Efficient Programming in S

Bill Venables and Brian Ripley

Workshop, 6 October 1998


```c
/*
 *  lqs.c by B. D. Ripley  Copyright (C) 1998
 */

#include <S.h>
#include <math.h>

/* corrected from R. Sedgewick ‘Algorithms in C’ */
static void shellsort(double *a, int N)
{
  int i, j, h;
  double v;

  for (h = 1; h <= N / 9; h = 3 * h + 1);
  for (; h > 0; h /= 3)
    for (i = h; i < N; i++) {
      v = a[i];
      j = i;
      while (j >= h && a[j - h] > v) { a[j] = a[j - h]; j -= h; }
      a[j] = v;
    }
}

/*
   Partial sort so that a[k] is in the correct place, smaller to left,
   larger to right
 */
static void psort(double *a, int N, int k)
{
  int L, R, i, j;
  double v, tmp;

  for (L = 0, R = N - 1; L < R; ) {
    v = a[k];
    for(i = L, j = R; i <= j;) {
      while (a[i] < v) i++;
      while (v < a[j]) j--;
      if (i <= j) { tmp = a[i]; a[i++] = a[j]; a[j--] =tmp; }
    }
    if (j < k) L = i;
    if (k < i) R = j;
  }
}


/* find qr decomposition, basis of qr.default() */
void F77_NAME(dqr)(double *qr, long *dx, long *pivot, double *qraux,
                   double *tol, double *work, long *rank);
```

```
/* solve for coefficients */
void F77_NAME(dqrsl)(double *qr, long *ldx, long *n, long *rank,
                     double *qraux,
                     double *y1, double *d1, double *y2, double *coef,
                     double *d3, double *d4, long *job, long *info);


static double *coef, *qraux, *work, *res, *yr, *xr, *means, *d2, *d2copy;
static long *pivot, *which, *which2, *bestone;
static int *ind;

/*
   Sampling k from 0:n-1 without replacement.
 */
static void sample_noreplace(long *x, int n, int k)
{
  int i, ii, j, nn=n;

  for (i = 0; i < n; i++) ind[i] = i;
    for (i = 0; i < k; i++) {
      j = nn * unif_rand();
      x[i] = ind[j];
      ind[j] = ind[--nn];
    }
}

/*
   Find all subsets of size k in order: this gets a new one each call
 */
static void next_set(long *x, int n, int k)
{
  int i, j, tmp;

  j = k - 1;
  tmp = x[j]++;
  while(j > 0 && x[j] >= n - (k - 1 -j)) tmp = ++x[--j];
  for(i = j+1; i < k; i++)  x[i] =  ++tmp;
}


static void lqs_setup(long *n, long *p, long *ps)
{
  coef = Calloc(*p, double);
  qraux = Calloc(*p, double);
  work = Calloc(2*(*p), double);
  res = Calloc(*n, double);
  yr = Calloc(*n, double);
  xr = Calloc((*n)*(*p), double);
  pivot = Calloc(*p, long);
  ind = Calloc(*n, int);
```



```
  which = Calloc(*ps, long);
  bestone = Calloc(*ps, long);
}

static void lqs_free()
{
  Free(coef); Free(qraux); Free(work); Free(res); Free(yr); Free(xr);
  Free(pivot); Free(ind); Free(which); Free(bestone);
}

/*
   Adjust the constant for an LMS fit. This is the midpoint of the
   qn contiguous observations of shortest length.
 */
static double lmsadj(double *x, int n, int qn, double *ssbest)
{
  int i, k = qn - 1;
  double len, best, adj;

  best = x[k] - x[0];
  adj = 0.5*(x[k] + x[0]);
  for(i = 1; i < n-k; i++){
    len = x[i+k] - x[i];
    if(len < best) {
      best = len;
      adj = 0.5*(x[i+k] + x[i]);
    }
  }
  *ssbest = 0.25*best*best;
  return(adj);
}

/*
   Adjust the constant for an LTS fit. This is the mean of the
   qn contiguous observations of smallest variance.
