Contents

Appendix A

R code

All files referenced in the current appendix are available in the R/ directory at:

https://github.com/tystan/thesis/.

A.1 Morphological operators

Description	File	Functions
Naive erosion and top-hat:		
	/00_erosion_slow.R	erode(), dilate(), tophat()
Line segment erosion:		
	/01_erosion_quick.R	erode_quick()
Naive erosion for unequally spaced values:		
	/02_cts_erosion_slow.R	erode_cts_slow()
Continuous line segment erosion:		
	/03_cts_erosion_quick.R	erode_cts_quick()

A.1.1 Naive erosion and top-hat

Below is a simple (and naive) implementation of an erosion, dilation and top-hat operator for $x \in \{1, 2, \ldots, n\} = X$ and $f(x) \in \mathbb{R} \ \forall x \in X$. Because of the assumed even spacing of the elements of X, the R-function below simply requires the vector of intensities, f, and the size of the SE.

Please note the code checks the SE size provided is an odd integer because symmetric SE is not possible with an even SE size. The maximum and minimum statements on line 11 of the code segment below, namely `max(1,i-k0)` and `min(nx,i+k0)`, check for when the SE sits over the 'edge' on the left or right of the series, respectively. This ensures only defined f values will be used.

```
1  ### f=f are evenly spaced intensity values
2  ### k is the SE length (size of B)
3  erode<-function(f,k) #f are the intensities
4  {
5     if(!(k %% 2)) return(NULL) #SE must be of odd length
6     nx<-length(f)
7     erode<-rep(0,nx)
8     # k0 is window coverage to the left and right of centre
9     k0<-(k-1)/2
10    for(i in 1:nx) #for each m/z point across the spectrum
11       erode[i]<-min(f[max(1,i-k0):min(nx,i+k0)])
12    return(erode)
13 }
14 ### dilate is the same as erode except use max instead of min, or:
15 dilate<-function(f,k) return(-erode(-f,k))
16 ### as defined τ_B(f) = f − (f ⊖ B) ⊕ B
17 tophat<-function(f,k) return(f-dilate(erode(f,k),k))
```

A.1.2 Line segment erosion

```
### f=f are evenly spaced intensity values
### k is the SE length (size of B)
erode_quick<-function(f,k){
  nx<-length(f)
  t1<-proc.time()[3] ### get start time
  if(k>=nx){
    cat("Warning: structuring element is >= in length as the input\n")
    cat("The input vector has been output \n")
    return(f)
  }else{
    if((k%%2) != 1){
      k<-k-1
      cat("Structring Element not symmetric, using SE length -1 =",k,"\n")
    }
    # k0 is window coverage to the left and right of centre
    k0<-(k-1)/2
    # check whether series is a length that is a multiple of k
    # if not, add points to series for algorithm then remove at end
    add.pix<-k-(nx%%k)
    isMultiple<-(add.pix==k)
    if(!isMultiple){
      f<-c(f,rep(+Inf,add.pix))
      rem.indxs<-(nx+1):(nx+add.pix)
      nx<-nx+add.pix
    }
    # intialise g,h
    g<-rep(0,nx); h<-rep(0,nx);
    r_min<-rep(0,nx)
    j<-nx
    # compute g,h values
    for(i in 1:nx){
      g[i]<-ifelse(i%%k==1,f[i],min(g[i-1],f[i]))
      h[j]<-ifelse(j%%k==0,f[j],min(h[j+1],f[j]))
      j<-j-1
    }
    # only g values are required at the left
    r_min[1:(k0+1)]<-g[(k0+1):k]
    # vectorised min calculations
    r_min[(k0+2):(nx-k0-1)]<-pmin(g[(k+1):(nx-1)],h[2:(nx-k)])
    # only h values are required at the left
    r_min[(nx-k0):nx]<-h[(nx-k+1):(nx-k0)]
    if(!isMultiple) r_min<-r_min[-rem.indxs]
    delta.t<-sprintf("%.2f",proc.time()[3]-t1) ### time elapsed
    cat("Completed morphological erosion in",delta.t,"seconds\n")
    return(r_min)
  }
}
```

A.1.3 Naive erosion for unequally spaced values

The function `get_lo_bounds()` creates a vector, `LO`, of all the lower bounds indexes such that `LO[i]` $= \arg\min_j x_j \geq x_i - k/2$ for each x_i, $i = 1, 2, \ldots, n$. This function implements an $O(n)$ algorithm using two pointers that move along the input vector X from left to right. One pointer is the current position, the other is a lagging pointer that moves along the vector when required. To find the upper bounds, the same algorithm would is employed but the pointers start from the right and move down the vector with the second point lagging to the right.

```
### get_lo_bounds() gets the index values of x[i]-k/2 for x[i], i = 1,2,...,n_x.
### returns a vector of indexes length n_x
### used by erode_cts_slow()
get_lo_bounds<-function(x,k)
{
  nx <- length(x)
  k0 <- k/2
  LO <- rep(0,nx)
  i <- 2
  i_lo <- 1
  LO[i_lo] <- i_lo # x[1] has lower bound 1
  while(i<=nx)
  {
    if((x[i]-x[i_lo])<=k0){
      LO[i] <- i_lo
      i <- i+1
    }else{
      i_lo <- i_lo+1
    }
  }
  return(LO)
}
### same as get_lo_bounds() but for x[i]+k/2
get_hi_bounds<-function(x,k)
{
  nx <- length(x)
  k0 <- k/2
  HI <- rep(0,nx)
  i <- nx-1
  i_hi <- nx
  HI[i_hi] <- i_hi # x[nx] has upper bound nx
  while(i>0)
  {
    if((x[i_hi]-x[i])<=k0){
      HI[i] <- i_hi
      i <- i-1
    }else{
      i_hi <- i_hi-1
    }
```

```
40    }
41    return(HI)
42  }
43  ### x=X are unevenly (or evenly) spaced locations of the intensities
44  ### f=f are the corresponding intensity values
45  ### k is the SE length (size of B)
46  erode_cts_slow<-function(x,f,k)
47  {
48    nx<-length(x)
49    r_min<-rep(0,nx)
50    # slow way to get LO and HI, removed to make comparison with erode_cts_quick() fair
51    #LO<-get_lo_bounds(x,k)
52    #HI<-get_hi_bounds(x,k)
53    k0<-k/2
54    # fast way to determine upper and lower indexes in R to avoid looping
55    LO<-nx-rev(findInterval(rev(-x),rev(-(x+k0))))+1
56    HI<-findInterval(x+k0,x)
57    for(i in 1:nx) r_min[i]<-min(f[LO[i]:HI[i]])
58    return(r_min)
59  }
```

