merlin—A unified modeling framework for data analysis and methods development in Stata

Michael J. Crowther
University of Leicester
Leicester, UK
and
Karolinska Institutet
Stockholm, Sweden
michael.crowther@le.ac.uk

Abstract. The challenges in statistics and data science are rapidly growing because access to a multitude of data types continues to increase, as well as the sheer quantity of data. Analysts are now presented with multivariate data, sometimes measured repeatedly, and often requiring the ability to model nonlinear relationships and hierarchical structures. In this article, I present the merlin command, which attempts to provide an extremely general framework for data analysis. From simple settings such as fitting a linear regression model or a Weibull survival model to more complex settings such as fitting a three-level logistic mixed-effects model or a multivariate joint model of multiple longitudinal outcomes (of different types) and a recurrent event and survival with nonlinear effects, merlin can fit them all. I will take a single dataset and attempt to show you the full range of capabilities of merlin and discuss some future directions for the implementation in Stata.

Keywords: st0616, merlin, modeling framework, outcome models, survival models, longitudinal models, Gaussian, Bernoulli, beta, Poisson, ordinal logistic, ordinal probit, gamma, exponential, Gompertz, Royston–Parmar, log-hazard, Weibull, time-dependent effects, restricted cubic splines, B-splines, fractional polynomial, random effects, multilevel, multivariate, hierarchical

1 Introduction
gsem’s introduction in Stata 14 brought an extremely broad class of mixed-effects models (among other things); most importantly (from my perspective), gsem is fast. gsem has full analytic derivatives for any model that you fit, including a model with any number of levels and random effects at each level. Of course, gllamm preceded gsem, providing a very flexible framework in Stata for modeling data (Rabe-Hesketh, Skrondal, and Pickles 2002, 2005). But inevitably, every program is limited to certain families of distributions, complexity of the linear predictors, and of course computational speed. There is also the widely used community-contributed cmp package by Roodman (2011), which implements joint modeling of multivariate processes, assuming joint normality. Bartus (2017) further demonstrated gsem’s capabilities in modeling simultaneous equation systems, including multilevel hazard equations with correlated random effects, with a focus on piecewise exponential models. merlin builds directly on these works but focuses on more flexible
Modeling framework for data analysis and methods development

parametric distributions, extending linking between outcome models, and nonlinear modeling capabilities.

Much of my previous work has centered on joint longitudinal survival analysis, implemented in the *stjm* command (Crowther, Abrams, and Lambert 2013), which can fit a joint model for a continuous, repeatedly measured biomarker and a single time to event. There is flexibility in that you can use splines or polynomials to model the biomarker over time, and there are lots of choices for your survival model, including spline-based approaches. The main benefit of *stjm* is the added flexibility in how to link the two submodels through things such as the expected value, or derivatives of it, which are commonly used in the joint model literature. Given this starting point, the natural extension is to allow for more than one biomarker, recurrent events, competing risks, etc. I have also released *stmixed* (Crowther, Look, and Riley 2014) for two-level parametric survival models, in particular for the Royston–Parmar spline-based model, and *stgenreg* for general hazard-based regression (Crowther and Lambert 2013, 2014), where users could specify their own functional form for the hazard function and *stgenreg* would use numerical quadrature to calculate the likelihood.

The aim of *merlin* is not only to bring together my previous programs but also to provide a whole lot more. In this article, I will introduce the core features and syntax of *merlin* and illustrate a variety of models that can be fit with it. More details on the methodological framework can be found in Crowther (2017). The article is structured as follows: In section 2, I describe the top-level syntax of the *merlin* command and describe many of the options. In section 3, I detail many examples, showing the wide variety of available models. In section 4, I conclude the article with a discussion of *merlin* and its potential.

### 2 The architecture of merlin

*merlin* is designed to be as flexible and general as possible. There is no real limit to what it can do, given that the user can readily extend it. You can specify any number of outcome models, which can be linked in any number of ways. Each outcome model, $i = 1, \ldots, M$, has a main complex predictor (you can have more, but I will leave that for another time) that comprises additive components, $c = 1, \ldots, C_i$, where each component comprises multiplicative elements, $e = 1, \ldots, E_{ic}$,

$$
g_i \{ \mu_i(y_i|x, b) \} = \sum_{c=1}^{C_i} \prod_{e=1}^{E_{ic}} \psi_{ice}(x, b)
$$

where for the $i$th model, $y_i$ is the observed response, $x$ is the full-design matrix, and $b$ are the stacked multivariate normal or $t$ distributed random effects. We have link function $g_i(\cdot)$ and expected value $\mu_i$. $\psi(\cdot)$ defines essentially an arbitrary function, of which some special cases are defined in the next section.

*merlin* models are fit using maximum likelihood, with any random effects integrated out using adaptive or nonadaptive Gaussian quadrature or Monte Carlo integration.
More details on the formulation of the likelihood can be found in Crowther (2017) or the Stata manual entry for \texttt{gsem} because the formulation and default integration options (adaptive Gauss–Hermite quadrature) are the same.

### 2.1 merlin syntax

\begin{verbatim}
merlin (model1) [ (model2) ] [ ... ] [ if ] [ in ] [, options ]
\end{verbatim}

where the syntax of a \textit{model} is

\begin{verbatim}
[ depvar ] [ component1 ] [ component2 ] [ ... ] [ if ] [ in ] [, model_options ]
\end{verbatim}

where the syntax of a \textit{component} is

\begin{verbatim}
element1 [ #element2 ] [ #element3 ] [ ... ] @real
\end{verbatim}

and each \textit{elementk} can take one of the forms described in the next section. At the end of each component, you may optionally specify a constraint on the parameter or parameters of the associated component by using \texttt{@}.

