Journal of Statistical Software

June 2012, Volume 49, Issue 7. *http://www.jstatsoft.org/*

The RcmdrPlugin.survival Package: Extending the R Commander Interface to Survival Analysis

John Fox
McMaster University

Marilia Sá Carvalho
Escola Nacional de Saúde Pública

Abstract

The R Commander graphical user interface to R is extensible via plug-in packages, which integrate seamlessly with the R Commander's menu structure, data, and model handling. The paper describes the **RcmdrPlugin.survival** package, which makes many of the facilities of the **survival** package for R available through the R Commander, including Cox and parametric survival models. We explain the structure, capabilities, and limitations of this plug-in package and illustrate its use.

Keywords: graphical user interface, **survival** package, R.

1. Introduction

This paper describes the **RcmdrPlugin.survival** package, which augments the **Rcmdr** ("R Commander") package (Fox 2005, 2007) to provide a graphical user interface (GUI) to many of the facilities of the **survival** package for R (Therneau 2012; Therneau and Grambsch 2000). The initial impetus for developing a survival-analysis plug-in for the R Commander came from a desire to introduce Brazilian medical researchers gently to the powerful facilities of the **survival** package (as discussed in Carvalho, Andreozzi, Codeço, Barbosa, Serrano, and Shimakura 2005). We anticipate that the capabilities of the **RcmdrPlugin.survival** package will grow, and hope that it may provide at least some researchers with a bridge to writing their own R commands.

After presenting a brief overview of the R Commander in Section 2 of the paper, meant to orient readers who have not previously encountered the **Rcmdr** package, we describe the use of the survival-analysis plug-in in Section 3. This section of the paper serves the dual function of furnishing a basic manual for the **RcmdrPlugin.survival** package and of establishing a basis for a discussion of the design and implementation of the **RcmdrPlugin.survival** package in Section 4. In Section 5 of the paper, we reflect on the challenges of designing a graphical interface

for survival analysis and on the consequent limitations of the **RcmdrPlugin.survival** package. The **Rcmdr** and **RcmdrPlugin.survival** packages are both available on the Comprehensive R Archive Network (CRAN) at `http://CRAN.R-project.org/`.

2. A brief overview of the R Commander and plug-ins

The R Commander (**Rcmdr**) package (Fox 2005) was originally conceived as a basic-statistics graphical user interface to R (R Development Core Team 2012a), suitable for supporting a first course in statistics taught with a text such as Moore (2010). The **Rcmdr** quickly expanded to include facilities for fitting, checking, and displaying linear and generalized linear models, along with some other methods, such as exploratory factor analysis, that are not usually

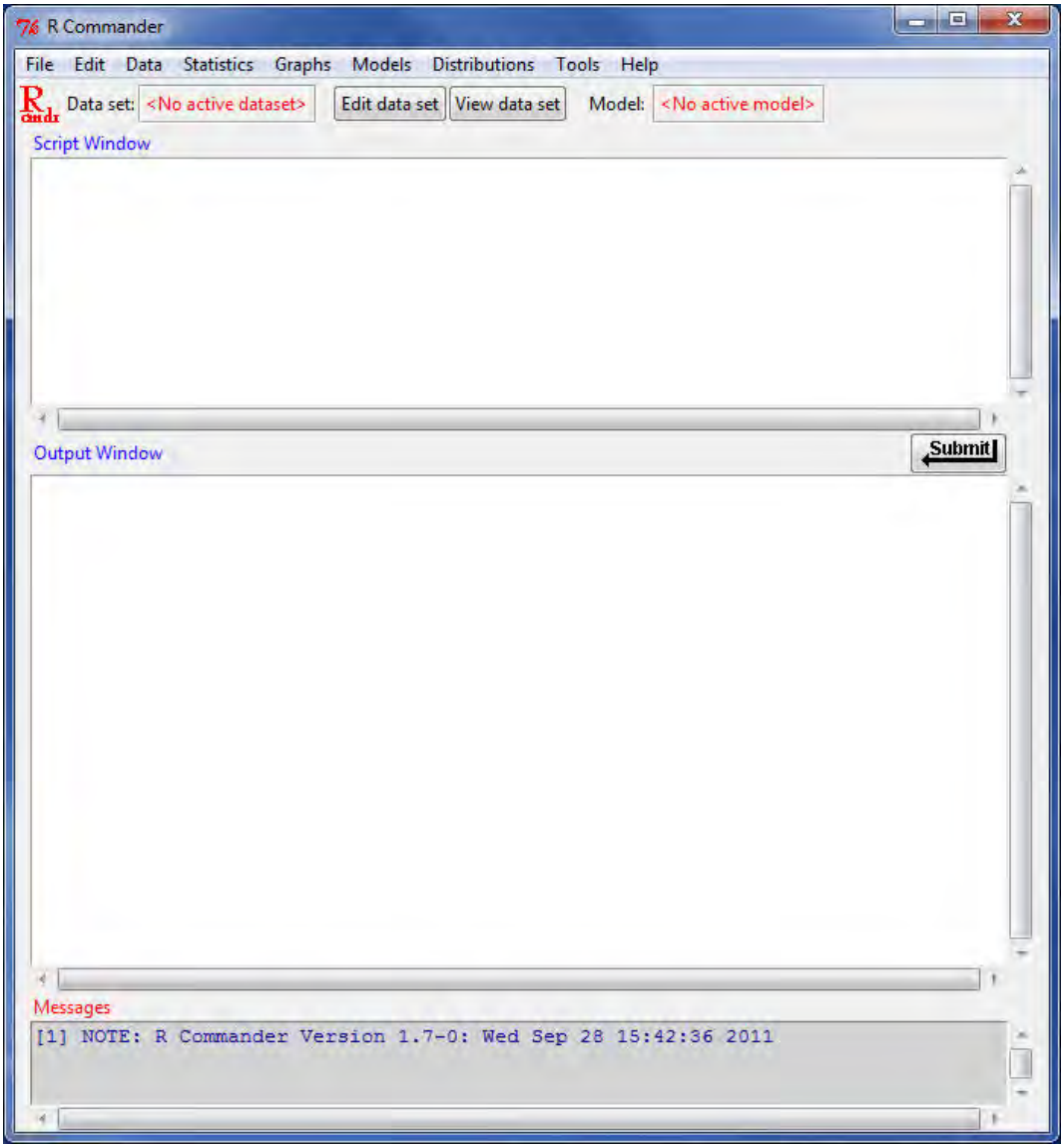

Figure 1: The **Rcmdr** interface at start-up.

part of the basic-statistics curriculum. Although its menu and dialog structure were always designed to be extensible, extending the **Rcmdr** became much more convenient with the introduction of plug-in packages, described in Fox (2007), and as we write this, 30 **Rcmdr** plug-ins are available on CRAN.

An original and continuing goal of the R Commander is to provide a simple-to-use and easy-to-install GUI for R that works on all of the major computing platforms on which R runs—Windows, Mac OS X, and Linux/Unix systems. To this end, the R Commander is implemented in Tcl/Tk (Welch and Jones 2003), an R interface to which is furnished by the **tcltk** package (R Development Core Team 2012a; Dalgaard 2001, 2002), which is included in the standard R distribution. A working Tcl/Tk compatible with the **tcltk** package is also part of the standard R installation for Windows and is usually included in Linux/Unix systems, so all that is required to get started with the **Rcmdr** interface is to install the package and its dependencies. Under Mac OS X, however, it is first necessary to install an X Windows version of Tcl/Tk compatible with the **tcltk** package.

The R Commander interface is shown in the "screen shot" in Figure 1, taken on a Windows 7 system. The interface comprises a menu bar, a toolbar, and three panels or windows—from top to bottom, a *script window* into which the **Rcmdr** writes the commands that it generates and executes; an *output window*, which shows commands and the associated printed output produced by them; and a *messages window*, which displays warnings, error messages, and other ancillary information. The script window is a rudimentary script editor, allowing the user to save and load scripts, and to edit, execute, and re-execute commands. Commands are invoked almost exclusively through the menus and the dialog boxes to which most menu items lead. This point-and-click interface is meant to be familiar and largely self-explanatory: Many of the menus have names common to most applications, such as `File`, `Edit`, `Tools`, and `Help`, while the others are natural for statistical applications, `Data`, `Statistics`, `Graphs`, `Models`, and `Distributions`.

The toolbar directly below the menu bar shows the name of the "active data set"; includes buttons for editing and displaying the active data set; and shows the name of the active statistical model. Data sets in the **Rcmdr** are R data frames. Most commands invoked via the **Rcmdr** menus are applied to the active data set, and while several data sets may be present in memory, only one is active at any given time. The user can change the active data set either by reading a new data set into memory via menu items under the `Data` menu (e.g., `Data` → `Import data` → `from text file, clipboard, or URL...`) or by selecting a data set already in memory by pressing the active data set button in the toolbar. When a statistical modeling function, such as `lm` or `glm`, creates a new statistical model object, that object becomes the "active model," to which commands generated via the `Models` menu apply. Statistical models are associated with particular data sets, and if more than one model is associated with the currently active data set, the user may choose among the models via the active model button in the toolbar.

3. Using the RcmdrPlugin.survival package

As is typical of **Rcmdr** plug-ins, the **RcmdrPlugin.survival** package can either be loaded directly, by the command `library("RcmdrPlugin.survival")`, or, with the **Rcmdr** running, via `Tools` → `Load Rcmdr plug-in(s)...`. The first part of the package name, **Rcmdr-**