A.1.4 Continuous line segment erosion

```
### x=X are unevenly (or evenly) spaced locations of the intensities
### f=f are the corresponding intensity values
### k is the SE length (size of B)
erode_cts_quick<-function(x,f,k){
  nx<-length(x)
  x_span<-x[nx]-x[1]
  t1<-proc.time()[3]
  isAppend<-FALSE
  if(k>=x_span){
    cat("Warning: structuring element spans the entire input set \n")
    cat("The input f vector has been output \n")
    return(f)
  }else{
    # check whether series is a length that is a multiple of k
    # if not, add a single point (x_1 + mk,∞) to series for algorithm,
    # then remove point at end
    m<-ceiling(x_span/k)
    mk<-m*k
    if(!((x[1]+mk) == x[nx])){
      x<-c(x,x[1]+mk)
      f<-c(f,+Inf)
      isAppend<-TRUE
      nx<-nx+1
      x_span<-x[nx]-x[1]
    }
    # create Θ
    k_blocks<-c(0,findInterval(x,seq(x[1],x[1]+(m-1)*k,by=k)),m+1)
    # p and q are used as current θ values running along the Θ vector
    p<-k_blocks[1]
    q<-k_blocks[nx+2]
    # intialise g,h
    g<-rep(0,nx)
    h<-rep(0,nx)
    r_min<-rep(0,nx)
    i<-1
    j<-nx
    while(i<=nx){
      this_p<-k_blocks[i+1]
      this_q<-k_blocks[j+1]
      g[i]<-ifelse(p==this_p,min(g[i-1],f[i]),f[i])
      h[j]<-ifelse(q==this_q,min(h[j+1],f[j]),f[j])
      p<-this_p
      q<-this_q
      i<-i+1
      j<-j-1
    }
    k0<-k/2
    # fast way to determine upper and lower indexes in R to avoid looping
    lo_bounds<-nx-rev(findInterval(rev(-x),rev(-(x+k0))))+1
```

```
50      hi_bounds<-findInterval(x+k0,x)
51      # case 3
52      r_min<-pmin(h[lo_bounds],g[hi_bounds])
53      # case 1: θ_{w_i^▽} = θ_{w_i^△} + 1
54      which_lo<-which(k_blocks[lo_bounds]==k_blocks[hi_bounds+1])
55      r_min[which_lo]<-h[lo_bounds[which_lo]]
56      # case 2: θ_{w_i^▽} + 1 = θ_{w_i^△}
57      which_hi<-which(k_blocks[lo_bounds+1]==k_blocks[hi_bounds+2])
58      r_min[which_hi]<-g[hi_bounds[which_hi]]
59      if(isAppend) r_min<-r_min[-nx]
60      cat("Completed morphological erosion (cts scale) in"
61        ,sprintf("%.2f",proc.time()[3]-t1),"seconds \n")
62      return(r_min)
63    }
64  }
```

A.2 Spectra normalisation

Description	File	Functions
Empirical quantile normalisation:		
	/04_quant_norm.R	quant_norm()
Pairwise spectra MA normalisation:		
	/05_ma_adj.R	ma_adj()

A.2.1 Empirical quantile normalisation

```
### Input: msData - the spectra intensities in matrix
###        where columns are spectra 1,2,...,n
quant_norm<-function(msData)
{
  msD<-msData
  nSpec<-ncol(msD)
  nDim<-nrow(msD)

  orders<-apply(msD, 2, order)
  reorders<-apply(orders, 2, order)
  # order each column into ascending order
  for(i in 1:nSpec) msD[,i]<-msD[orders[,i],i]
  #replace ordered columns with row means
  rmeans<-rowMeans(msD)
  for(i in 1:nSpec) msD[,i]<-rmeans
  #put back into the original order (with changed values)
  for(i in 1:nSpec) msD[,i]<-msD[reorders[,i],i]

  return(msD)
}
```

A.2.2 Pairwise spectra MA normalisation

```
### m_adj(): Used by ma_adj(), performs LOESS regression on ordered MA-vals
### Input: ordered dependent variable A with corresponding M
### Returns: adjusted M values, M_t^*
m_adj<-function(ordered_M,ordered_A)
{
  MA_finites<-is.finite(ordered_M) #only include values > -∞
  finites_M<-ordered_M[MA_finites]
  finites_A<-ordered_A[MA_finites]

  MAloess<-loess(finites_M~finites_A
    ,span=0.40,degree=2,family="symmetric",normalize=FALSE)

  finites_M<-finites_M-MAloess$fitted # make adjustments
  ordered_M[MA_finites]<-finites_M # and return adjusted values
  return(ordered_M)
}

### ma_adj(): Perform MA adjustment on two vectors
### Input: Two spectra vectors F_1 and F_2
### Returns: MA adjusted F_1 and F_2 values, F_1^* and F_2^* respectively
ma_adj<-function(F1,F2)
{
  t1<-proc.time()[3] ### get start time
  V1<-log2(F1) # Will produce -∞ for log_2(0)
  V2<-log2(F2)
  M<-V1-V2
  A<-(V1+V2)/2
  ### A is the dependent regression variable,
  ### ordering required for m_adj function
  ordered_indxs<-order(A)
  ordered_A<-A[ordered_indxs]
  ordered_M<-M[ordered_indxs]
  ordered_M<-m_adj(ordered_M,ordered_A)
  ### get indexes of original ordering
  orig_order<-order(ordered_indxs)
  M_dash<-ordered_M[orig_order]

  orig_finites<-is.finite(M) #update values requiring updating
  F1[orig_finites]<-2^(A[orig_finites]+M_dash[orig_finites]/2)
  F2[orig_finites]<-2^(A[orig_finites]-M_dash[orig_finites]/2)

  delta_t<-sprintf("%.2f",proc.time()[3]-t1) ### time elapsed
  cat("Completed MA Normalisation in",delta_t,"seconds \n")
  return(list(F1adj=F1,F2adj=F2))
}
```

