to the function, and did a common subexpression optimization (which doesn’t seem much faster, oh well).

> For a given strain, the following function returns a vector of 0:4 to indicate which nonreference nucleotide has the maximum read count at the corresponding position. The values 1..4 indicate that the max count occurred at A, G, C, T, resp. (Ties are resolved arbitrarily ($a < g < c < t$), which possibly deserves further attention.) The value 0 means all nonreference counts are below threshold, based *either* on absolute count *or* as a fraction of coverage. Default only excludes 0 counts.

```
nref.nuc.new <- function(strain=1, mask=T, thresh.count=0, thresh.rate=0.0,
                         snp.tables=snp.tables.01.26.14){
  # extract strain/mask subtable
  subtab <- snp.tables[[strain]][mask,]
  # get read count for max nonref nuc
        nref <- apply(subtab[, c('a', 'g', 'c', 't')], 1, max)
        # where does nref count match a (g,c,t, resp) count
        as <- ifelse(nref == subtab[,'a'],1,0)
        gs <- ifelse(nref == subtab[,'g'],2,0)
        cs <- ifelse(nref == subtab[,'c'],3,0)
        ts <- ifelse(nref == subtab[,'t'],4,0)
        # most positions will show 3 zeros and one of 1:4, so max identifies max nonref count;
        # ties broken arbitrarily  (a<g<c<t)
        merge <- pmax(as,gs,cs,ts)
        # but if max nonref count is zero or below threshold, return 0
        merge[nref == 0 | nref < thresh.count] <- 0
        merge[nref/subtab[,'Cov'] < thresh.rate] <- 0
        return(merge)
}
```

We use this to assess the degree of discordance between strains at the selected positions used for the pairwise scatter plots, with various levels of stringency in the `thresh.count, thresh.rate` parameters. [Note: thresh.rate and thresh.count tests are “strictly less,” so, e.g., thresh.count=1 does *not* eliminate singleton reads; threshcount=1.1 does.]

```
nrf.nuc.list <- function(mask=longsel, thresh.count=0, thresh.rate=0.0, snp.tables=snp.tables.01.26.14){
  nrf.nucs <- NULL
  for(st in 1:7){
    nrf.nucs[[st]] <- nref.nuc.new(strain=st, mask=mask, thresh.count=thresh.count,
                                   thresh.rate=thresh.rate, snp.tables=snp.tables)
  }
  return(nrf.nucs)
}
```

```
nrf.nucs1 <- nrf.nuc.list(mask=T, thresh.count=0.0, thresh.rate=0.00, snp.tables=seltab)
nrf.nucs2 <- nrf.nuc.list(mask=T, thresh.count=1.1, thresh.rate=0.00, snp.tables=seltab)
nrf.nucs3 <- nrf.nuc.list(mask=T, thresh.count=2.1, thresh.rate=0.00, snp.tables=seltab)
nrf.nucs4 <- nrf.nuc.list(mask=T, thresh.count=0.0, thresh.rate=0.05, snp.tables=seltab)
# 5 is identical to 4, since singletons are < 0.05, given min.cov=21
# nrf.nucs5 <- nrf.nuc.list(mask=T, thresh.count=1.1, thresh.rate=0.05, snp.tables=seltab)
nrf.nucs6 <- nrf.nuc.list(mask=T, thresh.count=2.1, thresh.rate=0.05, snp.tables=seltab)

discordance <- function(nrf.nucs, snp.tables=snp.tables.01.26.14){
  # nrf.nucs in 1:4 indicates max nonref nuc is AGCT, resp; 0 means max is 0
  # 'discord' simply counts positions where these differ between strain i & j
  discord <- matrix(NA,nrow=7,ncol=7)
  rownames(discord) <- names(snp.tables)
  colnames(discord) <- names(snp.tables)
  for(i in 1:6){
    for(j in (i+1):7){
      discord[i,j] <- sum(nrf.nucs[[i]] != nrf.nucs[[j]])
    }
  }
  # 'nzdiscord' counts positions where both strains are *nonzero* and differ; these
  # counts go in the upper triangle; lower triangle is count of jointly nonzero positions
  nzdiscord <- matrix(NA,nrow=7,ncol=7)
  rownames(nzdiscord) <- names(snp.tables)
  colnames(nzdiscord) <- names(snp.tables)
  for(i in 1:6){
    for(j in (i+1):7){
      nz <- nrf.nucs[[i]] != 0  & nrf.nucs[[j]] != 0
      nzdiscord[i,j] <- sum((nrf.nucs[[i]] != nrf.nucs[[j]])[nz])
      nzdiscord[j,i] <- sum(nz)
    }
  }
  return(list(discord,nzdiscord))
}

d1 <- discordance(nrf.nucs1, seltab); d1 # thresh.count=0.0, thresh.rate=0.00

# [[1]]
#      1007 1012 1013 1014 1015 3367 1335
# 1007   NA  246 2932  455  275 2937  323
# 1012   NA   NA 2915  449  283 2935  329
# 1013   NA   NA   NA 3035 2948 2723 2940
# 1014   NA   NA   NA   NA  466 3048  487
# 1015   NA   NA   NA   NA   NA 2973  319
# 3367   NA   NA   NA   NA   NA   NA 2950
# 1335   NA   NA   NA   NA   NA   NA   NA
#
# [[2]]
#      1007 1012 1013 1014 1015 3367 1335
# 1007   NA    6   67   12    3   61   11
# 1012 1690   NA   74   10   11   69   11
# 1013 1110 1152   NA   93   83   98   69
# 1014 1594 1626 1107   NA   13   79   14
# 1015 1682 1712 1148 1627   NA   75    9
# 3367 1084 1119 1972 1073 1111   NA   58
# 1335 1645 1672 1128 1600 1684 1097   NA
```