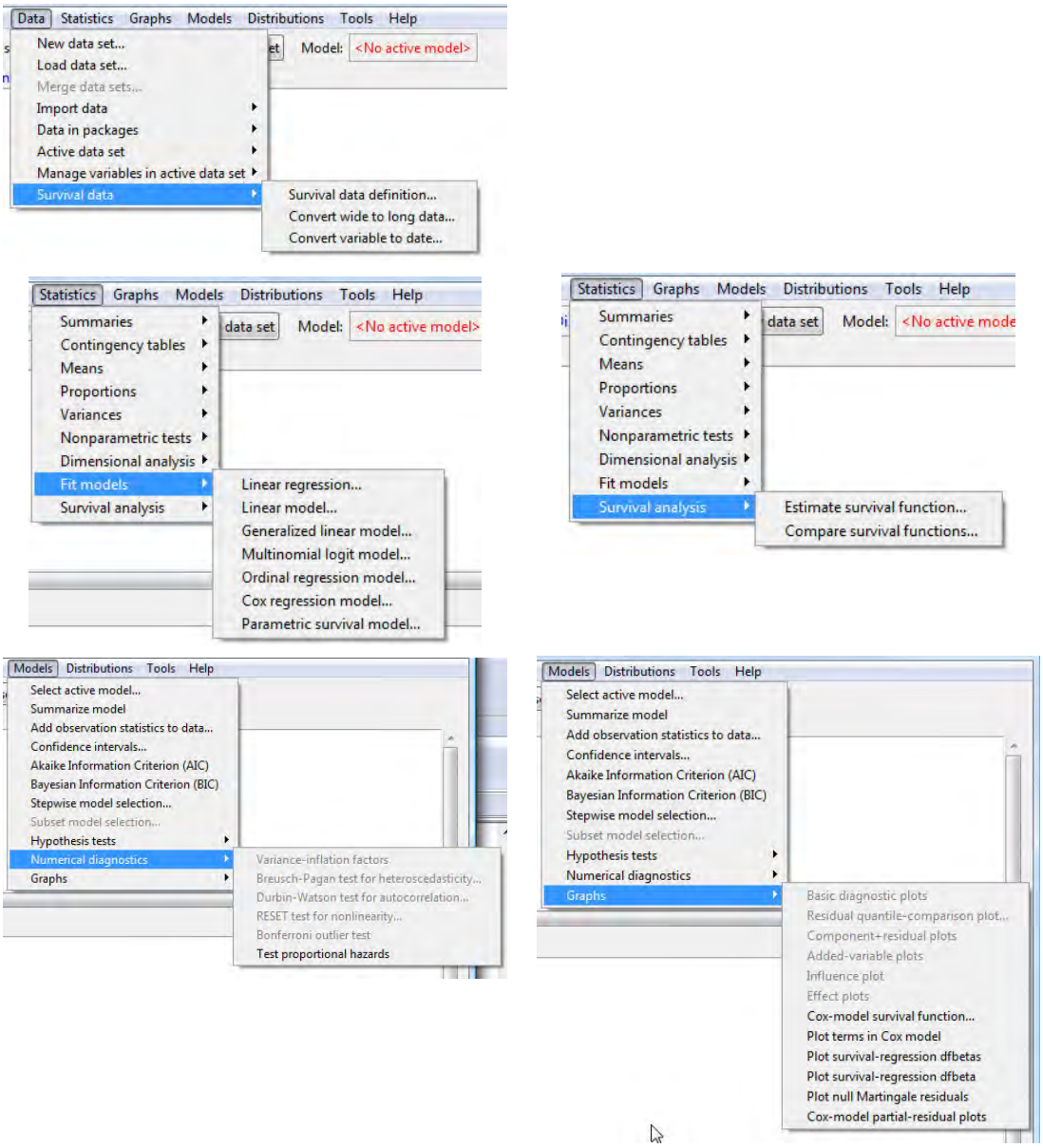

Figure 2: Menus and menu items added to the standard R Commander menus by the **RcmdrPlugin.survival** package.

Plugin., is conventional for an **Rcmdr** plug-in package, and insures that plug-ins sort directly after the **Rcmdr** on CRAN.

As illustrated in Figure 2, the **RcmdrPlugin.survival** package adds a variety of menus and menu items to the R Commander interface:

- A `Survival data` sub-menu is added to the `Data` menu, containing items for defining characteristics of survival data (such as time variables and an event indicator); for converting data sets from "wide" to "long" form, when data records may include more

than one period of observation for each individual; and for converting character data to date objects.

- The `Statistics` → `Fit models` menu acquires menu items for Cox regression models and for parametric survival models.
- A `Survival analysis` sub-menu is added to the `Statistics` menu, with items for estimating and comparing survival functions.
- A test for proportional hazards in the Cox model is added to `Models` → `Numerical diagnostics`.
- Several new items are added to the `Models` → `Graphs` menu, to graph survival functions based on a fitted Cox model; to plot terms in a Cox model; to plot dfbeta and dfbetas (influence diagnostics) for a survival regression model; to plot Martingale residuals for a survival regression; and to produce partial-residual plots for a Cox model.

As we will illustrate presently, typical work-flow in analyzing survival data follows several steps:

1. We begin by reading the data via an appropriate selection in the `Data` menu.
2. We may transform the data via the `Data` → `Active data set` and `Data` → `Manage variables in active data set` menus, and if necessary convert character data to dates using `Data` → `Survival data` → `Convert variable to date...`
3. We typically define time and event variables for the survival data set, and perhaps also define strata or clusters, with `Data` → `Survival data` → `Survival data definition....`
4. We may change the format of the data set via `Data` → `Survival data` → `Convert wide to long data....`
5. Preliminary examination of the data typically follows, using `Statistics` → `Survival analysis` → `Estimate survival function...`, and maybe `Statistics` → `Survival analysis` → `Compare survival functions....`
6. Regression modeling usually comes next, almost always with Cox models fit via `Statistics` → `Fit models` → `Cox regression model...`, but possibly with `Statistics` → `Fit models` → `Parametric survival model....`
7. A variety of standard diagnostics are available in the `Models` → `Numerical diagnostics` and `Models` → `Graphs` menus for checking the adequacy of the fitted model.
8. Finally, the `Models` → `Graphs` and `Models` → `Hypothesis tests` menus contain items for displaying and testing survival regression models.

3.1. Example: Brazilian hemodialysis data

As a more or less typical example of survival analysis, we employ, in simplified form, a data set analyzed by Carvalho, Henderson, Shimakura, and Sousa (2003), with data on all patients

undergoing hemodialysis treatment in publicly funded clinics in Rio de Janeiro State, Brazil, between January 1998 (month 1 of the study) and August 2000 (month 44). The principal purpose of this and the following example is to illustrate how the **RcmdrPlugin.survival** package is used, not to undertake a serious analysis of the data, and we also assume general familiarity with methods of survival analysis (as described, e.g., in Therneau and Grambsch 2000). The `Dialysis` data set is included in the **RcmdrPlugin.survival** package, and comprises 6805 patients and seven variables:

- `center`, a numeric code indicating in which of 67 medical centers the patient was treated. Although not strictly necessary, we find it convenient to convert this variable into a factor via `Data` → `Manage variables in active data set` → `Convert numeric variables to factors...`, selecting the option to use numbers as factor levels (dialog box not shown).
- The `age` of the patient, in years at entry into the study.
- `begin`, the month in which treatment began, coded `1` for patients already in treatment at the start of the study.
- `end`, the month in which observation terminated, either because of death or censoring.
- `time` under observation, that is `end` − `begin`.
- `event`, a numeric event indicator, coded `1` if the patient died while under observation or `0` if censored.
- The `disease` causing kidney failure requiring dialysis, coded `hypert` (hypertension), `congen` (congenital), `diabetes`, `renal`, and `other`. As is the default in R, these five factor levels are initially ordered alphabetically, and are put in the order recorded here using `Data` → `Manage variables in active data set` → `Reorder factor levels...`. Although reordering the levels of `disease` isn't strictly necessary, hypertension is the most common diagnosis in the data set, and one with a moderate risk of death, so we find it convenient to select this as the baseline level for dummy-coded contrasts used in the statistical modeling reported below; we want `other` as the last level for esthetic reasons; and the remaining three levels are given in alphabetical order.

We read the data via `Data` → `Data in packages` → `Read data from an attached package...`, completing the resulting dialog, as shown in Figure 3. A brief overview of the data set is provided by selecting `Statistics` → `Summaries` → `Active data set` from the R Commander menus, producing the following results in the output window, which also displays the commands that were generated to read and reorganize the data:[1]

```
> data(Dialysis, package="RcmdrPlugin.survival")
> Dialysis$center <- as.factor(Dialysis$center)
> Dialysis$disease <- factor(Dialysis$disease, levels=c('hypert',
+   'congen','diabetes','renal','other'))
> summary(Dialysis)
```

[1]The R Commander output window shows commands in red and the resulting output in dark blue. In this paper, we use typewriter italics for commands in the output window.

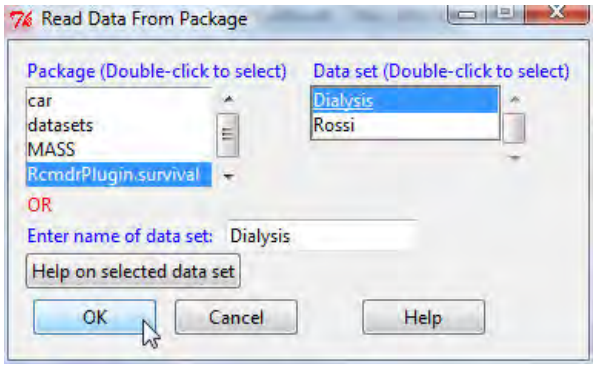

Figure 3: Reading the `Dialysis` data from the **RcmdrPlugin.survival** package.

```
    center          age            begin            end
2844   : 241   Min.   : 0.0   Min.   : 1.00   Min.   : 2.00
562    : 227   1st Qu.:42.0   1st Qu.:12.00   1st Qu.:31.00
561    : 222   Median :53.0   Median :23.00   Median :44.00
1029   : 218   Mean   :52.7   Mean   :22.78   Mean   :36.94
3064   : 184   3rd Qu.:65.0   3rd Qu.:33.00   3rd Qu.:45.00
1159   : 183   Max.   :97.0   Max.   :44.00   Max.   :45.00
(Other):5530
     event           time          disease
Min.   :0.0000   Min.   : 1.00   hypert  :2836
1st Qu.:0.0000   1st Qu.: 3.00   congen  : 142
Median :0.0000   Median :11.00   diabetes:1283
Mean   :0.2356   Mean   :14.16   renal   :1414
3rd Qu.:0.0000   3rd Qu.:22.00   other   :1130
Max.   :1.0000   Max.   :44.00
```

Before analyzing the `Dialysis` data, we must address the following issue: The natural time origin for patients in the study is the beginning of treatment, and for patients who entered the study between the 1st and 44th months, this is captured by the `time` variable in the data set. Some patients—so-called "prevalent cases"—began treatment before the beginning of the study, however. If we knew how many months of treatment had occurred before the beginning of the study, we could accommodate these patients by employing the counting-process approach to survival analysis, but in the data set we are unable to distinguish between patients who entered treatment in month 1 of the study and those who were already being treated in month 1. Consequently, we employ the R Commander subset dialog to remove these cases, `Data` → `Active data set` → `Subset active data set...` (Figure 4), leaving us with 6665 of the original 6805 cases.[2]

The **RcmdrPlugin.survival** package permits us to define persistent time and event variables, possibly along with strata and clusters, via `Data` → `Survival data` → `Survival data definition...`, producing the dialog box in Figure 5. The information is stored as attributes

[2]The original paper, Carvalho *et al.* (2003), employed a more complete version of the data set, including information on when prevalent patients began treatment, permitting their inclusion in the analysis employing the counting-process approach.

