rpsurv: Fast Royston-Parmar Flexible Parametric Survival Models

Imad El Badisy

2026-11-02

Contents

1 Introduction **1**

2 The Royston-Parmar model **2**
2.1 Spline basis 2
2.2 Linear predictor and scales 2
2.3 Log-likelihood 3
2.4 Analytic gradient 3

3 Fitting API and output **3**

4 Prediction and plots **4**

5 Time-varying effect vs. time-varying covariate **4**
5.1 Time-varying effect (non-proportional hazards) 5
5.2 Time-varying covariate 5

6 Diagnostics **6**

7 Validation against rstpm2 and flexsurv **6**

8 Speed benchmark **7**
8.1 What is delegated to C++ 7
8.2 Benchmark 7

9 Session info **9**

1 Introduction

`rpsurv` fits the Royston & Parmar (2002) family of flexible parametric survival models: the baseline distribution is represented by a restricted cubic spline in $\log t$ on a chosen transformed scale (proportional hazards, proportional odds, or probit), which lets a smooth, monotone hazard/survival shape be estimated without committing to a Weibull, log-normal, or other single parametric family, while still producing a fully parametric, smooth, extrapolable fit (unlike the Cox model).

`rpsurv` re-implements this model with the log-likelihood and its analytic gradient evaluated in C++ via `RcppParallel`, so that fitting scales to hundreds of thousands of rows in about a second (see the Speed benchmark section). Coefficients, standard errors, and log-likelihoods are validated against `rstpm2::stpm2` and `flexsurv::flexsurvspline` (the Validation section).

This vignette covers:

- the spline basis and the likelihood/gradient for all three scales,

- the fitting API and coxph-style output,
- prediction and plotting helpers,
- the distinction between a **time-varying effect** and a **time-varying covariate**, and how each is fit,
- residual diagnostics,
- the speed benchmark against `rstpm2` and `flexsurv`, and what is delegated to C++ to get there.

2 The Royston-Parmar model

2.1 Spline basis

Let $x = \log t$. A restricted (natural) cubic spline with boundary knots $k_{\min}, k_{\max}$ and interior knots $k_1, \ldots, k_m$ is written, in the Durrleman & Simon / Royston & Parmar parameterisation, as

$$s(x) = \gamma_0 + \gamma_1 x + \sum_{j=1}^{m} \gamma_{1+j}\, v_j(x),$$

where the linear term keeps the domain $x \in \mathbb{R}$ and each nonlinear basis term is

$$v_j(x) = (x - k_j)_+^3 - \lambda_j (x - k_{\min})_+^3 - (1 - \lambda_j)(x - k_{\max})_+^3, \qquad \lambda_j = \frac{k_{\max} - k_j}{k_{\max} - k_{\min}},$$

with $(u)_+ = \max(u, 0)$. This is exactly the basis in `rcs_basis()` (`R/splines.R`); by construction $s(x)$ is linear beyond the boundary knots, which keeps extrapolation well-behaved. Its derivative, needed for the hazard, is

$$v_j'(x) = 3(x - k_j)_+^2 - 3\lambda_j (x - k_{\min})_+^2 - 3(1 - \lambda_j)(x - k_{\max})_+^2.$$

Boundary knots default to the min/max of $\log t$ among **events**, and interior knots to equally spaced centiles of $\log t$ among events (`default_knots()`), matching `rstpm2`/`flexsurv` defaults.

2.2 Linear predictor and scales

Covariates enter proportionally on the modelled scale:

$$\eta(t \mid x) = s(\log t) + \beta^\top x.$$

`rpsurv` supports three transformations g of the survival function, selected via `scale`:

`scale`	$g(S) = \eta$	$S(\eta)$	Interpretation of $\exp(\beta)$
`"hazard"`	$\log(-\log S)$	$\exp(-\exp(\eta))$	hazard ratio (PH)
`"odds"`	$\log\{(1-S)/S\}$	$1/(1+\exp(\eta))$	odds ratio (PO)
`"normal"`	$\Phi^{-1}(1-S)$	$1 - \Phi(\eta)$	probit index

