An introduction to ERRA **05 September 2024**

James Kirchner
ETH Zurich and Swiss Federal Research Institute WSL

For ERRA version 1.0, Build 2024.09.05

This *introduction to ERRA* presents a series of vignettes that introduce some of the capabilities of the Ensemble Rainfall-Runoff Analysis (ERRA) script, and show how to invoke them. A bit of hand-holding is provided at the beginning for users who are not very familiar with R. This introduction concludes with documentation for ERRA function calls, largely reproducing the documentation embedded in the ERRA script itself.

ERRA is a model-independent, data-driven, nonparametric method for inferring nonstationary, nonlinear, and spatially heterogeneous responses to precipitation using RRDs (Runoff Response Distributions) and NRFs (Nonlinear Response Functions). ERRA and some of its basic capabilities are described in Kirchner, "Characterizing nonlinear, nonstationary, and heterogeneous hydrologic behavior using Ensemble Rainfall-Runoff Analysis (ERRA): proof of concept", Hydrology and Earth System Sciences, 2024, hereafter denoted K2024). *Users should read that paper*, since very little of it is reproduced here. The vignettes outlined in this "Introduction to ERRA" reproduce the analyses behind several key figures of that paper. The ERRA function call itself, and its outputs, are fully described at the end of this document.

The mathematical basis for ERRA is explained in Kirchner, "Impulse response functions for nonlinear, nonstationary, and heterogeneous systems, estimated by deconvolution and de-mixing of noisy time series" (Sensors, 22(9), 3291, https://doi.org/10.3390/s22093291, hereafter denoted K2022). The ERRA code is a shell, with a focus on hydrological applications, that runs the IRFnnhs.R script described in that paper (with several key technical modifications). ERRA also adds the capability to set up complex problems exploring rainfall-runoff relationships for precipitation subsets that are split by multiple interacting factors (via the `split_params` option, explained in Sections 8 and 9 below). It also provides for aggregation of time steps, as well as automatic calculation of peak height, width, timing, etc. for runoff response distributions.

1. Technical preliminaries

Before we start: ERRA calculates nonlinear and nonstationary runoff response distributions by framing them as large linear regression problems. These calculations can be made much faster using an efficient set of BLAS (Basic Linear Algebra Subprograms) routines. Base R currently (i.e., in 2024) does not use particularly efficient BLAS routines by default. Switching to a processor-optimized BLAS can speed up matrix operations by an order of magnitude or more, making much larger problems feasible. On Windows, OpenBLAS can be installed relatively straightforwardly as described at https://github.com/david-cortes/R-openblas-in-windows. On Linux machines running Intel processors, the Intel Math Kernel Library can be linked as described at https://www.intel.com/content/www/us/en/developer/articles/technical/quick-linking-intel-mkl-blas-lapack-to-r.html. A web search will reveal other approaches for other operating systems and processors. If an efficient BLAS is correctly installed, R will automatically use it, without any need for changes in the scripts themselves.

Note that an efficient BLAS is not required to run ERRA, but makes it much (much, much…) faster, particularly for large problems (see table below).

These scripts call functions from the following packages, which will need to be installed using the install.packages command if they are not already available:

```
caTools
data.table
dplyr
Matrix
matrixStats
Rcpp  (not required for ERRA_v1.0_no_Rcpp.R: see below)
```

ERRA uses C++ to do some heavy lifting in a few key places. It does this via Rcpp, which requires a C++ compiler. Windows users can get the GNU C++ compiler by installing Rtools, which can be downloaded from https://cran.r-project.org/. There is also a version of ERRA available that does not use Rcpp/C++ (ERRA_v1.0_no_Rcpp.R), but it can be somewhat slower, depending on the size and complexity of the task.

	Base R BLAS		OpenBLAS	
Test case (34 years of hourly data)	no Rcpp	with Rcpp	no Rcpp	with Rcpp
a) 100 lags, linear	2.9	2.6	2.0	1.0
b) 100 lags, nonlinear (4 xknots)	41.9	36.1	11.7	3.7
c) 200 lags, nonlinear (4 xknots)	124.3	114.4	13.2	8.9
d) 500 lags, nonlinear (4 xknots)	832.7	796.0	50.5	31.6
e) 500 lags, nonlinear (4 xknots), broken-stick (20 knots)	22.7	17.0	21.3	15.3
f) 1000 lags, nonlinear (4 xknots)	3566.2	3391.5	173.9	132.3
g) 1000 lags, nonlinear (4 xknots), broken-stick (20 knots)	40.8	28.7	39.0	28.9

Table 1. ERRA runtimes in seconds for a range of test cases (rows a-g), with/without OpenBLAS and with/without Rcpp, on the author's laptop (2022 vintage, 2.1 GHz, 12 cores, 16 threads). Test data set is 34 years of hourly precipitation and streamflows (shown in Figure xxf16 below; ~307,000 time steps). Analyses include (a) linear RRD analysis of 100 lags; (b,c,d,f) nonlinear NRF and weighted average RRD analysis of 100, 200, 500, and 1000 lags, respectively, evaluated between 4 internal xknots as well as the highest and lowest precipitation values (see Section 5 below); and (e and g) nonlinear NRF and weighted average RRD analysis of 500 and 1000 lags, respectively, approximated by a broken-stick piecewise-linear approximation (see Section 11 below) with 20 knots. In all of these test cases, the design matrix is approximately 307,000 rows high. In (a), it is ~100 columns wide, and in (b), (c), (d), and (f), it is ~500, ~1000, ~2500, and ~5000 columns wide, because evaluating the NRF between 4 internal xknots plus the highest and lowest precipitation values (5 intervals between 6 xknots in total) multiplies the width of the design matrix by a factor of 5. In (e) and (g), the design matrix is ~100 columns wide (20 knots x 5 intervals between xknots). Runtime scales very approximately as the design matrix's length multiplied by the square of its width, plus a variable amount of overhead (which is particularly significant for the broken-stick piecewise-linear method of Section 11). Complex problems are ~10 to ~25 times faster with the optimized OpenBLAS than with the non-optimized BLAS that is distributed with R. Rcpp can also reduce runtimes by large factors, particularly for relatively simple problems when the optimized BLAS is also used.

2. A first analysis: runoff response distributions at Roanoke River

We can get started with the following commands, which will be obvious to any R user:

```
setwd("~/ERRA introduction")
rm(list=ls())
source("ERRA_v1.0.R")
```

The first line sets the working directory – and should be changed to whatever your own working directory is, obviously! The second line clears all existing R objects from memory. The third line loads the ERRA script; this step may take some time, because some functions written in Rcpp need to be compiled and linked in C++.

The source data for this first analysis are found in the file "Roanoke_hourly_Q_and_airport_P_2006-2023.txt", which contains roughly 17 years of hourly streamflows at Roanoke, Virginia, and hourly precipitation measurements from Roanoke and Blacksburg airports, located at the eastern and western boundaries of the catchment:

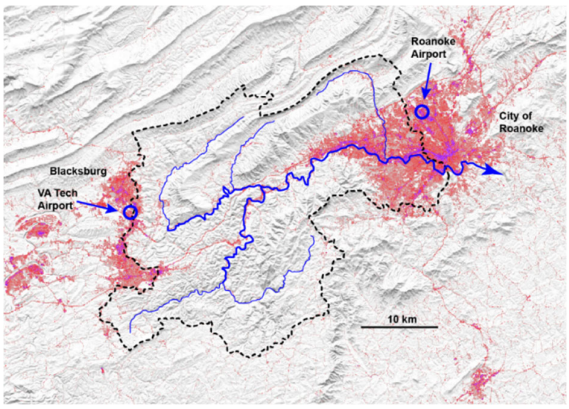

Figure xxf01 (Figure 2 of K2024) Map of the 995 km² Roanoke River catchment (Virginia, USA; perimeter shown by black dashed line), with major stream channels (blue lines) and locations of airport rain gauges (blue circles). Hillshade is from the USGS National Map Viewer (https://apps.nationalmap.gov/viewer/), with red and purple colors indicating impermeable surfaces in the National Land Cover Database.

A sample year of data is shown below.

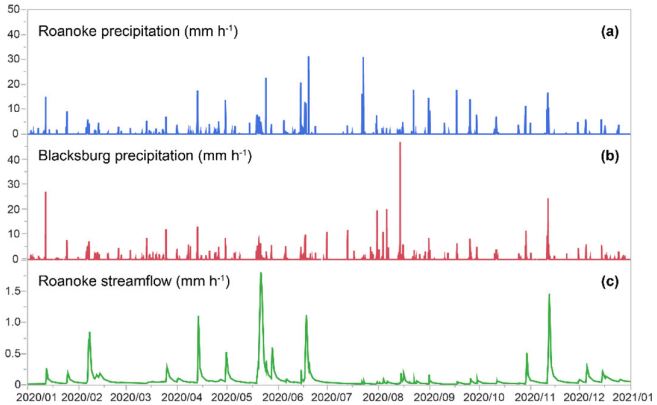

Figure xxf02 (Figure 3 of K2024) One year of hourly precipitation at Roanoke and Blacksburg airports and hourly streamflow at Roanoke gauge. Note that the streamflow axis is expanded 25x relative to the precipitation axes.

We can read in the test data with:

```
dat <- fread("Roanoke_hourly_Q_and_airport_P_2006-2023.txt", header=TRUE, sep="\t",
na.strings=c("NA",".","","#N/A"))
```

We can query what's in the resulting data table with `str(dat)`, which will yield, in this case:

Classes 'data.table' and 'data.frame': 285923 obs. of 5 variables:
$ date_time_UTC : chr "1990/10/01 05:00" "1990/10/01 06:00" "1990/10/01 07:00" "1990/10/01 08:00" ...
$ date_time_EST : chr "1990/10/01 00:00" "1990/10/01 01:00" "1990/10/01 02:00" "1990/10/01 03:00" ...
$ Blacksburg_METAR_P_mm.h: num NA NA NA NA NA NA NA NA NA NA ...
$ Roanoke_METAR_P_mm.h : num NA NA NA NA NA NA NA NA NA NA ...
$ Roanoke_Q_mm.h : num 0.00758 0.00758 0.0079 0.00779 0.00747 ...

Now let's copy precipitation and discharge into vectors with simpler names:

```
p <- 0.5*(dat$Roanoke_METAR_P_mm.h + dat$Blacksburg_METAR_P_mm.h)
q <- dat$Roanoke_Q_mm.h
```

Note that in this case we have averaged the Roanoke and Blacksburg precipitation records together, to get an approximation of the catchment-averaged precipitation.

Most users will only need to call one function, `ERRA`, and this will, in turn, call other stuff "under the hood". A complete list of `ERRA` options and what they do, and a description of all the outputs, is found in comment blocks directly below the `ERRA` function declaration at line 3000 of the ERRA script. These comment blocks have a lot of useful information, and *users are strongly encouraged to read them*. The information in these comment blocks is reproduced in the reference guide at the end of this document.

Running ERRA and interpreting console outputs

As a first analysis we can try

```
zz <- ERRA(p=p, q=q, m=168)
```

where `m=168` tells `ERRA` that we want to analyze lags of up to 168 time steps (i.e., one week).

Verbose reporting is invoked by default, so when we run `ERRA` using the command above, it will unspool a series of progress reports and timings:

initialization done...: 0.18
at h= 0 and time= 1.462 residual PACF (first 5 terms) = 0.9924 -0.4636 -0.1149 -0.067 0.1184
at h= 1 and time= 2.803 residual PACF (first 5 terms) = 0.7395 -0.2503 0.018 -0.0257 0.0419
AR coefficients (phi) = 0.9954331
at h= 2 and time= 4.463 residual PACF (first 5 terms) = 0.1775 -0.1298 -0.0557 -0.0926 -0.0183
AR coefficients (phi) = 1.736895 -0.7460434
practical (in)significance test passed at h=2
IRF finished...: 4.468 seconds
minimum column count of nonzero x values = 10507 (96% of maximum count, and 9% of rows with nonzero weight)
AR coefficients (phi) = 1.736895 -0.7460434
residual PACF (first 5 terms) = 0.1775 -0.1298 -0.0557 -0.0926 -0.0183
return from IRF...: 4.881 seconds
ERRA finished...: 5.115 seconds

Warning message:
In IRF(y = q, x = p_subsets, xnames = set_label, wt = wt, m = m, :
Residuals have lag-1 serial correlation=0.9924 Consider aggregating time steps.

(Your timings will of course be different!)

Setting `verbose=FALSE` will suppress most of this chatter, allowing only error and warning messages to get through. But let's take a look at what's here, and what it reveals about what the `ERRA` routine is doing.

The first 8 lines show `ERRA` iteratively trying successively higher orders of autoregressive noise correction, and the first five terms of the partial autocorrelation function of the residuals from each attempt. For ideal white noise – which we almost never find in the real world – these partial autocorrelations would all be zero. The parameter `h` can be used to specify a particular order of autoregressive noise correction, but the default `h=NULL` tells `ERRA` to automatically search for an appropriate level of noise correction. This proceeds by trying higher and higher values of `h` until the residual autocorrelation and partial autocorrelation become statistically insignificant, or until they become practically insignificant (i.e., too small to care about), as specified by the parameters `ARprob` and `ARlim`. In this case, that happens at `h`=2. The maximum value of `h` that will be tried is set by the parameter `max.AR` (which by default is 8); if `h=max.AR` a warning is issued but execution continues.

The line "*minimum column count of nonzero x values = 10507 (96% of maximum count, and 9% of rows with nonzero weight)*" can be important (although not in this case). This number, 10507, is the minimum count of nonzero precipitation values at any lag. If the precipitation series is split among multiple columns (to examine nonstationary or nonlinear behavior, for example), this number will be the minimum count of nonzero precipitation values at any lag in any column. The two percentages show that this number is 96% of the largest count of nonzero values in any column (because different numbers of nonzero values will be available at each lag), and only 9% of the total number of rows (because during most of the hours, no rain falls). In our example, we have started with a data set with over a quarter of a million rows (285923, to be exact), but most of these lack precipitation values (the hourly streamflow data began in 1990, but the hourly precipitation data only began in 2006). And many of the remaining time steps are rainless, so they do not appear in the tally of nonzero values.

If your RRDs seem strange, and particularly if they are sensitive to small differences in the input data, check these numbers. It is possible that because of the way in which you have split the precipitation time series, one of the subsets has very few nonzero values, and thus one or more of the coefficients cannot be reliably estimated.

The fitted autoregression coefficients are shown in the line
" *AR coefficients (phi) = 1.736895 -0.7460434*". These coefficients are large, have opposite signs, and sum to almost exactly 1, suggesting that the modeled noise is nearly perfectly autocorrelated, but also that this is not first-order autocorrelation but rather a combination of autoregressive and moving-average behavior. You can also see this in the line "*at h= 0 and time= 1.462 residual PACF (first 5 terms) = 0.9924 -0.4636 -0.1149 -0.067 0.1184*", which gives the first 5 terms of the partial autocorrelation function (PACF) of the residuals before any ARMA correction is applied. The phi coefficients correct for this ARMA noise as if it were pure AR noise (which will always be possible, if you include enough terms).