 */
static double ltsadj(double *x, int n, int qn, double *ssbest)
{
  int i, k = qn - 1;
  double ss, best, m1, m2, adj;

  /*printf("qn = %d\n", qn);*/
  m1 = m2 = 0.0;
  for(i=0; i < qn; i++) {
    m1 += x[i];
    m2 += x[i]*x[i];
  }
  adj = m1/qn;
  best = m2 - m1*m1/qn;
```

```
  for(i = 1; i < n-k; i++){
    m1 += x[i+k] - x[i-1];
    m2 += x[i+k]*x[i+k] - x[i-1]*x[i-1];
    ss = m2 - m1*m1/qn;
    if(ss < best) {
      best = ss;
      adj = m1/qn;
    }
  }
  *ssbest = best;
  return(adj);
}

/* the chi function for the S estimator: the integral of biweight */
static double chi(double x, double a)
{
  x /= a; x *= x;
  if(x > 1) return(1.0);
  else return(3*x - 3*x*x + x*x*x);
}

/*
   For lots of subsets of size *nwhich, compute the exact fit to those
   data points and the residuals from all the data points.
 */
void
lqs_fitlots(double *x, double *y, long *n, long *p, long *qn,
            long *lts, long *adj, long *sample, long *nwhich,
            long *ntrials, double *crit, long *sing, long *bestone,
            double *bestcoef, long *pk0, double *beta)
{
  long i, iter, j, k, k0 = *pk0, nn = *n, nnew = *nwhich, this, trial;
  long dx[2], rank, info, ny = 1, n100 = 100;
  double a = 0.0, tol = 1.0e-7, sum, thiscrit, best=1.e20, target,
    old, new, dummy;

  lqs_setup(n, p, nwhich);

  *sing = 0;
  target = (nn - *p)* (*beta);

  if(!*sample) {
    for(i = 0; i < nnew; i++) which[i] = i;
  } else seed_in((long *)NULL);

  for(trial = 0; trial < *ntrials; trial++) {

    if(!(*sample)) {if(trial > 0) next_set(which, nn, nnew);}
    else sample_noreplace(which, nn, nnew);
```



```
    for(j = 0; j < nnew; j++) {
      this = which[j];
      yr[j] = y[this];
      for(k = 0; k < *p; k++) xr[j + nnew*k] = x[this + nn*k];
    }

    /* compute fit, find residuals */
    dx[0] = nnew; dx[1] = *p;
    for(i = 0; i < *p; i++) pivot[i] = i+1;
    rank = *p;
    F77_NAME(dqr)(xr, dx, pivot, qraux, &tol, work, &rank);
    if(rank < *p) { (*sing)++; continue; }
    F77_NAME(dqrsl)(xr, &nnew, &nnew, &rank, qraux, yr, &dummy, yr,
                    coef, &dummy, &dummy, &n100, &info);

    for(i = 0; i < nn; i++) {
      sum = y[i];
      for(j = 0; j < rank; j++) sum -= coef[j] * x[i + nn*j];
      res[i] = sum;
    }

    if(*lts < 2) {/* lqs or lts estimation */
      /* find the constant subtracted from the residuals that minimizes
         the criterion. As this is a univariate problem, has an exact
         solution.  */
      if(*adj) {
        shellsort(res, nn);
        if(*lts) a = ltsadj(res, nn, *qn, &thiscrit);
        else a = lmsadj(res, nn, *qn, &thiscrit);
      } else {
        for(i = 0; i < nn; i++) {
          sum = res[i] - a;
          res[i] = sum*sum;
        }
        psort(res, nn, *qn-1); /* partial sort */
        if(!(*lts)) thiscrit = res[*qn-1];
        else {
          sum = 0.0;
          for(i = 0; i < *qn; i++) sum += res[i];
          thiscrit = sum;
        }
      }
    } else { /* S estimation */
      if(trial == 0) {