The hazard follows from differentiating $S(t) = S(\eta(t))$ with respect to t via the chain rule through $x = \log t$:

$$h(t) = \frac{f(t)}{S(t)} = -\frac{d\eta}{d\log t} \cdot \frac{1}{t} \cdot \frac{S'(\eta)}{S(\eta)}.$$

Writing $g(\eta) = \log h(t) - \log\!\left(\frac{d\eta}{d\log t}\right) + \log t$ (a function of η alone, on each scale), the three cases reduce to closed forms:

$$g(\eta) = \begin{cases} \eta & \text{hazard scale} \\ \eta - \log(1 + e^{\eta}) & \text{odds scale} \\ \log \phi(\eta) - \log\{1 - \Phi(\eta)\} & \text{probit scale} \end{cases}$$

so that $\log h(t) = \log\left(\frac{d\eta}{d \log t}\right) - \log t + g(\eta)$. This is exactly `scale_terms()` in `src/rpsurv_loglik.cpp`.

2.3 Log-likelihood

For a right-censored observation with event time t_i, status d_i, and (for left-truncated / counting-process data) entry time e_i, the contribution is

$$\ell_i(\beta) = d_i \log h(t_i) + \log S(t_i) - \log S(e_i),$$

with $\log S(e_i) \equiv 0$ when $e_i = 0$ (no truncation). Summing gives the full log-likelihood $\ell(\beta) = \sum_i \ell_i(\beta)$, maximised by `stats::optim(method = "BFGS")` using the analytic gradient below.

2.4 Analytic gradient

Because η and $\frac{d\eta}{d \log t}$ are both linear in β (through the design matrices X and $\dot{X} = dX/d \log t$), the gradient of ℓ_i with respect to β_k is

$$\frac{\partial \ell_i}{\partial \beta_k} = d_i \left[\frac{\dot{X}_{ik}}{\dot{\eta}_i} + g'(\eta_i) X_{ik} \right] + \frac{d \log S}{d\eta}(\eta_i)\, X_{ik} - \mathbb{1}[e_i > 0]\, \frac{d \log S}{d\eta}(\eta_{e_i})\, X^{(e)}_{ik},$$

where $X^{(e)}$ is the design row evaluated at $\log e_i$. The scale-specific pieces $g'(\eta)$ and $d \log S/d\eta$ are, again, closed form (hazard: $g' = 1$, $d \log S/d\eta = -e^{\eta}$; odds: $g' = S$, $d \log S/d\eta = -(1 - S)$; probit: $g' = -\eta + \phi(\eta)/S$, $d \log S/d\eta = -\phi(\eta)/S$). Evaluating ℓ and $\nabla \ell$ this way, rather than by finite differences, avoids the $O(p)$ extra likelihood evaluations per optimiser step that a numerical gradient would cost, which is the main source of `rpsurv`'s speed advantage together with parallel evaluation across observations (`RcppParallel::parallelReduce`, `src/rpsurv_loglik.cpp`).

An observation with $d_i = 1$ and $\dot{\eta}_i \le 0$ (a locally decreasing, i.e. invalid, hazard) is given a large penalty rather than `NaN`, likewise any row where $S(e_i) < S(t_i)$ (non-monotone survival across a truncation interval); both keep the optimiser inside the region where the model is a valid survival distribution.

3 Fitting API and output

```
data(brcancer)
brcancer$hormon <- as.numeric(brcancer$hormon)

fit <- rpsurv(Surv(rectime, censrec) ~ hormon, data = brcancer, df = 4, scale = "hazard")
summary(fit)
#> Royston-Parmar flexible parametric survival model
#> Call:
#> rpsurv(formula = Surv(rectime, censrec) ~ hormon, data = brcancer,
#>     df = 4, scale = "hazard")
#>
#> Covariate effects (hazard scale):
#>           coef exp(coef) [HR] se(coef)      z Pr(>|z|)
#> hormon -0.3641         0.6948   0.1249 -2.914  0.00356 **
#> ---
```

```
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Baseline spline terms:
#>                   coef  exp(coef)   se(coef)      z Pr(>|z|)
#> (Intercept) -1.960e+01  3.086e-09  3.257e+00 -6.016 1.79e-09
#> s(logt)1     2.993e+00  1.994e+01  5.966e-01  5.016 5.26e-07
#> s(logt)2    -4.429e-01  6.422e-01  5.711e-01 -0.776   0.4380
#> s(logt)3     1.357e+00  3.884e+00  8.188e-01  1.657   0.0974
#> s(logt)4    -8.841e-01  4.131e-01  4.176e-01 -2.117   0.0342
#>
#> n = 686 , number of events = 299 , parameters = 6
#> Log-likelihood = -2606   AIC = 5225   BIC = 5247
```