The autocorrelation in the residuals could arise from the characteristics of the measurement noise. It could also arise from the fact that the regression model is incomplete: it does not account for variations in

either ambient conditions or precipitation intensity, and the largest analyzed lag is 168 time steps (or one week), implying that if there has been no precipitation in the previous week, discharge must equal a constant. Whatever the origins of the autocorrelation in the residuals, the fitted AR coefficients can effectively whiten them, as shown by the lack of any large partial autocorrelations on the next line: *residual PACF (first 5 terms) = 0.1775 -0.1298 -0.0557 -0.0926 -0.0183*". The whitening of the residuals is important for ensuring that the RRD coefficients and their standard errors are not distorted by residual autocorrelation.

ERRA has also sent the following warning message to the console:

In IRF(y = q, x = p_subsets, xnames = set_label, wt = wt, m = m, :
Residuals have lag-1 serial correlation=0.9924 Consider aggregating time steps.

This message is triggered whenever the first-order serial correlation in the residuals exceeds the threshold set by `ser.corr.warn` parameter (which is set to 0.99 by default). The point of this is to alert users to cases where the serial correlation in the residuals is strong enough that it might cause problems in estimating the runoff response, even with ARMA noise correction. One can aggregate the time steps in the input and output using the `agg` parameter (more about this below) to reduce the first-order serial correlation, at the cost of potentially obscuring some features briefer than `agg` time steps in the runoff response. In the present case, aggregating the time steps to 2 hours has little effect on the RRD or the ARMA noise correction, so we keep the original time steps here.

Outputs from ERRA

The call to `ERRA` shown above puts the results of our analysis into a R `list` data structure and we have given this the name `zz`. What is in this list? We can get an overview by typing `str(zz)`:

```
List of 16
 $ options   : chr "m=168_h=2"
 $ RRD       :Classes ‘data.table’ and 'data.frame':  169 obs. of  3 variables:
  ..$ lagtime  : num [1:169] 0.333 1 2 3 4 ...
  ..$ RRD_p|all: num [1:169] 0.00106 0.00197 0.00289 0.00423 0.00469 ...
  ..$ se_p|all : num [1:169] 3.64e-05 3.01e-05 4.05e-05 4.94e-05 5.72e-05 ...
  ..- attr(*, ".internal.selfref")=<externalptr>
 $ peakstats :Classes ‘data.table’ and 'data.frame':            1 obs. of  10 variables:
  ..$ set_label: chr "p|all"
  ..$ tpeak    : num 15.4
  ..$ tpeak_se : num 0.924
  ..$ peakht   : num 0.00558
  ..$ peakht_se: num 0.000206
  ..$ width    : num 39.2
  ..$ width_se : num 1.56
  ..$ rc       : num 0.286
  ..$ rc_se    : num 0.00138
  ..$ n.nz     : num 10507
  ..- attr(*, ".internal.selfref")=<externalptr>
 $ criteria  :Classes ‘data.table’ and 'data.frame': 1 obs. of  1 variable:
  ..$ set_label: chr "p|all"
  ..- attr(*, ".internal.selfref")=<externalptr>
 $ Kbb       : num [1:169, 1:169] 3.31e-10 4.38e-10 4.93e-10 5.16e-10 5.24e-10 ...
 $ sets      : num [1:285923, 1] 1 1 1 1 1 1 1 1 1 1 ...
 $ set_label : chr "p|all"
 $ n         : int 285923
 $ n.eff     : num 116374
 $ n.nz      : num [1:169, 1] 10993 10987 10980 10976 10976 ...
 $ h         : num 2
```

```
$ phi      : num [1:2] 1.737 -0.746
$ resid    : num [1:285753] NA NA NA NA NA NA NA NA NA NA ...
$ resid.acf : num [1:50] 0.1775 -0.0942 -0.0949 -0.1049 -0.0375 ...
$ resid.pacf: num [1:50] 0.1775 -0.1298 -0.0557 -0.0926 -0.0183 ...
$ Qcomp    :Classes 'data.table' and 'data.frame':        285753 obs. of  6 variables:
 ..$ timestep : int [1:285753] 171 172 173 174 175 176 177 178 179 180 ...
 ..$ time    : num [1:285753] 171 172 173 174 175 176 177 178 179 180 ...
 ..$ weight  : num [1:325542] 0 0 0 0 0 0 0 0 0 0 ...
 ..$ P       : num [1:285753] NA NA NA NA NA NA NA NA NA NA ...
 ..$ Q       : num [1:285753] 0.00737 0.00726 0.00747 0.00747 0.00737 ...
 ..$ Qfitted  : num [1:285753] NA NA NA NA NA NA NA NA NA NA ...
 ..$ Qresidual: num [1:285753] NA NA NA NA NA NA NA NA NA NA ...
 ..- attr(*, ".internal.selfref")=<externalptr>
```

There's a lot in here, most of which we don't need to bother with for the moment (or possibly ever). But the first three main headings will be immediately useful:

`options` is a character string containing the most commonly used options that specify how the analysis was done. It is intended to help facilitate consistent, transparent file naming for the outputs, as we will see in a moment. Right now, it contains only the maximum lag `m=168` (in number of time steps), and the order of ARMA noise correction `h=2`. Note that `h` may be specified as an ERRA input option or may be determined by ERRA itself; in either case it is reported unless it is zero (meaning no ARMA noise correction was performed).

`RRD` is a data table containing the runoff response distribution. It has three columns:

`lagtime` the lag time

`RRD_p|all` RRD coefficients

Here, "`_p`" is the default designation for a single precipitation input. For multiple precipitation inputs we would have multiple columns designated `RRD_p1`, `RRD_p2`, etc., or if the input vectors have assigned names, the columns will be designated `RRD_<name1>`, `RRD_<name2>`, and so on. "`|all`" designates a precipitation input that has not been split via the `split_params` option to account for nonstationary conditions. If the precipitation input has been split (see Section 8 below for an example), "`|all`" will be replaced with a string that documents the splitting criteria that have been applied in that column.

`se_p|all` The standard errors of the RRD coefficients, with the same notation as explained above.

`peakstats` is a data table containing peak statistics and other quantities extracted from the runoff response distribution. Here, `RRD` has only one runoff response distribution, so `peakstats` will have only one line. In more complex cases, there will be one line for each RRD (each column in the data table RRD). The columns of `peakstats` are:

`set_label` the label for the corresponding RRD column

`tpeak` peak lag

`tpeak_se` standard error of peak lag

`peakht` peak height

`peakht_se` standard error of peak height

`width` width (at half maximum) of peak

`width_se` standard error of peak width

`rc` runoff coefficient (integral under RRD curve). Note that this integral is taken out to the maximum lag `m`, and thus will depend on `m` unless the RRD has converged to zero by then.

`rc_se` standard error of runoff coefficient

`n.nz` number of non-zero input values

Some users who are particularly familiar with R may choose to perform further analyses directly on the ERRA outputs in R itself. Many others will choose to transfer these outputs to other platforms for plotting and analysis. We can write some key results into tab-delimited text files for further analysis, viewing and plotting as follows:

```
fileID <- "Roanoke_avgP"
with(zz, {
  fwrite(RRD, paste0(fileID, "_RRD_", options, ".txt"), sep="\t")
  fwrite(peakstats, paste0(fileID, "_peakstats_", options, ".txt"), sep="\t")
})
```

Here the "fileID" is just a reusable label identifying the site (Roanoke) and the input (average precipitation). Each of the fwrite statements specifies a file name that combines this fileID, a string indicating what kind of output is in the file, and the options string.

The resulting RRD file, "Roanoke_avgP_RRD_m=168_h=2.txt", contains the data required to generate Figure xxf3a (Figure 4a in K2024). Similar analyses, using only the precipitation recorded at Roanoke or only the precipitation recorded at Blacksburg, yield Figure xxf3b (Figure 4b in K2024).

3. Spatial heterogeneity: comparing effects of precipitation falling at different locations

As explained in Section 2.3 of K2024, ERRA can reveal spatially heterogeneous runoff responses, even if streamflow is only measured at the catchment outlet, by deconvolving and demixing the effects of multiple precipitation inputs.

In ERRA, this is as simple as replacing the precipitation vector with multiple columns of a matrix, data table, or dataframe. If we assume that the precipitation recorded at Roanoke and Blacksburg each fall on roughly 50 percent of the catchment, this matrix can be constructed as:

```
p <- cbind(pR=0.5*dat$Roanoke_METAR_P_mm.h, pB=0.5*dat$Blacksburg_METAR_P_mm.h)
```

It's that simple. The symbols "pR" and "pB" have been supplied as labels for precipitation at Roanoke and Blacksburg, respectively, so that the corresponding results can be labeled accordingly.

The instruction to run ERRA is exactly the same…

```
zz <- ERRA(p=p, q=q, m=168)
```

… but now the RRD table has separate columns for the response to precipitation falling near Roanoke and precipitation falling near Blacksburg:

lagtime	RRD_pR|all	RRD_pB|all	se_pR|all	se_pB|all
0.333333333	0.002089008	6.45E-05	5.19E-05	6.04E-05
1	0.003405866	0.00029392	4.71E-05	5.46E-05
2	0.00486995	0.000479563	6.73E-05	7.85E-05
3	0.006635657	0.001035883	8.59E-05	0.00010085
4	0.006672661	0.001836145	0.000102918	0.000121566
5	0.005767346	0.002324199	0.000118346	0.000140516
6	0.005039116	0.002651304	0.00013226	0.000157611
7	0.004631435	0.003015556	0.000144748	0.000172997
8	0.004583704	0.003330818	0.000156011	0.000186911
9	0.004688239	0.003961965	0.000166178	0.000199556
10	0.004868277	0.004816589	0.00017531	0.000210994

This output table contains the data behind Figure xxf03c (Figure 4c of K2024):

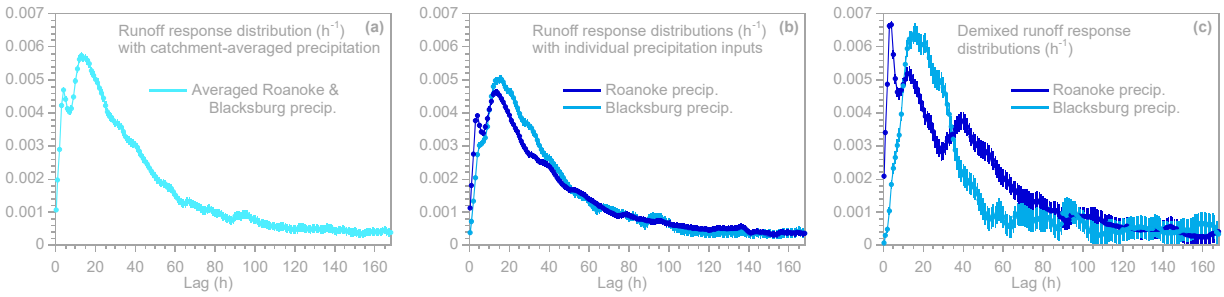

Figure xxf03 (Figure 4 of K2024) ERRA runoff response distributions calculated by deconvolving Roanoke River streamflow by different precipitation inputs. In (a), streamflow is deconvolved by the average of precipitation measured at Roanoke and Blacksburg. In (b) streamflow is deconvolved by either Roanoke precipitation alone (dark blue) or Blacksburg precipitation alone (light blue). In (c), by contrast, deconvolution and demixing are used to jointly determine the effects of Roanoke and Blacksburg precipitation on streamflow. This deconvolution/demixing analysis reveals that the effects of Roanoke and Blacksburg precipitation are markedly different. Error bars show one standard error.

4. Units and time steps

ERRA does not require its input data to be in any particular units. Nonetheless, the numerical results will be easiest to interpret if the input data are in compatible units (for example, mm per hour or mm per day).

ERRA likewise does not require its input data to be recorded at any particular time step. It implicitly performs all internal calculations in units of "per time step". To correctly translate those units of "time steps" to minutes, hours, days, or whatever, ERRA needs to know how long the input time steps are, measured in whatever units we want our outputs to be in.

That is the job of the `dt` parameter. It tells ERRA that the raw input data are recorded at intervals of `dt` in whatever time units you want to use (which will then be the time units for reporting the lag times, RRDs, and so forth). The default value, `dt=1`, tells ERRA that we want our results in the same time units as the time step of the data set. That means that if the data are hourly, the runoff response distribution will be reported in units of 1/hours. If we wanted our results in units of days, `dt` would be 1/24 (because our hourly time step is 1/24 of a day); if we had daily data but wanted our results in units of hours, `dt` would be 24 (because our daily time step equals 24 hours).

Note that `dt` affects the ***units*** in which the results are reported (and thus their numerical values), but not the ***time intervals*** at which the analysis is performed or results are reported, which are determined by the sampling interval of the underlying data, potentially as aggregated by the `agg` parameter (see Section 6 below for details).

I belabor this point because Runoff Response Distributions (RRDs) are, as the name implies, ***distributions*** (fractions per unit of time), and thus their numerical values are inherently dependent on their units. Specifying `dt=1` means that our RRDs will be reported in fractions per hour, which is more intuitive than, for example, fractions per minute. This means that the numerical values will also be 60 times greater than if the results were reported in fractions per minute (but also 24 times smaller than if they were reported in fractions per day, and so on).

5. Quantifying nonlinear runoff response at Saco River

As explained in Section 3 of K2024, ERRA implements a nonlinear deconvolution method to estimate runoff's nonlinear dependence on precipitation intensity. Instead of assuming that runoff response at each lag is proportional to precipitation and thus can be quantified by a single RRD coefficient, this approach instead assumes that runoff response is a piecewise-linear "broken stick" function of precipitation intensity, as shown in Figure xxf04 below. This function may vary, of course, from one lag to the next, just as RRD coefficients would also vary from one lag to the next. Thus ERRA can quantify how runoff response varies in shape, not just in amplitude, between high-intensity and low-intensity precipitation. This broken-stick model is illustrated in Figure 5 of K2024:

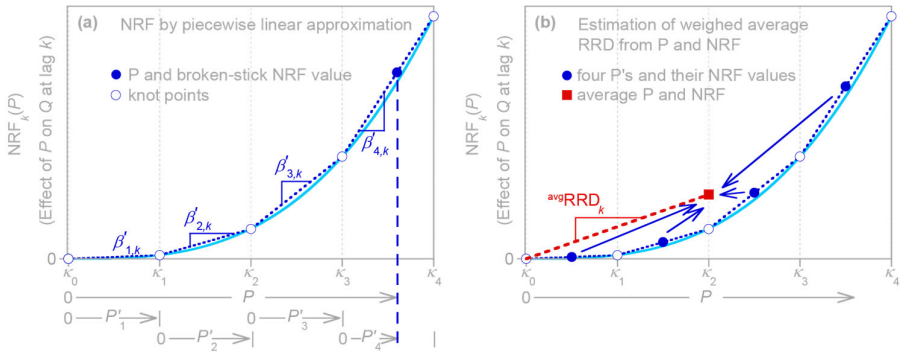

Figure xxf04 (Figure 6 of K2024) Nonlinear response function (NRF) estimation by piecewise-linear broken-stick approximation. (a) Estimation of the nonlinear dependence of Q on precipitation intensity P at a specified lag k. Splitting the precipitation value P (solid circle) into sub-ranges P'_ℓ between user-specified knots κ_ℓ (open circles) facilitates the fitting of slopes $\beta'_{\ell,k}$ between these knots by linear regression, yielding a nonparametric piecewise-linear continuous curve (see Eqs. 9-10 of K2024). (b) Estimation of the weighted-average runoff response distribution (RRD_k) as the ratio between the means of the NRF and P (red square) over all time steps (4 example values of P and NRF shown by solid blue circles).