A.3 Peak alignment

Description	File	Functions
Calculate W matrix for an N- and M-alignment:		
	/06_create_w.R	w_matrix()
Dendrogram peak alignment:		
	/07_dendro_peak_align.R	dendro_peak_align()

A.3.1 Calculate W matrix for an N- and M-alignment

```
1  ##############################################################################
2  ################################### Function #################################
3  ##############################################################################
4  #
5  # w_matrix(): Create a peak similarity matrix between an N- and M-alignment
6  #
7  ##############################################################################
8  ################################### Input ####################################
9  ##############################################################################
10 #
11 # Nmatchedpeaks: K x N matrix of matched pairs of the N alignment
12 # Npeaklists: a list object containing N matricies of
13 #         (time,intensityVector) pairs: (n_a x (n_Comp+1) matrix, a=1,...,N)
14 # Mmatchedpeaks: L x M matrix of matched pairs of the M alignment
15 # Mpeaklists: a list object containing M matricies of
16 #         (time,intensityVector) pairs: (n_b x (n_Comp+1) matrix, b=1,...,M)
17 #
18 ##############################################################################
19 #
20 # e.g. Nmatchedpeaks =
21 # [ 1 0 1 0 ]
22 # [ 0 1 2 1 ]
23 # [ 0 0 0 2 ]
24 # [ 2 2 0 0 ]
25 # [ 3 3 3 3 ]
26 # [ 0 4 0 4 ]
27 # [ . . . . ]
28 # [ . . . . ]
29 # [ . . . . ]
30 #
31 # here K x N (N=4) matrix
32 #
33 # Npeaklist=
34 # [[1]]
35 # [t_{1,1} t_{1,2} ... t_{1,n_1} ]
36 # [x_{1,1} x_{1,2} ... x_{1,n_1} ]
37 #
38 # [[2]]
39 # [t_{2,1} t_{2,2} ... t_{2,n_2} ]
40 # [x_{2,1} x_{2,2} ... x_{2,n_2} ]
41 #
42 # ...
43 #
44 # [[N]]
45 # [t_{N,1} t_{N,2} ... t_{N,n_N} ]
46 # [x_{N,1} x_{N,2} ... x_{N,n_N} ]
47 #
48 # where t_{i,j} is the time point j-th peak for the i-th spectrum
49 # where x_{i,j} is the vector of intensities (n_Comp long)
```

```
#      i.e. n_Comp x 1 matrix for the j-th peak for the i-th spectrum
# NB: each list item is a (n_Comp + 1) x n_N matrix
#
# the _i_ co-ord is the row in the Nmatchedpeaks
# the _a_ co-ord is the column number of Nmatchedpeaks (the spectrum number)
# the _p_ co-ord is the peak number for the _a_th spectrum
#
# the _j_,_b_ and _q_ co-ords are defined similarly for the M-alignment

w_matrix<-function(Nmatchedpeaks,Npeaklists,Mmatchedpeaks,Mpeaklists
          ,D,expon,lambda){

   K<-nrow(Nmatchedpeaks)
   N<-ncol(Nmatchedpeaks)
   L<-nrow(Mmatchedpeaks)
   M<-ncol(Mmatchedpeaks)
   W<-matrix(0,nrow=K,ncol=L)

   for(i in 1:K){
      for(j in 1:L){
         numerator<-0
         denominator<-0
         for(a in 1:N){
            p<-Nmatchedpeaks[i,a]
            if(p>0){
               for(b in 1:M){
                  q<-Mmatchedpeaks[j,b]
                  if(q>0){
                     t_a<-Npeaklists[[a]][1,p]
                     p_a<-Npeaklists[[a]][-1,p]
                     t_b<-Mpeaklists[[b]][1,q]
                     p_b<-Mpeaklists[[b]][-1,q]
                     numerator<-numerator+
                       PeakSim(p_a,t_a,p_b,t_b,D,expon,lambda)
                     denominator<-denominator+1
                  }
               }
            }
         }
         if(denominator>0) W[i,j]<-numerator/denominator
         else W[i,j]<-0
      }
   }
   return(W)
}
```

A.3.2 Dendrogram peak alignment

```
################################################################################
##################################### Function #################################
################################################################################
#
# dendro_peak_align(): for peak list data, create successive N- and M-alignments
#              until all spectra are aligned.
#
################################################################################
##################################### Input ####################################
################################################################################
#
# msD: MS Data, a T x n matrix of MS intensities. One column per spectra.
# peaklistlist: see below
# in.param: [ D expon lambda G maxM ]-tuple as a vector
#
################################################################################
##################################### Output ###################################
################################################################################
#
### A list containing the following elements:
# dendro:
# stepwise.peaks: a list where each element is the successive amalgamation data
# amalpeaks: the final matrix of aligned peaks. Columns are named spectra, rows
#    are
#
################################################################################

### peaklistlist=
# [[1]]
# [t_{1,1} t_{1,2} ... t_{1,n_1} ]
# [x_{1,1} x_{1,2} ... x_{1,n_1} ]
#
# [[2]]
# [t_{2,1} t_{2,2} ... t_{2,n_2} ]
# [x_{2,1} x_{2,2} ... x_{2,n_2} ]
#
# .
# .
# .
#
# [[N]]
# [t_{N,1} t_{N,2} ... t_{N,n_N} ]
# [x_{N,1} x_{N,2} ... x_{N,n_N} ]
#
# where t_{i,j} is the time point j-th peak for the i-th spectrum
# where x_{i,j} is the vector of intensities (nComp long i.e. nComp x 1 matrix)
#         for the j-th peak for the i-th spectrum
# NB: each list item is a (n_Comp+1) x n_N matrix