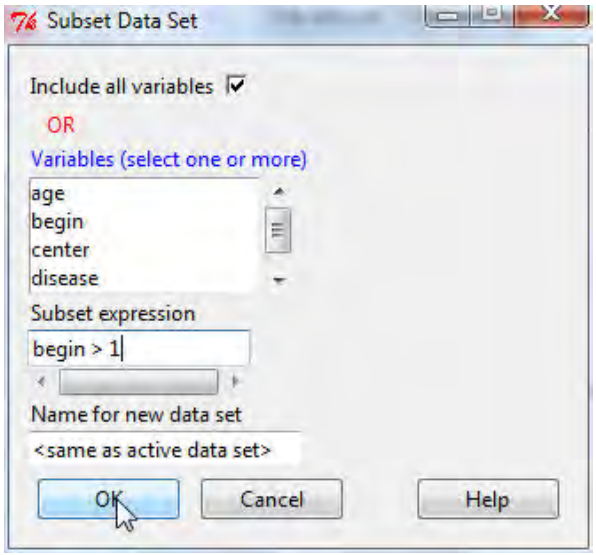

Figure 4: Removing patients from the `Dialysis` data who may have begun treatment before the start of the study.

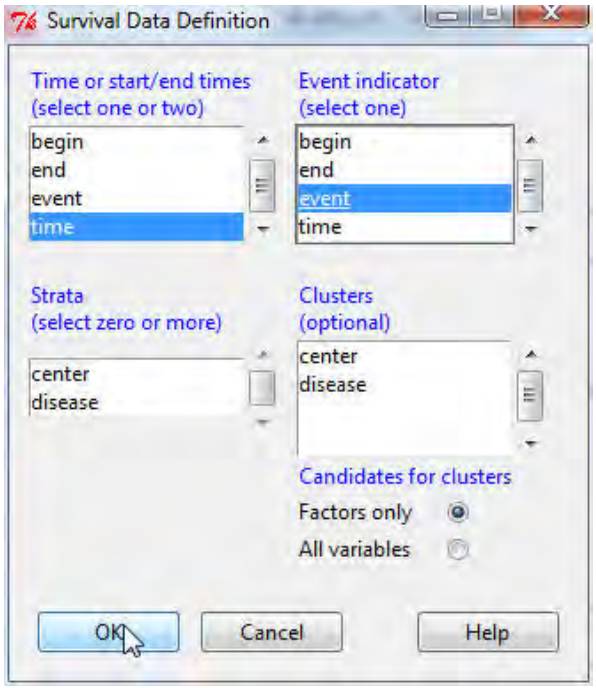

Figure 5: Defining time and event variables for the `Dialysis` data.

of the `Dialysis` data frame. Where appropriate, these selections can also be made or modified in individual dialog boxes for estimating survival functions, fitting survival regression models, etcetera. We find it convenient in the current case to define time and event variables (literally the variables `time` and `event`).

We proceed to estimate the survival function for the data set as a whole, using `Statistics`

→ Survival analysis → Estimate survival function..., leading to the dialog box in Figure 6, and producing the following output, along with a graph of the estimated estimated survival function, shown in Figure 7:

```
> .Survfit <- survfit(Surv(time, event) ~ 1, conf.type="log",
+   conf.int=0.95, type="kaplan-meier", error="greenwood",
+   data=Dialysis)

> summary(.Survfit)

Call: survfit(formula = Surv(time, event) ~ 1, data = Dialysis,
    conf.type = "log", conf.int = 0.95, type = "kaplan-meier",
    error = "greenwood")

 time n.risk n.event survival std.err lower 95% CI upper 95% CI
    1   6665     228    0.966 0.00223        0.961        0.970
    2   6035     245    0.927 0.00325        0.920        0.933
    3   5378     162    0.899 0.00382        0.891        0.906
    4   4942     109    0.879 0.00418        0.871        0.887
    5   4653      87    0.862 0.00446        0.854        0.871
    . . .
   40    224       2    0.599 0.01155        0.577        0.622
   41    166       2    0.592 0.01248        0.568        0.617

> plot(.Survfit, mark.time=TRUE)

> quantile(.Survfit, quantiles=c(.5,.6,.7,.8,.9))

               Survival Probability
Estimate         0.5 0.6 0.7 0.8 0.9
  lower 0.95 CL  NA  37  23  10   3
  quantile       NA  40  24  11   3
  upper 0.95 CL  NA  NA  27  12   4

> remove(.Survfit)
```

The widely spaced ellipses (. . .) in the output represent lines omitted for brevity. We knew that fewer than half the patients died during the study, and so requested estimates of quantiles above 0.5, including the 0.5 quantile to show that it is not estimable. The `quantile` method for `survfit` objects is not part of the **survival** package, but is supplied by **RcmdrPlugin.survival**.

We continue by comparing survival across levels of `disease`, via Statistics → Survival analysis → Compare survival functions..., leading to the dialog box in Figure 8, and producing the following output:

```
> survdiff(Surv(time,event) ~ disease, rho=0, data=Dialysis)
```

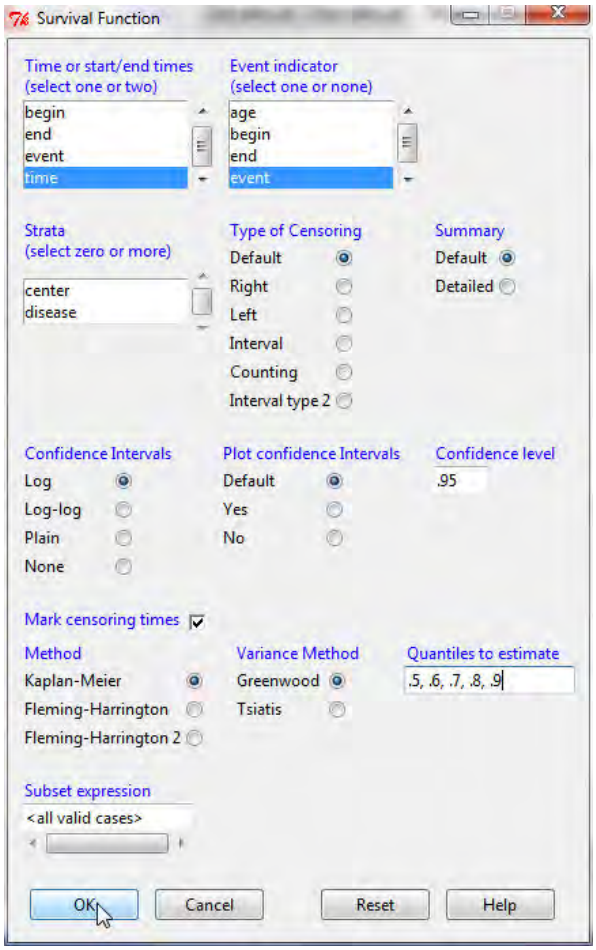

Figure 6: Dialog box for estimating a survival function. The `time` and `event` variables are pre-selected, and the default quantiles (`.25, .5, .75`) are replaced with `.5, .6, .7, .8, .9`; all other selections are defaults.

```
Call:
survdiff(formula = Surv(time, event) ~ disease, data = Dialysis,
    rho = 0)

                    N Observed Expected (O-E)^2/E (O-E)^2/V
disease=hypert   2782      633    674.4      2.54      4.57
disease=congen    139       19     38.6      9.93     10.41
disease=diabetes 1268      398    279.5     50.21     62.69
disease=renal    1378      289    325.7      4.14      5.35
disease=other    1098      224    244.8      1.77      2.15

 Chisq= 70.3  on 4 degrees of freedom, p= 1.95e-14
```

Differences in survival across the levels of `disease` are clearly highly statistically significant. We graph the estimated survival functions by `disease` strata using `Statistics` → `Survival analysis` → `Estimate survival function...` (dialog not shown), with the re-

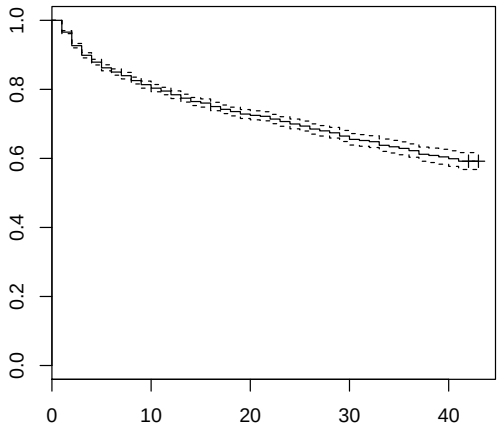

Figure 7: Estimated survival function for the `Dialysis` data. The broken lines give a 95% pointwise confidence envelope around the estimate.

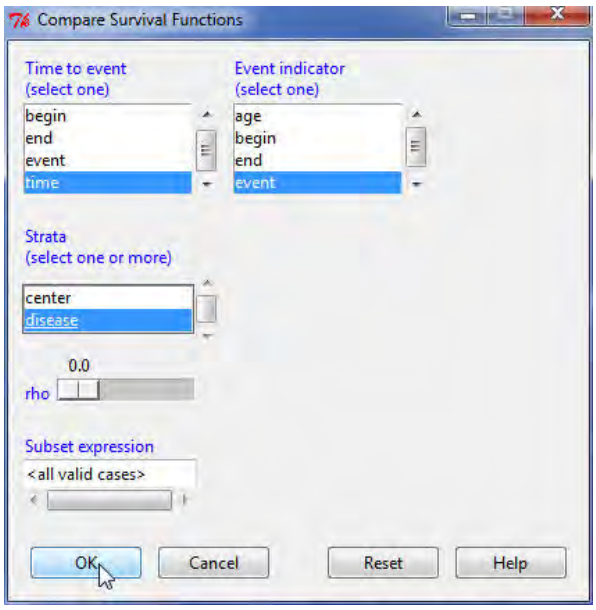

Figure 8: Dialog box for comparing survival functions across strata; `disease` is selected in the strata list box.

sult displayed in Figure 9. Survival is highest for those with congenital disease and lowest for those with diabetes.

Our next step is to fit a Cox model to the data, treating `age` and `disease` as covariates, `Statistics` → `Fit models` → `Cox regression model...`, which produces the dialog in Figure 10, and the following output:

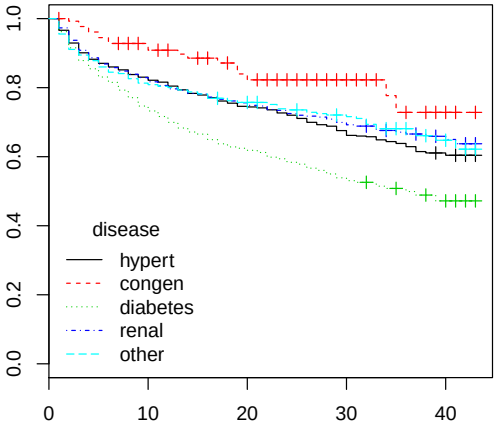

Figure 9: Estimated survival functions by `disease`.