The summary separates interpretable covariate effects (with hazard/odds ratios and Wald tests) from the baseline spline coefficients, which are nuisance parameters describing the shape of $h_0(t)$ and are not individually interpretable.

4 Prediction and plots

`predict()` returns survival, cumulative hazard, hazard, or the linear predictor, optionally with delta-method confidence limits; `plot()` wraps it for quick visualisation.

```
plot(fit, newdata = data.frame(hormon = c(0, 1)), col = c("steelblue", "firebrick"),
     main = "rpsurv: predicted survival")
```

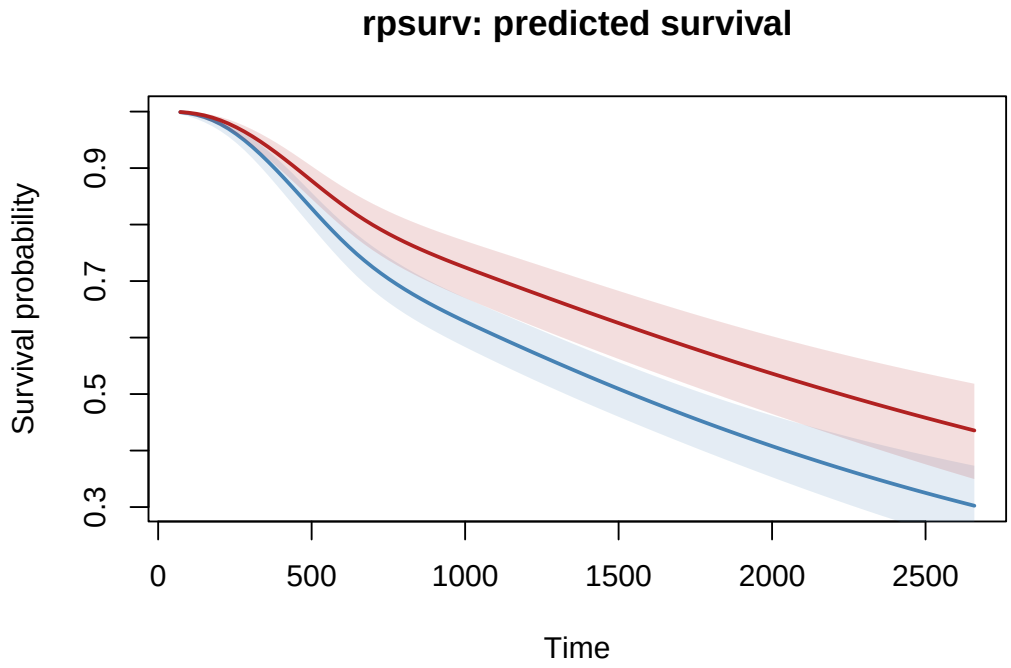


Figure 1: Predicted survival by hormonal therapy status.

5 Time-varying effect vs. time-varying covariate

These are two different extensions of the base model, and `rpsurv` supports both, separately or together.