        for(i = 0; i < nn; i ++) res[i] = fabs(res[i]);
        psort(res, nn, nn/2);
        old = res[nn/2]/0.6745;  /* MAD provides the initial scale */
      } else {
        /* only find optimal scale if it will be better than
           existing best solution */
```

```
      sum = 0.0;
      for(i = 0; i < nn; i ++) sum += chi(res[i], k0 * best);
      if(sum > target) continue;
      old = best;
    } /* now solve for scale S by re-substitution */
    for(iter = 0; iter < 30; iter++) {
      /*printf("iter %d, s = %f sum = %f %f\n", iter, old, sum, target);*/
      sum = 0.0;
      for(i = 0; i < nn; i ++) sum += chi(res[i], k0 * old);
      new = sqrt(sum/target) * old;
      if(fabs(sum/target - 1.) < 1e-4) break;
      old = new;
    }
    thiscrit = new;
  }

  if(thiscrit < best) {  /* first trial might be singular, so use fence */
    sum = 0.0;
    for(i = 0; i < nn; i ++) sum += chi(res[i], k0 * best);
    best = thiscrit;
    for(i = 0; i < nnew; i++) bestone[i] = which[i] + 1;
    for(i = 0; i < *p; i++) bestcoef[i] = coef[i];
    bestcoef[0] += a;
  }
 }
 *crit = best;
 if(*sample) seed_out((long *)NULL);
 lqs_free();
}
```



```
# file lqs.q
# copyright (C) 1998 B. D. Ripley
#
lqs <- function(x, ...) UseMethod("lqs")

lqs.formula <-
  function(formula, data, ...,
           method = c("lts" ,"lqs", "lms", "S", "model.frame"),
           subset, na.action = na.fail,
           model = FALSE, x.ret = FALSE, y.ret = FALSE, contrasts = NULL)
{
  call <- match.call()
  method <- match.arg(method)
  mf <- match.call(expand.dots = FALSE)
  mf$method <- mf$contrasts <- mf$model <- mf$x <- mf$y <- mf$... <- NULL
  mf[[1]] <- as.name("model.frame")
  mf <- eval(mf, sys.parent())
  if (method == "model.frame") return(mf)
  mt <- attr(mf, "terms")
  y <- model.extract(mf, "response")
  x <- model.matrix(mt, mf, contrasts)
  xint <- match("(Intercept)", dimnames(x)[[2]], nomatch = 0)
  if(xint) x <- x[, -xint, drop = FALSE]
  fit <- lqs.default(x, y, intercept = (xint>0), method = method, ...)
  fit$terms <- mt
  fit$call <- call
  fit$contrasts <- attr(x, "contrasts")
  if(model) fit$model <- mf
  if(x.ret) fit$x <- x
  if(y.ret) fit$y <- y
  fit
}

lqs.default <-
  function(x, y, intercept=T, method = c("lts", "lqs", "lms", "S"),
           quantile, control = lqs.control(...), k0 = 1.548, seed, ...)
{
  lqs.control <- function(psamp = NA, nsamp = "best", adjust = TRUE)
    list(psamp = psamp, nsamp = nsamp, adjust = adjust)

  choose <- function(n, k)
  {
    if(k > n) stop("must have k <= n")
    round(exp(lgamma(n + 1) - lgamma(k + 1) - lgamma(n - k + 1)))
  }

  n <- length(y)
  nmx <- deparse(substitute(x))
  if(is.null(dim(x))) {
    x <- as.matrix(x)
```

```
  dimnames(x)[[2]] <- nmx
} else x <- as.matrix(x)
p <- ncol(x)
if(any(is.na(x)) || any(is.na(y)))
  stop("missing values are not allowed")
nm <- dimnames(x)[[2]]
if(is.null(nm))
  nm <- if(p > 1) paste("X", 1:p, sep="") else if(p == 1) "X" else NULL
if(intercept) {
  x <- cbind(1, x)
  nm <- c("(Intercept)", nm)
}
p <- ncol(x)