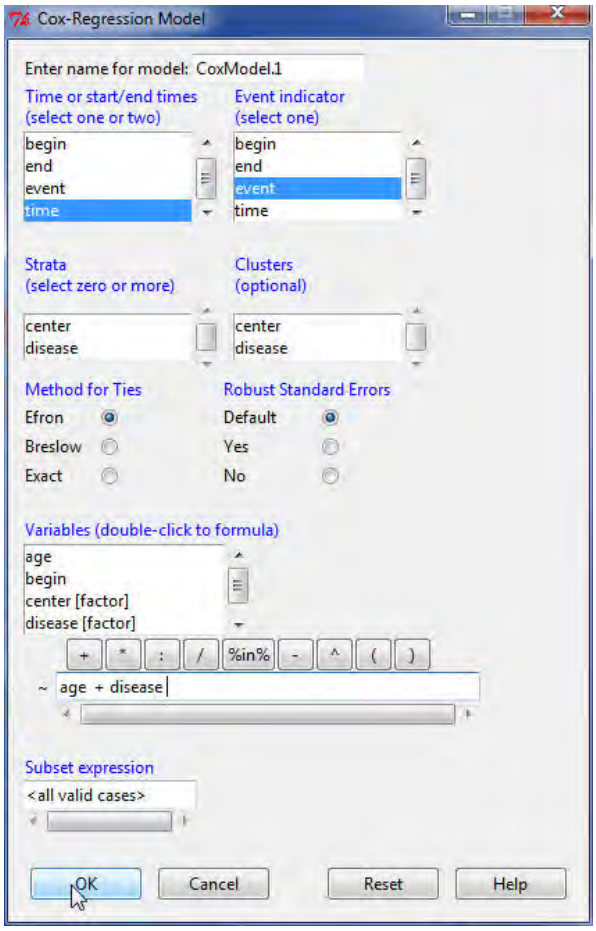

Figure 10: Cox regression dialog box.

```
> CoxModel.1 <- coxph(Surv(time,event) ~ age + disease,
+   method="efron", data=Dialysis)

> summary(CoxModel.1)

Call:
coxph(formula = Surv(time, event) ~ age + disease, data = Dialysis,
    method = "efron")

  n= 6665, number of events= 1563

                         coef exp(coef)  se(coef)      z Pr(>|z|)
age                  0.034463  1.035064  0.001775 19.416  < 2e-16 ***
disease[T.congen]   -0.724699  0.484470  0.232867 -3.112  0.00186 **
disease[T.diabetes]  0.293269  1.340803  0.064117  4.574 4.79e-06 ***
disease[T.renal]     0.042893  1.043826  0.071048  0.604  0.54603
disease[T.other]     0.028907  1.029329  0.077780  0.372  0.71015
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

                    exp(coef) exp(-coef) lower .95 upper .95
age                    1.0351     0.9661    1.0315    1.0387
disease[T.congen]      0.4845     2.0641    0.3069    0.7647
disease[T.diabetes]    1.3408     0.7458    1.1825    1.5203
disease[T.renal]       1.0438     0.9580    0.9081    1.1998
disease[T.other]       1.0293     0.9715    0.8838    1.1988

Rsquare= 0.067   (max possible= 0.979 )
Likelihood ratio test= 463.1  on 5 df,   p=0
Wald test            = 428.2  on 5 df,   p=0
Score (logrank) test = 441  on 5 df,   p=0
```

Thus, holding disease constant, each additional year of age increases the hazard of death by 3.5%; similarly, holding age constant, patients with congenital disease are substantially less likely to die and patients with diabetes substantially more likely to die than those with hypertension (the baseline level for `disease`). Partial-likelihood-ratio tests for the terms in the model are obtained from `Models` → `Hypothesis tests` → `ANOVA table...` (dialog box not shown, defaults taken), which employs the `Anova` function in the **car** package (Fox and Weisberg 2011):

```
> Anova(CoxModel.1, type="II")

Analysis of Deviance Table (Type II tests)
        LR Chisq Df Pr(>Chisq)
age       396.35  1  < 2.2e-16 ***
disease    37.99  4  1.127e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

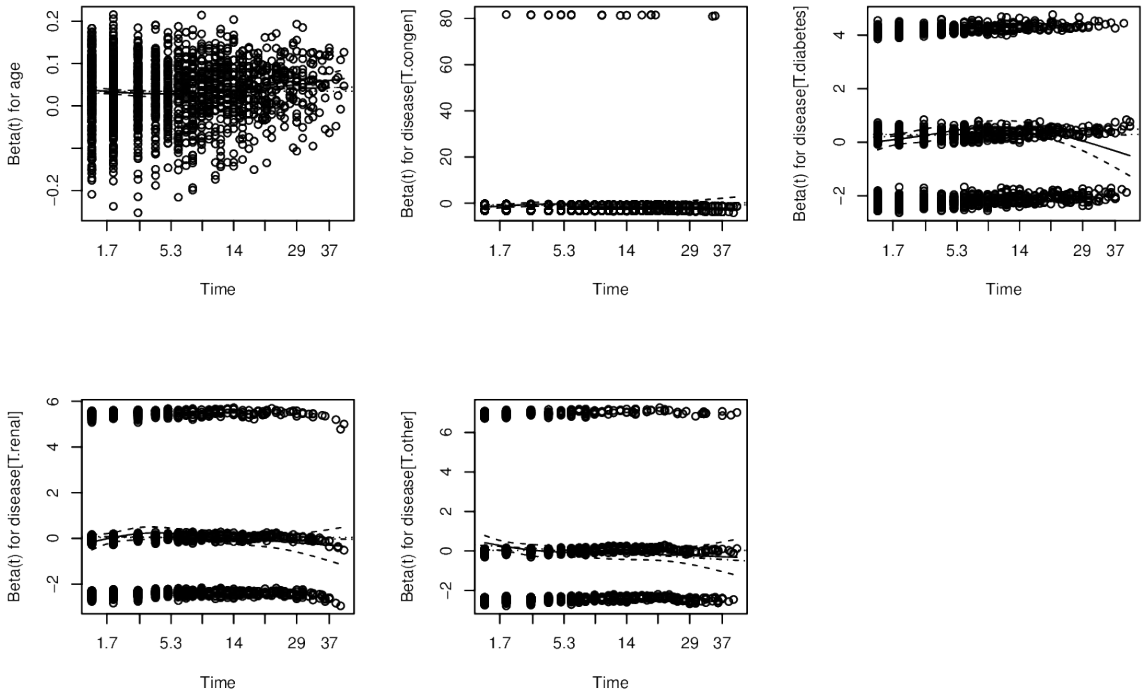

Figure 11: Plots of Schoenfeld residuals for each covariate in the model. The dotted horizontal line is drawn in each panel at the estimated coefficient for the corresponding covariate; the dot-dashed line is a least-squares fit, corresponding to the test for proportional hazards; the solid line is a nonparametric-regression smooth, which can detect more general forms of non-proportional hazards; and the broken lines show a 95% pointwise confidence envelope around the smooth.

Thus, both the `age` and `disease` terms in the Cox model are highly statistically significant.[3]

Having fit a Cox regression model to the `Dialysis` data, we would like to know whether the model adequately represents the data, and consequently we follow up with several diagnostics. Selecting `Models` → `Numerical diagnostics` → `Test proportional hazards` tests the proportional-hazards assumption of the Cox model and produces graphs of Schoenfeld residuals against time for each covariate (Figure 11):

```
                        rho  chisq       p
age                  0.0487  4.045 0.04431
disease[T.congen]    0.0404  2.549 0.11039
disease[T.diabetes]  0.0278  1.219 0.26952
disease[T.renal]    -0.0102  0.163 0.68635
disease[T.other]    -0.0626  6.105 0.01348
GLOBAL                   NA 17.976 0.00298
```

[3]There is some evidence for a `disease`-by-`age` interaction, which we will not pursue here.

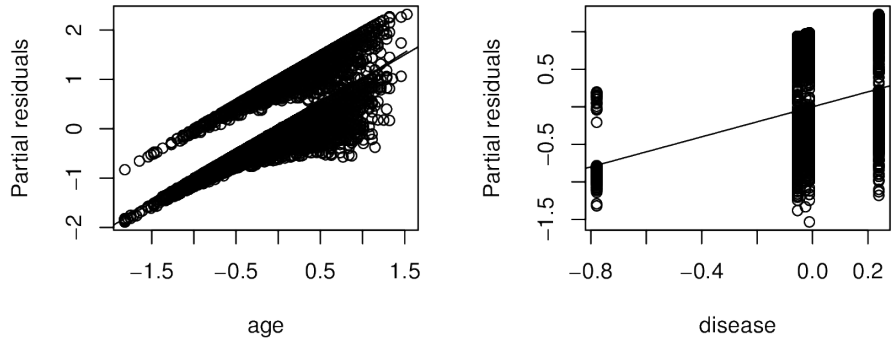

Figure 12: Partial-residual plots for the Cox model fit to the `Dialysis` data. The lines drawn on the plot are for a nonparametric-regression smooth and a linear least-squares fit.

There is, therefore, some evidence of non-proportional hazards, which we will not pursue here.[4]

Continuing with the diagnostics, partial-residual plots, produced by `Models` → `Graphs` → `Cox model partial-residual plots` and shown in Figure 12, suggest that the functional form of the model is reasonable; and index plots of dfbetas, produced by `Models` → `Graphs` → `Plot survival-regression dfbetas` and shown in Figure 13, suggest that none of the observations are unduly influential, as is to be expected in a data set this large.

The functional form of the Cox regression can also be checked by plotting Martingale residuals from a null model against the regressors, obtained by `Models` → `Graphs` → `Plot null Martingale residuals`, and producing the graphs shown in Figure 14. The plots for the dummy variables are not informative, of course, but the plot for `age` reveals a monotone and apparently nonlinear relationship between the log-hazard and `age`, with the slope increasing with `age`. We could deal with this nonlinearity by transforming `age` or by using a low-degree-of-freedom regression spline in `age`.

Finally, we consider modeling differences among the centers as a random effect, by entering `center` as a frailty term in the model. To do this, we type the term `frailty(center)` directly into the model formula box in the Cox-model dialog, as shown in Figure 15, producing the following output:

```
> CoxModel.2 <- coxph(Surv(time,event) ~ age + disease +
+   frailty(center), method="efron", data=Dialysis)

> summary(CoxModel.2)