Summary: With no filtering, thousands of selected positions are "discordant" between any pair of strains (counts in the upper triangle of the first matrix), but the vast majority are positions where one strain has some nonref reads while the other has none (remaining counts in the upper triangle of the second matrix; counts of jointly nonzero positions in lower triangle). The largest remaining is the count of 2597 between Wales and Gyre, which is 6% of all selected positions and 19% of positions where both have some nonref reads. However, these numbers drop sharply after filtering out nonref nucleotides receiving only one or two reads and/or cases where the max nonref read count is less than 5% of coverage (below). After these filtering steps, less that 1.5% of positions where both have some nonref reads are discordant between any pair of isolates, and the rate is about an order of magnitude lower between any pair of the 5 (upper triangle of last matrix below).

```
d2 <- discordance(nrf.nucs2, seltab); d2 # thresh.count=1.1, thresh.rate=0.00

# [[1]]
#      1007 1012 1013 1014 1015 3367 1335
# 1007   NA  165 2887  416  197 2911  237
# 1012   NA   NA 2867  408  179 2890  215
# 1013   NA   NA   NA 3003 2878 2741 2884
# 1014   NA   NA   NA   NA  431 3004  415
# 1015   NA   NA   NA   NA   NA 2922  201
# 3367   NA   NA   NA   NA   NA   NA 2914
# 1335   NA   NA   NA   NA   NA   NA   NA
#
# [[2]]
#      1007 1012 1013 1014 1015 3367 1335
# 1007   NA    2   24    3    2   23    3
# 1012 1587   NA   29    6    5   23    4
# 1013  965 1011   NA   21   30   40   25
# 1014 1430 1469  907   NA    7   25    8
# 1015 1582 1626 1017 1469   NA   27    4
# 3367  955  999 1810  911  996   NA   16
# 1335 1528 1573  977 1443 1591  960   NA

d3 <- discordance(nrf.nucs3, seltab); d3 # thresh.count=2.1, thresh.rate=0.00

# [[1]]
#      1007 1012 1013 1014 1015 3367 1335
# 1007   NA  211 2852  501  240 2878  253
# 1012   NA   NA 2864  495  194 2888  222
# 1013   NA   NA   NA 2971 2878 2743 2873
# 1014   NA   NA   NA   NA  524 2982  486
# 1015   NA   NA   NA   NA   NA 2901  220
# 3367   NA   NA   NA   NA   NA   NA 2897
# 1335   NA   NA   NA   NA   NA   NA   NA
#
# [[2]]
#      1007 1012 1013 1014 1015 3367 1335
# 1007   NA    2   18    2    1   16    1
# 1012 1484   NA   23    1    4   19    3
# 1013  903  950   NA   16   23   33   23
# 1014 1254 1307  808   NA    4   17    3
# 1015 1481 1556  955 1306   NA   20    1
# 3367  900  947 1758  814  953   NA   15
# 1335 1438 1505  921 1288 1517  916   NA

d4 <- discordance(nrf.nucs4, seltab); d4 # thresh.count=0.0, thresh.rate=0.05

# [[1]]
#      1007 1012 1013 1014 1015 3367 1335
# 1007   NA  199 2867  426  245 2908  272
# 1012   NA   NA 2869  412  202 2894  226
# 1013   NA   NA   NA 2996 2884 2754 2860
# 1014   NA   NA   NA   NA  436 3002  439
# 1015   NA   NA   NA   NA   NA 2923  253
# 3367   NA   NA   NA   NA   NA   NA 2889
# 1335   NA   NA   NA   NA   NA   NA   NA
#
# [[2]]
#      1007 1012 1013 1014 1015 3367 1335
# 1007   NA    2   16    3    1   17    1
# 1012 1484   NA   17    4    3   14    2
# 1013  913  917   NA   18   19   30   18
# 1014 1392 1404  875   NA    5   20    3
# 1015 1480 1507  930 1412   NA   14    0
# 3367  891  901 1735  871  906   NA   13
# 1335 1405 1433  880 1348 1438  861   NA

# d5 <- discordance(nrf.nucs5, seltab); d5 # thresh.count=1.1, thresh.rate=0.05
d6 <- discordance(nrf.nucs6, seltab); d6 # thresh.count=2.1, thresh.rate=0.05
```

```
# [[1]]
#      1007 1012 1013 1014 1015 3367 1335
# 1007   NA  206 2845  504  247 2876  262
# 1012   NA   NA 2850  481  203 2888  229
# 1013   NA   NA   NA 2962 2877 2754 2850
# 1014   NA   NA   NA   NA  510 2962  474
# 1015   NA   NA   NA   NA   NA 2914  253
# 3367   NA   NA   NA   NA   NA   NA 2880
# 1335   NA   NA   NA   NA   NA   NA   NA
#
# [[2]]
#      1007 1012 1013 1014 1015 3367 1335
# 1007   NA    2   16    2    1   16    1
# 1012 1453   NA   17    1    3   14    2
# 1013  892  914   NA   15   18   27   18
# 1014 1247 1282  800   NA    4   16    1
# 1015 1452 1499  921 1289   NA   14    0
# 3367  878  895 1720  802  902   NA   13
# 1335 1386 1427  876 1247 1434  860   NA

d6[[2]]/t(d6[[2]]) # div by transpose => rate in upper triangle

#          1007           1012        1013            1014          1015        3367         1335
# 1007       NA    0.001376462  0.01793722    0.0016038492  0.0006887052  0.01822323 0.0007215007
# 1012  726.500             NA  0.01859956    0.0007800312  0.0020013342  0.01564246 0.0014015417
# 1013   55.750   53.764705882          NA    0.0187500000  0.0195439739  0.01569767 0.0205479452
# 1014  623.500 1282.000000000 53.33333333              NA  0.0031031808  0.01995012 0.0008019246
# 1015 1452.000  499.666666667 51.16666667  322.2500000000            NA  0.01552106 0.0000000000
# 3367   54.875   63.928571429 63.70370370   50.1250000000 64.4285714286          NA 0.0151162791
# 1335 1386.000  713.500000000 48.66666667 1247.0000000000           Inf 66.15384615           NA
```