\textit{model_options} include the following:

- \texttt{family(family, fam_options)}, where \textit{family} can be one of
  - \texttt{gaussian}—Gaussian distribution
  - \texttt{bernoulli}—Bernoulli distribution
  - \texttt{beta}—beta distribution
  - \texttt{poisson}—Poisson distribution
  - \texttt{ologit}—ordinal with logistic link
  - \texttt{oprobit}—ordinal with probit link
  - \texttt{gamma}—gamma distribution
  - \texttt{exponential}—exponential distribution
  - \texttt{gompertz}—Gompertz distribution
  - \texttt{rp}—Royston–Parmar model (restricted cubic spline on log cumulative-hazard scale)
  - \texttt{loghazard}—general log-hazard model
  - \texttt{logchazard}—general log cumulative-hazard model
  - \texttt{weibull}—Weibull distribution
  - \texttt{user}—specify your own distribution
Modeling framework for data analysis and methods development

and fam_options include the following:

- `failure(varname)`—event indicator for a survival model
- `ltruncated(varname)`—left-truncation/delayed entry times for survival model
- `llfunction(func_name)`—name of a Mata function that returns your user-defined log-likelihood contribution
- `hfunction(func_name)`—name of a Mata function that returns your user-defined hazard function
- `chfunction(func_name)`—name of a Mata function that returns your user-defined cumulative hazard function
- `nap(#)`—estimate # ancillary parameters, which may be called in user-defined functions

- `timevar(varname)` specifies the variable that contains time; this is required to specify time-dependent effects. Generally, within a survival model, time must be explicitly handled by `merlin`. `timevar()` will be matched against any elements (see the next section) that may use it to make sure it is handled correctly.

There are many other suboptions available in `merlin`, which are fully documented in the help files.

Element types

One of the fundamental flexibilities of `merlin` is the variety of elements that can be used within your model.

- `varname` is an independent variable in your dataset.
- `rcs(varname, opts)` is a restricted cubic spline function, where `opts` include the following:
  - `df(#)` is degrees of freedom (# of internal knots + 1). Knots are placed at evenly spaced centiles of `varname`, with boundary knots assumed to be the minimum and maximum of `varname`.
  - `knots(knots_list)` specifies the knots, including the boundary knots.
  - `orthog` applies Gram–Schmidt orthogonalization, which can improve convergence.
  - `event` used in conjunction with `df()` specifies that internal knot locations be based on centiles of only observations of `varname` that experienced the survival event specified in `failure()`.
  - `log` creates splines of `log(varname)` instead of the default untransformed `varname`.
\textbf{- offset(varname)} specifies an offset to be added to varname before the rcs() function is built.

\textbullet \textbf{bs(varname, opts)} is a B-spline function, where opts include the following:

\textbullet \textbf{df(\#)} specifies the degrees of freedom (not strictly speaking) for the spline function, which allows you to specify internal knots at equally spaced centiles, instead of using knots(). df() is consistent with rcs() elements in how internal knots are chosen.

\bullet \textbf{knots(knots_list)} specifies the internal knots. They must be in ascending order.

\bullet \textbf{bknots(knots_list)} specifies the lower- and upper-boundary knot locations. They must be in ascending order. The default is the minimum and maximum of varname.

\bullet \textbf{event} used in conjunction with df() specifies that internal knot locations be based on centiles of only observations of varname that experienced the survival event specified in failure().

\bullet \textbf{log} creates splines of log(varname) instead of the default untransformed varname.

\bullet \textbf{intercept} includes the intercept basis function, which by default is not included.

\bullet \textbf{offset(varname)} specifies an offset to be added to varname before the bs() function is built.

\textbullet \textbf{fp(varname, opts)} is a fractional polynomial function (of order 1 or 2), where opts include the following:

\bullet \textbf{powers(numlist)} powers up to a second-degree fractional polynomial function, each of which can be one of \((-2, -1, -0.5, 0, 0.5, 1, 2, 3)\).

\bullet \textbf{offset(varname)} specifies an offset to be added to varname before the fp() function is built.

\textbullet \textbf{mf(function_name)} is a user-defined Mata function.

\bullet \textbf{M\# [cluster_level]} is a random effect defined at the cluster_level; for example, M1[centre] defines a random intercept at the centre level, and M2[centre>id] defines a random intercept at the id level.

\bullet \textbf{EV[depvar \#]} is the expected value of an outcome model.

\bullet \textbf{dEV[depvar \#]} is the first derivative with respect to time of the expected value of an outcome model.

\bullet \textbf{d2EV[depvar \#]} is the second derivative with respect to time of the expected value of an outcome model.
Modeling framework for data analysis and methods development

- \( iEV[depvar | #] \) is the integral with respect to time of the expected value of an outcome model.
- \( XB[depvar | #] \) is the expected value of a complex predictor.
- \( dXB[depvar | #] \) is the first derivative with respect to time of the expected value of a complex predictor.
- \( d2XB[depvar | #] \) is the second derivative with respect to time of the expected value of a complex predictor.
- \( iXB[depvar | #] \) is the integral with respect to time of the expected value of a complex predictor.

**merlin postestimation**

*merlin* comes with a set of postestimation tools available through the *predict* function, with the standard syntax of

`predict newvar [, statistic options]`

where *statistic* includes the following:

- **mu**—expected value of the response
- **eta**—expected value of the complex predictor
- **survival**—survival function
- **cif**—cumulative incidence function
- **hazard**—hazard function
- **chazard**—cumulative hazard function
- **rmst**—restricted mean survival time (integral of *survival*)
- **timelost**—time lost because of the event (integral of *cif*)
- **totaltimelost**—total time lost because of all events, within \((0, t]\)
- **mudifference**—differences in expected values of *depvar*
- **etadifference**—differences in expected value of complex predictor
- **hdifference**—differences in hazard functions
- **sdifference**—differences in survival functions
- **cifdifference**—differences in cumulative incidence functions
- **rmstdifference**—differences in restricted mean survival functions
options include the following:

- **fixedonly** calculates prediction based only on the fixed effects.
- **marginal** calculates prediction integrating out all random effects, that is, the population-averaged prediction.
- **outcome(#)** specifies the model to predict for; the default is outcome(1).
- **causes(numlist)** specifies which models contribute to the calculation; this is for use in competing risks models.
- **at(at_spec)** specifies covariate patterns at which to calculate statistic, for example, at(trt 1 age 54).
- **ci** calculates confidence intervals using the delta method through predictnl.
- **timevar(varname)** specifies a variable that contains time points at which to calculate statistic.
- **at1(at_spec)** specifies covariate values for the first contrast.
- **at2(at_spec)** specifies covariate values for the second contrast.
- **intpoints(#)** uses # integration points to compute marginal predictions.