if(nrow(x) != n) stop("x and y must have the same number of rows")
method <- match.arg(method)
lts <- 0; beta <- 0
if(method == "lqs" && missing(quantile)) quantile <- floor((n+p+1)/2)
if(method == "lms") quantile <- floor((n+1)/2)
if(method == "lts") {
  lts <- 1
  if(missing(quantile)) quantile <- floor(n/2) + floor((p+1)/2)
}
if(method == "S") {
  lts <- 2
  beta <- 0.5
  quantile <- ceiling(n/2)
  chi <- function(u, k0) { u <- (u/k0)^2; ifelse(u < 1, 3*u - 3*u^2 + u^3, 1) }
}
ps <- control$psamp
if(is.na(ps)) ps <- p
if(ps < p) {
  ps <- p
  warning("ps must be at least p")
}
adj <- control$adjust & intercept
nsamp <- eval(control$nsamp)
nexact <- choose(n, ps)
if(is.character(nsamp) && nsamp == "best")
  nsamp <- if(nexact < 5000) "exact" else "sample"
if(is.numeric(nsamp) && nsamp > nexact) {
  warning(paste("only", nexact, "sets, so all sets will be tried"))
  nsamp <- "exact"
}
if(nsamp == "exact") {
  nsamp <- nexact
  samp <- F
} else {
  if(nsamp == "sample") nsamp <- min(500*ps, 3000)
  samp <- T
}
```



```
  if(samp && !missing(seed)) {
    seed.keep <- .Random.seed
    on.exit(assign(".Random.seed", seed.keep, w=0))
    .Random.seed <<- seed
  }
  z <-  .C("lqs_fitlots",
          as.double(x), as.double(y), as.integer(n), as.integer(p),
          as.integer(quantile), as.integer(lts), as.integer(adj),
          as.integer(samp), as.integer(ps), as.integer(nsamp),
          crit=double(1), sing=integer(1), bestone=integer(ps),
          coefficients=double(p), as.integer(k0), as.double(beta)
          )[c("crit", "sing", "coefficients", "bestone")]
  z$sing <- paste(z$sing, "singular samples of size", ps, "out of", nsamp)
  z$bestone <- sort(z$bestone)
  names(z$coefficients) <- nm
  fitted <- drop(x %*% z$coefficients)
  z$fitted.values <- fitted
  z$residuals <- y - fitted
  c1 <- 1/qnorm((n + quantile)/(2*n))
  if(lts == 1) {
    s <- sqrt(z$crit/quantile)/sqrt(1 - 2*n*dnorm(1/c1)/(quantile*c1))
  } else if(lts == 0) {
    s <- sqrt(z$crit)*c1
  } else s <- z$crit
  res <- z$residual
  ind <- abs(res) <= 2.5*s
  s2 <- sum(res[ind]^2)/(sum(ind) - p)
  z$scale <- c(s, sqrt(s2))
  if(method == "S") { # IWLS refinement
    psi <- function(u, k0) (1  - pmin(1, abs(u/k0))^2)^2
    coef <- z$coef
    resid <- z$residuals
    scale <- s
    for(i in 1:30) {
      w <- psi(resid/scale, k0)
      temp <- lm.wfit(x, y, w, method="qr")
      resid <- temp$residuals
      s2 <- scale*sqrt(sum(chi(resid/scale, k0))/((n-p)*beta))
      if(abs(s2/scale - 1) < 1e-5) break
      scale <- s2
    }
    z$coef <- temp$coef
    z$fitted.values <- temp$fitted.values
    z$residuals <- resid
    z$scale <- scale
  }
  class(z) <- "lqs"
  z
}
```

```
print.lqs <- function (x, digits = max(3, .Options$digits - 3), ...)
{
  cat("\nCall:\n", deparse(x$call), "\n\n", sep = "")
  cat("Coefficients:\n")
  print.default(format(coef(x), digits = digits), quote = FALSE)
  cat("\nScale estimates", format(x$scale, digits = digits) ,"\n\n")
  invisible(x)
}

predict.lqs <- function (object, newdata)