This broken-stick function approximates the relationship between precipitation intensity (*P* in Figure xxf04) and runoff response (the nonlinear response function NRF in Figure xxf04) with a continuous function composed of linear segments connecting a set of "knot points" (the open circles of Figure xxf04). Because these knots are knots in the x-axis variable of our analysis (here, precipitation), they are referred to as "xknots" in ERRA. (This is because ERRA can also use a broken-stick approach to express the RRD as a function of unevenly-spaced lag times, and in that case, the knots are known simply as "knots").

Many users may immediately wonder how to select the precipitation values for the xknots. A fair question! Ideally we want to capture any important nonlinearities or thresholds, while also making sure that we have enough precipitation values between each of the knots so that the NRF is reliably estimated. Users may need to seek a balance between these two goals.

The lowermost and uppermost xknots are set by the algorithm. The lowermost xknot is at *P*=0, under the logical assumption that if rain doesn't fall, it cannot have any effect on current or future runoff (note that this is not the same as saying that current or future runoff will be zero). The uppermost xknot is automatically set at the maximum precipitation value in the data set (this is required by the mathematics underlying the broken-stick approach). In between these two limits, the remaining xknots are set according to the `xknot_type` and `xknots` parameters in `ERRA`. (Alternatively, setting either `xknot_type` or `xknots` to their default values of `NULL` will suppress the broken-stick approach entirely.)

The simplest way to invoke the broken-stick approach, ***which is also the best way for most users and for most purposes***, is to set `xknot_type="even"` and set (for example) `xknots=c(5, 20)` (each of these must be a positive integer). These instructions tell ERRA to construct 5 xknots between the lowermost and uppermost xknots (p=0 and p=max(p)), not including those endpoints. In this case, the broken-stick approximation would have 6 segments connecting the 7 total xknots (the two endpoints, plus the five in between). Now how are those five xknots set? The option `xknot_type="even"`, combined with `xknots=c(5, 20)`, tells ERRA to distribute these 5 xknots as evenly as it can, subject to the constraint that each interval between xknots should contain at least 20 nonzero, non-missing values.

Users can try different values for the `xknots` parameters, of course. But be aware that setting too many xknots may leave so few points in each interval between xknots that the broken-stick approximation may be highly uncertain (or in extreme cases, will computationally fail because of a loss of degrees of freedom). The second `xknots` parameter (here, 20) should be set large enough that the slope ($\beta'_{\ell,k}$) of the highest broken-stick segment can be accurately determined, but also small enough that the range of p that is spanned by this segment is not too large.

Users can also specify the values of xknots explicitly, if they wish. For example, `xknots = c(80, 90, 95, 99)` and `xknot_type="percentiles"` will tell `ERRA` to set the knots at the 80th, 90th, 95th, and 99th percentiles of the ***non-zero*** precipitation values in the data set. Other possible options for xknot_type are:

`xknot_type="values"`: xknots are expressed as raw values of *P*

`xknot_type="cumsum"`: xknots are expressed as percentages of ***the cumulative sum*** of p (so if `xknots=c(20, 50, 80)` for example, the xknot spacing will be set such that all smaller *P*'s sum to 20, 50, and 80 percent of the sum of all P's).

`xknot_type="sqsum"`: xknots are expressed as percentages of ***the cumulative sum of squares*** of p (so if `xknots=c(30, 50, 70)` for example, the xknot spacing will be set such that all smaller p's, squared and summed, add up to 30, 50, and 70 percent of the sum of all squared *P*'s). Assuming that p is strongly skewed with a mean near zero, these fractions should approximate the fractions of the total leverage in the data set.

If `xknot_type="percentiles"`, `"cumsum"`, or `"sqsum"` but the largest xknot is less than 1 (e.g., xknots=c(0.3, 0.5, 0.7), ERRA will assume that the user has accidentally specified the xknots as fractions rather than percentages, and will convert them to percentages instead.

Each xknot_type option can be abbreviated by its first letter, i.e., `e, p, v, c,` or `s` in place of `even`, `percentiles`, `values`, `cumsum`, and `sqsum`. In practical use so far, `xknot_type="even"` has been useful for generating initial estimates of the xknot values, which can then be fixed at nearby round-number values using `xknot_type="values"` for presentation purposes.

Users should note that if multiple precipitation time series are being analyzed (as in Section 3 above), or if they are being split (according to, e.g., antecedent wetness) to account for nonstationary behavior (as in Section 8 below), specifying `xknot_type="values"` will impose the same xknot spacing on all of these precipitation time series or subsets. Specifying any other `xknot_type` options will create differently

spaced xknots for each precipitation time series or subset. This is generally what you want, because you want to avoid having too few points available in each interval between the xknots.

Note that if you specify the xknots explicitly, you must not include the lowest and highest values (0 and the maximum precipitation). Those will be automatically added by ERRA, and including them in your specified xknots will cause problems.

Nonlinear functions like the one shown above depict the total runoff response (the effect of precipitation on runoff) as a function of precipitation intensity. Whereas the RRD is the runoff response *per unit of precipitation* (which would be the *slope* of a line from the origin to any point along this curve), the broken-stick function above estimates the *Nonlinear Response Function (NRF)*, which is the RRD times the precipitation rate itself. This more clearly depicts nonlinear relationships between precipitation intensity and runoff response. For example, if the NRF increased quadratically with precipitation intensity, the runoff response per unit of precipitation (that is, the RRD), would only increase linearly with precipitation intensity. And if the NRF increased linearly with precipitation intensity, the runoff response per unit of precipitation (again, the RRD) would be a constant.

Users should note that because the NRF is the product of the RRD times the precipitation rate (see Eq. 7 of K2024), it has different units. If, for example, precipitation and streamflow are measured in units of mm/h, the RRD will have units of 1/h, because it is the distribution over time of streamflow per unit of precipitation. The NRF, by contrast, will have units of mm/h^2, because it is the streamflow resulting *per time unit of precipitation at a given rate* (or in a given range), rather than *per unit of precipitation*.

Alternatively, users may choose to consider the time step to be embedded in the definition of the NRF itself, for example, an "hourly NRF" that quantifies streamflow that arises per hour of precipitation at a given rate, and is thus effectively measured in mm/h (with hourly precipitation intensity measured in mm) because the time scale is embedded in the definitions of the "hourly" NRF itself. This will be more intuitive for most users (and their audiences). At the risk of stating the obvious: both the RRD and NRF quantify *incremental* additions to streamflow (on top of the lagged effects of previous precipitation inputs), not the total streamflow that one should expect at any given lag.

Note that a nonlinear function like the one shown in Figure xxf04 above *will have different values for every lag*. From the resulting fitted parameters, users can examine NRF's for different precipitation intensities (i.e., runoff response vs. lag time, at fixed values of precipitation intensity); they can also examine how runoff response varies with precipitation intensity, at fixed values of lag time (through functions like the one shown in Figure xxf04 above).

The NRF can be estimated at each xknot point (these values are in the data table "knot_NRF", with the corresponding peakstats table "knot_peakstats"), or as a precipitation-weighted average over each broken-stick segment (these values are in the data table "NRF", with the corresponding table "peakstats"). The knot NRF has the advantage that it is estimated for an exactly specified precipitation intensity, but it has the important disadvantage that its estimation is less stable than the segment-averaged NRF's is. Therefore, the segment-averaged NRF is recommended for most analyses of nonlinear response to precipitation.

Lastly, note also that by default, results for the uppermost segment of the broken-stick approximation (and the highest xknot, corresponding to the single highest value of precipitation) will not be shown. The

reason for masking these results is that they can be erratic, particularly if they are often based on only a few highly skewed precipitation values. Users who nonetheless want to show these results can do so by setting the option `show_top_xknot=TRUE` (this is `FALSE` by default).

In K2024, I illustrate this approach using hourly precipitation and streamflow data from the Saco River at Conway, New Hampshire. Hourly precipitation data from the MOPEX study have been paired with hourly streamflows (from 15-minute USGS measurements) in the data file "MOPEX SacoR hourly.txt".

As before, we can read the data in with

```
dat <- fread("MOPEX SacoR hourly.txt", header=TRUE, sep="\t",
na.strings=c("NA",".","","#N/A"))
```

and then get an overview of what's in that file with `str(dat)`:

```
Classes 'data.table' and 'data.frame':    141984 obs. of  10 variables:
 $ date_time: chr  "1987/10/21 00:00:00" "1987/10/21 01:00:00" "1987/10/21 02:00:00" "1987/10/21 03:00:00" ...
 $ p       : num  0.07 1.54 0.04 0.04 0.66 1.02 0.29 1.47 0 0 ...
 $ PET     : num  0.0578 0.0578 0.0578 0.0578 0.0578 ...
 $ q       : num  0.0397 0.0397 0.0397 0.0397 0.0397 ...
 $ yr      : int  1987 1987 1987 1987 1987 1987 1987 1987 1987 1987 ...
 $ mm      : int  10 10 10 10 10 10 10 10 10 10 ...
 $ dd      : int  21 21 21 21 21 21 21 21 21 21 ...
 $ PET_mm/d : num  1.39 1.39 1.39 1.39 1.39 ...
 $ Tmax    : num  10.9 10.9 10.9 10.9 10.9 ...
 $ Tmin    : num  1.44 1.44 1.44 1.44 1.44 ...
```

The Saco River basin can have significant snow cover from roughly November until late April. To exclude these months, we can define a vector that is 1 during the snow-free months and zero otherwise:

```
fileID <- "SacoR_snowfree"
snowfree <- ifelse((dat$mm>4)&(dat$mm<11), 1, 0)
```

and then call ERRA using `Qfilter=snowfree`, which restricts the analysis to only time steps that fall within these snow-free months. Strictly speaking, it restricts the analysis to discharges (not precipitation time steps) that occur during the snow-free months (that's the reason for the Q in Qfilter). The first `m` time steps each May will also reflect the lag effects of precipitation falling in late April.

We can call ERRA as follows, `using xknots=c(6,40), xknot_type="even"` to let ERRA pick six xknots with at least 40 points in each precipitation interval, `show_top_xknot=TRUE` to show the full range of precipitation intensity, and `Qfilter=snowfree` to exclude snowy months:

```
zz <- ERRA(p=p, q=q, xknots=c(6,40), xknot_type="even", m=100,
Qfilter=snowfree, show_top_xknot=TRUE)
```