### 3 Examples

Given how varied and complex the syntax can appear, the easiest way to get to grips with merlin is through some examples. Throughout this section, I will use a single dataset (Murtaugh et al. 1994). It is a commonly used dataset from the joint longitudinal survival literature and will serve to illustrate many different analysis techniques, culminating in a detailed, complex multivariate hierarchical model. First, I load the data, which you can get from my website,

```stata
. use "https://www.mjcrowther.co.uk/data/jm_example"
---(Example dataset for joint modelling, Michael Crowther 2011)---
```

The dataset consists of information on 312 patients with primary biliary cirrhosis, of which 140 died during a maximum follow-up of 14.3 years. Covariates of interest include serum bilirubin and prothrombin index, both markers of liver performance, and treatment allocation, trt. Patients were randomized to either D-penicillamine or a placebo. In all analyses, I will work with the log of serum bilirubin, stored in logb. The data structure looks as follows:
Modeling framework for data analysis and methods development

merlin, just like gsem, treats models in wide format but observations within a model in long format. Hence, our survival outcome variables, stime and died, must have only one observation per id. If a patient had more than one row of data for his or her survival outcome, then we could fit a recurrent event model. The time variable records the times at which the biomarkers logb and prothrombin were recorded.

3.1 Linear mixed-effects regression

I will start with a very simple model, a linear regression of logb against time, assuming a Gaussian response:

```
. merlin (logb time, family(gaussian))
```

Fitting full model:

```
Iteration 0:  log likelihood = -3339.4091
Iteration 1:  log likelihood = -3044.5062
Iteration 2:  log likelihood = -2962.0950
Iteration 3:  log likelihood = -2961.4150
Iteration 4:  log likelihood = -2961.4150
Iteration 5:  log likelihood = -2961.4150
```

Mixed effects regression model

```
Number of obs = 1,945
Log likelihood = -2961.4144
```

| Coef.  | Std. Err.  | z    | P>|z|  | 95% Conf. Interval |
|--------|------------|------|------|------------------|
| logb:  |            |      |      |                  |
| time   | .0139443   | .0081287 | 1.72 | .086  | -.0019876 -.0298763 |
| _cons  | .5594103   | .0358095 | 15.62| .000  | .489225 .6295956   |
| sd(resid.) | 1.109201 | .0177842 | 1.074866 | 1.144611 |
```

Let's add some flexibility by using the rcs() element, which lets us model the change over time flexibly using restricted cubic splines with three degrees of freedom, that is, three spline terms.

```
. merlin (logb time, family(gaussian))
```

Fitting full model:

```
Iteration 0:  log likelihood = -3339.4091
Iteration 1:  log likelihood = -3044.5062
Iteration 2:  log likelihood = -2962.0950
Iteration 3:  log likelihood = -2961.4150
Iteration 4:  log likelihood = -2961.4150
Iteration 5:  log likelihood = -2961.4150
```

Mixed effects regression model

```
Number of obs = 1,945
Log likelihood = -2961.4144
```

| Coef.  | Std. Err.  | z    | P>|z|  | 95% Conf. Interval |
|--------|------------|------|------|------------------|
| logb:  |            |      |      |                  |
| time   | .0139443   | .0081287 | 1.72 | .086  | -.0019876 -.0298763 |
| _cons  | .5594103   | .0358095 | 15.62| .000  | .489225 .6295956   |
| sd(resid.) | 1.109201 | .0177842 | 1.074866 | 1.144611 |
```
M. J. Crowther

```
merlin (logb rcs(time, df(3)), family(gaussian)), nolog
variables created for model 1, component 1: _cmp_1_1_1 to _cmp_1_1_3
Fitting full model:
Mixed effects regression model Number of obs = 1,945
Log likelihood = -2960.7317

------------------------------------------------------------------------------
| Coef. Std. Err.      z    P>|z|      [95% Conf. Interval]   
|------------------------------------------------------------------------------
|        logb:                     
|   rcs():1        0.0636201   0.065479   0.97   0.331    -.0647164    .1919567
|   rcs():2        0.0057442   0.0115642  0.50   0.619    -.0169212    .0284096
|   rcs():3      -.0012839   0.0034269  -0.37   0.708    -.0080005    .0054327
|_cons            0.518356    0.0540793  9.59   0.000     0.4123626    0.6243494
|sd(resid.)      1.108811    0.0177782  1.074509  1.144209

If you prefer fractional polynomials or B-splines, then use the fp() or bs() element types. Given our observations are clustered within individuals, let's add a random intercept:

merlin (logb rcs(time, df(3)) M1[id]@1, family(gaussian)), nolog
variables created for model 1, component 1: _cmp_1_1_1 to _cmp_1_1_3
Fitting fixed effects model:
Fitting full model:
Mixed effects regression model Number of obs = 1,945
Log likelihood = -1871.1924

------------------------------------------------------------------------------
| Coef. Std. Err.      z    P>|z|      [95% Conf. Interval]   
|------------------------------------------------------------------------------
|        logb:                     
|   rcs():1        0.11573    0.0300614   3.85   0.000     0.0568107    0.1746493
|   rcs():2     -.0047586   0.0052904  -0.90   0.368    -.0151275    .0056104
|   rcs():3        0.0024164   0.0015696   1.54   0.124    -.0006599    .0054928
| M1[id]          .13        .13        .13        .13
|_cons            0.5311768   0.0666663   7.97   0.000     0.4051335    .6618403
|sd(resid.)      0.4866347   0.0085298  47.02005  50.36434

I have added M1[id]@1 to my complex predictor. This defines a single normally distributed random effect, called M1, defined at the id level. By default, any component within the complex predictor will have an estimated coefficient. Given our model will already estimate a fixed intercept, we want to constrain the random-effects coefficient to be 1 by specifying @1 at the end. Let's add a random linear slope and also orthogonalize my spline terms:

```
merlin (logb rcs(time, df(3)), family(gaussian)), nolog
variables created for model 1, component 1: _cmp_1_1_1 to _cmp_1_1_3
Fitting full model:
Mixed effects regression model Number of obs = 1,945
Log likelihood = -2960.7317