5.1 Time-varying effect (non-proportional hazards)

A covariate's *value* is fixed for a subject, but its *coefficient* $\beta(t)$ is allowed to change over follow-up: instead of βx, the linear predictor gets $x \cdot s_{\text{tve}}(\log t)$, i.e. its own spline in log time multiplying the covariate. Request this with `tve`:

```
fit_tve <- rpsurv(Surv(rectime, censrec) ~ hormon, data = brcancer,
                  df = 4, tve = "hormon", tve.df = 2)
coef(fit_tve)
#>     (Intercept)        s(logt)1        s(logt)2        s(logt)3        s(logt)4
#>    -19.19777720      2.93098413     -0.42391914      1.32597578     -0.87854493
#>          hormon hormon:s(logt)1 hormon:s(logt)2
#>     -2.34822990      0.32187562      0.03601483
```

The `hormon:s(logt)*` coefficients trace out how the hormonal-therapy hazard ratio evolves with time; a likelihood-ratio test against `fit` (both fitted by maximum likelihood, nested models) tests proportionality:

```
lrt_stat <- 2 * (fit_tve$loglik - fit$loglik)
lrt_df <- fit_tve$df - fit$df
pchisq(lrt_stat, lrt_df, lower.tail = FALSE)
#> [1] 0.8122301
```

5.2 Time-varying covariate

Here the covariate's *value itself* changes during follow-up (e.g. a biomarker updated at clinic visits). This has nothing to do with `tve`; it is a data-representation question. Split each subject's follow-up into consecutive intervals over which the covariate is constant, and supply counting-process data via `Surv(start, stop, status)`:

```
d <- brcancer
d$id <- seq_len(nrow(d))
half <- d$rectime / 2
first  <- data.frame(id = d$id, start = 0, stop = half, status = 0,
                     hormon = d$hormon, x1 = d$x1)
second <- data.frame(id = d$id, start = half, stop = d$rectime, status = d$censrec,
                     hormon = d$hormon, x1 = d$x1 + 1)  # covariate value changes
long <- rbind(first, second)
long <- long[long$start < long$stop, ]

fit_cp <- rpsurv(Surv(start, stop, status) ~ hormon + x1, data = long, df = 4)
fit_cp$counting
#> [1] TRUE
coef(fit_cp)
#>   (Intercept)      s(logt)1      s(logt)2      s(logt)3      s(logt)4
#> -19.826159484   2.991760891  -0.438400286   1.348304243  -0.878671736
#>        hormon            x1
#>  -0.385157266   0.004545335
```

`rpsurv` detects the 3-column `Surv` object and fits the left-truncated likelihood of Section 2 automatically: each interval contributes $\log S(\text{stop}) - \log S(\text{start})$, correctly accounting for the subject having already survived to `start` under the covariate value(s) of the *previous* interval. The same mechanism (`Surv(start, stop, status)`) also handles ordinary left truncation (delayed entry) when the covariate values do not change.