We can get an overview of the results with `str(zz)`:

```
List of 20
 $ options       : chr "m=100_h=3_nlin"
 $ NRF           :Classes 'data.table' and 'data.frame':         101 obs. of  15 variables:
```

```
..$ lagtime              : num [1:101] 0.333 1 2 3 4 ...
..$ NRF_p<1.49985|all    : num [1:101] 0.000555 0.000522 0.000644 0.000874 0.000944 ...
..$ NRF_p=1.49985-3.00637|all: num [1:101] 4.29e-04 3.01e-04 3.98e-04 3.44e-04 -6.64e-05 ...
..$ NRF_p=3.00637-4.51455|all: num [1:101] 0.000398 0.000327 0.001663 0.003685 0.006325 ...
..$ NRF_p=4.51455-6.0234|all : num [1:101] 0.00556 0.00645 0.01163 0.01853 0.02839 ...
..$ NRF_p=6.0234-7.61591|all : num [1:101] 0.00655 0.00613 0.01294 0.02224 0.03407 ...
..$ NRF_p=7.61591-9.14909|all: num [1:101] 0.00865 0.00771 0.01586 0.02547 0.03982 ...
..$ NRF_p>9.14909|all    : num [1:101] 0.0259 0.0298 0.0536 0.0842 0.106 ...
..$ se_p<1.49985|all     : num [1:101] 0.000115 0.00012 0.00018 0.000232 0.000275 ...
..$ se_p=1.49985-3.00637|all : num [1:101] 0.000242 0.000249 0.000369 0.000476 0.000569 ...
..$ se_p=3.00637-4.51455|all : num [1:101] 0.000361 0.000385 0.000588 0.000779 0.000952 ...
..$ se_p=4.51455-6.0234|all  : num [1:101] 0.000531 0.000568 0.000878 0.00118 0.001468 ...
..$ se_p=6.0234-7.61591|all  : num [1:101] 0.000705 0.000776 0.001236 0.001697 0.002142 ...
..$ se_p=7.61591-9.14909|all : num [1:101] 0.000864 0.000943 0.001479 0.002007 0.002514 ...
..$ se_p>9.14909|all     : num [1:101] 0.00131 0.00138 0.00215 0.00289 0.00353 ...
..- attr(*, ".internal.selfref")=<externalptr>
$ knot_NRF       :Classes 'data.table' and 'data.frame':     101 obs. of  15 variables:
..$ lagtime              : num [1:101] 0.333 1 2 3 4 ...
..$ knot_NRF_p=1.49985|all: num [1:101] 0.00124 0.00117 0.00144 0.00196 0.00211 ...
..$ knot_NRF_p=3.00637|all: num [1:101] -0.000492 -0.000683 -0.000783 -0.001481 -0.002531 ...
..$ knot_NRF_p=4.51455|all: num [1:101] 0.00153 0.00161 0.00477 0.01025 0.01758 ...
..$ knot_NRF_p=6.0234|all : num [1:101] 0.0105 0.0124 0.02 0.0287 0.0416 ...
..$ knot_NRF_p=7.61591|all: num [1:101] 0.00125 -0.00227 0.00339 0.01358 0.02389 ...
..$ knot_NRF_p=9.14909|all: num [1:101] 0.0192 0.022 0.0336 0.0424 0.0625 ...
..$ knot_NRF_p=16.93|all  : num [1:101] 0.0391 0.0453 0.0926 0.1658 0.1909 ...
..$ se_p=1.49985|all      : num [1:101] 0.000258 0.000269 0.000403 0.000519 0.000616 ...
..$ se_p=3.00637|all      : num [1:101] 0.000421 0.000466 0.000733 0.000994 0.001237 ...
..$ se_p=4.51455|all      : num [1:101] 0.000665 0.000765 0.001246 0.001731 0.00219 ...
..$ se_p=6.0234|all       : num [1:101] 0.000968 0.0011 0.001777 0.002467 0.003137 ...
..$ se_p=7.61591|all      : num [1:101] 0.00123 0.00143 0.00233 0.00324 0.00411 ...
..$ se_p=9.14909|all      : num [1:101] 0.00144 0.00172 0.00287 0.00404 0.0052 ...
..$ se_p=16.93|all        : num [1:101] 0.00337 0.00374 0.00602 0.00839 0.01069 ...
..- attr(*, ".internal.selfref")=<externalptr>
$ wtd_avg_RRD    :Classes 'data.table' and 'data.frame':    101 obs. of  3 variables:
..$ lagtime         : num [1:101] 0.333 1 2 3 4 ...
..$ wtd_avg_RRD_p|all: num [1:101] 0.00061 0.000587 0.000921 0.001384 0.001789 ...
..$ se_p|all        : num [1:101] 1.00e-04 9.98e-05 1.43e-04 1.77e-04 2.03e-04 ...
..- attr(*, ".internal.selfref")=<externalptr>
$ peakstats      :Classes 'data.table' and 'data.frame':        7 obs. of  14 variables:
..$ setwm_label  : chr [1:7] "p<1.49985|all" "p=1.49985-3.00637|all" "p=3.00637-4.51455|all" "p=4.51455-6.0234|all" ...
..$ wtd_meanp    : num [1:7] 0.669 2.207 3.671 5.193 6.702 ...
..$ pvol         : num [1:7] 0.669 2.207 3.671 5.193 6.702 ...
..$ tpeak        : num [1:7] 44 19.8 14.8 12 11.6 ...
..$ tpeak_se     : num [1:7] 1.244 0.855 0.579 0.287 0.234 ...
..$ NRF_peakht   : num [1:7] 0.00228 0.01192 0.04222 0.10318 0.14245 ...
..$ NRF_peakht_se: num [1:7] 0.000318 0.000518 0.001946 0.00339 0.004411 ...
..$ width        : num [1:7] 25.4 28 20.2 16.3 13.5 ...
..$ width_se     : num [1:7] 5.158 11.653 1.126 0.574 0.392 ...
..$ rc           : num [1:7] 0.145 0.199 0.355 0.573 0.437 ...
..$ rc_se        : num [1:7] 0.008 0.00622 0.00671 0.00837 0.00924 ...
..$ rsum         : num [1:7] 0.0968 0.4397 1.3038 2.9772 2.9303 ...
..$ rsum_se      : num [1:7] 0.00535 0.01373 0.02463 0.04348 0.06196 ...
..$ n.nz         : num [1:7] 14143 1986 745 357 181 ...
..- attr(*, ".internal.selfref")=<externalptr>
$ knot_peakstats  :Classes 'data.table' and 'data.frame':    7 obs. of  14 variables:
..$ setbp_label      : chr [1:7] "p=1.49985|all" "p=3.00637|all" "p=4.51455|all" "p=6.0234|all" ...
..$ knot             : num [1:7] 1.5 3.01 4.51 6.02 7.62 ...
..$ knot_pvol        : num [1:7] 1.5 3.01 4.51 6.02 7.62 ...
```

```
 ..$ tpeak          : num [1:7] 44 16.7 13.6 11.3 11.8 ...
 ..$ tpeak_se       : num [1:7] 1.244 2.864 0.85 0.228 0.487 ...
 ..$ knot_NRF_peakht  : num [1:7] 0.00511 0.02504 0.06465 0.15406 0.12834 ...
 ..$ knot_NRF_peakht_se: num [1:7] 0.000712 0.002594 0.004342 0.004567 0.007006 ...
 ..$ width          : num [1:7] 25.4 20.4 19.7 14.7 12.3 ...
 ..$ width_se       : num [1:7] 5.158 8.839 1.889 0.645 0.575 ...
 ..$ rc             : num [1:7] 0.145 0.23 0.461 0.676 0.183 ...
 ..$ rc_se          : num [1:7] 0.008 0.0114 0.014 0.0158 0.016 ...
 ..$ knot_rsum      : num [1:7] 0.217 0.691 2.082 4.073 1.391 ...
 ..$ knot_rsum_se   : num [1:7] 0.012 0.0343 0.063 0.0955 0.1222 ...
 ..$ n.nz           : num [1:7] 14143 1986 745 357 181 ...
 ..- attr(*, ".internal.selfref")=<externalptr>
$ avgRRD_peakstats:Classes 'data.table' and 'data.frame':1 obs. of  9 variables:
 ..$ set_label: chr "p|all"
 ..$ tpeak    : num 16.8
 ..$ tpeak_se : num 0.638
 ..$ peakht   : num 0.00557
 ..$ peakht_se: num 0.000182
 ..$ width    : num 32.9
 ..$ width_se : num 9.63
 ..$ rc       : num 0.259
 ..$ rc_se    : num 0.00317
 ..- attr(*, ".internal.selfref")=<externalptr>
$ criteria       :Classes 'data.table' and 'data.frame':          7 obs. of  3 variables:
 ..$ setwm_label: chr [1:7] "p<1.49985|all" "p=1.49985-3.00637|all" "p=3.00637-4.51455|all" "p=4.51455-6.0234|all" ...
 ..$ wtd_meanp  : num [1:7] 0.669 2.207 3.671 5.193 6.702 ...
 ..$ pvol       : num [1:7] 0.669 2.207 3.671 5.193 6.702 ...
 ..- attr(*, ".internal.selfref")=<externalptr>
$ Kbb            : num [1:707, 1:707] 7.42e-09 1.35e-08 1.70e-08 1.82e-08 1.80e-08 ...
$ sets           : num [1:141984, 1] 1 1 1 1 1 1 1 1 1 1 ...
$ set_label      : chr "p|all"
$ n              : int 141984
$ n.eff          : num 56965
$ n.nz           : num [1:101, 1:7] 14284 14285 14284 14286 14286 ...
$ h              : num 3
$ phi            : num [1:3] 2.219 -1.625 0.401
$ resid          : NULL
$ resid.acf      : num [1:47] 0.00023 -0.01177 0.00209 0.02185 0.0116 ...
$ resid.pacf     : num [1:47] 0.00023 -0.01177 0.0021 0.02171 0.01165 ...
$ Qcomp          :Classes 'data.table' and 'data.frame':        141881 obs. of  6 variables:
 ..$ timestep : int [1:141881] 104 105 106 107 108 109 110 111 112 113 ...
 ..$ time     : num [1:141881] 104 105 106 107 108 109 110 111 112 113 ...
 ..$ weight   : num [1:325542] 0 0 0 0 0 0 0 0 0 0 ...
 ..$ P        : num [1:141881] 2.5 0.42 0 0 0 0 0 0 0 0 ...
 ..$ Q        : num [1:141881] NA NA NA NA NA NA NA NA NA NA ...
 ..$ Qfitted  : num [1:141881] NA NA NA NA NA NA NA NA NA NA ...
 ..$ Qresidual: num [1:141881] NA NA NA NA NA NA NA NA NA NA ...
```

I know, It's a lot. But just as we saw before in Section 2, there will be a lot of details that most users will never need, although there are some basics that users should know about. The data table `NRF` contains a column of lag times and columns for the NRF's and associated standard errors for each precipitation range. For example, the column heading `NRF_p=7.6659-9.23908|all` indicates the NRF for the stated precipitation intensity range (for an un-named precipitation source "`p`"), with `|all` indicating that we have analyzed all the precipitation, meaning that we have not split the precipitation according to things like antecedent wetness (but note that any filtering of the data according to ***discharge*** time, such as our elimination of snowy months by specifying `Qfilter=snowfree`, is not reflected in this label)

There is a corresponding data table `knot_NRF` which contains the NRFs evaluated at the individual knot points (which are specified by exact values of *P* rather than ranges, for example a column heading might be `knot_NRF_p=9.23908|all`). As indicated above, these "knot NRFs" are usually more noisy than the NRFs estimated over each interval of precipitation intensity.

There are peak statistics tables, with the names `peakstats` and `knot_peakstats`, which contain the peak statistics for the NRFs and the knot NRFs. The peak statistics tables for the NRFs and knot NRFs include some new elements, including `wtd_meanp` (mean precipitation in the interval of precipitation intensity corresponding to a given NRF), `knot` (precipitation intensity corresponding to a given knot NRF), `pvol` or `knot_pvol` (precipitation volume, that is, precipitation rate multiplied by time step, including any aggregation of the time step), and `rsum` or `knot_rsum` (runoff volume in, for example, mm/h, calculated by integrating the area under the NRF or knot NRF).

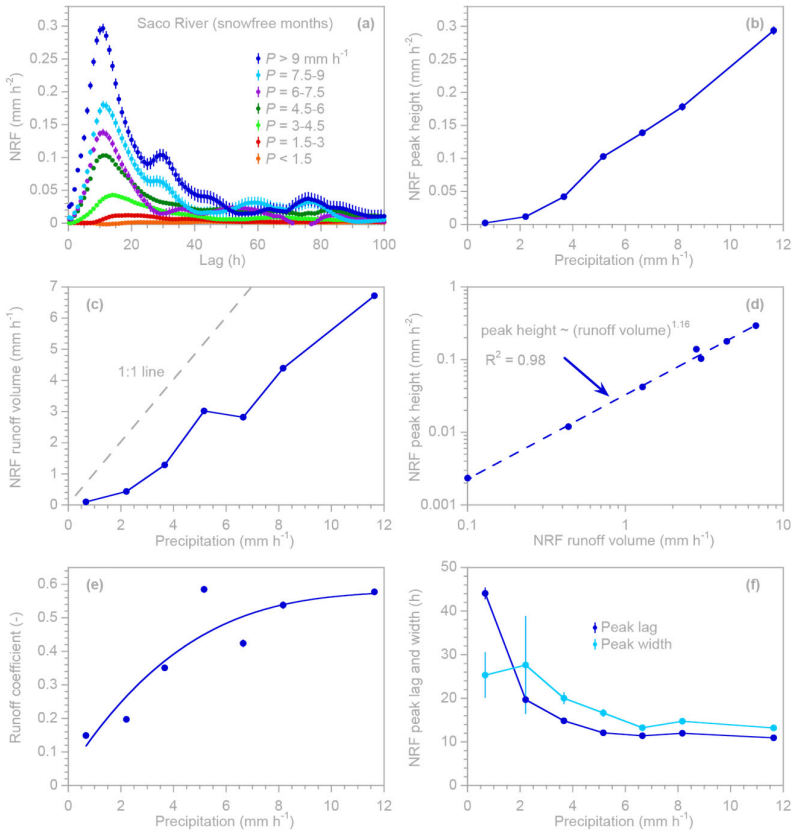

Figure xxf05 (Figure 7 of K2024) Nonlinear rainfall-runoff behavior at Saco River near Conway, NH, USA, inferred from ERRA nonlinear response functions (NRF). (a) NRF as a function of lag time for specified ranges of precipitation intensity. (b) Peak height of NRF as a function of precipitation intensity, showing nonlinearity in peak runoff response. (c) Effect of precipitation intensity on NRF runoff volume, measured as the total area under each curve in panel (a) and expressed in mm of streamflow per hour of precipitation at the stated intensity. Runoff volume increases more slowly than precipitation intensity does (i.e., the gap between the blue curve and the gray 1:1 line continues to grow), suggesting that losses to evapotranspiration and/or long-term storage are not constant, but instead increase with increasing precipitation intensity. (d) Log-log relationship between NRF peak height and NRF runoff volume, with a scaling exponent of 1.16 indicating that peak height is modestly more sensitive to precipitation intensity than total runoff volume is. Thus as precipitation intensity increases, peak height grows more-than-proportionally relative to average runoff response. (e) Runoff coefficient (ratio between NRF runoff volume and precipitation intensity) as a function of precipitation intensity, with an arbitrary smooth curve to guide the eye. (f) Lag-to-peak and peak width (at half maximum) as functions of precipitation intensity, showing that NRF peaks are earlier and narrower at higher precipitation rates. Source data span 1987-2003; months from November through April were omitted to exclude effects of snow. Error bars show one standard error, where this is larger than the plotting symbols.

One thing to note is that although the instructions `xknots=c(6,40), xknot_type="even"` tell ERRA to have at least 40 points between each pair of xknots, in our example `n.nz` (the number of nonzero precipitation values) in the uppermost precipitation interval was only 39. This can occur when lines of the design matrix are excluded for various reasons (such as a lack of precipitation data after the xknots are determined.

We can take the xknot values that were found by this procedure and approximate them with rounder values for ease of communication. This gives us xknots at 1.5, 3, 4.5, 6, 7.5 and 9 mm/h. We can then run ERRA with these values, as follows:

```
zz <- ERRA(p=p, q=q, xknots=c(1.5, 3, 4.5, 6, 7.5, 9), xknot_type="values",
m=100, Qfilter=snowfree, show_top_xknot=TRUE)
```

And write out the key data tables for plotting:

```
with(zz, {                                    # write the output to files
  fwrite(peakstats, paste0(fileID, "_peakstats_", options, ".txt"), sep="\t")
  fwrite(NRF, paste0(fileID, "_NRF_", options, ".txt"), sep="\t")
})
```

This results in the files `SacoR_snowfree_NRF_m=100_h=3_nlin.txt` and `SacoR_snowfree_peakstats_m=100_h=3_nlin.txt`, which contain the numerical values behind Figure xxf05 (Figure 7 of K2024) and are also used in Figures 8 and 9 of K2024 as well. The designation `m=100` means that the maximum lag was 100 time steps, `h=3` means that third-order noise correction was applied, and `nlin` means that nonlinear deconvolution was used.

6. Aggregating time steps

Typically, users will conduct ERRA analyses at the native resolution of the available time series. In some cases, however, it may be advantageous to aggregate those time series to coarser resolution (larger time steps). One motivation might be computational efficiency; if our time series has 10-minute time steps but we want to examine lags out to 1 week, that would be 1008 lag periods. The width of the design matrix will scale as the number of lag periods, times the number of different subsets being analyzed to assess nonstationarity (more about this below), times the number of knots if one wants to assess the nonlinearity of the system response. The order of difficulty of the matrix calculations in ERRA scales as the ***square***, or as an even higher power, of the width of the design matrix. Thus having so many lag periods can multiply the computational burden of these calculations dramatically.

A second and potentially more important reason for time step aggregation is that when time steps are much shorter than the response time of the hydrologic system, it becomes difficult to statistically distinguish the effects of inputs at one lag time from those of inputs at nearby lags. Figure xxf06 presents an illustrative example from the 3818 km^2 Clinch River basin. The left-hand panel shows NRFs estimated from hourly precipitation and streamflow data; the right-hand panel shows NRFs estimated from the same data, aggregated to 6-hour time steps. The NRFs in the left-hand panel have large uncertainties and exhibit coherent but spurious fluctuations around the expected rise and fall of the hydrological response. Both the large uncertainties and the spurious fluctuations are consistent with the strong first-order autocorrelation (ρ=0.9991) in the residuals. Aggregating the time steps to 6-hour intervals reduces the

first-order autocorrelation in the residuals to ρ=0.98, greatly reducing the uncertainties and largely eliminating the spurious features in the NRF.

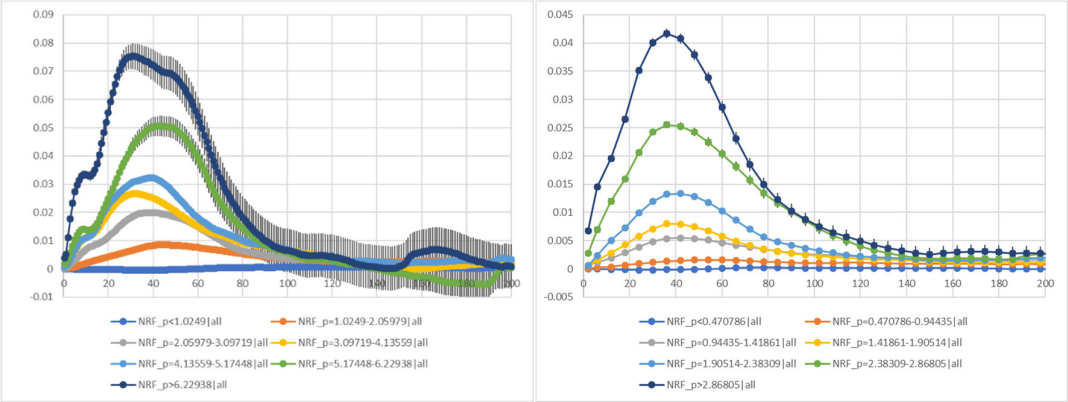

Figure xxf06. Nonlinear response functions (NRFs) estimated for Clinch River at Tazewell, TN, from hourly precipitation and streamflow data (left panel) and hourly data aggregated to 6-hour time steps (right panel). Error bars indicate one standard error. Time step aggregation narrows the distribution of precipitation values; thus the precipitation ranges and NRF values in the two panels differ.