------------------------------------------------------------------------------
| Coef. Std. Err.      z    P>|z|      [95% Conf. Interval]   
|------------------------------------------------------------------------------
|        logb:                     
|   rcs():1        0.0636201   0.065479   0.97   0.331    -.0647164    .1919567
|   rcs():2        0.0057442   0.0115642  0.50   0.619    -.0169212    .0284096
|   rcs():3      -.0012839   0.0034269  -0.37   0.708    -.0080005    .0054327
|_cons            0.518356    0.0540793  9.59   0.000     0.4123626    0.6243494
|sd(resid.)      1.108811    0.0177782  1.074509  1.144209
```

Modeling framework for data analysis and methods development

```
merlin (logb rcs(time, df(3) orthog) time#M2[id]@1 M1[id]@1, > family(gaussian)), nolog
variables created for model 1, component 1: _cmp_1_1_1 to _cmp_1_1_3
Fitting fixed effects model:
Fitting full model:
```

Mixed effects regression model  Number of obs  =  1,945
Log likelihood = -1531.5158

```

|            | Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|------------|-------|-----------|------|-----|----------------------|
| logb:      |       |           |      |     |                      |
| rcs():1    | 0.547 | 0.044     | 12.44| 0.000| 0.4607322 - 0.6330828|
| rcs():2    | -0.365| 0.012     | -3.16| 0.002| -0.500718 - -0.2303272|
| rcs():3    | 0.011 | 0.010     | 1.061| 0.289| 0.000092 - 0.0216916|
| time#M2[id]| 1.267 | 0.207     | 6.168| 0.000| 0.855875 - 0.6783488|
| M1[id]     | 1.035 | 0.073     | 13.92| 0.000| 0.891802 - 1.1787827|
| _cons      | 1.035 | 0.073     | 13.92| 0.000| 0.891802 - 1.1787827|
| sd(resid.) | 0.345 | 0.007     | 50.46| 0.000| 0.3317063 - 0.357957 |
| id:        |       |           |      |     |                      |
| sd(M1)     | 1.012 | 0.043     | 23.47| 0.000| 0.9299325 - 1.0944496|
| sd(M2)     | 0.346 | 0.007     | 50.46| 0.000| 0.3317063 - 0.357957 |

Note by orthogonalizing the splines, we change the interpretation of the intercept. I now have a component with more than one element, time#M2[id]@1—a variable time has been multiplied by a random effect called M2 defined at the id level, with its coefficient constrained to be 1. If I called it M1 again, then there would only be a single random effect in my model but used twice, which would not be sensible in this case. By default, each level’s random effects are assumed to have covariance(diagonal). We can relax this by estimating a correlation,

```
merlin (logb rcs(time, df(3) orthog) time#M2[id]@1 M1[id]@1, > family(gaussian)), covariance(unstructured) nolog
variables created for model 1, component 1: _cmp_1_1_1 to _cmp_1_1_3
Fitting fixed effects model:
Fitting full model:
```

Mixed effects regression model  Number of obs  =  1,945
Log likelihood = -1518.484

```

|            | Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|------------|-------|-----------|------|-----|----------------------|
| logb:      |       |           |      |     |                      |
| rcs():1    | 0.598 | 0.044     | 13.74| 0.000| 0.512481 - 0.6839992|
| rcs():2    | -0.377| 0.011     | -3.38| 0.001| -0.49926 - -0.2558463|
| rcs():3    | 0.016 | 0.009     | 1.794| 0.075| 0.007672 - 0.0245042|
| time#M2[id]| 1.746 | 0.082     | 21.24| 0.000| 1.583972 - 1.90865 |
| M1[id]     | 1.079 | 0.080     | 13.72| 0.000| 0.922693 - 1.235721 |
| _cons      | 0.346 | 0.007     | 50.46| 0.000| 0.3317063 - 0.357957 |
| sd(resid.) | 0.986 | 0.042     | 23.47| 0.000| 0.906653 - 1.072046 |
| id:        |       |           |      |     |                      |
| sd(M1)     | 0.986 | 0.042     | 23.47| 0.000| 0.906653 - 1.072046 |
| sd(M2)     | 0.346 | 0.007     | 50.46| 0.000| 0.3317063 - 0.357957 |
| corr(M2,M1)| 0.434 | 0.074     | 5.872| 0.000| 0.324954 - 0.543296 |
```
Estimating the correlation shows evidence of positive correlation between intercept and slope.

### 3.2 Survival and time-to-event analysis

Many standard time-to-event models are available in `merlin`, including most of those in `streg`, and some more flexible distributions are also built in. This includes the Royston–Parmar model and a log-hazard scale equivalent model, both using restricted cubic splines to model the baseline. To fit a survival model with `merlin`, we simply add the `failure()` option to specify the event indicator.

#### Weibull proportional hazards model

Let’s start with a simple Weibull proportional hazards model, adjusting for treatment, with all parameters estimated on the log scale.

```
. merlin (stime trt, family(weibull, failure(died)))
Fitting full model:
  Iteration 0: log likelihood = -2000.3067
  Iteration 1: log likelihood = -512.23492
  Iteration 2: log likelihood = -511.85192
  Iteration 3: log likelihood = -511.84742
  Iteration 4: log likelihood = -511.84742
Mixed effects regression model
Number of obs = 312
Log likelihood = -511.84742

| Coef.  | Std. Err. |  z    | P>|z| | [95% Conf. Interval] |
|--------|-----------|------|-----|----------------------|
| stime: |           |      |     |                      |
| trt    | -0.0004536| .169053 | -0.00 | 0.998                | -.3317914 -.3308841 |
| _cons  | -2.815926 | .2054485 | -13.71 | 0.000                | -3.218597 -2.413254 |
| log(gamma) | .0740757 | .0752114 | 0.98 | 0.325 | -0.0733359 .2214873 |
```

#### Spline-based survival model

The Royston–Parmar model is now widely used, using restricted cubic splines to model the baseline log cumulative-hazard function and any time-dependent effects:

```
. merlin (stime trt, family(rp, failure(died) df(3)))
variables created: _rcs1_1 to _rcs1_3
Fitting full model:
  Iteration 0: log likelihood = -644.90661
  Iteration 1: log likelihood = -513.25003
  Iteration 2: log likelihood = -511.50222
  Iteration 3: log likelihood = -511.49315
  Iteration 4: log likelihood = -511.49315
```
Mixed effects regression model  

Number of obs = 312  
Log likelihood = -511.12311

| Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|-------|-----------|-----|------|----------------------|
| stime: |           |     |      |                      |
| trt    | 0.0001165 | 0.1690663 | 0.000 | 0.999 | -0.3312474 to 0.3314804 |
| _cons  | -1.088126 | 0.1258163 | -8.650 | 0.000 | -1.334722 to -0.8415308 |

Warning: Baseline spline coefficients not shown - use ml display

I model the baseline with 3 spline terms using the df() option. Note the spline coefficients are not shown by default (they are essentially uninterpretable). You can show all the parameters by using ml display.