```

```
50  dendro_peak_align<-function(msD,peaklistlist,in.param)
51  {
52
53    D<-in.param[1]
54    nC<-nrow(peaklistlist[[1]])-1
55    expon<-in.param[2]
56    lambda<-in.param[3]
57    G<-in.param[4]
58    maxM<-in.param[5]
59
60    nPat<-length(peaklistlist)
61    Pats<-1:nPat
62
63    cat("Calculating merge sequence for spectra \n")
64
65    fordist<-t(msD$intensity)
66    hc<-hclust(as.dist(fordist,diag=FALSE,upper=FALSE),"average")
67
68    ### find amalgamation sequence
69    ### see ?hclust for information on the merge matrix:
70    #    "an n-1 by 2 matrix. Row i of merge describes the merging of clusters
71    #    at step i of the clustering. If an element j in the row is negative,
72    #    then observation -j was merged at this stage. If j is positive then
73    #    the merge was with the cluster formed at the (earlier) stage j of the
74    #    algorithm. Thus negative entries in merge indicate agglomerations of
75    #    singletons, and positive entries indicate agglomerations of
76    #    non-singletons."
77    amalg<-hc$merge
78
79    nAmal<-nrow(amalg)
80    # alignment of peaklistlist (a.pll)
81    a.pll<-vector(length=nAmal,mode="list")
82    Npeaks<-NULL
83    Npeaklist<-NULL
84    Mpeaks<-NULL
85    Mpeaklist<-NULL
86
87    # start
88    for(aindx in 1:nAmal)
89    {
90
91      # if patsToGetN or patsToGetM are positive,
92      #    ... it is a single spectrum (1-alignment)
93      # if negative, it is a previous N/M-alignment (N,M>1)
94      patsToGetN<--amalg[aindx,1]
95      patsToGetM<--amalg[aindx,2]
96
97      printPatsN<-sprintf("%03d",patsToGetN)
98      printPatsM<-sprintf("%03d",patsToGetM)
99      amalg.str<-"Amalgamting patient"
100     if(patsToGetN>0 && patsToGetM>0){
101       cat(amalg.str,"s ",printPatsN," and ",printPatsM,"\n",sep="")
```

```
102     }else if(patsToGetN>0 && patsToGetM<0){
103       cat(amalg.str,printPatsN,"to previously amalgamated patients\n")
104     }else if(patsToGetN<0 && patsToGetM>0){
105       cat(amalg.str,printPatsM,"to previously amalgamated patients\n")
106     }else{
107       cat("Amalgamting two clusters of previously amalgamated patients\n")
108     }
109
110     ### prepare N-Alignment data
111     if(patsToGetN>0){ # if a single spectrum (1-alignment)
112       Npeaks<-matrix(1:ncol(peaklistlist[[patsToGetN]]),ncol=1)
113       Npeaklist<-peaklistlist[patsToGetN]
114     }else{ # if a previously aligned N-alignment (N>1)
115       Npeaks<-a.pll[[-patsToGetN]]
116       patsToGetN<-as.numeric(colnames(Npeaks))
117       Npeaklist<-peaklistlist[patsToGetN]
118     }
119
120     ### prepare M-Alignment data
121     if(patsToGetM>0){ # if a single spectrum (1-alignment)
122       Mpeaks<-matrix(1:ncol(peaklistlist[[patsToGetM]]),ncol=1)
123       Mpeaklist<-peaklistlist[patsToGetM]
124     }else{ # if a previously aligned M-alignment (M>1)
125       Mpeaks<-a.pll[[-patsToGetM]]
126       patsToGetM<-as.numeric(colnames(Mpeaks))
127       Mpeaklist<-peaklistlist[patsToGetM]
128     }
129
130     ### use Wmatrix() function
131     Wm<-Wmatrix(Npeaks,Npeaklist,Mpeaks,Mpeaklist,D,expon,lambda)
132     ### use S-W alignment function to estimate maximum path
133     ### see: https://code.google.com/p/swalign/
134     estPM<-SWalign(Wm,G,maxM)
135     ### estPM is a data.frame of (i,j) locations of the maximum path
136     ### the data.frame is 2 columns for i,j points
137
138     nN<-ncol(Npeaks) ### no. of peaks in N-align
139     nM<-ncol(Mpeaks) ### no. of peaks in M-align
140     nK<-nrow(estPM)  ### no. of peaks in new N:M-align
141     ### apllTemp:
142     ###      (a)lignment of (p)eak (l)ist (l)ist, (temp)orary
143     ### Matrix of peak indicators. The nK rows represent the nK peaks
144     ###    from the N:M-alignment.
145     ### Entries apllTemp[i,j] are ==
146     ### { 0 if that N:M-aligned peak does not exist in spec j (column j)
147     ### { _else_ a non-zero indicator, the peak number from within the
148     ###                              1-alignment from spectrum j (column j)
149     apllTemp<-matrix(0,nrow=nK,ncol=nN+nM)
150     mzValsTemp<-NULL
151     AveMzValsTemp<-NULL
152     for(n.k in 1:nK)
153     {
```

```
154         if(estPM[n.k,1]>0) # if the peak exists in the N-alignment
155         {
156           # transfer peak info from N-align to new N:M-align matrix
157           apllTemp[n.k,1:nN]<-Npeaks[estPM[n.k,1],]
158           for(i in 1:nN) if(apllTemp[n.k,i]>0) mzValsTemp<-
159               c(mzValsTemp,Npeaklist[[i]][1,apllTemp[n.k,i]])
160         }
161         if(estPM[n.k,2]>0) # if the peak exists in the M-alignment
162         {
163           # transfer peak info from N-align to new N:M-align matrix
164           apllTemp[n.k,(nN+1):(nN+nM)]<-Mpeaks[estPM[n.k,2],]
165           for(i in (nN+1):(nN+nM)) if(apllTemp[n.k,i]>0) mzValsTemp<-
166               c(mzValsTemp,Mpeaklist[[i-nN]][1,apllTemp[n.k,i]])
167         }
168         # get ave m/z of all aligned peaks
169         AveMzValsTemp<-c(AveMzValsTemp,mean(mzValsTemp))
170         mzValsTemp<-NULL
171       }
172       ### change row order if averaging m/z has changed peak location order
173       mzReOrder<-order(AveMzValsTemp)
174       apllTemp<-apllTemp[mzReOrder,]
175 
176       allPat<-c(patsToGetN,patsToGetM)
177       colnames(apllTemp)<-allPat
178       ### clean up N:M-alignment to preserve spectrum order
179       patOrder<-order(allPat)
180       apllTemp<-apllTemp[,patOrder]
181 
182       a.pll[[aindx]]<-apllTemp
183     }
184     ### return list() object of peak amalgamation/alignment,
185     ###     including intermediate steps
186     outlist<-list(dendro=hc,stepwise.peaks=a.pll[-nAmal],amalpeaks=a.pll[[nAmal]])
187     return(outlist)
188 
189 }
```