The above analysis implicitly assumes that the reference is among the nucleotides with a significant read count. How accurate is this? E.g., Is it ever missing? How often does the reference nucleotide not rank 1st or second in its read count?

```
rbind(
  max         =unlist(lapply(seltab,function(x){max(x[,'.match'])})),
  median      =unlist(lapply(seltab,function(x){median(x[,'.match'])})),
  min         =unlist(lapply(seltab,function(x){min(x[,'.match'])})),
  'under 10' =unlist(lapply(seltab,function(x){sum(x[,'.match']<10)})),
  'under 5'  =unlist(lapply(seltab,function(x){sum(x[,'.match']<5)})),
  'under 3'  =unlist(lapply(seltab,function(x){sum(x[,'.match']<3)})),
  'absent'   =unlist(lapply(seltab,function(x){sum(x[,'.match']==0)})),
  'under 10%'=unlist(lapply(seltab,function(x){sum(x[,'.match']/x[,'Cov']<.10)})),
  'under 5%' =unlist(lapply(seltab,function(x){sum(x[,'.match']/x[,'Cov']<.05)}))
)

#           1007 1012 1013 1014 1015 3367 1335
# max         95  150  149  103  146  150  150
# median      32   58   35   23   54   35   85
# min          1    0    0    2    1    0    1
# under 10    45    3  812  171    5 1006    3
# under 5      4    3  728   17    3  921    1
# under 3      1    3  580    2    2  702    1
# absent       0    1  171    0    0  194    0
# under 10%    2    3  698    2    3  894    2
# under 5%     1    3  520    0    3  638    1
```

In short, among the 5213 selected positions, the reference nucleotide is nearly always seen, but is seen only in a low proportion of reads (say, $< 10\%$) at about 500 positions in Italy/Wales.

Turning to rank, the following code chunk extracts the 6 read counts for each selected position and calculates their ranks.

```
rankem <- function(tab){
  count.mat <- as.matrix(tab[,c('a','g','c','t','.match', 'Cov')])
  return(t(apply(count.mat,1,rank)))
}
selrank <- lapply(seltab,rankem)
```

Total coverage is necessarily the largest value, possibly tied, so it will have rank 6 (not tied) or 5.5 (tied with some other position, which also gets rank 5.5—ranks are averaged in case of ties). Rank of .match is the interesting quantity. As shown in the table below, the most common case is that its rank is 5.5—i.e., all reads matched reference and it is tied with Cov. The second most common case is when there are some nonref reads, but not as many as match the reference; .match will be second largest in this case, with rank 5. If ref and a single nonref are tied for max, both get rank 4.5; this happened in 49–300 positions. .match gets rank 4 if it is the second largest read count, (exceeded by some nonref nuc and of course by total coverage). (A 3-way tie for max would also give rank 4, but seems unlikely; I didn't check for this case.) The remaining cases, where 2 or more nonref nucs have higher counts than the reference nuc, happen at only 1 of the 5213 positions in the big 5 and less than 50 positions in Italy or Wales.

```
for(i in 1:7){cat(names(selrank)[i], ':',  sort(unique(selrank[[i]][,'.match']),decreasing=T),'\n')}

# 1007 : 5.5 5 4.5 4
# 1012 : 5.5 5 4.5 4 2.5
# 1013 : 5.5 5 4.5 4 3.5 3 2.5 2
# 1014 : 5.5 5 4.5 4
# 1015 : 5.5 5 4.5 4
# 3367 : 5.5 5 4.5 4 3.5 3 2.5 2
# 1335 : 5.5 5 4.5 4

howmanyeq <- function(th){unlist(lapply(selrank,function(x){sum(x[,'.match'] == th)}))}
howmanylt <- function(th){unlist(lapply(selrank,function(x){sum(x[,'.match'] <  th)}))}
smary <- NULL
smary <- rbind(smary,'ref + Cov tied; no nonref reads'=howmanyeq(5.5))
smary <- rbind(smary,'ref is max, but some nonrefs'   =howmanyeq(5  ))
smary <- rbind(smary,'ref + some nonref tied for max' =howmanyeq(4.5))
smary <- rbind(smary,'ref is second highest'          =howmanyeq(4  ))
smary <- rbind(smary,'ref is tied for 2nd'            =howmanyeq(3.5))
smary <- rbind(smary,'ref is third highest'           =howmanyeq(3  ))
smary <- rbind(smary,'ref is tied for 3rd'            =howmanyeq(2.5))
smary <- rbind(smary,'aggregate of previous 3 cases'  =howmanylt(4  ))
smary

#                                  1007 1012 1013 1014 1015 3367 1335
# ref + Cov tied; no nonref reads 3433 3373 1908 3362 3357 1949 3391
# ref is max, but some nonrefs    1324 1426 2068 1498 1388 1852 1533
# ref + some nonref tied for max    59   37   54   24   45   43   30
# ref is second highest            397  376  998  329  423 1163  259
# ref is tied for 2nd                0    0    7    0    0    5    0
# ref is third highest               0    0    7    0    0    7    0
# ref is tied for 3rd                0    1  168    0    0  192    0
# aggregate of previous 3 cases      0    1  185    0    0  206    0
```

Finally, just for more clarity on what the filtering is doing, here is a complex version of one of the pairs plots (NOT intended for the paper, but for us), showing a comparison between the unfiltered data (filt1 above) versus the most aggressive filter tried above (filt6: delete single and double reads, and use max nonref over that plus ref as nonref fraction). This is Italy vs Wales.