Assess proportional hazards

Let’s assess proportional hazards in the effect of treatment by forming an interaction between treatment and log time:

```
. merlin (stime trt trt#fp(stime, powers(0)), family(rp, failure(died) df(3))
> timevar(stime))
```

variables created: _rcs1_1 to _rcs1_3
variables created for model 1, component 2: _cmp_1_2_1 to _cmp_1_2_1

Fitting full model:

Iteration 0: log likelihood = -511.49315

Iteration 1: log likelihood = -511.12311 (not concave)

Iteration 2: log likelihood = -511.12311

Mixed effects regression model  

Number of obs = 312  
Log likelihood = -511.12311

| Coef. | Std. Err. | z    | P>|z| | [95% Conf. Interval] |
|-------|-----------|-----|------|----------------------|
| stime: |           |     |      |                      |
| trt    | -0.2856646 | 0.3732703 | -0.770 | 0.444 | -1.017261 to 0.4459317 |
| trt#fp() | 0.1390466 | 0.1624313 | 0.860 | 0.390 | -0.1788549 to 0.44578641 |
| _cons  | -1.058687 | 0.1293453 | -8.181 | 0.000 | -1.312199 to -0.8051752 |

Warning: Baseline spline coefficients not shown - use ml display

We had to specify our timevar() so merlin knows to handle it differently. Equivalently, we could have used the rcs() element type:

```
. merlin (stime trt trt#rcs(stime, df(1) log), family(rp, failure(died) df(3))
> timevar(stime))
```

(output omitted)
Add a nonlinear effect

We can keep building this model by investigating the effect of age, modeled flexibly using fractional polynomials:

```
. merlin (stime trt fp(age, pow(1 1)) trt#fp(stime, powers(0)), family(rp, failure(died) df(3)) timevar(stime), nolog
variables created: _rcs1_1 to _rcs1_3
variables created for model 1, component 2: _cmp_1_2_1 to _cmp_1_2_2
variables created for model 1, component 3: _cmp_1_3_1 to _cmp_1_3_1
Fitting full model:
Mixed effects regression model Number of obs = 312
Log likelihood = -496.24176
```

| Coef. Std. Err. | z    | P>|z| [95% Conf. Interval] |
|----------------|------|--------|----------------------|
| stime:         |      |        |                      |
| trt            | -0.4355148 | .3744547 | -1.16 | 0.245 | -1.169432 | .2984028 |
| fp():1         | 0.0154085  | .2168584 | 0.07  | 0.943 | -.4096261 | .4404432 |
| fp():2         | 0.0061645  | .0436779 | 0.14  | 0.888 | -.0794425 | .0917716 |
| trt#fp()       | 0.1340058  | .1639403 | 0.82  | 0.414 | -.1873114 | .4553229 |
| _cons          | -3.032566  | 2.298932 | -1.32 | 0.187 | -7.53839  | 1.475288 |

Warning: Baseline spline coefficients not shown - use ml display

Delayed entry/left-truncation

We can allow for delayed entry/left-truncation by using the ltruncated() option. Because we have no delayed entry in this dataset, I simulate it as an illustration:

```
. set seed 42590
. generate t0 = runiform()*stime*0.5
(1,633 missing values generated)
. merlin (stime trt fp(age, pow(1 1)) trt#fp(stime, powers(0)), family(rp, failure(died) df(3) ltruncated(t0)) timevar(stime)), nolog
variables created: _rcs1_1 to _rcs1_3
variables created for model 1, component 2: _cmp_1_2_1 to _cmp_1_2_2
variables created for model 1, component 3: _cmp_1_3_1 to _cmp_1_3_1
Fitting full model:
Mixed effects regression model Number of obs = 312
Log likelihood = -443.56809
```

| Coef. Std. Err. | z    | P>|z| [95% Conf. Interval] |
|----------------|------|--------|----------------------|
| stime:         |      |        |                      |
| trt            | -0.5295243 | .5488949 | -0.96 | 0.335 | -1.605339 | .54629 |
| fp():1         | -0.0826438 | .3254012 | -0.25 | 0.800 | -.7204184 | .5551308 |
| fp():2         | 0.025127  | .0655395 | 0.38  | 0.701 | -.1033279 | .153582 |
| trt#fp()       | 0.1219145  | .1739769 | 0.70  | 0.483 | -.219074  | .462903 |
| _cons          | -9.572162  | 3.433243 | -2.80 | 0.780 | -7.686249 | 5.771817 |

Warning: Baseline spline coefficients not shown - use ml display
### 3.3 Competing risks