Aggregating the input data can also help in clarifying the nonlinear response to precipitation intensity. If (say) the characteristic response time of the catchment is on the order of days, we would not expect that minute-to-minute variations in precipitation intensity will have a detectable effect on runoff; instead we would expect runoff to exhibit clearer responses to precipitation intensity variations that are averaged over timescales comparable to the catchment's response time. For this reason, it makes sense to aggregate high-frequency measurements so that the catchment's response is captured in several time steps, but not dozens or hundreds.

Time-step aggregation is invoked in ERRA by setting the `agg` parameter to an integer greater than 1. If `agg=3`, for example, time steps 1, 2, and 3 are averaged together, time steps 4, 5, and 6 are averaged together, and so on. Any numeric vectors or matrices are averaged; any boolean matrices or vectors are aggregated by majority rule.

To repeat the Saco River analyses shown in Section 5, but with aggregation of time steps to 3, 6, and 12 hours, one can use the following calls to ERRA:

```
zz <- ERRA(p=p, q=q, xknots=c(1, 2, 3, 4, 5, 6), xknot_type="values", agg=3,
m=34, Qfilter=snowfree, show_top_xknot=TRUE)

zz <- ERRA(p=p, q=q, xknots=c(0.7, 1.4, 2.1, 2.8, 3.5, 4.2),
xknot_type="values", agg=6, m=17, Qfilter=snowfree, show_top_xknot=TRUE)

zz <- ERRA(p=p, q=q, xknots=c(0.5, 1.0, 1.5, 2.0, 2.5, 3.0),
xknot_type="values", agg=12, m=9, Qfilter=snowfree, show_top_xknot=TRUE)
```

From these examples, readers will note that when time steps are aggregated, `m` should be reduced proportionally, since fewer lag steps will be required to cover the same range of lag time. Note also that when the time steps are aggregated, the xknots must be set at different values, because (for example) the distribution of 6-hour precipitation is different from the distribution of hourly precipitation. Users should also keep in mind that aggregating time steps will multiply the shortest time lags that are visible in ERRA.

The instructions above generate the analyses behind Figure 8 of K2024 (Figure xxf07 below). As these examples show, aggregation not only changes the distribution of precipitation rates (and thus the precipitation intervals over which the NRFs are calculated), it also changes the values of the NRFs themselves, although the overall pattern of behavior is similar.

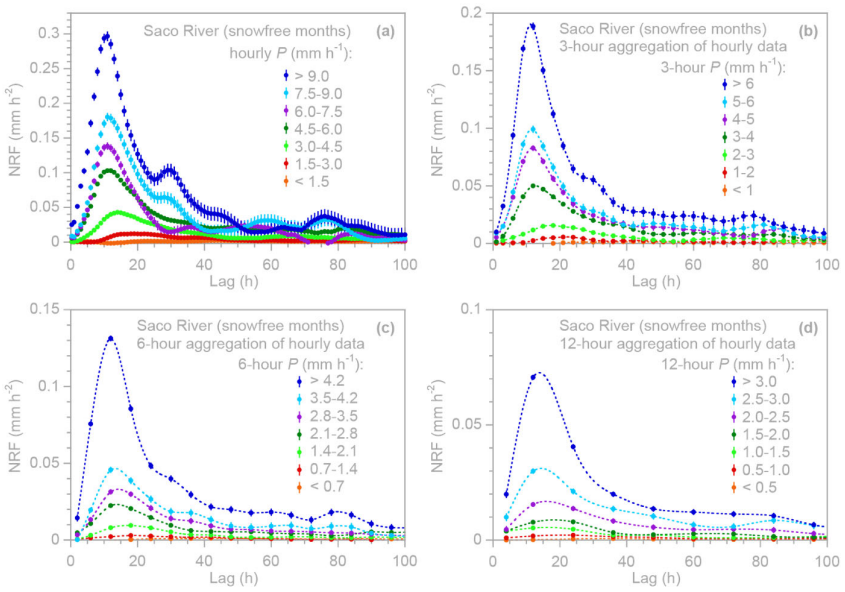

Figure xxf07 (Figure 8 of K2024). Effects of time step aggregation on inferred nonlinear runoff response of Saco River. Panels (a)-(d) show ERRA nonlinear response functions (NRFs) as functions of lag time for specified ranges of precipitation intensity, using hourly precipitation and streamflow time series (a), and the same data aggregated by averaging over 3 hours (b), 6 hours (c), and 12 hours (d). Average precipitation intensity is lower, and runoff responses are correspondingly more muted, when precipitation and discharge are averaged over longer spans of time. Source data span 1987-2003; months from November through April were omitted to exclude any effects of snow. Error bars show one standard error.

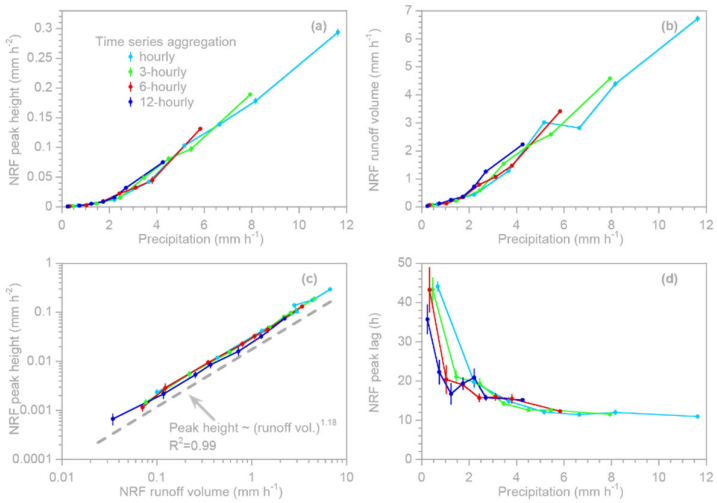

Figure xxf08 (Figure 9 of K2024). Nonlinear responses of peak heights, peak lags, and runoff volumes to variations in precipitation intensity at Saco River, estimated from precipitation and streamflow data aggregated over different time intervals. Time step aggregation reduces average precipitation intensity and damps the resulting nonlinear response functions (NRFs; Figure xxf07). However, the nonlinear response curves relating peak heights (a) and runoff volumes (b) to precipitation intensity plot on top of one another at different levels of time step aggregation, as do the power-law relationships between peak heights and runoff volumes (c; dashed gray line is offset from the data for clarity). Time step aggregation has little effect on lag-to-peak (d) at high precipitation intensities, but may have a more noticeable effect at low precipitation intensities. Source data span 1987-2003; months from November through April were omitted to exclude any effects of snow. Error bars show one standard error.

The same calls to ERRA provide the data for Figure xxf08 above (Figure 9 of K2024), comparing the peak statistics for different levels of time step aggregation at Saco River. Time step aggregation implies changes in the knot values of precipitation intensity and changes in NRF peak heights, peak lags, and runoff volumes. However, as Figure xxf08 shows, the relationships between these peak statistics and precipitation intensity are consistent across the different levels of time step aggregation shown here (at least at Saco River; it is not known how widely this behavior holds).

7. Weighted average runoff response distributions

In nonlinear analyses like those described in Section 6 above, it would be theoretically possible to consider each NRF as an RRD for each range of precipitation, by dividing by the precipitation rate (inverting Eq. 7 of K2024). However, these RRDs would not reflect the nonlinear dependence of runoff response on precipitation intensity, at least in a way that is straightforward to interpret. In the case of a quadratic runoff response like the one shown schematically in Figure xxf04, the RRD would be a *linear* function of precipitation rate, and in the case of linear runoff response (a linear increase in NRF with precipitation rate), the RRD would be the same for different precipitation rates. A further difficulty is that for precipitation near zero, with a small but uncertain runoff response, the RRD would be very noisy.

Instead, to characterize the average runoff response of a nonlinear system, ERRA estimates the precipitation-weighted average RRD (`wtd_avg_RRD`) as the average NRF divided by the average precipitation, shown schematically by the slope of the dashed line leading to the red square in Figure xxf04. The corresponding peak statistics table is `avgRRD_peakstats`. This can be particularly helpful in quantifying how changes in ambient conditions (such as antecedent wetness, see below) affect runoff response in nonlinear systems, if users do not care much about the nonlinearity itself, but instead are primarily interested in the average system response.

As noted in Section 3.4 of K2024, this precipitation-weighted average is a more realistic estimate of the average catchment runoff response than an RRD calculated without taking nonlinearity into account. By contrast, a "regular" RRD (estimated without using xknots to account for nonlinear response) will be weighted by the leverage of each data point, which is approximately P^2 in typical highly skewed precipitation distributions; it will therefore mostly reflect runoff response to high-intensity precipitation. Thus even if one does not want to explore the nonlinearity in the system's response, accounting for that nonlinearity may be important for accurately quantifying the system's average response (that is, the precipitation-weighted RRD reported in `wtd_avg_RRD`, rather than the P^2-weighted RRD that results if `xknots==NULL`.

Figure xxf09 below (Figure 10 of K2024) compares the weighted average RRD to its "unweighted" (but in practice P^2-weighted) counterpart for five MOPEX sites in the southeastern US. The source data files (MOPEX AnacostiaR hourly.txt, MOPEX NantahalaR hourly.txt, MOPEX SFNewR hourly.txt, MOPEX FrenchBroadR hourly.txt, and MOPEX ClinchR hourly.txt) contain hourly precipitation and streamflow beginning in 1985-1990 (depending on the site) and extending through 2003. Generating the necessary data for Figure xxf09 is straightforward, combining analyses with and without xknots:

```
dat <- fread("MOPEX NantahalaR hourly.txt", header=TRUE, sep="\t",
na.strings=c("NA",".","","#N/A"))    # read the data file
fileID <- "Nantahala_hourly"

p <- dat$p
```

```
q <- dat$q

# run ERRA without nonlinear analysis
zz <- ERRA(p=p, q=q, m=100)
# run ERRA with nonlinear analysis
ww <- ERRA(p=p, q=q, m=100, xknots=c(6,20), xknot_type="even")

with(ww, {                                          # write the output to files
  fwrite(cbind(zz$RRD, wtd_avg_RRD[,-1]), paste0(fileID, "_RRDs_", options,
".txt"), sep="\t")  # combine RRD and weighted_average_RRD
  fwrite(peakstats, paste0(fileID, "_peakstats_", options, ".txt"), sep="\t")
})
```

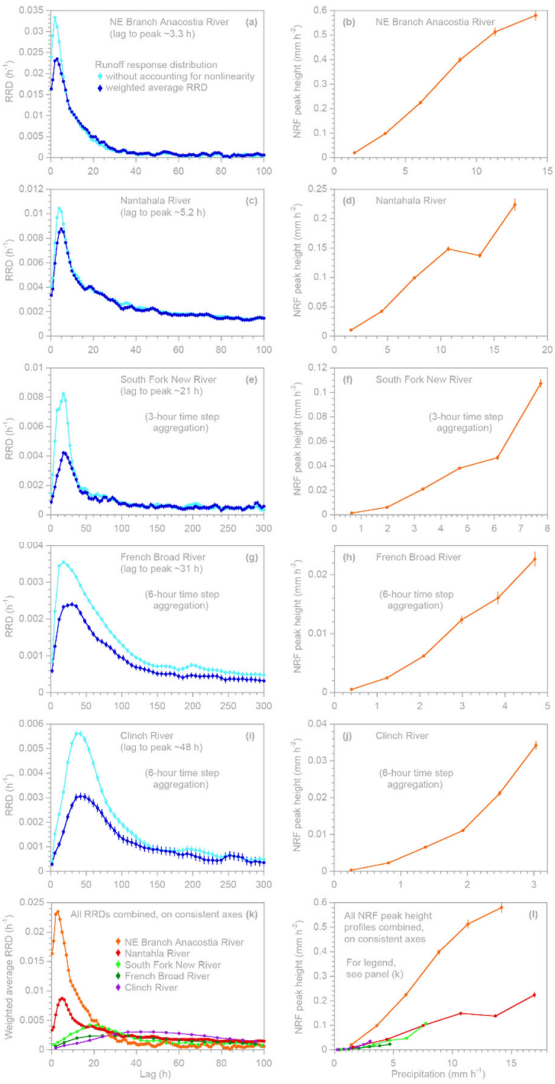

Figure xxf09 (Figure 10 of K2024). Runoff Response Distributions (RRDs) calculated with (dark blue) and without (light blue) accounting for nonlinear response to precipitation intensity (left panels), and profiles of Nonlinear Response Function (NRF) peak height as a function of precipitation intensity (right panels), for five mesoscale river basins in the southeastern US. Basins are the Northeast Branch of the Anacostia River at Riverdale, MD (a, b: 189 km^2), Nantahala River near Rainbow Springs, NC (c, d: 134 km^2), South Fork New River near Jefferson, NC (e, f: 531 km^2), French Broad River at Asheville, NC (g, h: 2448 km^2), and Clinch River above Tazewell, TN (i, j: 3818 km^2). Weighted average RRDs, calculated via Eq. (12) of K2024 to take account of nonlinear response (dark blue, left panels), are less strongly peaked and peak a bit later than RRDs calculated without taking nonlinearity into account (light blue, left panels). Note that the axis scales differ substantially among the different panels. All sites' weighted average RRDs and NRF peak height profiles are shown on consistent axes in panels (k) and (l). At several sites with delayed and long-lasting runoff response, time steps were aggregated to 3 or 6 hours to prevent the residual autocorrelation from becoming excessive. Error bars indicate one standard error.

8. Quantifying nonstationary rainfall-runoff behavior

As outlined in Section 4 of K2024, ERRA allows users to split the precipitation record according to different conditions prevailing at the time that the precipitation falls, and to track the landscape's runoff responses under these different conditions, even if they overlap with one another (for example, even if the runoff response to precipitation falling on dry ground overlaps with the runoff response to later precipitation that falls on wetter ground). Users who want to understand the underlying mathematics are referred to Section 4 of K2022 or the briefer summary in Section 4 of K2024; users who want to understand how this works computationally are referred to the scripts themselves. Here I will instead describe how to set up such an analysis and how to interpret the results.

Splitting the precipitation record according to ambient conditions is controlled by the `split_params` option in `ERRA`. Users tell ERRA how to split the precipitation time series by specifying a list of splitting parameters. Here is an example of a `split_params` list, which splits by antecedent discharge (as a proxy for antecedent wetness):

```
antQparams <- list(
    crit = list(q) ,
    crit_label = c("antQ") ,
    crit_lag = 1 ,
    breakpts = list(c(0.1, 0.2, 0.5, 1)) ,
    pct_breakpts = FALSE ,
    thresh = 0.0 ,
    by_bin = FALSE
)
```

`crit` specifies a `list` of one or more variables that will be used as splitting criteria. This must be specified as a `list`, even if, as in this case, it consists of just a single vector of data. In this case, we specify the discharge vector `q`, because we will use lagged values of discharge as an indication of antecedent wetness conditions. If we specify multiple criteria, they will be nested in the order specified (see Section 9 for an example with two nested criteria).