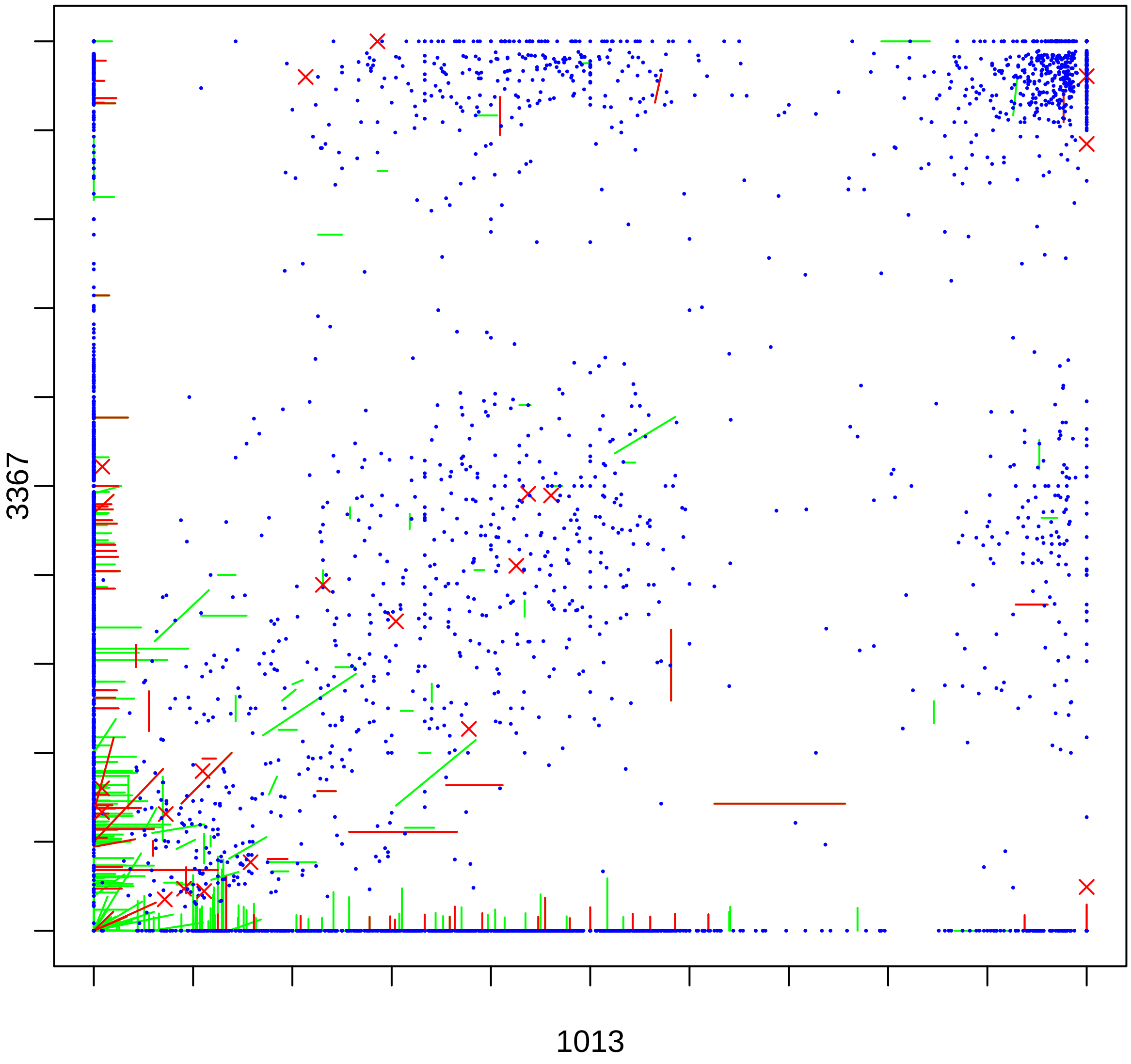
- Blue dots: as in the earlier plots, but these are the subset of points that do not move perceptibly (not more than 0.01 Euclidean distance), and are "consistent," i.e., have their max nonref reads on the same nonref nuc in both strains; 8040 of these.
- Red x's: didn't move, but are inconsistent; 49 of these. Most of them are near the axes where they are hard to see in the clutter.
- Green lines: connect consistent points before/after filtering; about 1400 of these.
- Red lines: points that moved, and were inconsistent; 458 of these.

- Black dotted lines overlaid on red or green lines: a handful of the biggest movers; details are given in the printout below.

The “post-filtering” end on each line is the one closer to the origin.