Now I will look at a multiple-outcome survival model, for example, the competing risks setting. This is done by specifying cause-specific hazard models. Remember our data structure is simplest in wide format, and within a competing risks setting, our survival time is either censoring or event time; regardless of the event, we simply need cause-specific event indicators. Because this dataset has only all-cause mortality, I randomly assign half of them to represent death from cancer, `cancer`, and the other half to death from other causes, `other`. We can then model each cause-specific hazard in any way we like, for example,

```
. generate cancer = 1 if died==1 & runiform()<0.5
    (1,870 missing values generated)
. generate other = 1 if died==1 & cancer!=1
    (1,880 missing values generated)
. replace cancer = 0 if died==0 | other==1
    (237 real changes made)
. replace other = 0 if died==0 | cancer==1
    (247 real changes made)
. merlin (stime trt rcs(stime, df(3) log), family(loghazard, failure(cancer))
    timevar(stime)) (stime trt, family(rp, failure(other) df(3))), nolog
variables created: _rcs2_1 to _rcs2_3
variables created for model 1, component 2: _cmp_1_2_1 to _cmp_1_2_3
Fitting full model:
```

| Coef. Std. Err. | z   | P>|z| | [95% Conf. Interval] |
|----------------|-----|------|----------------------|
| stime:         |     |      |                      |
| trt            | 0.0506408 | 0.2311454 | 0.22 | 0.827 | -0.4023959 | 0.5036776 |
| rcs():1        | 0.0149 | 0.2605904 | 0.06 | 0.954 | -0.4958477 | 0.5258478 |
| rcs():2        | 0.7667253 | 0.9978469 | 0.77 | 0.442 | -1.189019 | 2.722469 |
| rcs():3        | -1.445388 | 1.832751 | -0.79 | 0.430 | -5.037514 | 2.146737 |
| _cons          | -3.282338 | 0.3259485 | -10.07 | 0.000 | -3.921185 | -2.643491 |
| stime:         |     |      |                      |
| trt            | -0.0618849 | 0.2481256 | -0.25 | 0.803 | -0.5482022 | 0.4244324 |
| _cons          | -1.85768 | 1.837268 | -10.11 | 0.000 | -2.217778 | -1.497582 |

Warning: Baseline spline coefficients not shown - use ml display

This models cause 1 with splines on the log-hazard scale and cause 2 with splines on the log cumulative-hazard scale.

Let’s calculate the cumulative incidence functions for each cause using the `predict` tools. I will generate a time variable and calculate the predictions for a patient in the treated group.

```
. range tvar 0 10 100
    (1,845 missing values generated)
. predict cifi1, cif outcome(1) causes(1 2) timevar(tvar) at(trt 1)
    (1846 missing values generated)
```
By default, `causes()` will include all models, but I am being explicit here for clarity. To create a stacked plot, we need to add the two cumulative incidence functions and then use `area` graphs:

```
. predict cif2, cif outcome(2) causes(1 2) timevar(tvar) at(trt 1)
(1846 missing values generated)
```

To create a stacked plot, we need to add the two cumulative incidence functions and then use `area` graphs:

```
. generate totalcif1 = cif1 + cif2
(1,846 missing values generated)
. twoway (area totalcif1 tvar)(area cif2 tvar), name(g1,replace)
   > xtitle("Time since entry") ytitle("Cumulative incidence")
   > title("Treated group") legend(cols(1)
   >   order(1 "Prob. of death because of cancer"
   >   2 "Prob. of death because of other causes"))
   > ylabel(,angle(h) format(%2.1f)) ylabel(0(0.1)1)
```

![Figure 1. Stacked cumulative incidence functions](image)

### 3.4 Joint modeling of longitudinal and survival data

Now I will move into the field of joint longitudinal survival models. Joint models were first proposed by linking the submodels through the random effects only, such as

```
. merlin (stime trt M1[id] , family(weibull, failure(died)))
   > (logb time M1[id]@1, family(gaussian))
Fitting fixed effects model:
Fitting full model:
Iteration 0:  loglikelihood = -3095.8911
Iteration 1:  loglikelihood = -2494.924
Iteration 2:  loglikelihood = -2369.5547
Iteration 3:  loglikelihood = -2310.8764
Iteration 4:  loglikelihood = -2310.5375
Iteration 5:  loglikelihood = -2310.5374
```
Mixed effects regression model

|                | Coef.  | Std. Err. | z     | P>|z| | [95% Conf. Interval] |
|----------------|--------|-----------|-------|-----|----------------------|
| **stime:**     |        |           |       |     |                      |
| trt            | 0.02319 | 0.17490 | 0.13  | 0.895 | -0.3196098 - 0.3659871 |
| M1[id]         | 1.23391 | 0.10339 | 11.93 | 0.000 | 1.031273 - 1.436557  |
| _cons          | -3.84620 | 0.28871 | -14.31 | 0.000 | -4.372892 - 3.319509 |
| log(gamma)     | 0.42218 | 0.07112 | 5.94  | 0.000 | 0.2827999 - 0.5615706 |
| **logb:**      |        |           |       |     |                      |
| time           | 0.09766 | 0.00432 | 22.61 | 0.000 | 0.0891962 - 0.1061268 |
| M1[id]         | 1       |          |       |     |                      |
| _cons          | 0.57775 | 0.06496 | 8.89  | 0.000 | 0.4506404 - 0.7050897 |
| sd(resid.)     | 0.49121 | 0.08584 | 0.47467 | 1.203824 |
| **id:**        |        |           |       |     |                      |
| sd(M1)         | 1.10726 | 0.04706 | 1.019099 | 1.203797 |