6 Diagnostics

```
coxsnell_plot(fit)
```

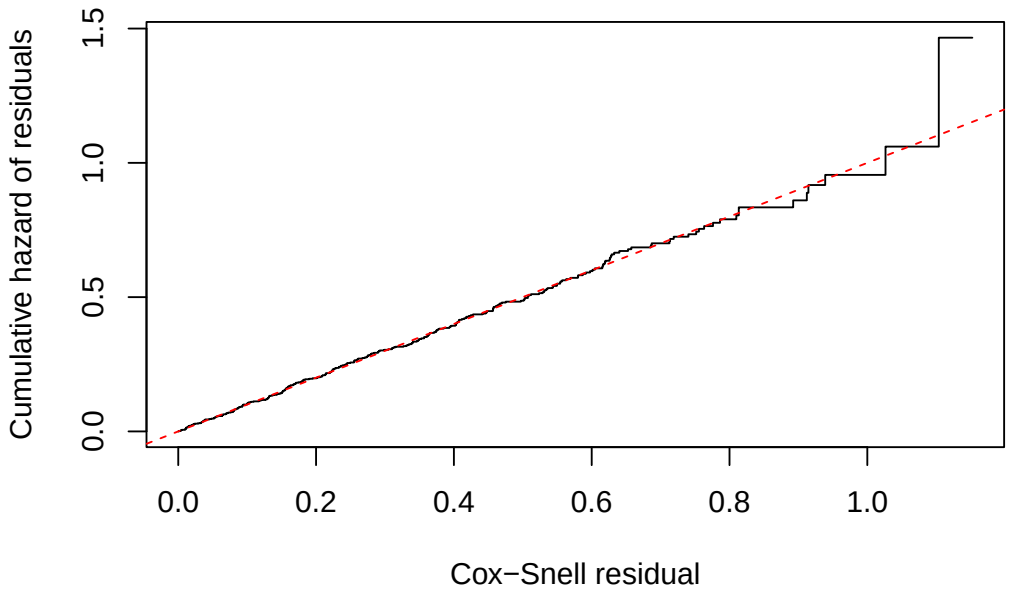


Figure 2: Cox-Snell residual check: points should lie near the diagonal.

```
dev_resid <- residuals(fit, type = "deviance")
summary(dev_resid)
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> -1.51819 -1.07107 -0.68466 -0.03882  0.98077  3.40559
```

`km_compare_plot()` overlays the model-implied survival curve on the nonparametric Kaplan-Meier estimate, by strata of a covariate: a direct visual check of whether the chosen spline `df` and `scale` capture the data:

```
km_compare_plot(fit, by = "hormon")
```

7 Validation against rstpm2 and flexsurv

```
suppressPackageStartupMessages(library(rstpm2))
ref <- stpm2(Surv(rectime, censrec) ~ hormon, data = brcancer, df = 4)
data.frame(
  term = c("hormon", "logLik"),
  rpsurv = c(coef(fit)["hormon"], fit$loglik),
  stpm2  = c(coef(ref)["hormon"], -ref@min)
)
#>          term        rpsurv         stpm2
#> hormon hormon    -0.3640642    -0.3640574
#>        logLik -2606.4716870 -2606.4716869
```

Coefficients, standard errors, and log-likelihoods agree with `rstpm2` to 4-5 decimal places across all three scales (`hazard`, `odds`, `normal`) and with `flexsurv::flexsurvspline` on the hazard scale; see `tests/testthat/test-rpsurv.R` for the full parity suite, which also covers left truncation.

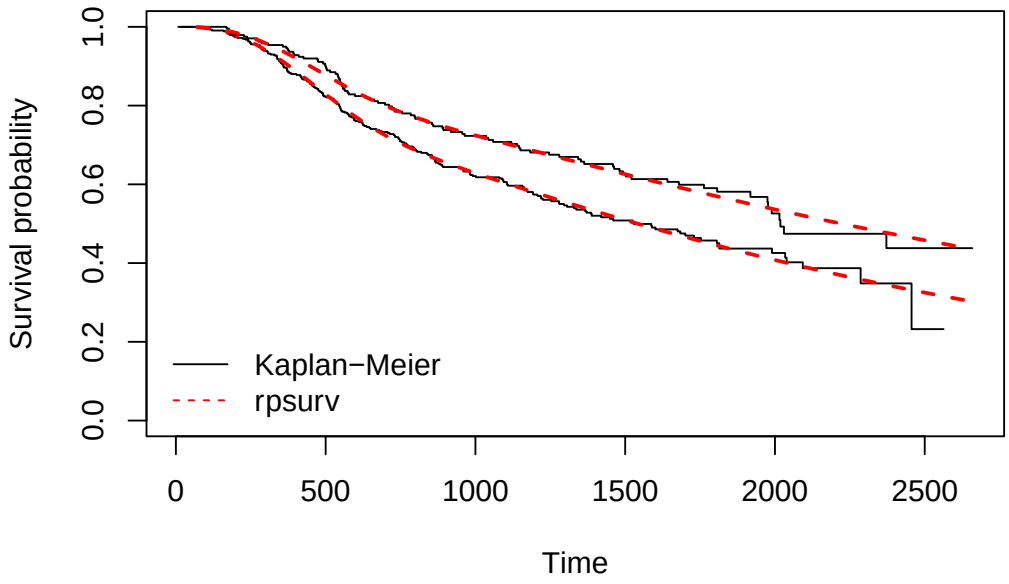


Figure 3: Model vs. Kaplan-Meier, by hormonal therapy status.