My summary: Not much changes; most of what does change, changes near the axes—tiny nonref counts pushed to zero by the filtering. These were probable read errors, but don’t matter much either way. There are some big moves away from the axes, and they are individually interesting, but I don’t think they are frequent enough to change our story. Example interesting vignette: Chr1 2632162 is plausibly tri-allelic in these 2 strains, with 6 and 7 reads on the two nonreference nucs. This generates a big move under julie.filter2 since the 6 discarded reads represent almost 20% of coverage, *and* by chance the 6/7 counts flip position, making it inconsistent.

```r
alt.pairs <- function(x=3,y=6,loose=filt1,tight=filt6,nnl=nrf.nucs1,nnt=nrf.nucs6,
                      stab=seltab){
  #x = 3 : wales
  #y = 6 : italy
  samp <- loose$sample # use same sample in all
  lnrfall <- loose$nrfall[samp,]
  tnrfall <- tight$nrfall[samp,]
  stab <- lapply(stab,function(x){x[samp,]})
  nnlx <- nnl[[x]][samp]
  nnly <- nnl[[y]][samp]
  nntx <- nnt[[x]][samp]
  nnty <- nnt[[y]][samp]
  epsilon <- 1e-2
  del <- sqrt( (lnrfall[,x] - tnrfall[,x])^2 + (lnrfall[,y] - tnrfall[,y])^2 )
  unmoved <- del < epsilon
  moved <- !unmoved
  cat('unmoved:', sum(unmoved), ' moved:', sum(moved), '\n')

  # Flag a few of the biggest movers
  bigmoves <- which(del>.15)
  cat('\nSome big movers:\n')
  print(bigmoves)
  cat('\nDetails:')
  bigwhere <- as.integer(sub('Chr1:','',rownames(lnrfall)[bigmoves]))
  for(i in bigwhere){
    cat('\n')
    print(seecounts(i,snp.tables=snp.tables))
  }

  # flag inconsistent nonrefs
  nn <- cbind(nnlx, nntx, nnly, nnty)
  # inconsistent if row max > row min, ignoring zeros
  nnmax <- apply(nn,1,max)
  nnmin <- ifelse(nn==0,5,nn) # don't let zero hide a nonzero in min
  nnmin <- apply(nnmin,1,min)
  nnmin <- ifelse(nnmin==5,0,nnmin) # put back zeros (all-zero row will have max=min=0)
  inconsistent <- (nnmax != nnmin)
  cat('Inconsistent:', sum(inconsistent),
      'moved/un:', sum(inconsistent & moved), sum(inconsistent & unmoved), '\n')

  library(compactr)
  # empty plot
  eplot(ylab=colnames(lnrfall)[y], xlab=colnames(lnrfall)[x],
        xlim=0:1, xat=(0:10)/10, xticklab=(0:10)/10,
        ylim=0:1, yat=(0:10)/10, yticklab=(0:10)/10 )

  # goal: for points that move after filtering, connect old/new position with
  # a line.  given vectors of x & y coords, 'lines' function will connect all
  # in order but stops when it hits NA, so w matrix below interleaves
  # old x/y, new x/y, NA/NA repeatedly.
  x1 <- lnrfall[moved,x]
  y1 <- lnrfall[moved,y]
  x2 <- tnrfall[moved,x]
  y2 <- tnrfall[moved,y]
  z  <- rep(NA,sum(moved))
  w  <- matrix(as.vector(t(cbind(x1,y1,x2,y2,z,z))),ncol=2,byrow=T)
  lines(w,col='green')

  # redraw the inconsistent ones in red
  redl <- inconsistent[moved]
  reds <- as.vector(t(cbind(redl,redl,redl)))
  lines(w[reds,],col='red',lwd=1)

  # highlight the few big movers in black
  for(i in bigmoves){
    lines(c(lnrfall[i,x],tnrfall[i,x]),c(lnrfall[i,y], tnrfall[i,y]),
          col='black',lwd=2,lty='dotted')
  }
  # below marked end points of old/new segments, but on reflection, the new
  # (tighter filtering) coords are always <= the old ones
  #points(lnrfall[  moved,x], lnrfall[  moved,y], pch=3, cex=.4, col='orange')
  #points(tnrfall[  moved,x], tnrfall[  moved,y], pch=20,cex=.4, col='yellow')

  # now draw unmoved points, red x if inconsistent
  pts.col <- (ifelse(inconsistent,'red','blue'))[unmoved]
  pts.pch <- (ifelse(inconsistent,4,20))[unmoved]
  pts.cex <- (ifelse(inconsistent,1,.2))[unmoved]
  points(lnrfall[unmoved,x], lnrfall[unmoved,y], pch=pts.pch, cex=pts.cex, col=pts.col)
}
alt.pairs()

# unmoved: 4788  moved: 425
#
# Some big movers:
# named integer(0)
#
# Details:Inconsistent: 98 moved/un: 78 20
```



Finally, what should we show in the main text figure? Not this, it's too complicated. Showing it either before filtering (the original plots) or after preserves the interesting 4-lobed structure, so that's not an issue. The main issue is what is clearest/simplest to explain. I currently favor julie.filter1=0 and julie.filter2=TRUE, but flag inconsistent points, say, in red. Explanation is roughly we kept all data, but only used the most frequent nonref nucleotide, flagging cases where that is different between the two strains; it will be obvious by eye that most points are consistent, and the rare inconsistent ones cluster near the axes.

9.3 Other measures of discordance

I think the analysis in the previous subsection is the most straightforward way to address the question, but I tried some other stuff earlier that gives other ways of viewing it. I will leave it here for posterity, but just skim it or skip it.

How many selected positions are called SNPs in at least one strain?

```
the.snps <- (snp.tables[[1]]$snp==1)
for(i in 2:7){
  the.snps <- the.snps | (snp.tables[[i]]$snp==1)
}
n <- sum(the.snps[selected])
cat(n, 'of', n.selected, '=', n/n.selected*100, '%\n')

# 3820 of 5213 = 73.27834 %
```

Are they “consistent SNPs,” as in shared-SNPs analysis?

Uh oh. As of 10/2015, “../00common/mycache/consistent.rda” no longer exists, so following code breaks. I think this has been subsumed by one of the several files named:

```
paste('../00common/mycache/filtered.snps', which.snp.tables(), 'rda', sep='.')
```