By including my random intercept M1 in the complex predictor for the survival outcome, and not specifying a constraint, we will obtain an association parameter for the relationship between the subject-specific random intercept of the biomarker and survival. Alternatively, and now more commonly, we can use the EV[] element type to link the time-dependent expected value of the biomarker directly to the risk of event. Given we are modeling time, timevar() must be specified in both submodels because it will be integrated over in the survival model likelihood contribution.
There is no limit in how many associations and what form we specify; for example, we can also link the cumulative value of the biomarker (think of it as cumulative exposure) directly to survival.

```
. merlin (stime trt EV[logb] iEV[logb], family(weibull, failure(died)))
> timevar(stime)) (logb fp(time, pow(1)) M1[id]$1, family(gaussian)
> timevar(time)), nolog
variables created for model 2, component 1: _cmp_2_1_1 to _cmp_2_1_1
Fitting fixed effects model:
Fitting full model:
Mixed effects regression model
Number of obs = 1,945
Log likelihood = -2306.8166

|              | Coef. Std. Err. | z     | P>|z|   | [95% Conf. Interval] |
|--------------|-----------------|-------|-------|---------------------|
| stime:       |                 |       |       |                     |
| trt          | .0504003        | .1769845 | .28 | .776 | -.2964829 .3972834 |
| EV[]         | 1.454449        | .1759921 | 8.26 | .000 | 1.109511 1.799387 |
| iEV[]        | -.0397346       | .0309956 | -1.28 | .200 | -.1004848 .0210156 |
| _cons        | -4.897322       | .4150164 | -11.80 | .000 | -5.710739 -4.083905 |
| log(gamma)   | .288128         | .1054812 | 2.73 | .006 | .0813887 .4948674 |
| logb:        |                 |       |       |                     |
| fp()         | .0982338        | .0043145 | 22.77 | .000 | .0897775 .10669 |
| M1[id]       | 1               | .   | .   | .   |                     |
| _cons        | .578134         | .0650171 | 8.89 | .000 | .4507029 .7055651 |
| sd(resid.)   | .4912445        | .0085862 | 8.89 | .000 | .4747009 .5083647 |
| id:          |                 |       |       |                     |
| sd(M1)       | 1.108471        | .0471066 | 1.019885 | 1.204753 |
```
Nonlinearities in the association and time-dependent effects can also be specified.

```
. merlin (stime trt EV[logb] EV[logb]#fp(stime, pow(0)), family(weibull,
> failure(died)) timevar(stime)) (logb fp(time, pow(1)) M1[id]@1,
> family(gaussian) timevar(time)), nolog
variables created for model 1, component 3: _cmp_1_3_1 to _cmp_1_3_1
variables created for model 2, component 1: _cmp_2_1_1 to _cmp_2_1_1
Fitting fixed effects model:
Fitting full model:
Mixed effects regression model
Log likelihood = -2306.9583
Number of obs = 1,945

|               | Coef.   | Std. Err. |    z  |   P>|z|   | [95% Conf. Interval] |
|---------------|---------|-----------|-------|-------|---------------------|
| stime:        |         |           |       |       |                     |
| trt           | .0474131| .1769711  | 0.27  | 0.789 | -.2994439 to .3942701|
| EV[       | 1.426937| .1685874  | 8.46  | 0.000 | 1.096511 to 1.757362|
| EV[       #fp()| -.1238264| .1034565  | -1.20 | 0.231 | -.3265974 to .0789447|
| _cons        | -.039484| .5227464  | -9.64 | 0.000 | -6.064048 to -4.014919|
| log(gamma)   | .3605693| .1549547  | 2.33  | 0.020 | .0568638 to .6642749|
| logb:        |         |           |       |       |                     |
| fp()          | .0980068| .0043171  | 22.70 | 0.000 | .0895455 to .1064682|
| M1[id]       | 1.00000 | .000000   | 1.00  | 0.000 | .000000 to .000000  |
| _cons        | .5792938| .0650543  | 9.00  | 0.000 | .4517897 to .7067979|
| sd(resid.)   | .4912266| .0085855  | 8.90  | 0.000 | .4746863 to .5083473|
| id:          |         |           |       |       |                     |
| sd(M1)       | 1.108874| .0471116  | 1.02  | 0.027 | 1.020277 to 1.205165|
```

3.5 A final model

Now I am going to combine many of the previous models into one merlin model. This is purely for illustration but hopefully gives you an idea of just how flexible merlin can be. I will now generate a binary, repeatedly measured variable, catpro, that is prothrombin merely categorized into above and below 12 (this is clearly an unnecessary thing to do but purely for illustration). I will also use the artificial competing risks outcomes created above and so combine a joint model for two repeatedly measured outcomes, one continuous and one binary, with cause-specific competing risks survival models. Each cause-specific hazard model will have a different distribution, and we can allow for time-dependent effects and nonproportional hazards.
. generate byte catpro = prothrombin > 12
. merlin (stime trt M2[id] M1[id] rcs(stime, df(3) log),
>   family(loghazard, failure(cancer)) timevar(stime))
> (stime trt trt#fp(stime, pow(0)) EV[logb] M1[id],
>   family(weibull, failure(other)) timevar(stime))
> (logb rcs(time, df(3) orthog) M1[id]@1, family(bernoulli) timevar(time)),
>   covariance(unstructured) nolog
variables created for model 1, component 4: _cmp_1_4_1 to _cmp_1_4_3
variables created for model 2, component 2: _cmp_2_2_1 to _cmp_2_2_1
variables created for model 3, component 1: _cmp_3_1_1 to _cmp_3_1_3
variables created for model 4, component 1: _cmp_4_1_1 to _cmp_4_1_1
Fitting fixed effects model:

Fitting full model:
Mixed effects regression model

Number of obs = 1,945
Log likelihood = -2908.5363

| Coef. | Std. Err. | z   | P>|z| | [95% Conf. Interval] |
|-------|-----------|-----|------|----------------------|
| stime:
| trt   | .0679753  | .2579126 | 0.26 | 0.792 | -.4375241 | .5734747 |
| M2[id]| .5788921  | .1429466 | 4.05 | 0.000 | .2987219 | .8590622 |
| M1[id]| .6114203  | .2338957 | 2.61 | 0.009 | .1529932 | 1.069847 |
| rcs():1 | .0557385  | .2684125 | 0.21 | 0.835 | -.4703403 | .5818172 |
| rcs():2 | 1.011354  | 1.054479 | 0.96 | 0.338 | -.1055387 | 3.078096 |
| rcs():3 | -2.173861 | 1.942443 | -1.12 | 0.263 | -5.980979 | 1.632368 |
| _cons | -5.393272 | .5535125 | -9.74 | 0.000 | -6.478137 | -4.308408 |
| stime:
| trt   | -.0432462 | .4161506 | -0.10 | 0.917 | -.8588863 | .7723939 |
| trt#fp() | -.0011682 | .2759248 | -0.00 | 0.997 | -.5419709 | .5396345 |
| EV[logb] | 1.077974 | 1.307879 | 0.82 | 0.410 | -1.485423 | 3.64137 |
| M1[id] | .1663005  | 1.281548 | 0.13 | 0.897 | -2.345488 | 2.678089 |
| _cons | 5.311917  | 1.307879 | 3.96 | 0.000 | -4.001499 | 13.62538 |
| log(gamma) | .2805833 | .2963242 | 0.95 | 0.344 | -.3002015 | .8613681 |
| logb:
| rcs():1 | .3045531  | .1233323 | 23.01 | 0.000 | .2786163 | .3304898 |
| rcs():2 | .0679448  | .0118596 | 5.73 | 0.000 | .0467003 | .0891933 |
| rcs():3 | .0136825  | .0114779 | 1.19 | 0.233 | -.008139 | .0361788 |
| M1[id] | 1.1663005 | 1.281548 | 0.13 | 0.897 | -2.345488 | 2.678089 |
| _cons | 5.311917  | 1.307879 | 3.96 | 0.000 | -4.001499 | 13.62538 |
| sd(resid.) | .4865402 | .008522 | 47.01209 | 5035329 |
| catpro:
| fp() | .5175683  | .0452748 | 11.43 | 0.000 | .4288312 | .6063053 |
| M2[id] | 1.115696  | .0473045 | 2.38549 | 3.450572 |
| _cons | -4.612572 | .3375689 | -13.66 | 0.000 | -5.274195 | -3.950949 |
| id:
| sd(M1) | 1.115696  | .0473045 | 2.38549 | 3.450572 |
| sd(M2) | 2.876868  | .2669049 | 10.80549 | 3.450572 |
| corr(M2,M1) | .7832839 | .0349725 | 16.13549 | 3.450572 |