{
  if (missing(newdata)) return(object$fitted.values)
  X <- model.matrix(delete.response(terms(object)), newdata,
                    contrasts = object$contrasts)
  drop(X %*% object$coefficients)
}

residuals.lqs <- function(x) x$residuals
```

GUI programming

A simple dialog box

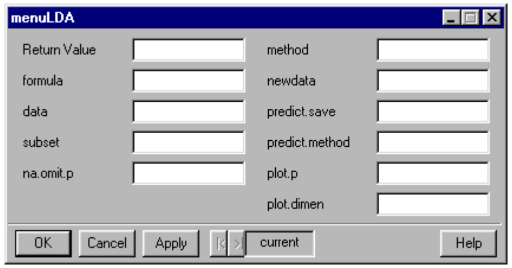

Figure 1: Default dialog box

```
menuLDA <-
function(formula, data, subset, na.omit.p=T,
         method="moment", newdata=NULL, predict.save=NULL,
         predict.method="plug-in", plot.p=F, plot.dimen=99)
{
  fun.call <- match.call()
  fun.call[[1]] <- as.name("lda.formula")
  if(na.omit.p) fun.call$na.action <- as.name("na.omit")
  else fun.call$na.action <- as.name("na.fail")
  fun.args <- is.element(arg.names(fun.call),
     c("formula", "data", "subset", "na.action", "method"))
  fun.call <- fun.call[c(T, fun.args)]
  ldaobj <- eval(fun.call)
  tabPredict.lda(ldaobj, newdata, predict.save, predict.method)
  if(plot.p) tabPlot.lda(ldaobj, plot.dimen=plot.dimen)
  invisible(ldaobj)
}

tabPredict.lda <-
function(object, newdata=NULL, predict.save=NULL,
         predict.method="plug-in")
{
  if(!is.null(predict.save)) {
    if(is.null(newdata))
      predobj <- predict(object, method=predict.method)
    else
      predobj <- predict(object, newdata=newdata,
                         method=predict.method)
    if(exists(predict.save, where = 1)) {
```



```
      newsave.name <- unique.name(predict.save, where = 1)
      assign(newsave.name, predobj, where = 1)
      warning(paste("Predictions saved in", newsave.name))
    } else assign(predict.save, predobj, where = 1)
  }
  invisible(NULL)
}

tabPlot.lda <- function(object, plot.dimen=99)
{
  plot.lda(object, dimen=plot.dimen)
  invisible(NULL)
}
```

A customized dialog box

```
guiCreate("FunctionInfo", Name = "menuLDA",
          Function = "menuLDA",
          HelpCommand = "help(lda)",
          DialogHeader = "Linear Discriminant Analysis",
          PropertyList = c("ldaModelPage", "ldaPlotPage",
            "ldaPredictPage", "SPropPFEnableButton"),
          CallBackFunction = "backLDA",
          ArgumentList = "#0=ldaSaveAs, #1=SPropPFFormula,
  #2=SPropDataFrameList, #3=SPropSubset, #4=SPropOmitMissing,
  #5=ldaFitMethod, #6=SPropPredictNewdata,
  #7=SPropSavePredictObject, #8=ldaPredictMethod,
  #9=ldaPlot, #10=ldaPlotDimen")

# pages

guiCreate("Property", Name="ldaModelPage", Type="Page",
          DialogPrompt="Model",
          PropertyList=c("ldaModelData", "SPropPFFormulaG",
            "ldaMethod", "ldaSaveModel"))
guiCreate("Property", Name="ldaPredictPage", Type="Page",
          DialogPrompt="Predict",
          PropertyList=c("SPropPredictNewdata", "ldaPredictMethod",
            "SPropSavePredictObject"))
guiCreate("Property", Name="ldaPlotPage", Type="Page",
          DialogPrompt="Plot",
          PropertyList=c("ldaPlot", "ldaPlotDimen"))

# groups

guiCreate("Property", Name="ldaModelData", Type="Group",
          DialogPrompt="Data",
          PropertyList=c("SPropDataFrameList", "SPropSubset",