but not totally sure and I think the structure has changed, too. E.g., I think the embedded var name “consistent” should now be `filtered.snps$Data$consistent.` *or something. Maybe I’ve fixed this below, maybe not; but not on the critical path tonight...*

```
#old:
#load('../00common/mycache/consistent.rda')
#str(consistent)
#sum(consistent[[2]])
#conswhere <- names(consistent[[2]][consistent[[2]]])
#new:
load('../00common/mycache/filtered.snps.Chr1-qfiltered.rda')
str(filtered.snps$Data$consistent.snps)

# List of 4
#  $ : Named logi [1:47499] TRUE FALSE TRUE TRUE TRUE TRUE ...
#   ..- attr(*, "names")= chr [1:47499] "Chr1:333" "Chr1:417" "Chr1:435" "Chr1:438" ...
#  $ : Named logi [1:47499] TRUE TRUE TRUE TRUE TRUE TRUE ...
#   ..- attr(*, "names")= chr [1:47499] "Chr1:333" "Chr1:417" "Chr1:435" "Chr1:438" ...
#  $ : Named logi [1:47499] TRUE TRUE TRUE TRUE TRUE TRUE ...
#   ..- attr(*, "names")= chr [1:47499] "Chr1:333" "Chr1:417" "Chr1:435" "Chr1:438" ...
#  $ : Named logi [1:47499] TRUE TRUE TRUE TRUE TRUE TRUE ...
#   ..- attr(*, "names")= chr [1:47499] "Chr1:333" "Chr1:417" "Chr1:435" "Chr1:438" ...

consistent <- filtered.snps$Data$consistent.snps
sum(consistent[[2]])

# [1] 47108

conswhere <- names(consistent[[2]][consistent[[2]]])

conswhere[1:10]

#  [1] "Chr1:333" "Chr1:417" "Chr1:435" "Chr1:438" "Chr1:465" "Chr1:560" "Chr1:567" "Chr1:723"
#  [9] "Chr1:735" "Chr1:858"

conswherei <- as.integer(substr(conswhere,6,99))
str(conswherei)

#  int [1:47108] 333 417 435 438 465 560 567 723 735 858 ...

seecounts(conswherei[sample.int(length(conswherei),10)],snp.tables=snp.tables) # eyeball a few

#     chr     pos Ref Strain   A   G  C   T SNP  exon indel nrf rat
# 1  Chr1  833617   C
# 2                     1007   0   0 25  0   0 FALSE FALSE
# 3                     1012   0   0 51  0   0 FALSE FALSE
# 4                     1013   0   0 28 16   1 FALSE FALSE
# 5                     1014   0   0 13  0   0 FALSE FALSE
# 6                     1015   0   0 50  0   0 FALSE FALSE
# 7                     3367   0   0 22 25   1 FALSE FALSE
# 8                     1335   0   0 95  0   0 FALSE FALSE
# 9  Chr1 1150017   A
# 10                    1007  19   0  0  8   1  TRUE FALSE
# 11                    1012  30   0  0 17   1  TRUE FALSE
# 12                    1013  65   0  0  0   0  TRUE FALSE
# 13                    1014  10   0  0  3   1  TRUE FALSE
# 14                    1015  30   0  0 21   1  TRUE FALSE
# 15                    3367  51   0  0  0   0  TRUE FALSE
# 16                    1335  49   0  0 33   1  TRUE FALSE
# 17 Chr1 1812033   A
# 18                    1007  21  23  0  0   1  TRUE FALSE
# 19                    1012  35  32  0  0   1  TRUE FALSE
# 20                    1013  22  26  0  0   1  TRUE FALSE
# 21                    1014  19   8  0  0    1  TRUE FALSE
# 22                    1015  18  48  0  0    1  TRUE FALSE
# 23                    3367  26   0  0  0    0  TRUE FALSE
# 24                    1335  10   3  0  0    1  TRUE FALSE
# 25 Chr1 2799854   A
# 26                    1007   9   0  0  0    0 FALSE FALSE
# 27                    1012  35   0  0  0    0 FALSE FALSE
# 28                    1013  23   0 15  0    1 FALSE FALSE
# 29                    1014   8   0  0  0    0 FALSE FALSE
# 30                    1015  32   0  0  0    0 FALSE FALSE
# 31                    3367   0   0 23  0    1 FALSE FALSE
# 32                    1335 119   0  0  0    0 FALSE FALSE
# 33 Chr1  631862   C
# 34                    1007   0   0 13  0    0 FALSE FALSE
# 35                    1012   0   0 20  0    0 FALSE FALSE
# 36                    1013   0   0 12  0    0 FALSE FALSE
# 37                    1014   0   0  6  0    0 FALSE FALSE
# 38                    1015   0   0  5 10    1 FALSE FALSE
# 39                    3367   0   0 24  0    0 FALSE FALSE
# 40                    1335   0   0 27  0    0 FALSE FALSE
# 41 Chr1 2774888   G
# 42                    1007   0  21  0  0    0  TRUE FALSE
# 43                    1012   0  44  0  0    0  TRUE FALSE
# 44                    1013   0   4 13  0    1  TRUE FALSE
# 45                    1014   0   7  0  0    0  TRUE FALSE
# 46                    1015   0  53  0  0    0  TRUE FALSE
# 47                    3367   0  44  0  0    0  TRUE FALSE
# 48                    1335   0 111  0  0    0  TRUE FALSE
# 49 Chr1 2902959   C
# 50                    1007   0   0 34  0    0 FALSE FALSE
# 51                    1012   0   0 73  0    0 FALSE FALSE
# 52                    1013   0   0 51 22    1 FALSE FALSE
# 53                    1014   0   0 24  0    0 FALSE FALSE
# 54                    1015   0   0 63  0    0 FALSE FALSE
# 55                    3367   0   0 34 19    1 FALSE FALSE
# 56                    1335   0   0 90  0    0 FALSE FALSE
# 57 Chr1 2067941   G
# 58                    1007   0  21 10  0    1  TRUE FALSE
# 59                    1012   0  15 23  0    1  TRUE FALSE
# 60                    1013   0  32  0  0    0  TRUE FALSE
# 61                    1014   0   4  1  0    0  TRUE FALSE
# 62                    1015   0  21 21  0    1  TRUE FALSE
# 63                    3367   0  40  0  0    0  TRUE FALSE
# 64                    1335   0  31 34  0    1  TRUE FALSE
# 65 Chr1 1970279   G
# 66                    1007   0  15  0  2    0 FALSE FALSE
# 67                    1012   0  12  0  5    1 FALSE FALSE
# 68                    1013   0  16  1  1    0 FALSE FALSE
# 69                    1014   0   3  0  0    0 FALSE FALSE
# 70                    1015   0  18  0  3    0 FALSE FALSE
# 71                    3367   0  13  0  0    0 FALSE FALSE
# 72                    1335   0  16  0  2    0 FALSE FALSE
# 73 Chr1  181945   C
# 74                    1007   0   0 31  0    0 FALSE FALSE
# 75                    1012   0   0 55  0    0 FALSE FALSE
# 76                    1013  15   0 19  0    1 FALSE FALSE
# 77                    1014   0   0 26  0    0 FALSE FALSE
# 78                    1015   0   0 51  0    0 FALSE FALSE
# 79                    3367   0   0 32  0    0 FALSE FALSE
# 80                    1335   0   0 56  0    0 FALSE FALSE

longcons <- vector('logical',length(the.snps))
longcons[conswherei] <- T
cat('selected:', n.selected, 'consistent:',length(conswherei), 'both:', sum(longcons & longsel),
    '=', sum(longcons & longsel)/n.selected*100, '%\n')

# selected: 5213 consistent: 47108 both: 3794 = 72.77959 %
```