A.4 Surrogate variable analysis

Description	File	Functions
Get SVA adjusted expression matrix:	/08_do_sva.R	do_sva()

A.4.1 Get SVA adjusted expression matrix

Please note the function `getH()` (line 47 below) is the code available in the `DanteR` package to determine the number of significant surrogate variables. The function `mulReg(Y,X)` performs sequential linear regressions on the columns of the input `Y` using a fixed effects design matrix `X`. `mulReg()` returns a list containing the following vectors and matrices: $\texttt{RES}_{n\times P}$, residual matrix after `Y` has been regressed; $\texttt{BETA}_{d\times P}$, matrix of the regression coefficients, P columns for each regression; $\texttt{TVALS}_{d\times P}$, the corresponding t-statistics; $\texttt{PVALS}_{d\times P}$ the corresponding p-values of `TVALS`; $\texttt{FPVALS}_{P\times 1}$, p-value for each linear regression corresponding to the null model F-statistic.

```
1  ################################### FUNCTION ###################################
2  #### do_sva: Perform SVA using the model:
3  ####        Y_j = μ_j + Xα_j + Zβ_j + Wδ_j + e_j
4  ################################### INPUTS #####################################
5  #### Y: is a n × p matrix, where each p columns are regressed
6  #### Intecept: boolean; do we want to fit a mean value? (yes, in most cases)
7  #### X: is a n × d_α design matrix of the factors of interest
8  #### Z: is a n × d_β design matrix of the incidental experimental factors
9  #### nosigsv: the number (referred to as H in some papers) of significant
10 ####         eigen vecs if NULL, the function will determine. If less than
11 ####         1, no W computed
12 #### verbose: boolean, whether the surragate variable matrix, W is returned
13 #### seed: an integer to feed into 'set.seed()' for reproducable results
14 ################################### OUTPUTS ####################################
15 #### Ytilde: the Y matrix with Zβ_j + Wδ_j removed
16 #### pvals: the p-values of Ytilde regressed on μ_j + Xα_j + Zβ_j + Wδ_j
17 #### tvals: the corresponding t-statistics
18 #### betas: the corresponding α_j, β_j, δ_j estimates
19 #### paramlabels: a combination of I (intercept), X, Z, W to signify the
20 ####         relevent rows of p-vals/tvals/betas
21 #### W: the eigen vectors matrix
22 #### H: the number of columns of W (used eigen-vectors)
23 ################################################################################
24 do_sva<-function(
25   Y,Intercept=TRUE,X=NULL,Z=NULL,nosigsv=NULL,verbose=FALSE,seed=NULL
26 ){
27   n<-nrow(Y)
28   thisInt<-IXZ<-NULL
29   if(Intercept) thisInt<-matrix(1,nrow=n,ncol=1,dimnames=list(NULL,"Intcpt"))
30   if(is.null(thisInt) && is.null(X) && is.null(Z))
31   {
32     cat("At least one of: Intercept, X and Z must be specified \n")
33     return(NULL)
34   } else IXZ<-cbind(thisInt,X,Z)
35   kparam<-ncol(IXZ)
36   colmarkers<-rep("",kparam)
37   indx<-0
38   if(!is.null(thisInt)) colmarkers[indx<-indx+1]<-"I"
```

```
39    if(!is.null(X)) colmarkers[(indx<-indx+1):(indx<-indx+ncol(X)-1)]<-"X"
40    if(!is.null(Z)) colmarkers[(indx<-indx+1):(indx<-indx+ncol(Z)-1)]<-"Z"
41
42    RIXZ<-multReg(Y,IXZ,createNAvals=TRUE,seed=seed)
43    thissvd<-svd(RIXZ$RES)
44
45    W<-H<-NULL
46    if(is.null(nosigsv)){
47      H<-getH(RIXZ$RES,IXZ,nullsig=0.1,verbose=FALSE)
48      if(H<1) cat("No significant surrogate variables found \n")
49    }else{
50      H<-nosigsv
51    }
52    if(H<1){
53      cat("No surrogate variables will be used \n")
54    }else{
55      cat("Using H=",H," significant surrogate variables \n",sep="")
56      W<-as.matrix(thissvd$u[,1:H])
57      colnames(W)<-paste("W",1:H,sep="")
58      colmarkers<-c(colmarkers,rep("W",H))
59    }
60    IXZW<-cbind(IXZ,W)
61    Rtilde<-multReg(Y,IXZW)
62    removecols<-colmarkers %in% c("Z","W")
63    ZBetaWDelta<-0
64    if(sum(removecols)) ZBetaWDelta<-as.matrix(IXZW[,removecols]) %*%
65                  as.matrix(Rtilde$BETA[removecols,])
66    Ytilde<-Y-ZBetaWDelta
67    if(verbose) return(list(Ytilde=Ytilde,paramlabels=colmarkers,W=W,H=H))
68    else return(Ytilde)
69  }
```