```

```
            "SPropOmitMissing"))
guiCreate("Property", Name="ldaMethod", Type="Group",
          DialogPrompt="Fit method",
          PropertyList="ldaFitMethod")
guiCreate("Property", Name="ldaSaveModel", Type="Group",
          DialogPrompt="Save Model Object",
          PropertyList="ldaSaveAs")

# basic properties

guiCreate("Property", Name="ldaFitMethod",
          DialogPrompt="Method",
          DialogControl="List Box",
          OptionList=c("moment", "mle", "mve", "t"),
          DefaultValue="moment", UseQuotes=T)
guiCreate("Property", Name="ldaSaveAs", DialogPrompt="Save As",
          DialogControl="String", DefaultValue="last.lda")

guiCreate("Property", Name="ldaPredictMethod", DialogPrompt="Method",
          DialogControl="List Box",
          OptionList=c("plug-in", "predictive", "debiased"),
          DefaultValue="plug-in", UseQuotes=T)

guiCreate("Property", Name="ldaPlot", DialogPrompt="Plot LDs",
          DialogControl="Check Box", DefaultValue=F)
guiCreate("Property", Name="ldaPlotDimen", DialogPrompt="Dimen",
          DialogControl="Integer Spinner", DefaultValue=3, Disable=T)
```


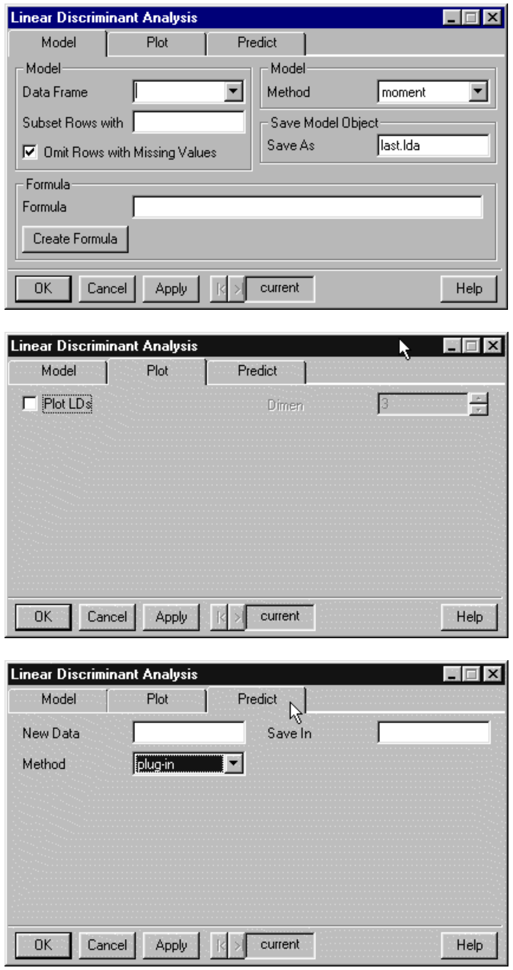

Figure 2: Customized dialog box showing all three pages. Note that the plot dimension is initially greyed-out.

```
# callback function

backLDA <- function(data)
{
  activeprop <- cbGetActiveProp(data)
  activevalue <- cbGetCurrValue(data, activeprop)
  switch(activeprop,
         ldaPlot = {
           if(activevalue == "F") {
             data <- cbSetEnableFlag(data, "ldaPlotDimen", F)
           } else {
             data <- cbSetEnableFlag(data, "ldaPlotDimen", T)
           }
         }
         ,
         SPropPFButton = {
           data <- backform1(data) # code copied from backLM
         }
         ,
         SPropPFEnableButton = {
           data <- cbSetEnableFlag(data, "SPropDataFrameList", T)
           data <- cbSetEnableFlag(data, "SPropPFFormula", T)
           data <- cbSetEnableFlag(data, "SPropPFButton", T)
         }
         )
  data
}

backform1 <- function(data)
{
  if(exists(cbGetCurrValue(data, "SPropDataFrameList"))) {
    guiModify(classname = "Property",
              Name = "SPropCFFormulaID",
              DialogControl = "Invisible",
              DefaultValue = cbGetDialogId(data))
    guiModify(classname = "Property",
              Name = "SPropCFData",
              DialogControl = "Invisible",
              DefaultValue = cbGetCurrValue(data,
                "SPropDataFrameList"))
    guiModify(classname = "Property",
              Name = "SPropCFFormula",
              DefaultValue = cbGetCurrValue(data, "SPropPFFormula"))
    guiDisplayDialog(classname = "Function", Name = "makeFormulaLm")
    data <- cbSetEnableFlag(data, "SPropDataFrameList", F)
    data <- cbSetEnableFlag(data, "SPropPFFormula", F)
    data <- cbSetEnableFlag(data, "SPropPFButton", F)
  } else guiCreate("MessageBox", String =
  "Data Frame doesn't exist. Enter Data Frame before building formula.")