`crit_label` specifies a short text label for the variable in `crit` (or a vector of such labels, if `crit` contains multiple criterion variables). Here our label is `antQ`, reflecting discharge lagged by 1 time step.

`crit_lag` is an integer (or vector, if there are multiple criteria) specifying the lag in ***number of time steps*** (not time units) to be applied to the variable(s) specified in `crit`. Here we specify a lag of 1 time step. Note that these lags are specified in time steps ***following*** any data aggregation implied by the `agg` parameter in `ERRA`. For example, if we aggregated the data to 6-hourly time steps by specifying `agg=6`, setting `crit_lag=2` would create a lag of 12 hours (two aggregated time steps).

`breakpts` is a `list` of vectors, one for each variable in `crit`, specifying the values at which the precipitation record should be split. Again, this must be specified within a `list()` function, even if it consists of a single vector (or even a single value).

`pct_breakpts` is a boolean flag (or a vector of such flags, if there are multiple criteria), indicating whether the breakpoints specified in `breakpts` are expressed in percentiles (`TRUE`) or in the units of the criterion variables themselves (`FALSE`). If pct_breakpts is TRUE but all the breakpoints are less than 1,

ERRA will assume that the user has accidentally specified the breakpoints as fractions rather than percentages, and will convert them to percentages instead.

`thresh` is a numeric value (or a vector of such values, if there are multiple criteria), specifying the thresholds above which percentiles will be calculated for the variable(s) specified in `crit`. If `pct_breakpts` is `FALSE` for the variable in question, thresh has no effect. But if `pct_breakpts` is `TRUE` (and thus the splitting breakpoints are set by percentiles), those percentiles will only be calculated from values greater than `thresh`. This has the advantage that users do not need to know what fraction of the `crit` distribution is below `thresh`, which may be large in highly skewed distributions (such as antecedent precipitation, where many values may be zero and one wants to specify percentiles of the non-zero values instead). All values below or equal to thresh will be included in the lowest bin of `crit`, below the first breakpoint; they just won't be used to convert the percentile breakpoints to their corresponding values of the `crit` variable.

`by_bin` is a boolean flag (or vector of such flags, for multiple criteria), specifying whether, for nested criteria, percentile breakpoints should be calculated separately for each bin that they are nested into. Why would this be useful? Consider a case where we want to split precipitation according to cold vs. warm conditions, and according to antecedent discharge within both cold and warm conditions. If `by_bin=FALSE`, then (say) a breakpoint of 95 percent would be set according to the 95th percentile of antecedent discharge over both warm and cold conditions combined. But if this breakpoint value exceeds all of the antecedent discharges under cold conditions, the analysis will fail since the above-95-percent bin under cold conditions will have nothing in it. Setting `by_bin=TRUE` avoids this problem by calculating the breakpoint percentiles separately for each nested bin (separately for warm and cold conditions, in our example). If `by_bin=TRUE`, then `pct_breakpts` must be `TRUE` for the corresponding criterion variable, or an error and a hard stop will result (because the breakpoints can't vary in nested bins, if they are specified as values rather than percentiles). `by_bin` also has no effect on the first criterion variable (which is not nested within any other criterion), but here also a value of `TRUE` or `FALSE` must be specified nonetheless (and if `by_bin=TRUE`, even for the first or only variable, then `pct_breakpoints` must also be `TRUE`).

Breakpoint values must not lie outside the range of values of the corresponding criterion variable (or subsets of those values, for criteria that are nested inside other criteria). This will trigger an error message and a hard stop, because it would generate an empty column of the design matrix. This problem can be avoided by specifying breakpoints as percentiles, particularly for criteria that are nested inside of other criteria, and specifying `by_bin=TRUE` for those criteria (this has the drawback that the numerical values of the breakpoints will usually vary from bin to bin, however).

The breakpoints must not result in bins that contain no non-zero precipitation values, because this will lead to columns of zeroes in the design matrix, and thus to singularities in the crossproducts matrix. If you get a message like "*Error in solve.default(C, xy): Lapack routine dgesv: system is exactly singular: U[146,146] = 0*", this is usually the cause. In this example it is column 146 of the design matrix that is the problem, so that will give you a clue where to look. One way these singularity errors commonly arise (*don't ask me how I know…*) is when `pct_breakpts=FALSE` but the elements in `breakpts` are actually specified in percentages, or vice versa, because this commonly leads to one or more of the specified bins having no members. These singularity errors are not a computational glitch; the user has simply asked for the solution of a mathematically impossible problem.

Whew! I know it may seem complicated, but these lists of splitting parameters allow users to employ many different strategies for splitting the precipitation record according to multiple, potentially nested, criteria in order to explore how catchment runoff response varies under different ambient conditions.

In K2024, this approach is demonstrated using hourly precipitation and weather data from the 3.6 km^2 Hafren catchment at Plynlimon in mid-Wales. The Plynlimon data set used here, "Plynlimon hourly data for ERRA" runs from the mid-1970's through 2010, comprising over 300,000 hourly time steps at multiple stream gauges and weather stations, all of which have been meticulously quality-controlled by hand, as outlined in the documentation "Plynlimon AWS and discharge data editing notes.pdf". At some future time, this data set will be available through the data portal of the UK Centre for Ecology and Hydrology, and a link to that archive will be provided as soon as it is known.

The data are loaded, as usual, with:

```
dat <- fread("Plynlimon hourly data for ERRA.txt", header=TRUE, sep="\t",
na.strings=c("NA",".","","#N/A"))
```

and the relevant precipitation and discharge time series are extracted with:

```
p <- dat$P_Hafren_Tanllwyth_Hore_UpperHore_mm.hr
q <- dat$Hafren_avgQ_mm.hr
fileID <- "Hafren"
```

Figure xxf10a shows RRDs for Hafren streamflows, with precipitation split by antecedent discharge (antQ) using the splitting parameters presented above. The call to ERRA is simply:

```
zz <- ERRA(p=p, q=q, m=100, split_params=antQparams)
```

and the necessary outputs can be saved as

```
with(zz, {
  fwrite(RRD, paste0(fileID, "_RRD_", options, ".txt"), sep="\t")
})
```

The column headings of this output file are in the form of `RRD_p|antQ0.1-0.2`, which can be read as the RRD for a single precipitation time series with the default label `p`, and a range of `0.1-0.2` of the criterion variable `antQ` (antecedent discharge), as defined in the splitting parameter list.

Analyzing the effects of antecedent wetness will often require also accounting for the nonlinear effects of precipitation intensity, because precipitation intensity and ambient conditions are often correlated with one another. For example, the same wet weather conditions that lead to high antecedent wetness will also often increase the likelihood of intense precipitation. This would lead to distorted estimates of how either antecedent wetness or high-intensity precipitation would affect the runoff response, unless the two factors are analyzed jointly. In ERRA, this can be accomplished straightforwardly.

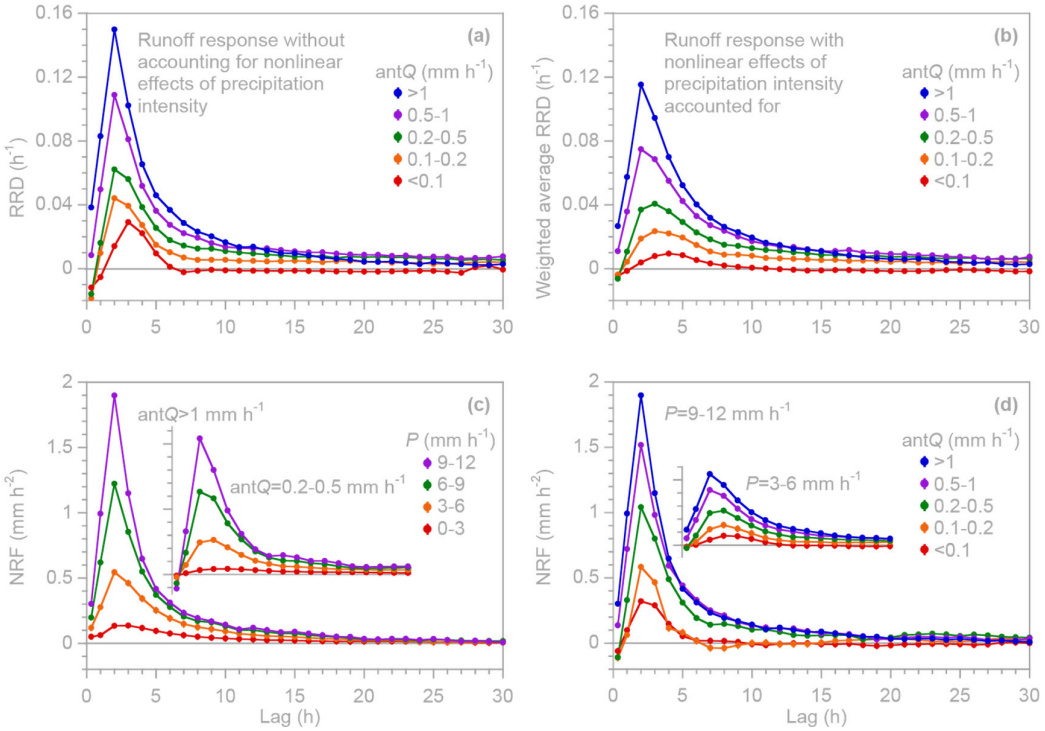

Figure xxf10 (Figure 12 of K2024). Runoff responses at Hafren, estimated via Eq. (16) of K2024 as functions of precipitation intensity *P* and antecedent wetness (using 1-hour antecedent discharge, ant*Q*, as a proxy). (a) Runoff response distributions (RRDs) for different ranges of ant*Q*, calculated using Eq. (15) of K2024 without accounting for nonlinear effects of *P* or co-variation between *P* and ant*Q*. Peak runoff response is somewhat exaggerated, due to the greater leverage of high precipitation values (see text). (b) Precipitation-weighted average RRDs for different ranges of ant*Q*, with nonlinear effects of *P*, and co-variation between *P* and ant*Q*, taken into account via Eq. (15) of K2024. Peak runoff responses in weighted average RRDs (b) are less pronounced relative to unweighted RRDs (a), particularly under drier antecedent conditions. (c) Effects of variations in precipitation intensity *P* (shown by different colors) for two example ranges of ant*Q* (as proxy for antecedent wetness), illustrating how runoff response under drier antecedent conditions (lower ant*Q*: inset figure) is less pronounced across all ranges of precipitation intensity. (d) Effects of variations in ant*Q* (shown by different colors, as proxy for antecedent wetness) for two example ranges of precipitation intensity *P*, illustrating how runoff response to lower-intensity precipitation (lower *P*: inset figure) is less pronounced across all ranges of ant*Q*. Insets in (c) and (d) are on the same axis scales as the main figures, but are cropped and offset for compact presentation. Error bars indicate one standard error, where this is larger than the plotting symbols.

For example, Figure xxf10b,c,d shows NRFs and weighted average RRDs that account for both antecedent wetness and the nonlinear dependence of runoff response on precipitation intensity. The necessary analysis can be run by adding xknots to the ERRA function call outlined above:

```
zz <- ERRA(p=p, q=q, xknots=c(3, 6, 9, 12), xknot_type="values", m=100,
split_params=antQparams)
```

and saving the results with:

```
with(zz, {
  fwrite(peakstats, paste0(fileID, "_peakstats_", options, ".txt"), sep="\t")
  fwrite(wtd_avg_RRD, paste0(fileID, "_avgRRD_", options, ".txt"), sep="\t")
  fwrite(NRF, paste0(fileID, "_NRF_", options, ".txt"), sep="\t")
})
```

The peakstats file from this same function call also provides the results shown in Figure 13 of K2024 (Figure xxf11 below).

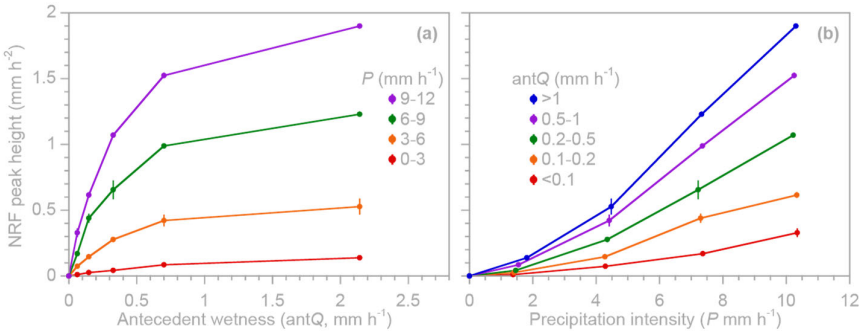

Figure xxf11 (Figure 13 of K2024). Peak runoff response at Hafren as a joint function of precipitation intensity *P* and antecedent wetness (using 1-hour antecedent discharge, ant*Q*, as a proxy). (a) Peak height of the nonlinear response function (NRF) as a function of antecedent wetness for four ranges of precipitation intensity (shown by different colors). (b) Peak height as a function of precipitation intensity for five ranges of antecedent wetness (shown by different colors). Error bars indicate one standard error, where this is larger than the plotting symbols.

9. Splitting by multiple criteria to quantify nonstationary behavior

What if we want to analyze how runoff response to precipitation varies depending on both the antecedent vapor pressure deficit just before the rain falls, ***and*** the antecedent wetness (as proxied by the antecedent discharge an hour beforehand)? The `ERRA` script can do this; in fact, it can split the precipitation time steps according to an ***unlimited*** number of nested criteria (unlimited, that is, subject to the availability of sufficient data).

In Figure xxf12 (Figure 15 of K2024), peak runoff response at Hafren is compared for different precipitation intensities, different antecedent wetness conditions (antQ), and different antecedent vapor pressure deficit conditions (antVPD). Here is how this is done. First we extract the columns of precipitation, discharge, and VPD from the data set:

```
p <- dat$P_Hafren_Tanllwyth_Hore_UpperHore_mm.hr
q <- dat$Hafren_avgQ_mm.hr
vpd <- dat$AllSitesAWS_VPD
```

Then we set up a list of splitting parameters as follows:

```
qbreakpts <- c(0.1, 0.2, 0.5, 1)
vbreakpts <- c(0.0932) #60% of VPD distribution
antVPD_Qparams <- list(crit = list(vpd, q) ,
                   crit_label = c("antVPD", "antQ") ,
                   crit_lag = c(1, 1) ,
                   breakpts = list(vbreakpts, qbreakpts) ,
                   pct_breakpts = c(FALSE, FALSE) ,
                   thresh = c(-999, 0.0) ,
                   by_bin = c(FALSE, FALSE)
) # end antVPD_Qparams
```

The vector `qbreakpts` and the single-element vector `vbreakpts` contain the splitting breakpoints for antecedent discharge and antecedent VPD, respectively. The splitting criterion list `crit = list(vpd, q)` tells ERRA to first divide the precipitation records into two ranges of antecedentVPD (above and below

0.0932, which is the 60th percentile of the VPD distribution), and then to divide each of these bins further into five different bins of antecedent discharge. The line `crit_label = c("antVPD", "antQ")` provides text labels for each of the splitting criteria, and the line `crit_lag = c(1, 1)` says that precipitation will be filtered into separate bins based on the VPD and discharge in the previous time step (i.e., lagged by 1 time step relative to precipitation). The line `breakpts = list(vbreakpts, qbreakpts)` specifies the breakpoints in VPD and discharge, respectively, that will be used to filter the precipitation time series into bins. The line `pct_breakpts = c(FALSE, FALSE)` says that the breakpoints for both of these variables are specified as values rather than percentiles. Because the breakpoints are specified as values, the line `thresh = c(-999, 0.0)` has no effect, but if they were percentiles instead, it would specify the lower thresholds of values that would be included in estimating those percentiles. Lastly, `by_bin = c(FALSE, FALSE)` has no effect because `qbreakpts` is specified as values rather than percentiles, but if they were percentiles instead, the second element of `by_bin` would specify whether those percentile breakpoints should be applied separately to each VPD bin (`TRUE`) or jointly to both VPD bins (`FALSE`).