8 Speed benchmark

8.1 What is delegated to C++

Everything on the hot path of fitting is C++, not just the likelihood sum:

- the log-likelihood **and** its analytic gradient are computed in a single parallel pass (`RcppParallel::parallelReduce`) over observations, rather than two separate passes;
- `optim(method = "BFGS")` calls the objective (`fn`) and the gradient (`gr`) separately at every trial point, which would otherwise mean two full data passes per iteration for the *same* point; `rpsurv` caches the fused C++ result keyed on the parameter vector, so each distinct point is evaluated exactly once;
- the restricted cubic spline basis itself (`rcs_basis()`, both the basis and its derivative) is computed by `rcs_basis_cpp()` in C++ rather than with R's vectorised-but-still-interpreted arithmetic;
- the closed-form starting values (`rp_start_values()`) are computed from a bounded subsample (default 20,000 rows) of the Kaplan-Meier curve; they only need to be in the right neighbourhood for BFGS, so this avoids an $O(n \log n)$ sort of the full data purely for initialisation.

What remains in R (`model.frame()`/`model.matrix()` formula parsing, one `cbind()` to assemble the design matrix) runs once per fit, not once per optimiser iteration, and is a small fraction of total time at the sample sizes below.

8.2 Benchmark

The figure and table compare wall-clock fit time against both canonical flexible parametric survival packages, `rstpm2::stpm2` and `flexsurv::flexsurvspline`, on simulated Weibull-hazard data with two covariates (reproducible via `data-raw/benchmark.R`, shipped with the package source). `flexsurv::flexsurvspline` is materially slower per fit, so it is only run up to $n = 2 \times 10^4$ to keep the benchmark tractable; `rstpm2::stpm2` is run up to $n = 2 \times 10^5$.

n	rpsurv (s)	stpm2 (s)	flexsurvspline (s)
1e+03	0.031	0.017	0.121
5e+03	0.037	0.058	0.636
2e+04	0.080	0.238	2.330

n	rpsurv (s)	stpm2 (s)	flexsurvspline (s)
5e+04	0.165	1.290	NA
1e+05	0.247	2.410	NA
2e+05	0.413	4.912	NA
5e+05	1.182	NA	NA