I.e., nearly all positions selected for the pairs plots that are called SNPs somewhere, are classified as “consistent” in the shared-SNPs analysis. How many selected positions have 0,1,2,3 nonzero nonref counts, per strain:

```
see.nrf.counts <- function(sel=selected, lo=0, snp.tables=snp.tables.01.26.14){
  n.sel <- length(sel)
  counts <- NULL
  aggreg8 <- matrix(0,n.sel,6)
  colnames(aggreg8) <- c('Cov','a','g','c','t','.match')
  for(st in 1:7){
    # extract the selected subset of positions in one strain
    sel.df <- snp.tables[[st]][sel,]
    # count nonzero nonref nucleotides (more generally, if lo > 0, count those > lo)
    sel.nz <- (sel.df$a > lo) + (sel.df$g > lo) + (sel.df$c > lo) + (sel.df$t > lo)
    #print(summary(sel.nz))
    # use "histogram" to count them
    counts <- rbind(counts, hist(sel.nz,breaks=-1:3,plot=F)$counts)
    # also aggregate counts across all 7
    aggreg8[,'Cov']    <- aggreg8[,'Cov']    + sel.df[,'Cov']
    aggreg8[,'.match'] <- aggreg8[,'.match'] + sel.df[,'.match']
    for(nuc in c('a','g','c','t')){
      aggreg8[,nuc] <- aggreg8[,nuc] + ifelse(sel.df[,nuc] <= lo, 0, sel.df[,nuc])
    }
  }
  # same counting for the aggregate (if >lo individually, then >lo in aggregate, so >0 test ok here)
  agg.nz <- (aggreg8[,'a'] > 0) + (aggreg8[,'g'] > 0) + (aggreg8[,'c'] > 0) + (aggreg8[,'t'] > 0)
  counts <- rbind(counts, hist(agg.nz,breaks=-1:3,plot=F)$counts)
  rownames(counts) <- c(names(snp.tables), 'aggregate')
  colnames(counts) <- c('zero', 'one', 'two', 'three')
  return(counts)
}
see.nrf.counts(lo=0, snp.tables=snp.tables)

#           zero  one two three
# 1007      3433 1740  40     0
# 1012      3373 1799  40     1
# 1013      1908 3179 123     3
# 1014      3362 1771  73     7
# 1015      3357 1807  49     0
# 3367      1949 3163 101     0
# 1335      3391 1764  57     1
# aggregate    0 4525 655    33
```

In short, of the 5213 selected positions, in all strains, very roughly half of these positions have *no* nonreference reads, most of the rest have nonreference reads for a single nucleotide, 2–10% percent have nonzero read counts for two different nonref nucleotides and a fraction of a per cent show reads for all three. These numbers shift sharply when aggregating read counts across all 7 strains. *However*, many of these totals seem to be largely driven by very low read counts; deleting singletons (1 read in a given nonref nuc in any strain) reduces the 3616 positions having three nonref nucleotides to a mere 87, and deleting doubletons further reduces it to 18:

```
see.nrf.counts(lo=1, snp.tables=snp.tables)

#           zero  one two three
# 1007      3578 1629   6     0
# 1012      3511 1692  10     0
# 1013      2055 3123  33     2
# 1014      3575 1614  23     1
# 1015      3489 1711  13     0
# 3367      2050 3133  30     0
# 1335      3558 1639  16     0
# aggregate    2 5062 142     7

see.nrf.counts(lo=2, snp.tables=snp.tables)

#           zero  one two three
# 1007      3675 1535   3     0
# 1012      3574 1633   6     0
# 1013      2111 3075  26     1
# 1014      3744 1462   6     1
# 1015      3550 1659   4     0
# 3367      2089 3102  22     0
# 1335      3623 1580  10     0
# aggregate   22 5088 100     3
```

9.4 Discordance in UN-selected positions

Out of curiousity, how many positions in Italy have all nonzero counts on all three nonref nucs, across all of Chr1, not just the set selected for the pairwise comparisons?

```
# look for 3 nonref cases, all of Chr1
italy.nz <- (snp.tables[[6]]$a > 0) + (snp.tables[[6]]$g > 0) +
            (snp.tables[[6]]$c > 0) + (snp.tables[[6]]$t > 0)
italy.3  <- sum(italy.nz == 3)
italy.which3 <- which(italy.nz == 3)
cat('Italy has', italy.3, 'positions with nonzero read counts for all 3 nonref nucs.\n')
```

```
# Italy has 17 positions with nonzero read counts for all 3 nonref nucs.

# coverage summary
summary(snp.tables[[6]]$Cov[italy.nz == 3])

#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#     5.0    23.0    76.0   103.8   139.0   363.0

# nonref count summary
summary(snp.tables[[6]]$Cov[italy.nz == 3] - snp.tables[[6]]$.match[italy.nz == 3])

#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#    3.00    5.00   11.00   19.41   22.00   99.00

# first few examples
snp.tables[[6]][italy.which3[1:9],]

#         snp  Chr     Pos Ref Cov  a g  c  t n .match  exon indel  chr     pos rawCov
# 258626    1 Chr1  258626   T  28  1 1 13  0 0     13  TRUE FALSE Chr1  258626     51
# 346940    0 Chr1  346940   C 175  1 1  0  1 0    172 FALSE FALSE Chr1  346940    229
# 700466    0 Chr1  700466   G  33 10 0  1  1 0     21 FALSE FALSE Chr1  700466     47
# 718157    0 Chr1  718157   G  23  2 0  1  1 0     19  TRUE FALSE Chr1  718157     35
# 1116985   0 Chr1 1116985   C 133 24 5  0  5 0     99  TRUE FALSE Chr1 1116985    187
# 1117075   1 Chr1 1117075   G 363 72 0 14 13 0    264  TRUE FALSE Chr1 1117075    421
# 1117218   0 Chr1 1117218   A  99  0 1  2  8 0     88  TRUE FALSE Chr1 1117218    140
# 1117276   0 Chr1 1117276   G  76  8 0  1  2 0     65  TRUE FALSE Chr1 1117276     87
# 1224654   0 Chr1 1224654   T   8  2 3  1  0 0      2  TRUE FALSE Chr1 1224654     18
```