These splitting parameters can be invoked by the following call to ERRA, generating the data for Figure xxf12:

```
zz <- ERRA(p=p, q=q, xknots=c(2, 4, 6, 10), xknot_type="values", m=100,
split_params=antVPD_Qparams)

with(zz, fwrite(peakstats, paste0(fileID, "_peakstats_", options, ".txt"),
sep="\t") )
```

This analysis splits the precipitation time series into a total of 50 separate variables, representing two categories of VPD (above and below the 60th percentile), five categories of antecedent discharge (<0.1, 0.1-0.2, 0.2-0.5, 0.5-1, and >1), and five categories of precipitation intensity (<2, 2-4, 4-6, 6-10, and >10), of which four are shown (the highest is not shown because `show_top_xknot` is `FALSE` by default). The resulting calculations are roughly $(2*5*5)^2$=2500 times more difficult than a simple RRD calculation without any splitting of the precipitation time series, but are nonetheless tractable on an ordinary laptop (even when, in this case, there are over 300,000 time steps of each variable to be handled). A further problem is that as the precipitation time series is split among more and more categories, the chances rise that one or more categories will contain only zeroes or missing precipitation values (which makes the underlying equations unsolvable). If the minimum number of non-zero, non-missing precipitation values among any of the precipitation columns is less than `low.data.warn` (a user-specifiable parameter that is 16 by default), ERRA issues a warning but continues to run. If it is less than `low.data.fail` (user specifiable; 8 by default), ERRA issues an error message and stops.

Figure xxf12 shows that the ERRA call shown above can quantify how variations in antecedent VPD, ***and*** in antecedent wetness, ***and*** in precipitation intensity, jointly influence runoff response at Hafren, even though all of these effects are overprinted on one another in streamflow. As noted in K2024, whenever these potential drivers are correlated with one another, seeing their influences clearly will require analyzing them jointly, as in the figure below. By contrast, Figure 14 of K2024 provides a cautionary tale of what can happen if these connections are overlooked. In Figure 14 of K2024, the estimated effect of VPD variations is huge, because VPD is correlated with antecedent wetness; the strong effect of antecedent wetness appears to be coming from VPD instead, because antecedent wetness is a "hidden variable" that does not appear in Figure 14 of K2024 and its effects are instead aliased as effects of VPD.

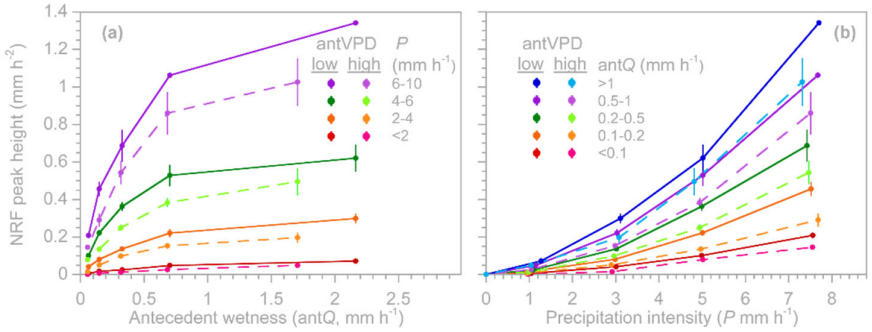

Figure xxf12 (Figure 15 of K2024). Peak runoff response at Hafren, as a joint function of precipitation intensity *P* and antecedent wetness (using 1-hour lagged discharge, ant*Q*, as a proxy), for low antecedent vapor pressure deficit (antVPD) conditions (darker colors and solid lines: lowest 60% of VPD values) and high antVPD conditions (lighter colors and dashed lines: highest 40% of VPD values). (a) Peak height of the nonlinear response function (NRF) as a function of antecedent wetness for four ranges of precipitation intensity (shown by different colors). (b) Peak height as a function of precipitation intensity for five ranges of antecedent wetness (shown by different colors). Precipitation intensity ranges are more limited than in Figs. 12-13 to provide enough time steps in each combination of *P* and antVPD. Error bars indicate one standard error, where this is larger than the plotting symbols. Jointly accounting for the effects of precipitation intensity and antecedent wetness shows that, all else equal, peak runoff response is roughly 20-30% lower following periods of high VPD (dashed lines) than following low VPD (solid lines). However, peak runoff response following both low and high VPD exhibits similar dependency on antecedent wetness and precipitation intensity.

10. Robust estimation

ERRA includes the capability for robust estimation via Iteratively Reweighted Least Squares (IRLS). Robust estimation can be invoked by setting `robust=TRUE` (the default is `FALSE`). Robust estimation has the advantage of making the results less sensitive to outliers in the underlying data. IRLS achieves this by downweighting points that have unusually large residuals. Applying such techniques to rainfall-runoff analysis can be tricky, however, because precipitation distributions are typically very strongly skewed, with the consequence that runoff responses to rare, intense precipitation inputs can be flagged as outliers and effectively ignored. If runoff response depends nonlinearly on precipitation intensity (as it typically does), robust estimation may yield an RRD that looks substantially different, even if the underlying data contain no true outliers.

As a cautionary tale, we can illustrate this phenomenon using the Hafren time series data that were analyzed in Sections 8 and 9 above. We tell ERRA to estimate an "ordinary" RRD, ignoring nonlinear response to precipitation, with and without robust estimation:

```
zz1 <- ERRA(p=p, q=q, m=100, robust=FALSE)         # without robust estimation

zz1 <- ERRA(p=p, q=q, m=100, robust=TRUE)         # now WITH robust estimation
```

and we compare the results. As the left panel in Figure xxf13 below shows, robust estimation yields an RRD with a substantially smaller peak. To caution users that this may happen, ERRA prints a warning whenever it detects that robust estimation is being used without accounting for nonlinear effects of precipitation intensity:

```
Robust estimation has been selected together with xknots==NULL.  This may
result in underestimation of runoff response due to downweighting of high
precipitation inputs.  Users are generally advised to use robust estimation
```

```
only when xknots are also used to account for nonlinear response to
precipitation intensity.
```

We can test the effects of robust estimation on runoff response estimates from nonlinear deconvolution (i.e., estimates that account for nonlinear effects of precipitation intensity). To do that, we first calculate a non-robust estimate of that runoff response :

```
zz <- ERRA(p=p, q=q, m=100, xknots=c(3, 6, 9, 12), xknot_type="values",
robust=FALSE)
```

and then compare it with a robust estimate:

```
zz <- ERRA(p=p, q=q, m=100, xknots=c(3, 6, 9, 12), xknot_type="values",
robust=TRUE)
```

The results are shown in the right panel of Figure xxf13 below. As one can see, when the nonlinear effects of precipitation intensity are taken into account (right panel), robust and non-robust estimation yield very similar results. When those nonlinear effects are not accounted for (left panel), the non-robust RRD peak is much higher, and the robust RRD is similar to both the robust and non-robust RRDs in the right-hand panel. One should not infer from this one example that robust estimation always has equivalent effects to nonlinear deconvolution, because they are doing fundamentally different things. Robust estimation reduces the influence of unusual points, and nonlinear deconvolution quantifies nonlinear effects. They both diminish the influence of high-intensity precipitation on the RRD results, but in different ways that are coincidentally similar in magnitude in this case.

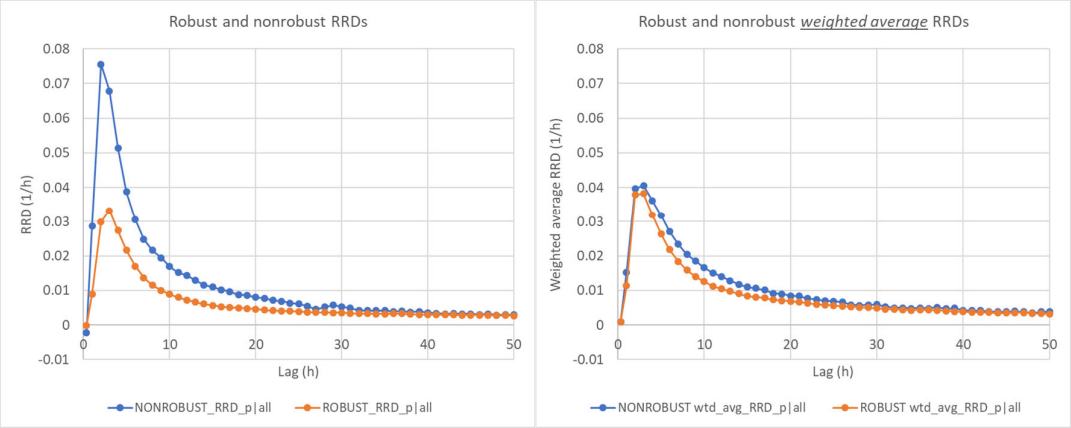

Figure xxf13. Robust and non-robust RRDs for Hafren data. When nonlinear effects of precipitation intensity are ignored (left panel), robust estimation yields a much smaller runoff response because it reduces the influence of rare, high-intensity precipitation inputs. When nonlinear effects of precipitation intensity are accounted for using xknots (right panel), robust and non-robust estimation methods yield similar RRDs.

11. Piecewise-linear RRDs for multiscale runoff responses

Runoff responses often span a range of timescales, with a strong, sharp peak at short lags, but a long tail extending to very long lags. In theory, of course, such multi-scale behavior could be captured by evaluating the RRD and/or NRF for a very large number of very closely spaced lags (for example, 720 hourly lags to capture responses out to timescales of 30 days). In practice, this can result in a very noisy tail, because the delayed runoff response is too weak to adequately constrain the large number of individual lag

coefficients. Wouldn't it make sense to somehow estimate the RRD at closely spaced lags for short lag times, and widely spaced lags for long lag times? It would indeed make sense, and the ERRA script provides a way to do exactly this. As explained in Section 6 of K2022, ERRA can make a piecewise-linear approximation to the RRD, as defined by knot points that are geometrically distributed in lag time with close spacing at short lags, and increasingly wide spacing at longer lags (see Figure xxf14 below).

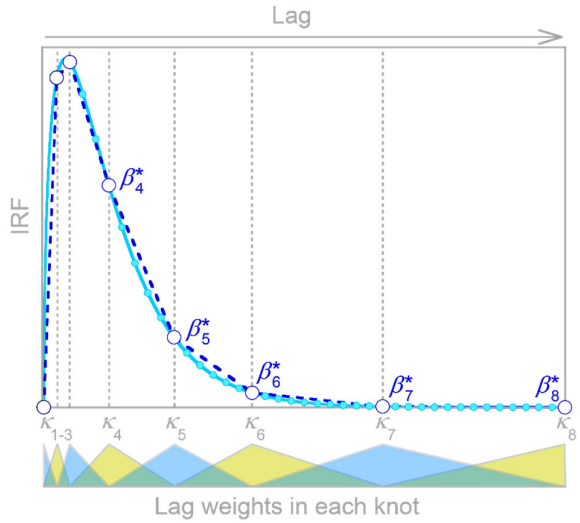

Figure xxf14 (Figure 15 of K2022). Piecewise linear approximation (dashed dark blue line) to an impulse response function (IRF, light blue line), with 40 conventional IRF coefficients (light blue dots) replaced by values at 8 knots (open circles). The overlapping yellow and blue triangles depict the relative influence of each lag on the even and odd numbered knots, respectively (see Equations 51-52 of K2022).

Users should note that piecewise-linear broken-stick approximations are used for two different purposes in ERRA. First, as described in Section 5 above, a broken-stick model is used to quantify the nonlinear effects of precipitation intensity on runoff response. In that application, the knots are called xknots. Second, as described here and shown in Figure xxf14 above, a broken-stick model can also be used to approximate the RRD or NRF between irregularly spaced knots (which, here, are actually called knots).

Invoking this second piecewise-linear approximation, as shown in Figure xxf14 above, is simple in ERRA: users just need to specify the number of knots they want, using the `nk` parameter (where `nk` stands for "number of knots"). If `nk` is greater than two and less than the maximum lag `m`, ERRA will construct a geometric series of `nk` knots between lags 0 and `m`. (or more precisely, as close to a geometric series as possible, given that the knots must be at integer lags).

Figure 16 of K2024 (Figure xxf15 below) demonstrates how this approach can be used to quantify the long tails of recession curves, using the Hafren dataset from Sections 8-10 above as a test case. It shows weighted average RRDs calculated at lags of up to 1000 hours using both non-robust and robust estimation (light blue and dark blue, respectively):

```
zz <- ERRA(p=p, q=q, xknots=c(5, 20), m=1000, h=3, xknot_type="even",
robust=FALSE)
zz <- ERRA(p=p, q=q, xknots=c(5, 20), m=1000, h=3, xknot_type="even",
robust=TRUE)
```

compared to the same weighted average RRDs calculated for 40 knots between lags of 0 and 1000 hours (orange and yellow for non-robust and robust, respectively:

```
zz <- ERRA(p=p, q=q, xknots=c(5, 20), m=1000, nk=40, h=3, xknot_type="even",
robust=FALSE)

zz <- ERRA(p=p, q=q, xknots=c(5, 20), m=1000, nk=40, h=3, xknot_type="even",
robust=TRUE)
```

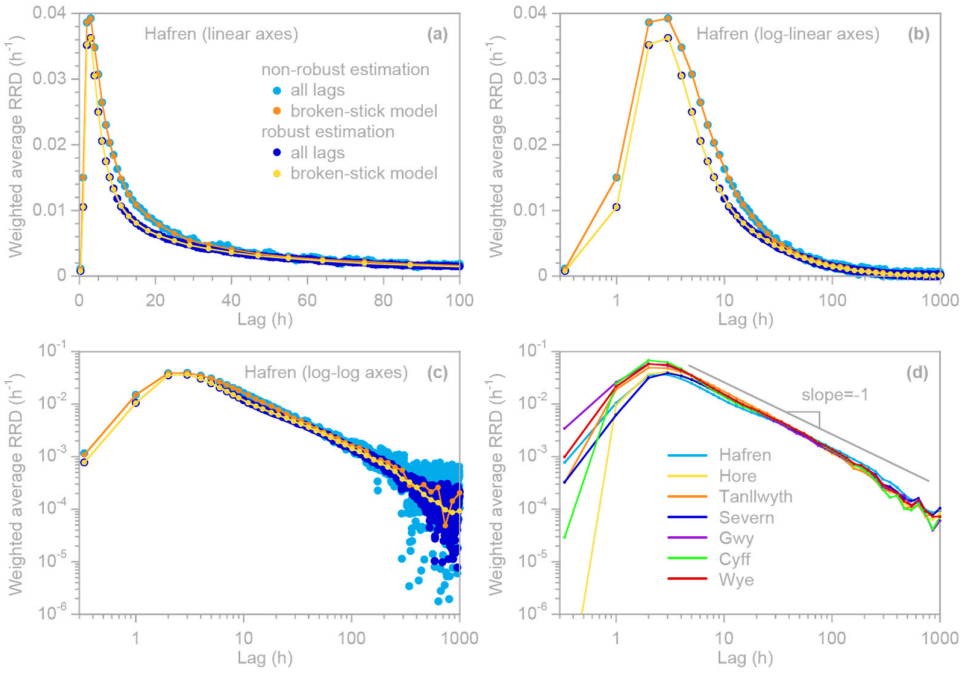

Figure xxf15 (Figure 16 of K2024). Long-tail recession behavior at Hafren and six other Plynlimon streams. Panels (a), (b), and (c) show precipitation-weighted average runoff response distributions (RRDs) for Hafren on linear, log-linear, and log-log axes, respectively. Each of these panels shows RRDs calculated at every lag using non-robust and robust estimation (light blue and dark blue, respectively), and calculated using a piecewise-linear broken-stick model over a geometric progression of lag intervals using non-robust and robust estimation (orange and yellow, respectively). At larger lags, the broken-stick approach averages the runoff response over longer lag intervals, thus greatly reducing the scatter in the long tail of the RRD. Robust estimation further reduces the scatter in the RRD tail by limiting the influence of individual data points with large residuals. Panel (d) shows robust broken-stick RRDs for 7 Plynlimon streams. All 7 streams have recession limbs that scale as approximately τ^{-1} over more than two orders of magnitude in lag time τ.