Call:
coxph(formula = Surv(time, event) ~ age + disease + frailty(center),
    data = Dialysis, method = "efron")
```

[4]The evidence for non-proportional hazards is greatly reduced if we include interactions between `disease` and `age`.

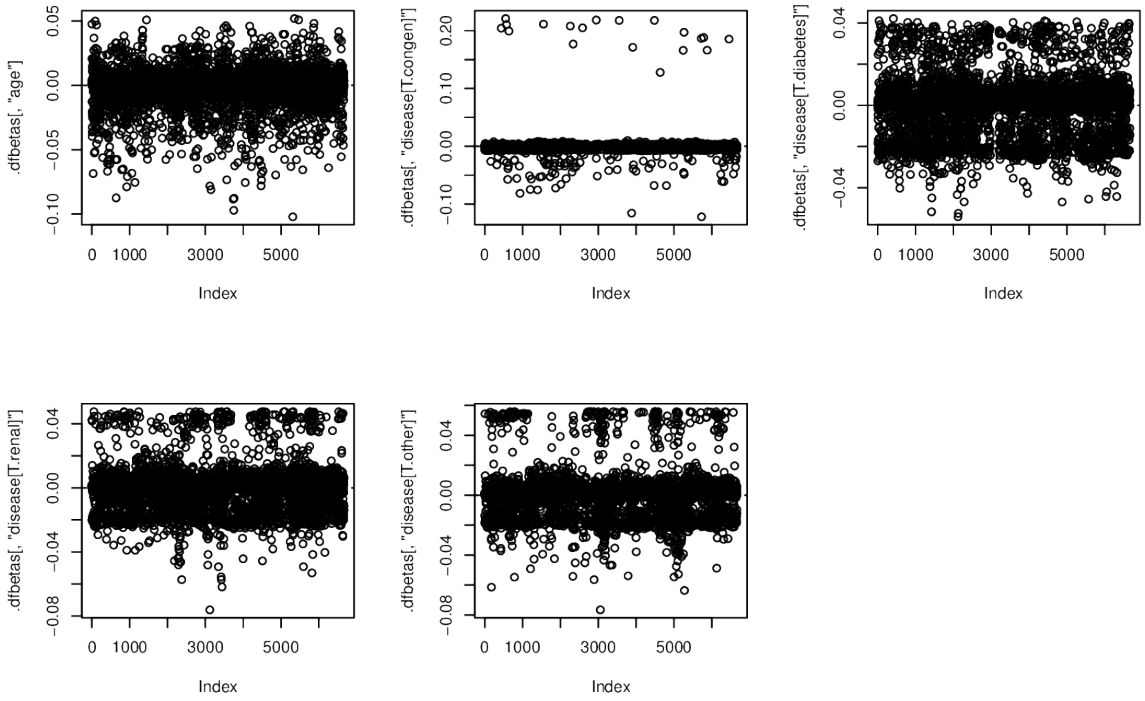

Figure 13: Index plots of dfbetas for the Cox model fit to the `Dialysis` data.

```
  n= 6665
                     coef    se(coef) se2     Chisq  DF   p
age                  0.0351 0.00182  0.00182 370.75  1.0 0.0e+00
disease[T.congen]   -0.6531 0.23440  0.23426   7.76  1.0 5.3e-03
disease[T.diabetes]  0.3844 0.06726  0.06706  32.66  1.0 1.1e-08
disease[T.renal]    -0.0312 0.07727  0.07680   0.16  1.0 6.9e-01
disease[T.other]     0.1344 0.08885  0.08807   2.29  1.0 1.3e-01
frailty(center)                              794.42 60.8 0.0e+00

                    exp(coef) exp(-coef) lower .95 upper .95
age                      1.04      0.966     1.032     1.039
disease[T.congen]        0.52      1.921     0.329     0.824
disease[T.diabetes]      1.47      0.681     1.287     1.676
disease[T.renal]         0.97      1.032     0.833     1.128
disease[T.other]         1.14      0.874     0.961     1.361

Iterations: 10 outer, 34 Newton-Raphson
     Variance of random effect= 0.695   I-likelihood = -12401.2
Degrees of freedom for terms=  1.0  4.0 60.8
Rsquare= 0.167   (max possible= 0.979 )
Likelihood ratio test= 1222  on 65.7 df,   p=0
Wald test            = 442  on 65.7 df,   p=0
```

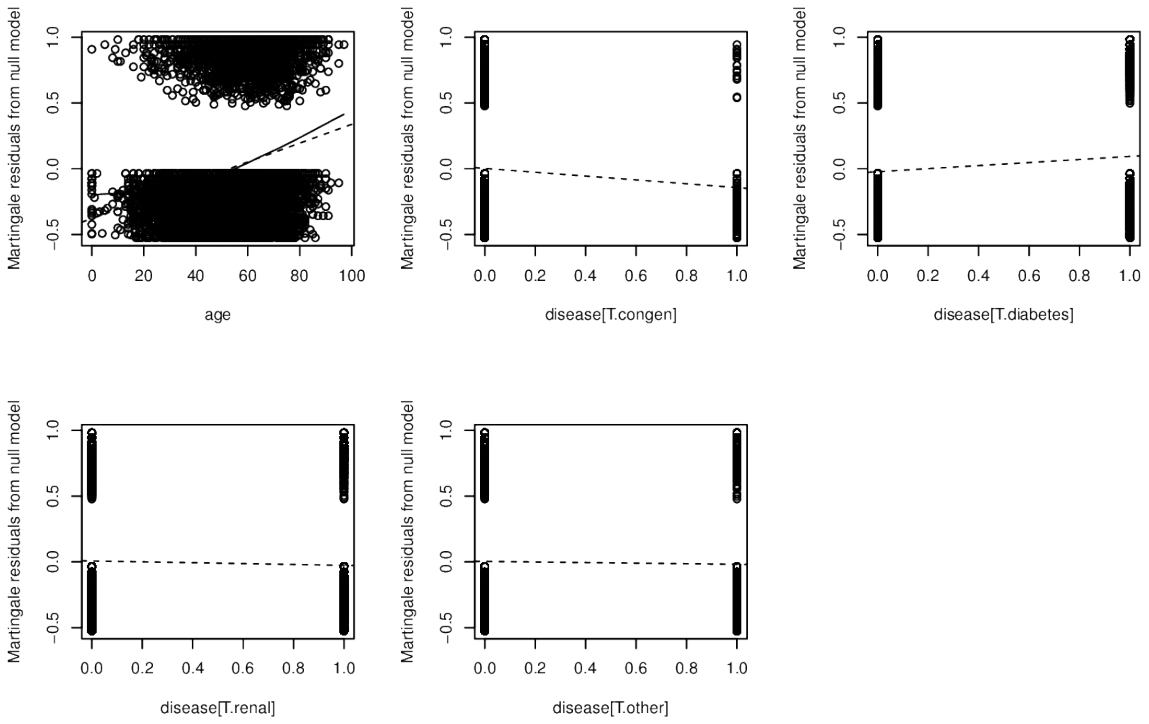

Figure 14: Plots of null Martingale residuals for the Cox model fit to the `Dialysis` data. The broken lines are fit by linear least squares, and simply connect the mean residuals at the values 0 and 1 of the dummy regressors for `disease`; the solid line in the upper-left panel for `age` is a nonparametric-regression smoother.

The frailty term is highly statistically significant, but the regression coefficients and their standard error have not changed much.

3.2. Example: Recidivism data

To illustrate further the use of the **RcmdrPlugin.survival** package, we turn to data on criminal recidivism originally from Rossi, Berk, and Lenihan (1980) and analyzed extensively by Allison (1995) in a text on survival analysis using SAS. The data set, included in the **RcmdrPlugin.survival** package and read, as before, via the R Commander `Data` → `Data in packages` → `Read data from an attached package...` dialog, pertains to 432 convicts who were released from Maryland state prisons in the 1970s and followed up for one year after release. Half the released convicts were assigned at random to an experimental treatment in which they were given financial aid, and the other half did not receive aid. The `Rossi` data set includes the following variables:

- `week` of first arrest after release or censoring; all censored observations are censored at 52 weeks.
- `arrest`, an event indicator, coded `1` if the released convict was rearrested and `0` otherwise.

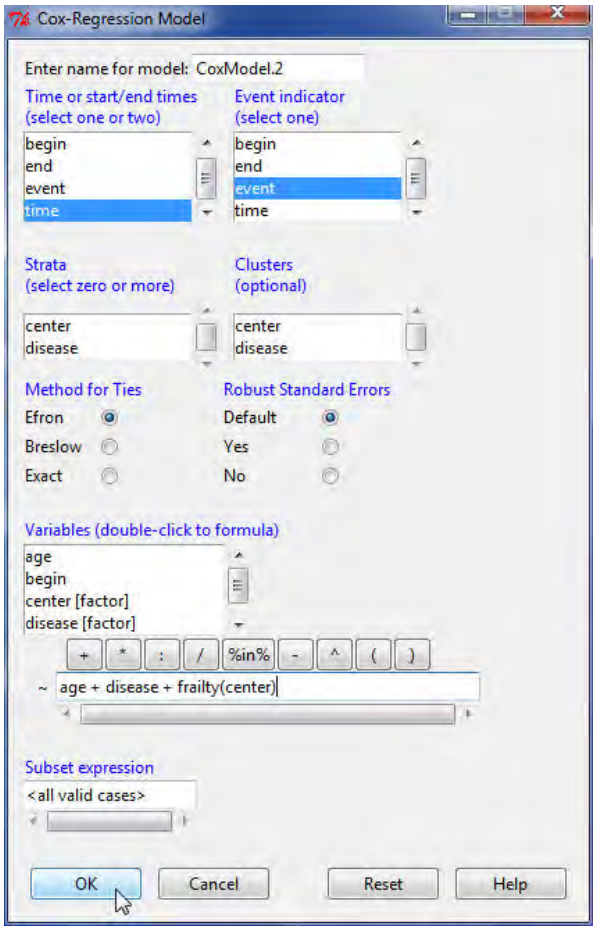

Figure 15: Cox regression dialog box, with the term `frailty(center)` typed directly into the model formula.

- `fin`, a factor with levels `yes` if the convict received financial aid and `no` if he did not.
- The convict's `age` in years at the time of release.
- The convict's `race`, a factor with levels `black` and `other`.
- `wexp`, the convict's full-time work experience prior to incarceration, a factor coded `no` or `yes`.
- `mar`, the convict's marital status at the time of release, a factor coded `married` or `not married`.
- `paro`, whether or not the convict was released on parole, a factored coded `no` or `yes`.
- `prio`, the number of convictions prior to the current conviction.
- `educ`, level of education, coded numerically: `2` = 6th grade or less; `3` = 7th to 9th grade; `4` = 10th to 11th grade; `5` = 12th grade; and `6` = some college.
- `emp1`, employment status in the first week after release, a factor with levels `no` and `yes`.

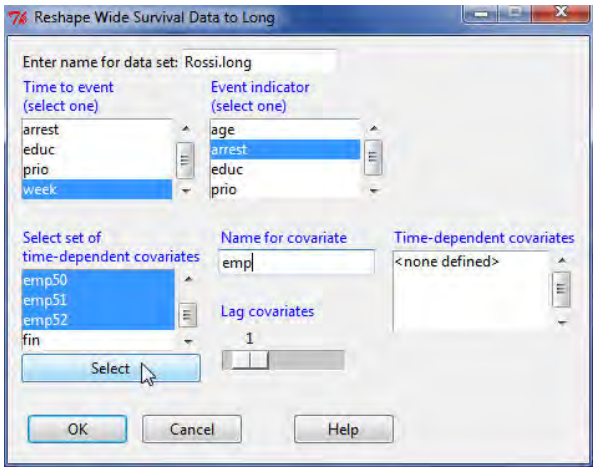

Figure 16: Changing the `Rossi` data from wide to long format. The time-dependent covariates in the original wide form of the data, `emp1` through `emp52`, are selected in the variable list box, and a name for the corresponding time-dependent covariate in the long form of the data, `emp`, is typed into the text box. Pressing `Select` will queue the operation of creating `emp` in the resulting long data set.

- `emp2` through `emp52`, recording employment status in weeks 2 through 52. Once a convict is rearrested, the values of subsequent employment indicators are missing (coded `NA`).

The data, therefore, are in "wide" format, with employment status recorded in the last 52 variables of each data record. To analyze the data with employment status as a time-varying covariate, we have to reorient the data into "long" format, with one record in counting-process form for each week during which a convict remains at risk of rearrest. As well, we would like to lag employment status by one week; otherwise, the effect of employment status will likely be exaggerated, because a convict is unable to work at the end of the week in which he is rearrested.

We can accomplish these data-management tasks with `Data` → `Survival data` → `Convert wide to long data...`, producing the dialog box shown in Figure 16. In this dialog, we selected the time variable `week` and event indicator `arrest`; picked the time-dependent covariates `emp1` through `emp52` in the list box at the lower left; typed in the name `emp` for the resulting time-varying covariate; set the lag slider to 1; and are about to press the `Select` button to define the time-dependent covariate. Were there more than one time-varying covariate, we would repeat these operations. Finally, we press the `OK` button to create the new data set, which is named `Rossi.long` by default, and which becomes the active data set in the R Commander. Although it is moderately complex and can handle many wide-to-long conversions, the reshape dialog is also substantially limited in its capabilities—for example, it can only deal with covariates that are measured at each point in time at the same level of precision as the time-to-event variable. The `unfold` function that the dialog calls is somewhat more flexible. We return to this and other limitations of the **RcmdrPlugin.survival** GUI in Section 5 of the paper.

While the original data set had 432 rows and 62 variables, the new data set has 19,377 rows and 14 variables, one row for each period of observation. The new data set has variables

`start` and `stop` for the beginning and end of each period of observation, and a new event indicator `arrest.time`, coded `1` in the week in which a convict is rearrested and `0` otherwise. Time-constant covariates are simply copied into each record pertaining to a given convict. The first few rows of the data set, with data for the first two convicts, look like this:

```
     start stop arrest.time week arrest fin age  race wexp
1.2      1    2           0   20      1  no  27 black   no
1.3      2    3           0   20      1  no  27 black   no
1.4      3    4           0   20      1  no  27 black   no
. . .
1.19    18   19           0   20      1  no  27 black   no
1.20    19   20           1   20      1  no  27 black   no
2.2      1    2           0   17      1  no  18 black   no
2.3      2    3           0   17      1  no  18 black   no
2.4      3    4           0   17      1  no  18 black   no
. . .
2.16    15   16           0   17      1  no  18 black   no
2.17    16   17           1   17      1  no  18 black   no

             mar paro prio educ emp
1.2  not married  yes    3    3  no
1.3  not married  yes    3    3  no
1.4  not married  yes    3    3  no
. . .
1.19 not married  yes    3    3  no
1.20 not married  yes    3    3  no
2.2  not married  yes    8    4  no
2.3  not married  yes    8    4  no
2.4  not married  yes    8    4  no
. . .
2.16 not married  yes    8    4  no
2.17 not married  yes    8    4  no
```

Thus, convict 1 was rearrested in the 20th week and convict 2 in the 17th week following his release.

We proceed to fit a Cox model to the recidivism data (via the dialog in Figure 17), producing the following results:

```
> CoxModel.3 <- coxph(Surv(start,stop,arrest.time) ~ age + educ +
+   emp + fin + mar + paro + prio + race + wexp, method="efron",
+   data=Rossi.long)

> summary(CoxModel.3)

Call:
coxph(formula = Surv(start, stop, arrest.time) ~ age + educ +
    emp + fin + mar + paro + prio + race + wexp, data = Rossi.long,
    method = "efron")
```

Figure 17: Dialog for fitting a Cox model to the `Rossi` data. Because the data are in counting-process format, two time variables are selected, `start` and `stop`.

```
  n= 19377, number of events= 113

                      coef exp(coef)  se(coef)      z Pr(>|z|)
age                -0.050567  0.950691  0.021703 -2.330 0.019807 *
educ               -0.194915  0.822904  0.133078 -1.465 0.143012
emp[T.yes]         -0.795258  0.451465  0.218027 -3.648 0.000265 ***
fin[T.yes]         -0.330882  0.718290  0.192232 -1.721 0.085203 .
mar[T.not married]  0.329440  1.390189  0.383518  0.859 0.390345
paro[T.yes]        -0.058040  0.943612  0.196512 -0.295 0.767724
prio                0.084767  1.088463  0.029352  2.888 0.003878 **
race[T.other]      -0.352430  0.702978  0.310237 -1.136 0.255955
wexp[T.yes]        -0.009608  0.990438  0.214080 -0.045 0.964204
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
                   exp(coef) exp(-coef) lower .95 upper .95
age                   0.9507     1.0519    0.9111    0.9920
educ                  0.8229     1.2152    0.6340    1.0681
emp[T.yes]            0.4515     2.2150    0.2945    0.6922
fin[T.yes]            0.7183     1.3922    0.4928    1.0470
mar[T.not married]    1.3902     0.7193    0.6556    2.9480
paro[T.yes]           0.9436     1.0598    0.6420    1.3870
prio                  1.0885     0.9187    1.0276    1.1529
race[T.other]         0.7030     1.4225    0.3827    1.2913
wexp[T.yes]           0.9904     1.0097    0.6510    1.5068

Rsquare= 0.003   (max possible= 0.067 )
Likelihood ratio test= 49.41  on 9 df,   p=1.389e-07
Wald test            = 44.89  on 9 df,   p=9.67e-07
Score (logrank) test = 48.02  on 9 df,   p=2.532e-07
```

The covariates age, employment status, and number of prior arrests have statistically significant coefficients; the coefficient of financial aid, the focus of the study, is just statistically significant by a one-sided test (halving the reported two-sided p value of .085).

The **RcmdrPlugin.survival** package offers several options for graphing the results of a fitted Cox model. We show one approach here, using `Models` → `Graphs` → `Cox-model survival function...`, which leads to the dialog in Figure 18. We select *Plot at specified values of predictors*, set the slider to 2 rows, and fill in the values with the medians of age, education, and the number of prior arrests, and with the most common levels for the factors: `not married` for marital status, `no` for employment status, `black` for race, and `yes` for parole and work experience. We set financial aid to `no` and `yes` in turn. The resulting graph is displayed in Figure 19.

4. The design of the RcmdrPlugin.survival package

Rcmdr plug-in packages (Fox 2007) are ordinary R packages (see R Development Core Team 2012b) with some extra features. Plug-in packages consist of several components.

The `DESCRIPTION` file for the **RcmdrPlugin.survival** package includes the line `Models: coxph, survreg, coxph.penal` to instruct the R Commander that `coxph`, `survreg`, and `coxph.penal` objects represent statistical models, manipulable through the R Commander `Models` menu.

The file `menus.txt`, included in the `inst/etc` directory of the source package, and shown in slightly edited form in Figure 20, contains directives for adding to the R Commander menus.

There are up to seven fields, separated by white space, in each line of the `menus.txt` file:

- The first field in each directive indicates what `type` of entry—`menu` or menu `item`—is defined by the directive; it is also possible to specify `remove` to delete an existing menu or item.
- The second field supplies a name for a new menu or the name of the menu under which an item is to be installed; in either case this must be a valid R variable name. For

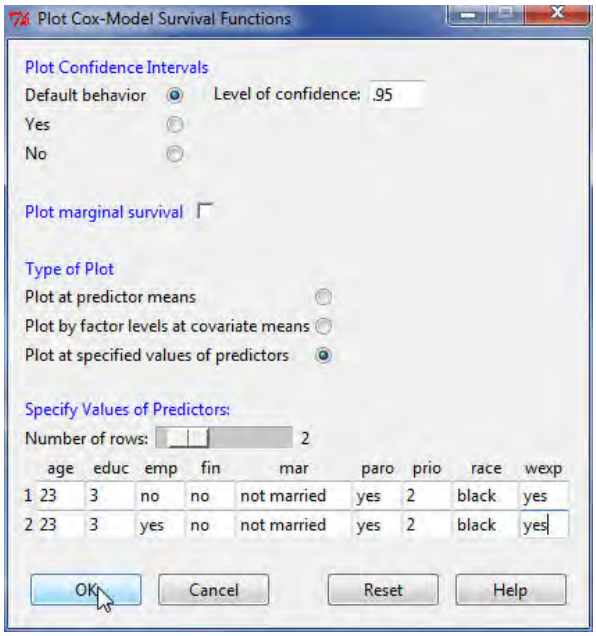

Figure 18: Dialog box for graphing the survival function from a Cox model.

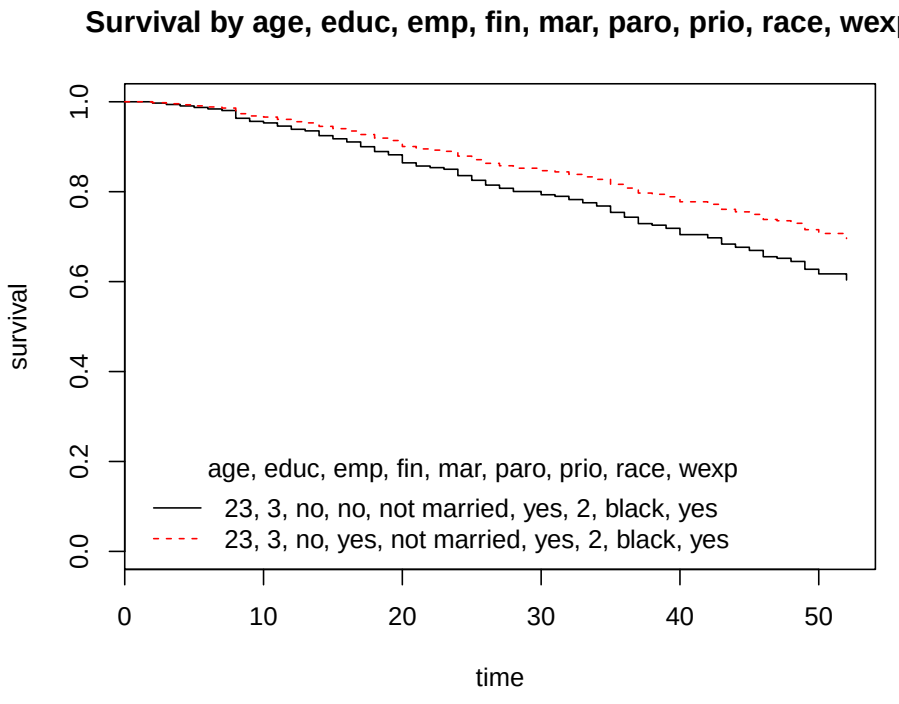

Figure 19: Estimated survival by whether or not the convict received financial aid, at typical values of the other covariates.