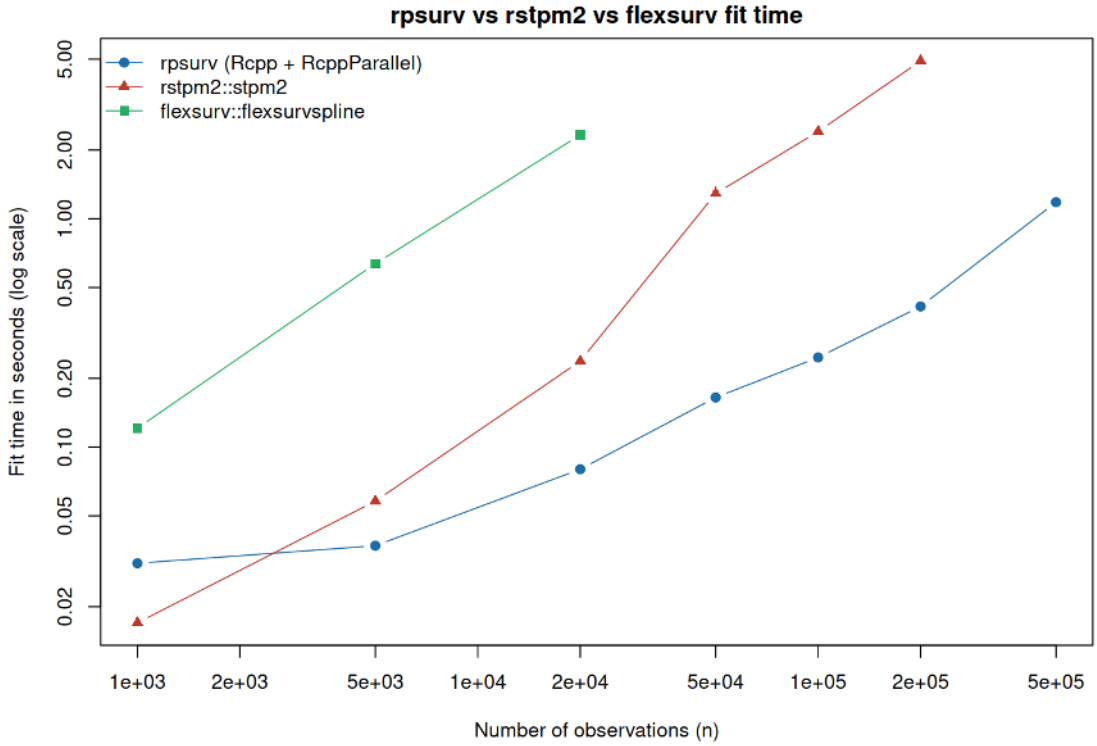


Figure 4: Fit time vs. sample size (log-log axes).

At $n = 2 \times 10^5$, `rpsurv` is about 11.9x faster than `rstpm2::stpm2`, and continues to scale to $n = 5 \times 10^5$ (`rstpm2` was not run at that size). Against `flexsurv::flexsurvspline`, at $n = 2 \times 10^4$ `rpsurv` is about 29.1x faster. The advantage widens with n in both comparisons, consistent with `rpsurv` doing $O(n)$ work per optimiser step against a single parallel pass, versus repeated R-level likelihood evaluation (numerical gradients, or gradients built from R-level matrix algebra) in the other two packages. The benchmark code itself:

```
simulate_data <- function(n, seed = 1) {
  set.seed(seed)
  x1 <- rnorm(n); x2 <- rbinom(n, 1, 0.5)
  lp <- 0.5 * x1 - 0.3 * x2
  u <- runif(n)
  event_time <- (-log(u) / (0.01 * exp(lp)))^(1 / 1.2)
  cens_time <- rexp(n, 0.008)
  data.frame(time = pmin(event_time, cens_time),
             status = as.numeric(event_time <= cens_time), x1 = x1, x2 = x2)
}
d <- simulate_data(200000)
system.time(rpsurv(Surv(time, status) ~ x1 + x2, data = d, df = 4))
```

```
system.time(stpm2(Surv(time, status) ~ x1 + x2, data = d, df = 4))
system.time(flexsurvspline(Surv(time, status) ~ x1 + x2, data = d, k = 3))
```

9 Session info

```
sessionInfo()
#> R version 4.5.1 (2025-06-13)
#> Platform: x86_64-pc-linux-gnu
#> Running under: Ubuntu 24.04.3 LTS
#>
#> Matrix products: default
#> BLAS/LAPACK: /usr/lib/x86_64-linux-gnu/libmkl_rt.so;  LAPACK version 3.8.0
#>
#> locale:
#>  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
#>  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
#>  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
#>  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
#>  [9] LC_ADDRESS=C               LC_TELEPHONE=C
#> [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
#>
#> time zone: Africa/Casablanca
#> tzcode source: system (glibc)
#>
#> attached base packages:
#> [1] splines   stats     graphics  grDevices utils     datasets  methods
#> [8] base
#>
#> other attached packages:
#> [1] rstpm2_1.6.9   rpsurv_0.8.1   survival_3.8-6
#>
#> loaded via a namespace (and not attached):
#>  [1] nlme_3.1-168        cli_3.6.6           knitr_1.51
#>  [4] rlang_1.3.0         xfun_0.60           otel_0.2.0
#>  [7] RcppParallel_6.2.1  htmltools_0.5.9     tinytex_0.60
#> [10] stats4_4.5.1        rmarkdown_2.31      grid_4.5.1
#> [13] evaluate_1.0.5      MASS_7.3-65         fastmap_1.2.0
#> [16] mvtnorm_1.4-2       yaml_2.3.12         numDeriv_2016.8-1.1
#> [19] compiler_4.5.1      Rcpp_1.1.2          bbmle_1.0.25.1
#> [22] mgcv_1.9-4          lattice_0.22-5      digest_0.6.39
#> [25] Matrix_1.7-3        tools_4.5.1         bdsmatrix_1.3-7
```