There are four outcome models and 2 random effects, so it takes some time, but we get there in about 16 minutes on my laptop. It is trivial to extend this model, for example, by adding random slopes or different ways of linking the submodels or
incorporating nonlinear effects, be they for continuous covariates or time. Importantly, we can use the predict function to obtain easily understandable measures of risk, such as cumulative incidence functions, even from such a complex joint model.

4 Discussion

I have given a broad overview of merlin’s capabilities, and potential, in the field of data analysis. I have described the fundamental syntax and, through worked examples, illustrated some of the areas of statistical modeling that can be applied using merlin.

4.1 The curse of generality

When you implement a software package that can do many things, one of the challenging tasks is to balance said generality with computational speed. merlin is not the quickest. Its current implementation is a gf0 evaluator, which means it uses Stata’s internal finite difference routines within the ml engine to calculate the score and Hessian, which means a lot of calls to the evaluator program. Furthermore, merlin has to cover many different settings and options, which inevitably means storing a lot of information within its main object. You will get speed gains if you implement a specific command, from scratch, designed for a specific setting.

4.2 Wrapper commands

First, the syntax of merlin is not the simplest. This is of course because it has to accommodate a lot of different options and techniques. This opens up more room to go wrong. Second, it is rather challenging to obtain good starting values for a command that can fit anything, and good starting values can be crucial to improve convergence.

This motivates the writing of wrapper commands, that is, ado-files specifically written to handle specific classes of merlin models. This allows a much simpler and cleaner syntax and the ability to hard-code initial-value fitting routines. Under the hood, and unbeknown to most users, the shell command will call merlin. The first of these is stmixed (Crowther 2019). I am working on a few more of these.

4.3 Concluding remarks

There is a multitude of future directions in which to take merlin. This includes adding analytic derivatives for computational speed gains, extending the allowed random-effects distributions to allow things such as mixtures of Gaussians, and providing more tools for postestimation, including dynamic prediction capabilities. In a future article, I plan to focus on the user-defined and custom distribution capabilities of merlin.
5 Programs and supplemental materials

To install a snapshot of the corresponding software files as they existed at the time of publication of this article, type

```
    . net sj 20-4
    . net install st0616 (to install program files, if available)
    . net get st0616 (to install ancillary files, if available)
```

The latest stable version of `merlin` can be installed by typing

```
    . ssc install merlin
```

The development version can be installed using

```
    . net install merlin, from(https://www.mjcrowther.co.uk/code/merlin)
```

Further examples can be found at https://www.mjcrowther.co.uk/software/merlin/tutorials_stata/.

6 References

Bartus, T. 2017. Multilevel multiprocess modeling with gsem. *Stata Journal* 17: 442–461. https://doi.org/10.1177/1536867X1701700211.

Crowther, M. J. 2017. Extended multivariate generalised linear and non-linear mixed effects models. ArXiv Working Paper No. arXiv:1710.02223. http://arxiv.org/abs/1710.02223.

———. 2019. Multilevel mixed-effects parametric survival analysis: Estimation, simulation, and application. *Stata Journal* 19: 931–949. https://doi.org/10.1177/1536867X19893639.

Crowther, M. J., K. R. Abrams, and P. C. Lambert. 2013. Joint modeling of longitudinal and survival data. *Stata Journal* 13: 165–184. https://doi.org/10.1177/1536867X1301300112.

Crowther, M. J., and P. C. Lambert. 2013. stgenreg: A Stata package for general parametric survival analysis. *Journal of Statistical Software* 53(12): 1–17.

———. 2014. A general framework for parametric survival analysis. *Statistics in Medicine* 33: 5280–5297.

Crowther, M. J., M. P. Look, and R. D. Riley. 2014. Multilevel mixed effects parametric survival models using adaptive Gauss–Hermite quadrature with application to recurrent events and individual participant data meta-analysis. *Statistics in Medicine* 33: 3844–3858.

Murtaugh, P. A., E. R. Dickson, G. M. V. Dam, M. Malinchoc, P. M. Grambsch, A. L. Langworthy, and C. H. Gips. 1994. Primary biliary cirrhosis: Prediction of short-term survival based on repeated patient visits. *Hepatology* 20: 126–134.
Rabe-Hesketh, S., A. Skrondal, and A. Pickles. 2002. Reliable estimation of generalized linear mixed models using adaptive quadrature. *Stata Journal* 2: 1–21. https://doi.org/10.1177/1536867X0200200101.

———. 2005. Maximum likelihood estimation of limited and discrete dependent variable models with nested random effects. *Journal of Econometrics* 128: 301–323.

Roodman, D. 2011. Fitting fully observed recursive mixed-process models with cmp. *Stata Journal* 11: 159–206. https://doi.org/10.1177/1536867X1101100202.

**About the author**

Michael J. Crowther is an associate professor of biostatistics at the University of Leicester. He works heavily in methods and software development, particularly in the field of survival analysis. This work was supported by an MRC New Investigator Research Grant (MR/P015433/1).