```
# type menu/item        operation/parent label                          command/menu     activation          install?

menu   survivalMenu     statisticsMenu   ""                             ""               ""                  ""
item   survivalMenu     command          "Estimate survival function..." Survfit         "activeDataSetP()"  "packageAvailable('survival')"
item   survivalMenu     command          "Compare survival functions..." Survdiff        "activeDataSetP()"  "packageAvailable('survival')"
item   statisticsMenu   cascade          "Survival analysis"            survivalMenu     ""                  "packageAvailable('survival')"

menu   survDataMenu     dataMenu         ""                             ""               ""                  ""
item   survDataMenu     command          "Survival data definition..."  SurvivalData     "activeDataSetP()"  "packageAvailable('survival')"
item   survDataMenu     command          "Convert wide to long data..." Unfold          "activeDataSetP()"  "packageAvailable('survival')"
item   survDataMenu     command          "Convert variable to date..."  toDate          "activeDataSetP()"  "packageAvailable('survival')"
item   dataMenu         cascade          "Survival data"                survDataMenu     ""                  "packageAvailable('survival')"

item   statModelsMenu   command          "Cox regression model..."      CoxModel         "activeDataSetP()"  "packageAvailable('survival')"
item   statModelsMenu   command          "Parametric survival model..." survregModel    "activeDataSetP()"  "packageAvailable('survival')"

item   diagnosticsMenu  command          "Test proportional hazards"    CoxZPH           "coxphP()"          "packageAvailable('survival')"

item   modelsGraphsMenu command          "Cox-model survival function..." PlotCoxph      "coxphP()"          "packageAvailable('survival')"
item   modelsGraphsMenu command          "Plot terms in Cox model"      TermPlots        "coxphP() && !highOrderTermsP()"
                                                                                                            "packageAvailable('survival')"
item   modelsGraphsMenu command          "Plot survival-regression dfbetas" CoxDfbetas   "coxphP() || survregP()"
                                                                                                            "packageAvailable('survival')"
item   modelsGraphsMenu command          "Plot survival-regression dfbeta" CoxDfbeta     "coxphP() || survregP()"
                                                                                                            "packageAvailable('survival')"
item   modelsGraphsMenu command          "Plot null Martingale residuals" MartingalePlots "coxphP()"         "packageAvailable('survival')"
item   modelsGraphsMenu command          "Cox-model partial-residual plots" PartialResPlots "coxphP()"       "packageAvailable('survival')"
```

Figure 20: The `menus.txt` file for the **RcmdrPlugin.survival** package. Some of the lines in the file are wrapped to fit on the page.

example, the first directive in Figure 20 defines a new menu named `survivalMenu`, while the second directive installs a menu item under the `survivalMenu` menu.

- The third field specifies the name of the parent menu for a newly defined submenu (such as `statisticsMenu` for `survivalMenu`), or one of two operations: `command` for a menu item (as in the second directive in Figure 20), or `cascade` to place a submenu under its parent.
- The remaining fields apply to `item` directives. The fourth field specifies a label, as a character string enclosed in quotes, for the menu or menu item. By convention, a menu item that leads to a dialog box ends in `...`.
- The fifth field specifies the parent menu for a `cascade` operation, or a *callback function* for a menu item—the name of an R function to be invoked when the menu item is selected by the user.
- The sixth field specifies an optional R command to be executed to determine whether a menu item is activated or, alternatively, "grayed out." This expression, enclosed in quotes, should return the logical value `TRUE` or `FALSE`. The **Rcmdr** package provides a variety of "predicate" functions for testing commonly used conditions for menu-item activation; for example, the function `activeDataSetP` returns `TRUE` if there is an active data set and `FALSE` otherwise.
- The seventh, and final, field is also optional—a quoted R command that evaluates to `TRUE` or `FALSE`, and determines whether a menu or menu item is installed; for example, the command `packageAvailable('survival')` is `TRUE` if the **survival** package is installed on the user's system and `FALSE` otherwise.

Missing fields in the `menu.txt` file are specified with the place-holder `""`, and may be omitted if they occur at the end of a line. At startup, the `Rcmdr` processes menu directives sequentially, and so, for example, a menu (such as `survivalMenu`) must be defined before menu items can be installed in it.

Callback functions may generate R commands directly or, more commonly, construct dialog boxes that solicit user input to formulate commands. An example of the former is shown in Figure 21, a callback function that produces index plots of dfbetas for Cox regression models and parametric survival models, accessed through `Models` → `Graphs` → `Plot survival regression dfbetas` (see Figure 13 on page 16 for an example). As a general matter, callback functions take no arguments. As is typical, `CoxDfbetas` constructs R commands as character strings and passes these to the **Rcmdr**-supplied function `doItAndPrint`, which causes the command to be executed; entered into the R Commander script window; and entered into the R Commander output window, along with the printed output, if any, generated by the command. The `logger` command, invoked near the end of `CoxDfbetas`, simply enters a command into the script and output windows without executing it. `CoxDfbetas` also makes use of the **Rcmdr** `ActiveModel()` command to return the name of the active statistical model. **Rcmdr**-supplied functions such as `doItAndPrint` and `ActiveModel` are documented in the online help file `?Rcmdr.Utilities`, and described in Fox (2005) and Fox (2007).

Figure 22 shows a typical callback function, `Survdiff`, for creating an R Commander dialog box, in this case for comparing survival functions in different strata via the `survtest` function

```
CoxDfbetas <- function(){
  command <- paste(".dfbetas <- as.matrix(residuals(",
    ActiveModel(), ', type="dfbetas"))', sep = "")
  doItAndPrint(command)
  command <- if (coxphP())
    paste("colnames(.dfbetas) <- names(coef(",
      ActiveModel(), "))", sep = "")
    else paste("colnames(.dfbetas) <- rownames(summary(",
      ActiveModel(), ")$table)", sep = "")
  doItAndPrint(command)
  ncol <- ncol(.dfbetas)
  doItAndPrint(paste(".mfrow <- par(mfrow = mfrow(", ncol, "))", sep = ""))
  for (col in colnames(.dfbetas)){
    doItAndPrint(paste('plot(.dfbetas[,"', col, '"])', sep = ""))
    doItAndPrint("abline(h=0, lty=2)")
  }
  doItAndPrint("par(mfrow=.mfrow)")
  logger("remove(.dfbetas, .mfrow)")
  remove(.dfbetas, .mfrow, envir = .GlobalEnv)
}
```

Figure 21: Callback function `CoxDfbetas` to produce index plots of dfbetas in Cox models and parametric survival models (produced by the `survreg` function).

in the **survival** package (see Figure 8 on page 11, along with the associated output). The `Survdiff` function makes use of several **Rcmdr**-supplied utilities, including:

- `initializeDialog` to set up the dialog box;
- `getSelection` to return the selection from a list box;
- `errorCondition` to signal a user error;
- `trim.blanks` to remove leading and trailing blanks from a text string;
- `doItAndPrint` to execute the command generated from the dialog;
- `OKCancelHelp` to create the standard set of **Rcmdr** buttons at the bottom of the dialog;
- `Factors` to ascertain the names of factors (i.e., categorical variables) in the active data set;
- `variableListBox` to create a variable list box;
- `getFrame` to return the enclosing frame of a variable list box;
- `labelRcmdr` to create a standard **Rcmdr** label widget;
- `dialogSuffix` to perform a variety of housekeeping tasks to complete the dialog; and

- the `getDialog` and `putDialog` functions to permit the dialog to "remember" its state from one invocation to the next, a new facility introduced in version 1.7-0 of the **Rcmdr**.[5] The defaults provided to `getDialog` are used if previous values haven't been stored, and dialog state is refreshed when the data set changes or when the `Refresh` button is pressed in a dialog.

Other commands used in the dialog are provided by the **RcmdrPlugin.survival** package:

- when there are two time variables selected, `startStop` determines which is the start and which the end of a period of observation;
- `numericOrDate` returns the names of numeric and date variables in the active data set.

Several commands are provided directly by the **tcltk** package: `tclvalue`, `tkfocus`, `tkframe`, `tclVar`, `tkscale`, and `tkgrid`. Finally, the `gettext` function, which is part of base R, makes provision for messages and other text in the **RcmdrPlugin.survival** package to be translated from English into other languages, a feature shared with **Rcmdr** package. Currently, four translations are available for the **RcmdrPlugin.survival** package, into Brazilian Portuguese, French, Russian, and Spanish.

The **RcmdrPlugin.survival** package also provides some facilities that could have been included in the **survival** package but were not, such a `plot` method for `coxph` objects and a `quantile` method for `survfit` objects.

5. Concluding remarks: Design challenges and limitations

It is difficult, and probably ill-advised, to expose all of the facilities of complex statistical software such as the **survival** package through a GUI: A complete GUI for the **survival** package would likely be hard to use. On the other hand, to the extent that the work-flow in a statistical analysis can be decomposed into semi-independent operations of manageable complexity, it should be possible to provide menus and dialog boxes to perform these operations. This is essentially the approach that is taken in the R Commander, and that approach is inherited by the survival-analysis plug-in described in this paper.

Although it might at first blush seem counter-intuitive, we believe that the object-orientation of R, and in particular of the statistical modeling functions in R, facilitates the development of GUIs, because the encapsulation of a statistical model, for example, in an R object allows us to modularize the process of data analysis, providing menu items and dialogs for standard manipulation of the model. This approach also fits well with the essentially iterative character of statistical modeling—as, of course, does the canonical command-driven approach in R. Contrast this with software in which all operations related to a statistical model must be specified in a single dialog or command, typically producing voluminous output. Similarly, the fact that R data frames are modifiable objects allows us to add information, subsequently available to the GUI, that is specifically relevant to survival analysis—such as the names of time and event variables in the data set.

[5]This mechanism is activated by setting `options(Rcmdr = list(retain.selections = TRUE))`, which is the default as of **Rcmdr** version 1.8-1