}
```


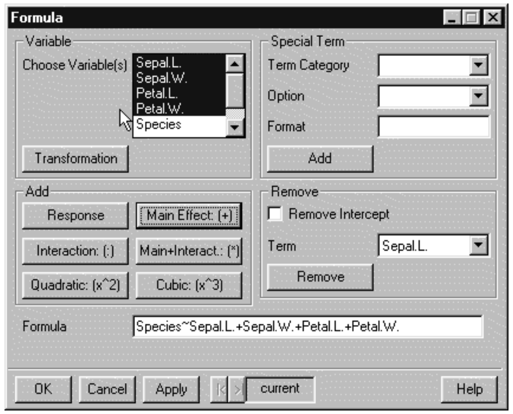

Figure 3: Formula-entry popup-up

```
# function to clean up

menuLDAPropDel <- function()
{
  if(!exists("guiRemove")) return(NULL)
  guiRemove("FunctionInfo", Name="menuLDA")
  guiRemove("Property", Name="ldaModelPage")
  guiRemove("Property", Name="ldaModelData")
  guiRemove("Property", Name="ldaMethod")
  guiRemove("Property", Name="ldaSaveModel")
  guiRemove("Property", Name="ldaFitMethod")
  guiRemove("Property", Name="ldaSaveAs")
    ....
}
```

Adding items to the menus

```
# To create a new menu

guiCreate("MenuItem",
          Name="SPlusMenuBar$MASS", Type="Menu",
          MenuItemText= "&MASS", Index=11, OverWrite=F)
guiCreate("MenuItem",
          Name="SPlusMenuBar$MASS$LDA", Type="MenuItem",
          Action="Function", Command="menuLDA",
          MenuItemText= "&LDA...")

# to remove this use
guiRemove("MenuItem", Name="SPlusMenuBar$MASS")

# or to add to Statistics menu:

guiCreate("MenuItem",
  Name="SPlusMenuBar$Statistics$Multivariate$Discrimination",
  Type="Menu", MenuItemText= "&Discrimination")
guiCreate("MenuItem",
  Name="SPlusMenuBar$Statistics$Multivariate$Discrimination$LDA",
  Type="MenuItem", Action="Function", Command="menuLDA",
  MenuItemText= "&LDA...")
guiCreate("MenuItem",
  Name="SPlusMenuBar$Statistics$Multivariate$Discrimination$QDA",
  Type="MenuItem", Action="Function", Command="menuQDA",
  MenuItemText= "&QDA...")
# to remove this use
guiRemove("MenuItem",
  Name="SPlusMenuBar$Statistics$Multivariate$Discrimination")


# Context-sensitive menus

guiCreate("MenuItem", Name="lda", Type="Menu",
          DocumentType="lda")
guiCreate("MenuItem", Name="lda$print", Type="MenuItem",
          DocumentType="lda", Action="Function",
          Command="tabResults.lda", MenuItemText="Print",
          ShowDialogOnRun=F)
guiCreate("MenuItem", Name="lda$plot", Type="MenuItem",
          DocumentType="lda", Action="Function",
          Command="tabPlot.lda", MenuItemText="Plot...",
          ShowDialogOnRun=T)
guiCreate("MenuItem", Name="lda$predict", Type="MenuItem",
          DocumentType="lda", Action="Function",
          Command="tabPredict.lda", MenuItemText="Predict...",
          ShowDialogOnRun=T)
```



```
guiCreate("ClassInfo", Name="lda", ContextMenu="lda",
          DoubleClickAction = "tabResults.lda")

tabResults.lda <- function(object) invisible(print.lda(object))
guiCreate("FunctionInfo", Name = "tabPlot.lda",
          Function = "tabPlot.lda",
          DialogHeader = "LDA plot",
          PropertyList = c("SPropInvisibleReturnObject",
            "SPropCurrentObject","ldaPlot2Page",
            "SPropPFEnableButton"),
          ArgumentList = "#0=SPropInvisibleReturnObject,
                   #1=SPropCurrentObject, #2=ldaPlotDimen2")
guiCreate("Property", Name="ldaPlot2Page", Type="Page",
          DialogPrompt="Plot",
          PropertyList="ldaPlotDimen2")
guiCreate("Property", Name="ldaPlotDimen2", DialogPrompt="Dimen",
          DialogControl="Integer Spinner", DefaultValue=3)

guiCreate("FunctionInfo", Name = "tabPredict.lda",
          Function = "tabPredict.lda",
          DialogHeader = "LDA prediction",
          PropertyList = c("SPropInvisibleReturnObject",
          "SPropCurrentObject","ldaPredictPage",
            "SPropPFEnableButton"),
          ArgumentList = "#0=SPropInvisibleReturnObject,
            #1=SPropCurrentObject, #2=SPropPredictNewdata,
            #3=SPropSavePredictObject, #4=ldaPredictMethod")
```