A.5 Pairwise fusion linear discriminant analysis

Description	File	Functions
Create a PFDA object:		
	/09_create_pfda_obj.R	create_pfda_obj()
Predict class for new data and a PFDA object:		
	/10_pfda_predict.R	pfda_predict()

A.5.1 Create a PFDA object

```
####### FUNCTION: create_pfda_obj()
### estimate parameters of PF-DA model, so that a discrim function created

####### input:
### X: a n x p matrix, of n obs and p variables
### Xclass: a vector of length n of the classes (must be a factor variable)
### priors: a vector of length K (#classes) with elements in (0,1)

create_pfda_obj<-function(X,Xclass,lambdar=1,priors=NULL,alph=NULL,wts=NULL)
{
  N<-length(Xclass)
  P<-ncol(X)
  nks<-table(Xclass)
  classnames<-levels(Xclass)
  K<-length(classnames)

  ### if not supplied, make π̂_k data proportions
  if(is.null(priors)) priors<-nks/N

  if(length(priors)!=K){
    cat("The length of priors and the total number
        of groups must be equal \n")
    return(NULL)
  }else if(is.null(alph) & (N<P)){
    cat("Alpha is suggested for n<p data \n")
  }else if(N!=nrow(X)){
    cat("The length of Xclass and the number
        of rows in X must agree \n")
    return(NULL)
  }else if(!all(nks>1)){
    cat("There needs to be at least two obs in each
        group for variances to be computed \n")
    return(NULL)
  }

  Xclassint<-as.integer(Xclass)
  transMeans<-colMeans(X)
  X<-X-matrix(rep(transMeans,N),nrow=N,byrow=TRUE)

  ##### create μ_k = [μ_k1,...,μ_kp] vectors
  ##### place on top of each other to get KxP matrix
  MuMat<-matrix(0,nrow=K,ncol=P)
  for(k in 1:K) MuMat[k,]<-colMeans(X[Xclassint==k,])
  MuIter<-MuMat

  ##### create Σ
  Sigma<-matrix(0,nrow=P,ncol=P)
  for(k in 1:K)
  {
```

```
    rowuse<-which(Xclassint==k)
    Sigma<-Sigma+length(rowuse)*cov.wt(X[rowuse,],cor=FALSE
                                       ,center=TRUE,method="ML")$cov
  }
  Sigma<-Sigma/N
  ##### extract Diag elements
  sigmasqs<-diag(Sigma)

  if(!(is.null(alph) | is.null(wts))) sigmasqs<-sigmasqs+alph*wts
  else if(!is.null(alph)) sigmasqs<-sigmasqs+rep(alph,P)

  #### Now start iterative estimation of the ℓ1 penalised means
  #### Note "squig" is used for the ML estimates
  G<-matrix(0,nrow=K,ncol=K)
  deltatol<-1e-10
  deltaMu<-1
  maxIter<-500
  itcount<-0
  while(deltaMu>(1e-5) && itcount<maxIter)
  {
    deltaMuNumer<-0
    deltaMuDenom<-0
    itcount<-itcount+1

    ### For each of the features
    for(j in 1:P)
    {
      beta.t.j<-MuIter[,j]
      musqig.j<-MuMat[,j]
      sqigY<-X[,j]
      sqigX<-matrix(0,nrow=N,ncol=K)
      G<-matrix(0,nrow=K,ncol=K)

      ### ∑_{k=1}^{K-1} ∑_{k_dash=k+1}^{K}
      for(k in 1:(K-1))
      {
        for(kdash in (k+1):K)
        {
          PFweight<-1/abs(musqig.j[k]-musqig.j[kdash])
          ### assign updated iterations, or tol value if "zero"
          muDiffIter<-max(abs(beta.t.j[k]-beta.t.j[kdash]),deltatol)
          G[k,kdash]<-G[kdash,k]<- -PFweight/muDiffIter
        }
      }
      for(k in 1:K)
      {
        sqigX[which(Xclassint==k),k]<-1
        ### note the diag elements of G can be calculated as the sum of the column
        G[k,k]<- -sum(G[,k])
      }
      #### M̂ = (B^T B + λσ_j^2 G)^{-1} B^T J
      MuIter[,j]<-solve(t(sqigX)%*%sqigX+lambdar*sigmasqs[j]*G)%*%(t(sqigX)%*%sqigY)
```

```
      deltaMuNumer<-deltaMuNumer+sum(abs(MuIter[,j]-beta.t.j))
      deltaMuDenom<-deltaMuDenom+sum(abs(beta.t.j))
    }
    ### our break loop value
    deltaMu<-deltaMuNumer/deltaMuDenom

  }
  cat("Iterations performed to aquire a solution:",itcount
      ,"| Final tol val:",deltaMu," \n")

  MuIter[which(MuIter<deltatol)]<-0

  ### $\frac{1}{2}\sum_{j=1}^{p}\hat{\mu}_{kj}^2$ constant term
  consts<-rep(0,K)
  for(k in 1:K) consts[k]<-0.5*sum(MuIter[k,]^2/sigmasqs)
  ### $\hat{\mu}_{kj}^2/\sigma_p^2$ term
  lins<-vector(mode="list",length=K)
  for(k in 1:K) lins[[k]]<-MuIter[k,]/sigmasqs

  ### return calculated information as list object
  return(list(classes=classnames,consts=consts,lins=lins,prior=priors
              ,meanadj=transMeans,MuIter=MuIter,InitMu=MuMat))
}
```