```
Survdiff <- function(){
  require("survival")
  defaults <- list(time1 = NULL, event = NULL, strata = NULL, rho = "0",
    subset = NULL)
  dialog.values <- getDialog("Survdiff", defaults)
  if (!activeDataSetP()) return()
  currentModel <- FALSE
  initializeDialog(title = gettext("Compare Survival Functions",
    domain = "R-RcmdrPlugin.survival"))
  onOK <- function(){
    time <- getSelection(timeBox)
    if (length(time) == 1) time1 <- time
    else {
      errorCondition(recall = Survdiff,
        message = gettext("You must select a time-to-event variable.",
        domain = "R-RcmdrPlugin.survival"))
      return()
    }
    event <- getSelection(eventBox)
    if (length(event) == 0) {
      errorCondition(recall = Survdiff,
        message = gettext("You must select an event indicator.",
        domain = "R-RcmdrPlugin.survival"))
      return()
    }
    strata <- getSelection(strataBox)
    if (length(strata) == 0) {
      errorCondition(recall = Survdiff,
        message = gettext("You must select strata.",
        domain = "R-RcmdrPlugin.survival"))
      return()
    }
    rho <- tclvalue(rhoValue)
    subset <- tclvalue(subsetVariable)
    putDialog("Survdiff",
      list(time1 = time1, event = event, strata = strata, rho = rho,
      subset = subset))
    closeDialog()
    if (trim.blanks(subset) == gettext("<all valid cases>",
        domain = "R-RcmdrPlugin.survival") || trim.blanks(subset) == ""){
      subset <- ""
    }
    else{
      subset <- paste(", subset=", subset, sep = "")
    }
    formula <- paste("Surv(", time1, ",", event, ")", sep = "")
    formula <- paste(formula, " ~ ", paste(strata, collapse = " + "), sep = "")
    command <- paste("survdiff(", formula, ", rho=", rho,
      ', data=', ActiveDataSet(), subset, ")", sep = "")
```

Figure 22: Callback function `Survdiff` to produce a dialog box for testing differences in survival functions (continued on the next page).

```
    doItAndPrint(command)
    tkfocus(CommanderWindow())
  }
  OKCancelHelp(helpSubject = "survdiff", reset = "Survdiff")
  survFrame <- tkframe(top)
  .activeDataSet <- ActiveDataSet()
  .numeric <- NumericOrDate()
  .factors <- Factors()
  time1 <- if(!is.null(dialog.values$time1)) dialog.values$time1
    else eval(parse(text = paste('attr(', .activeDataSet, ', "time1")', sep = "")))
  time1 <- if (!is.null(time1)) which(time1 == .numeric) - 1
  event <- if(!is.null(dialog.values$event))dialog.values$event
    else eval(parse(text = paste('attr(', .activeDataSet, ', "event")', sep = "")))
  event <- if (!is.null(event)) which(event == Numeric()) - 1
  strata <- if(!is.null(dialog.values$strata)) dialog.values$strata
    else eval(parse(text = paste('attr(', .activeDataSet, ', "strata")', sep="")))
  strata <- if (!is.null(strata)) which(is.element(.factors, strata)) - 1 else -1
  timeBox <- variableListBox(survFrame, NumericOrDate(),
    title = gettext("Time to event\n(select one)",
    domain = "R-RcmdrPlugin.survival"),
    initialSelection = time1)
  eventBox <- variableListBox(survFrame, Numeric(),
    title = gettext("Event indicator\n(select one)",
    domain = "R-RcmdrPlugin.survival"),
    initialSelection = event)
  strataBox <- variableListBox(survFrame, Factors(),
    title = gettext("Strata\n(select one or more)",
    domain = "R-RcmdrPlugin.survival"),
    selectmode = "multiple", initialSelection = strata)
  rhoFrame <- tkframe(top)
  rhoValue <- tclVar(dialog.values$rho)
  rhoSlider <- tkscale(rhoFrame, from = 0, to = 1, showvalue = TRUE,
    variable = rhoValue, resolution = 0.1, orient = "horizontal")
  subsetBox(subset.expression = dialog.values$subset)
  tkgrid(getFrame(timeBox), labelRcmdr(survFrame, text = "  "),
    getFrame(eventBox), sticky = "sw")
  tkgrid(labelRcmdr(survFrame, text = ""))
  tkgrid(getFrame(strataBox), sticky = "nw")
  tkgrid(survFrame, sticky = "nw")
  tkgrid(labelRcmdr(rhoFrame, text = "rho", foreground = "blue"), rhoSlider,
    sticky = "sw")
  tkgrid(rhoFrame, sticky = "nw")
  tkgrid(labelRcmdr(top, text = ""))
  tkgrid(subsetFrame, sticky = "w")
  tkgrid(labelRcmdr(top, text = ""))
  tkgrid(buttonsFrame, sticky = "w")
  dialogSuffix(rows = 9, columns = 1)
}
```

Figure 22: (concluded) Callback function `Survdiff` to produce a dialog box for testing differences in survival functions.

To the extent that the work-flow of data analysis isn't easily broken into relatively simple steps, however, the design of a GUI becomes challenging. We believe that this is particularly true of data manipulation, where it is difficult in a GUI to make provision for all eventualities that commonly arise, and where many data sets have idiosyncratic features. It is, therefore, often much more natural to manipulate data by writing simple programs than through a GUI, and the R user who takes the time to master simple programming is consequently at a distinct advantage.

A case in point is the dialog provided by the **RcmdrPlugin.survival** package for reshaping data from wide to long form, discussed briefly in Section 3. As we explained, this dialog is capable of handling wide data sets with a regular structure in which time-varying covariates are measured at equally spaced intervals of the same precision as the time-to-event variable in the data set. The `unfold` function that the dialog invokes can handle more complex time-varying covariate structures, and other facilities in R, such as the standard `reshape` function and the **reshape** package (Wickham 2007) provide still greater flexibility. Writing a more general GUI for any of these facilities would be challenging and quite likely of dubious value. Indeed, given the requisite programming skill, it is often simpler to write a quick-and-dirty function for manipulating data than to use a general function. Notwithstanding these general caveats, however, we hope to improve the survival-data handling facilities of the **RcmdrPlugin.survival** as the opportunity to do so arises.

The R Commander plug-in architecture has the advantage of embedding a GUI for a particular area of data analysis, such as survival analysis, in a more general framework, providing, for example, substantial infrastructure for reading, manipulating, and summarizing data. The R Commander also provides some general facilities to the plug-in developer for writing menus and dialogs. Of course, the resulting plug-in must conform to the general structure of the R Commander, which is oriented towards the step-by-step analysis of one rectangular data set at a time.

We must confess to having mixed feelings about GUIs for R, and for statistical software more generally. We believe that most serious users of R would be better served by learning to write commands. Nevertheless, GUIs such as the R Commander do have a legitimate role: for use in basic and perhaps intermediate-level statistics courses, where teaching complex computational tools such as R can distract from the principal focus on statistical ideas; among casual users of statistical software, whose memory of commands would otherwise be taxed; and in other special circumstances where ease of use is critical.

Acknowledgments

Work on the **Rcmdr** and **RcmdrPlugin.survival** packages was supported by a grant to John Fox from the Social Sciences and Humanities Research Council of Canada and by the Senator William McMaster Chair in Social Statistics at McMaster University. We are grateful to two anonymous referees for helpful comments on an earlier version of this paper.

References

Allison PD (1995). *Survival Analysis Using the SAS System: A Practical Guide*. SAS Institute,

Cary, NC.

Carvalho MS, Andreozzi VL, Codeço CT, Barbosa, Serrano MT, Shimakura SE (2005). *Análise de Sobrevida: Teoria e Aplicaçõnes em Saúde*. Editora FIOCRUZ, Rio de Janeiro.

Carvalho MS, Henderson R, Shimakura S, Sousa IPSC (2003). "Survival of Hemodialysis Patients: Modleing Differences in Risk of Dialysis Centers." *International Journal for Quality in Health Care*, **15**(3), 189–196.

Dalgaard P (2001). "A Primer on the R-Tcl/Tk Package." *R News*, **1**(3), 27–31. URL http://CRAN.R-project.org/doc/Rnews/.

Dalgaard P (2002). "Changes to the R-Tcl/Tk package." *R News*, **2**(3), 25–27. URL http://CRAN.R-project.org/doc/Rnews/.

Fox J (2005). "The R Commander: A Basic-Statistics Graphical User Interface to R." *Journal of Statistical Software*, **14**(9), 1–42. URL http://www.jstatsoft.org/v14/i09/.

Fox J (2007). "Extending the R Commander by 'Plug-In' Packages." *R News*, **7**(3), 46–52. URL http://CRAN.R-project.org/doc/Rnews/.

Fox J, Weisberg S (2011). *An R Companion to Applied Regression*. 2nd edition. Sage Publications, Thousand Oaks.

Moore DS (2010). *The Basic Practice of Statistics*. 5th edition. W. H. Freeman and Company, New York.

R Development Core Team (2012a). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

R Development Core Team (2012b). *Writing R Extensions*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-11-9, URL http://www.R-project.org/.

Rossi PH, Berk RA, Lenihan KJ (1980). *Money, Work, and Crime: Some Experimental Results*. Academic Press, New York.

Therneau T (2012). ***survival**: A Package for Survival Analysis in S*. R package version 2.36-12, URL http://CRAN.R-project.org/package=survival.

Therneau TM, Grambsch PM (2000). *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, New York.

Welch BB, Jones K (2003). *Practical Programming in Tcl and Tk*. 4th edition. Prentice Hall PTR, Upper Saddle River.

Wickham H (2007). "Reshaping Data with the **reshape** Package." *Journal of Statistical Software*, **21**(12), 1–20. URL http://www.jstatsoft.org/v21/i12/.

Affiliation:

John Fox
Department of Sociology
McMaster University
Hamilton, Ontario, Canada L8S 4M4
E-mail: jfox@mcmaster.ca
URL: http://socserv.socsci.mcmaster.ca/jfox/

Marilia Sá Carvalho
Escola Nacional de Saúde Pública
FIOCRUZ
Rua Leopoldo Bulhôes
1480/8deg Andar
Rio de Janeiro, RJ 210041-210, Brazil
E-mail: carvalho@procc.fiocruz.br

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 49, Issue 7 *Submitted:* 2010-12-26
June 2012 *Accepted:* 2011-12-28