Answer: 17. Their overall coverage is unremarkable (except perhaps the extreme outliers), but the fact that the total nonref count is ≤ 7 in three quarters of the cases (implying a max count of no more than 5 in these cases) and the mean is < 9 suggests that these are dominated by low-count sporadic read errors rather than biology. However, the second example above (Chr1 9310) looks like a plausible tri-allelic (but not quad-) position.

Section 9 Summary: Based on the criteria outlined in subsection 7.1, the nonreference frequencies at positions selected for display in the pairwise scatter plots definitely include counts for discordant nucleotides, e.g., a read for a nonreference "A" at some position in one strain where a read for a "G" is seen in a different strain. But in the vast majority of cases, discordant counts are small, and the big picture does not change if we do some further filtering to remove them.

10 Tangents

1: I wondered what the analogous plots would look like for the big desert region. Answer is more or less as expected: based on these 5686 points, the 5 NE have a thin, somewhat correlated, scatter of points above freq .2, and a cloud below .2. There is no clear signal to suggest whether this cloud is low-freq alleles vs read errors. In this region, Italy and Wales have many apparent SNPs absent from the 5 NE, showing a pattern of sharing similar to the (largely) nondesert in the main fig 1b above.

```
pi <- order(des[[1]][[1]][,3],decreasing=T)
big <- pi[1]

nrf.6plus1(snp.tables=snp.tables,
           mask=seg.mask(des[[1]][[1]][big,1],des[[1]][[1]][big,2],
                         snp.tables=snp.tables))

# non-null mask 320006 positions.
# nrf.6plus1: From a region of length: 320006 we identified all positions satisfying:
#  21 <= coverage <= 150 in *all* 7 isolates,
# and having 0.1 <= nr.frac <= 1.0 in *at least 1* of them.
# From these 409 positions, we sampled 10000 to plot.
```

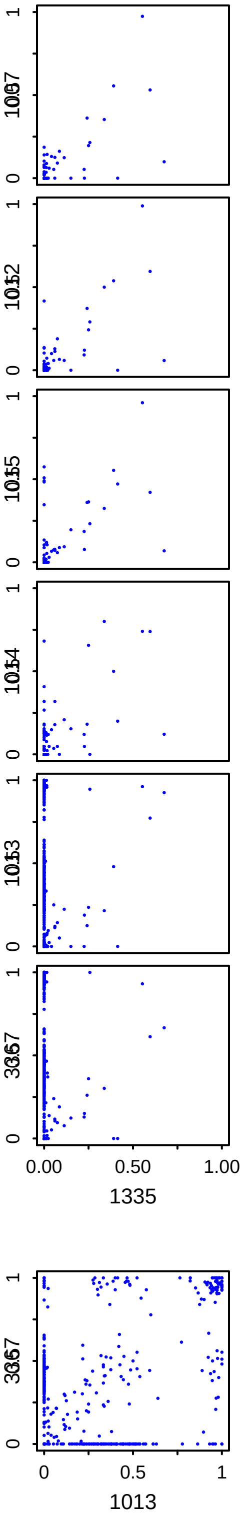


Figure 1: Comparative nonref allele proportions in Chr1 big desert.

11 To Do/Improvements?

On 2015/11/7 & 8 I did a bit more exporlatory stuff to see whether we could clarify the Gyre data. My conclusion is — not without a lot of work. It looks better in the un-q-filtered data, but of course then IT/Wales have peak at 0.8 rather than 1. With current q-filtering, the later problem is fixed, but mean coverage in Gyre drops sharply, to about 12:

```
summary(snp.tables[[4]]$Cov) # weak coverage in Gyre

#    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#    0.00    7.00   11.00   12.43   16.00  212.00

unlist(lapply(snp.tables, function(x){median(x$Cov)}))

# 1007 1012 1013 1014 1015 3367 1335
#   27   47   40   11   45   43   79
```

With the current filtering, based on min/max count limits applied uniformly across all strains, presently [10, 120] for the scatter-smooths plots, the "10" end means that we loose a lot of Gyre while scooping up noise in the others, while the 120 end gets collapsed repeats etc in gyre. I tried lowering the 10 to 9, 7, even 5 (thinking this might get more good data in Gyre), and tried raising it 15 or 20 (thinking it would get less data, but perhap the het positions would stand out better). But it didn't help; all figures looked pretty much the same. Setting julie.filter1=2 *did* remove big spikes near zero (plausibly 1 or 2 isolated read errors at many positions, still not removed by q-filters), and created a very broad hump (roughly 0.2-0.8), but this seems a bit arbitrary; not fruitful to show it in the paper. With the 7x difference in median coverage across strains, I think a better way to choose the positions included in the scatters would be to find the set of positions that are, say, $\mu \pm \sigma$ simultaneously in all 7, using per-strain values for μ, σ. I don't expect the plots would change qualitatively, but I think this would cover more positions without pulling in unusually noisy ones, maybe giving a somewhat cleaner picture. I have *not* tried this yet.

I also ran it once on full genome data. As expected, count of eligible position increased by about 10x, and histos are a bit smoother, but otherwise not an increment worth the effort. E.g., there are more outlier dots which just add clutter.

SVN ID I miss you Id: Fig1-mscat.rnw 2017-07-21 or later ruzzo