A.5.2 Predict class for new data and a PFDA object

```
####### FUNCTION: pfda_predict()
### estimate probabilities and class of new inputs

####### input:
### ldaobj: an object created by create_pfda_obj()
### Xnew: a n x p matrix, of n obs and p variables.
###         May also be a single numeric vector (n=1) of length p
### priors: a vector of length K (#classes) with elements in (0,1)

pfda_predict<-function(ldaobj,Xnew,priors=NULL)
{
  if(is.vector(Xnew,mode="numeric")){
    Xnew<-matrix(Xnew,nrow=1)
  }else if(!is.matrix(Xnew)){
    cat("Xnew must be numeric and either a vector or matrix \n ")
  }

  Nnew<-nrow(Xnew)
  ### centre each feature at 0, as done on the training data
  Xnew<-Xnew-matrix(rep(ldaobj$meanadj,Nnew),nrow=Nnew,byrow=TRUE)
  Xclasses<-ldaobj$classes
  K<-length(Xclasses)
  if(is.null(priors)) priors<-ldaobj$prior

  outposteriors<-matrix(0,nrow=Nnew,ncol=K)
  colnames(outposteriors)<-Xclasses

  ### discrim function: δ_k(x_new)
  for(i in 1:Nnew){
    for(k in 1:K){
      outposteriors[i,k]<-log(priors[k]) - ldaobj$consts[k]
                                  + sum(Xnew[i,]*ldaobj$lins[[k]])
    }
  }

  ### predicted class is arg max
  ### p(G_K|x_new) = P(x_new|G_k)P(G_k) / Σ_{i=1}^K P(x_new|G_i)P(G_i)
  predclasses<-rep(0,Nnew)
  for(i in 1:Nnew){
    outposteriors[i,]<-exp(outposteriors[i,])
    predclasses[i]<-which.max(outposteriors[i,])
    outposteriors[i,]<-outposteriors[i,]/sum(outposteriors[i,])
  }
  outpred<-factor(Xclasses[predclasses])

  return(list(pred=outpred,posteriors=outposteriors))
}
```

A.6 Pareto Fronts for variable ranking

Description	File	Functions
Calculate dominating features:		
	/11_dom_feat.c	dom_feat()
Pareto Front wrapper functions:		
	/12_pareto_fronts.R	pareto_ranking()

A.6.1 Calculate dominating features

Below is the core of the Pareto Front code, finding features that are the dominated as per the definition. Written in `C` to be compiled to a `.so` file (or `.dll` on Windows operating systems) that in turn can be loaded into `R`.

```c
#include <R.h>

void dom_feat(int *n, double *obja, double *objb, int *domvec)
{
  int i, j, nonDomI;
  for (i=0; i<*n; i++)
  {
    j=0;
    nonDomI=1;
    while(nonDomI && j<*n)
    {
      if((obja[i]<obja[j]) && (objb[i]<=objb[j])) nonDomI=0;
      else if((objb[i]<objb[j]) && (obja[i]<=obja[j])) nonDomI=0;
      j+=1;
    }
    domvec[i]=-nonDomI+1;
  }
}
```

A.6.2 Pareto Front wrapper functions

```
# require animation package for kfcv() function
library(animation)
# load compiled C code (shared object)
dyn.load("dom_feat.so")

##########################################
##########################################
#
# Pairwise case of Pareto Fronts
#     obj is the 2 × n matrix of the two vectors of length n for
#      the features/observations of the 2 criteria/objective functions
#     istomin is a boolean vector of whether the criteria obj_1, obj_2
#      are to be minimised (=TRUE), respectively
#
##########################################
##########################################

#
# This function returns a vector of 'dominated' observations (Boolean,
# length n vector) FALSE=Pareto front, TRUE=dominated observation
#

dom_features_pw<-function(obj,istomin)
{
  #if to be minimised then just make negative and maximise
  obj[,istomin]<- -obj[,istomin]
  n<-as.integer(nrow(obj))
  obj1<-as.double(obj[,1])
  obj2<-as.double(obj[,2])
  domvec<-as.integer(rep(0,n)) #output vector

  return(as.logical(.C("dom_feat",n,obj1,obj2,domvec)[[4]]))
}

##########################################
##########################################
#
# General case
#     objmatrix is n × m matrix. n features/observations and
#     m criteria/objective functions istominvec is a boolean vec
#     of length m to say whether the criteria are to be minimised
#
##########################################
##########################################

#
# This function returns a vector of 'dominated' observations (Boolean,
# length n vector) FALSE=Pareto front, TRUE=dominated observation
# same as pairwise but the input can take more than two objective functions
```

```
#

dom_features<-function(objmatrix,istominvec)
{
  n<-as.integer(nrow(objmatrix))
  m<-ncol(objmatrix)
  objmatrix[,istominvec]<- -objmatrix[,istominvec]
  vecdomvec<-rep(1,n)
  indxs<-combn(m,2)
  nm<-ncol(indxs)
  i<-0
  while(i<nm)
  {
    i<-i+1
    # call pairwise function, take the intersection of previous
    # dominated observations remembering the intersection(s)
    # of dominated in the same as unions(s) of Pareto fronts
    vecdomvec<-vecdomvec * .C("dom_feat"
              ,n
              ,as.double(objmatrix[,indxs[1,i]])
              ,as.double(objmatrix[,indxs[2,i]])
              ,as.integer(rep(0,n)))[[4]]
  }
  return(as.logical(vecdomvec))
}

##########################################
##########################################
#
# Sucessive Pareto Fronts
#     noFronts is the # of pareto fronts required
#
#     fn returns a vector of length n
#           each element is labelled the pareto front #,
#           0 is dominated even after noFronts found
#
##########################################
##########################################

pareto_fronts<-function(noFronts,objmatrix,istominvec)
{
  objmatrix[,istominvec]<- -objmatrix[,istominvec]
  n<-as.integer(nrow(objmatrix))
  m<-ncol(objmatrix)
  pfvec<-rep(0,n) #output vector
  #once a front is found we need to set the correponding values to ∞ or
  # −∞ so they won't be chosen again
  #try: as.numeric(c(TRUE,FALSE,TRUE))*2-1 to see what the next line is doing
  #if Min then set 1, ifMax then set -1 (the sign of the Inf if we find front
  #   and have to put to a value)
  ourInfs<-min(objmatrix)-1
  allFrontsFound<-FALSE
```

```
102
103   i<-0
104   while(i<noFronts && !allFrontsFound) #go thru all fronts required
105   {
106     i<-i+1
107     df<-dom_features(objmatrix,rep(FALSE,m)) #general m obj vectors function
108     # pf.i are the indexs of the output vector that need to be updated
109     # with the pareto front number
110     pf.i<-(!df) & (pfvec<1)
111     pfvec[pf.i]<-i
112     # re-assign values were pareto front found
113     objmatrix[pf.i,]<-ourInfs
114     if(all(pfvec>0)) allFrontsFound<-TRUE
115   }
116   return(pfvec)
117 }
118
119 ##########################################
120 ##########################################
121 #
122 # Leave-one-out/k-fold feature ranking
123 #
124 # returns a vector of length n with values ∈ (0,1] for feature importance
125 #
126 ##########################################
127 ##########################################
128
129 #
130 # Same inputs of previous functions, with folds (aka k-fold cross
131 # validation) and reps is the number of times we re-do the cross
132 # validation fold=1 or the length of the input (i.e. n) creates
133 # leave-one-out cross validation
134
135 pareto_ranking<-function(objmatrix,istominvec,noFronts=20,folds=1,reps=5)
136 {
137   objmatrix[,istominvec]<- -objmatrix[,istominvec]
138   m<-ncol(objmatrix)
139   n<-nrow(objmatrix)
140   pfmetric<-rep(0,n) #output vector
141   nfolds<-n
142   if(folds>1) nfolds<-folds
143   if(nfolds==n) reps<-1
144
145   blocks<-kfcv(nfolds,n)
146   block.nos<-rep(1:nfolds,blocks)
147
148   for(r in 1:reps)
149   {
150     indxs<-sample(1:n) #fresh randomisation each repetition
151     k.f.mat<-cbind(indxs,block.nos) # create the fold 'blocks' of data
152     for(i in 1:nfolds)
153     {
```

```
      rows<-k.f.mat[k.f.mat[,2]==i,1] # find the ith fold to leave out
      # call general function with ith fold removed
      calcfronts<-pareto_fronts(noFronts,objmatrix[-rows,],rep(FALSE,m))
      # which are non-dominated
      whichnondom<-calcfronts>0
      # if you are ont the first front you get 1, second=1/2, third=1/3,
      # ..., jth=1/j, else 0
      pfmetric[-rows][whichnondom]<-
          pfmetric[-rows][whichnondom]+1/calcfronts[whichnondom]
    }
  }
  #now divide by maximum posible value i.e. (nfolds-1)*reps so output in [0,1]
  pfmetric<-pfmetric/((nfolds-1)*reps)
  return(pfmetric)
}

##############################################################################
# Below are three metrics that can possibly describe the value of
#   variables/fetures to discriminate betwwen classes
##############################################################################

# minIntraClassVar(): find the minimum WITHIN class variance of the K groups
# interClassVar(): find the variance of means/centroids of the K groups
# maxInterClassDist(): possibly correlated with interClassVar, find the dist
#              MAXIMUM between the K group's centroids/means

# The rationale of the last metric is that a variable/feature that only
# seperates two of the K classes  is undervalued by the Fisher score because
# it may not separate the K-2 classes remaining well.
# ... And a separation of two classes (in conjunction with other variables)
#      is important information for the discriminant model

##############################################################################
#### ds: a data.frame or matrix (numeric values only/factors will be dealt
####  with as integers)  class vec: a vector correspong to the class of
####    the rows of ds
##############################################################################

minIntraClassVar<-function(ds,class.vec){

  dsfs<-ds
  if(!is.matrix(dsfs)) dsfs<-data.matrix(dsfs)

  p<-ncol(dsfs)
  K<-length(levels(class.vec))
  n.all<-length(class.vec)
  n.i<-0

  intraClassVar<-Inf
  mean.j<-colMeans(dsfs)

  for(i in 1:K){
```

```
206     true.vec<-(as.integer(class.vec)==i)
207     n.i<-length(which(true.vec))
208     mean.class<-colMeans(as.matrix(dsfs[true.vec,],ncol=p))
209     var.class<-colSums(as.matrix((dsfs[true.vec,]-rep(mean.class,each=n.i))^2,ncol=p))
210     intraClassVar<-pmin(intraClassVar,var.class/(n.i-1))
211   }
212   return(intraClassVar)
213 }
214
215 interClassVar<-function(ds,class.vec){
216
217   dsfs<-ds
218   if(!is.matrix(dsfs)) dsfs<-data.matrix(dsfs)
219
220   p<-ncol(dsfs)
221   K<-length(levels(class.vec))
222
223   interClassVar<-0
224   mean.j<-colMeans(dsfs)
225
226   for(i in 1:K){
227     true.vec<-(as.integer(class.vec)==i)
228     n.i<-length(which(true.vec))
229     mean.class<-colMeans(as.matrix(dsfs[true.vec,],ncol=p))
230     var.class<-((mean.j-mean.class)^2)
231     interClassVar<-interClassVar+var.class
232   }
233   return(interClassVar/(K-1))
234 }
235
236 maxInterClassDist<-function(ds,class.vec){
237
238   dsfs<-ds
239   if(!is.matrix(dsfs)) dsfs<-data.matrix(dsfs)
240
241   p<-ncol(dsfs)
242   K<-length(levels(class.vec))
243
244   interClassDist<--Inf
245   mean.j<-colMeans(dsfs)
246
247   for(i in 1:K){
248     true.vec<-(as.integer(class.vec)==i)
249     mean.class<-colMeans(as.matrix(dsfs[true.vec,],ncol=p))
250     interClassDist<-pmax(interClassDist,abs(mean.j-mean.class))
251   }
252   return(interClassDist)
253 }
```