As panel (c) in Figure xxf15 makes clear, one advantage of the piecewise-linear broken-stick model is that the average RRD or NRF at long lag times can be estimated with much greater precision. A further advantage is that estimating the broken-stick model may be much faster than estimating many evenly spaced lags to cover the same lag interval (although there is significant overhead in setting up the broken-stick approach, so there is usually no speed advantage unless `m` is larger than several hundred).

However, users should note that, as explained in K2022, the AR noise correction can misbehave when the piecewise linear approximation is used. In particular, the automatic AR selection method may not converge, because the residual PACF may never become small enough. Thus a more pragmatic approach to selecting the AR order parameter `h` is (as shown above) to make a generous guess and then verify either that the residual PACF is well behaved, or that the AR coefficients are dying away to nearly zero (so that further increases in `h` would not help).

12. Comparing fitted and observed hydrographs

With the proviso that *the objective of ERRA is not to predict hydrographs*, ERRA does provide time series of fitted and observed discharges via the `Qcomp` data table. Here I illustrate how these can be used to check the plausibility of ERRA results, using the rainfall-runoff time series from the Hafren catchment at Plynlimon (which has also been featured in Sections 8-11 above). As Figure xxf16 shows, streamflow at Plynlimon is higher during the winter, both because more precipitation falls then, and because evapotranspiration rates are much lower. Two prominent storms are also visible in the time series. The first, on 05 July 1976, dropped 26 mm of rain in 1 hour, but produced little runoff response. The second, on 15 August 1977, dropped a total of 92 mm of rain in 3 hours, and generated the highest streamflows (by about a factor of 3) in the entire 35-year record.

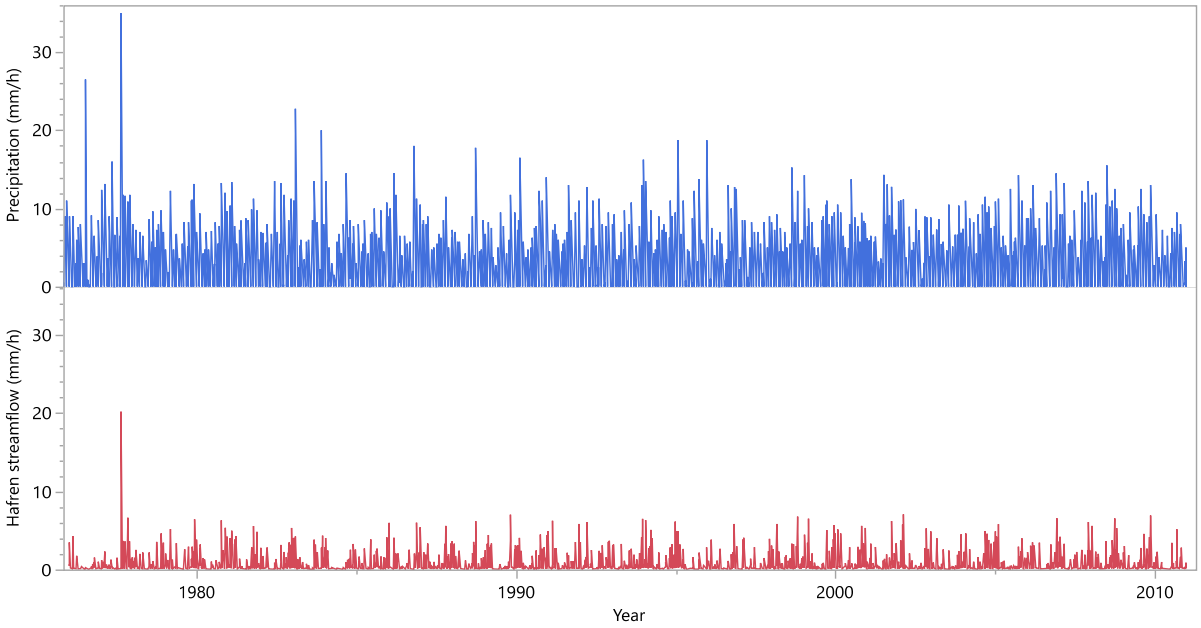

Figure xxf16. Hourly precipitation and streamflow at the Hafren catchment at Plynlimon.

We can begin by setting parameters to divide the precipitation time series according to antecedent wetness, as proxied by antecedent discharge:

```
breakpts <- c(0.1, 0.2, 0.5, 1)
antQparams <- list(crit = list(q) ,
                   crit_label = c("antQ") ,
                   crit_lag = 1 ,
                   breakpts = list(breakpts) ,
                   pct_breakpts = c(FALSE) ,
                   thresh = 0.0 ,
                   by_bin = c(FALSE)
) # end split_params
```

And then perform a nonlinear, nonstationary rainfall-runoff analysis using the methods of Sections 8 and 11 together:

```
zz <- ERRA(p=p, q=q, xknots=c(4, 20), xknot_type="even", m=1000, nk=40, h=3,
split_params=antQparams)
```

We can then get an overview of what is in the Qcomp table with

```
str(zz$Qcomp)
```

which yields:

```
Classes 'data.table' and 'data.frame':    325542 obs. of  7 variables:
 $ timestep : int  1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 ...
 $ time     : num  1004 1005 1006 1007 1008 ...
 $ weight   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ P        : num  NA NA NA NA NA NA NA NA NA NA ...
 $ Q        : num  NA NA NA NA NA NA NA NA NA NA ...
 $ Qfitted  : num  NA NA NA NA NA NA NA NA NA NA ...
 $ Qresidual: num  NA NA NA NA NA NA NA NA NA NA ...
```

where `timestep` is the serial numbering for the time series, `time` (which equals `timestep*dt`) is the time since the start of the time series, `weight` is the weight each discharge value had in the fitting procedure (see more about this below), `P` and `Q` are the measured precipitation and discharge, `Qfitted` is the fitted discharge, and `Qresidual` is the difference between `Q` and `Qfitted`.

One can overlay the measured and fitted discharge time series to get an idea of how they correspond. Figure xxf17, below, shows the entire time series and an expanded view from late 2004.

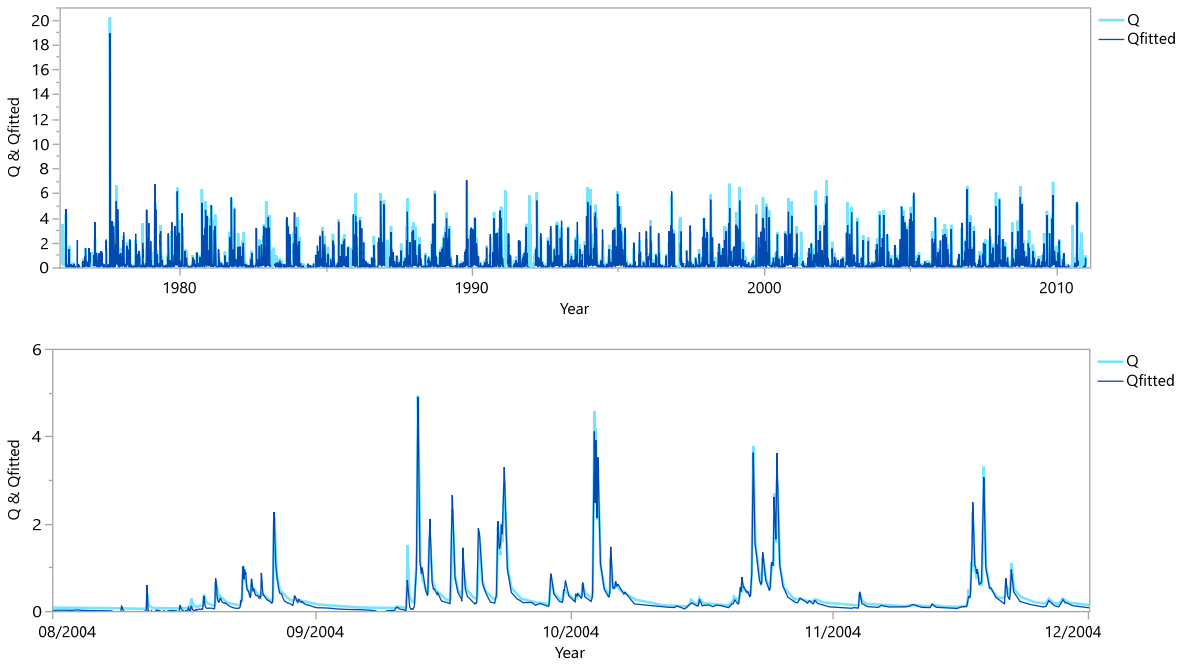

Figure xxf17. Observed (light blue) and fitted (dark blue) hourly discharge time series for the entire record (top panel) and four months in late 2004 (bottom panel).

From Figure xxf17, one can see that the measured and fitted discharge time series correspond reasonably well. We need to be careful to use the word *fitted* rather than *predicted* here, since these are not a priori predictions. Split-sample cross-validation tests are needed to determine whether this approach can

adequately predict streamflows that it has not already been fitted to. This can be facilitated by setting `Qfilter` to 1 for time steps that will be used for calibration, and setting `Qfilter` to 0 for time steps where the deconvolution model will not be fitted, but will instead be used to estimate "out-of-sample" values. For example, to fit the deconvolution model to the first 10 years of data and test it against the remaining 25 years of data, one can use the following function call:

```
zz <- ERRA(p=p, q=q, xknots=c(4, 20), xknot_type="even", m=1000, nk=40, h=3,
split_params=antQparams, Qfilter=ifelse((dat$yr<=1985), 1, 0))
```

If one wishes to estimate goodness-of-fit statistics for the results, they can be calculated as:

```
with(zz$Qcomp, {
  NSE_in_sample <-  1-var(Qresidual[(weight>0)],
na.rm=TRUE)/var(Q[(weight>0)&(!is.na(Qresidual))])
  NSE_out_of_sample <-  1-var(Qresidual[(weight==0)],
na.rm=TRUE)/var(Q[(weight==0)&(!is.na(Qresidual))])
})
```

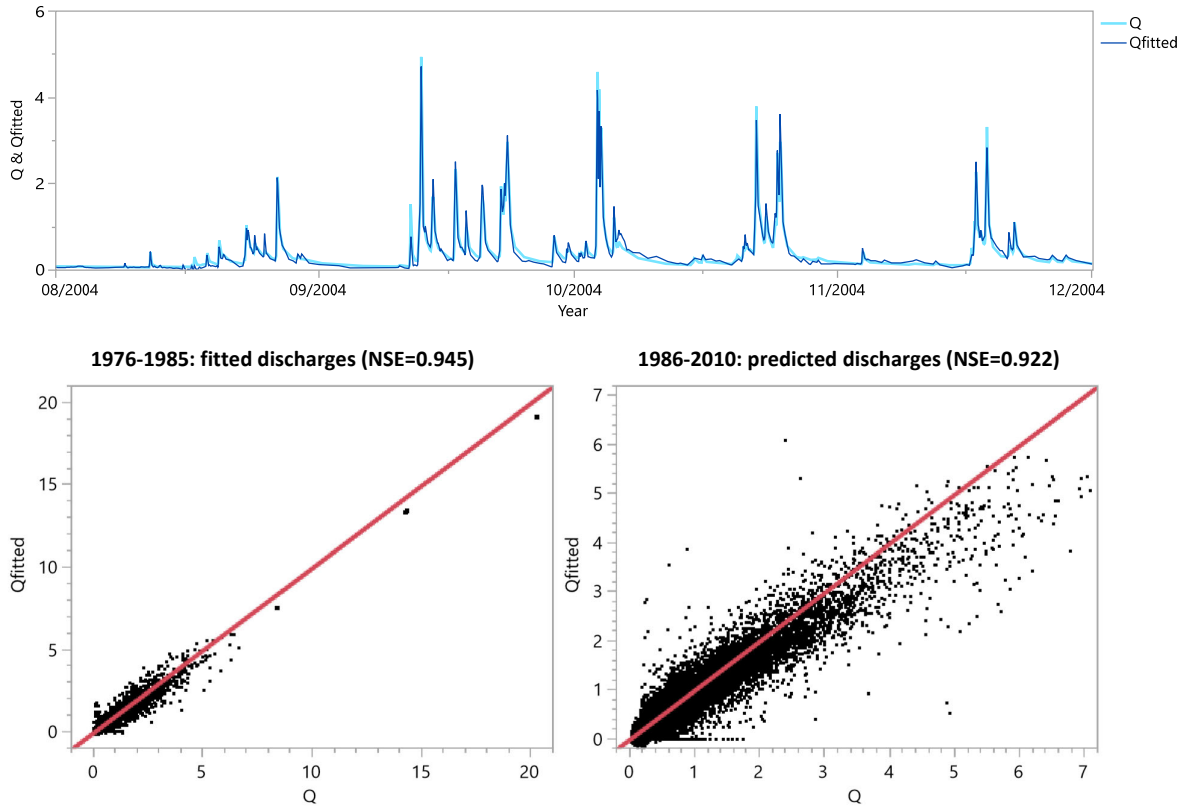

Figure xxf18. Split-sample cross-validation test for ERRA fits to 1976-1985 streamflows at Hafren. Top panel shows example time series of measured and predicted streamflows for four months in late 2004 (out-of-sample predictions). Bottom left panel shows measured and fitted discharges during 1976-1985 (calibration period); bottom right panel shows measured and predicted discharges during 1986-2010 (out-of-sample predictions). 1:1 lines are shown in red.

Figure xxf18, above, shows the results of this exercise. The time series in the top panel of Figure xxf18 is an out-of-sample prediction, but is nonetheless broadly similar to that shown in the bottom panel of Figure xxf17, which is an in-sample fit to the data. The bottom left panel shows that the fitted streamflows during 1976-1985 correspond closely to the measured values (including the flood of record), but this is not unexpected since the deconvolution parameters have been fitted to these data. The bottom right panel

shows that out-of-sample predicted values from 1986-2010 also correspond closely to the measured values (although the visual impression suggests greater scatter, just because the axis ranges are smaller).

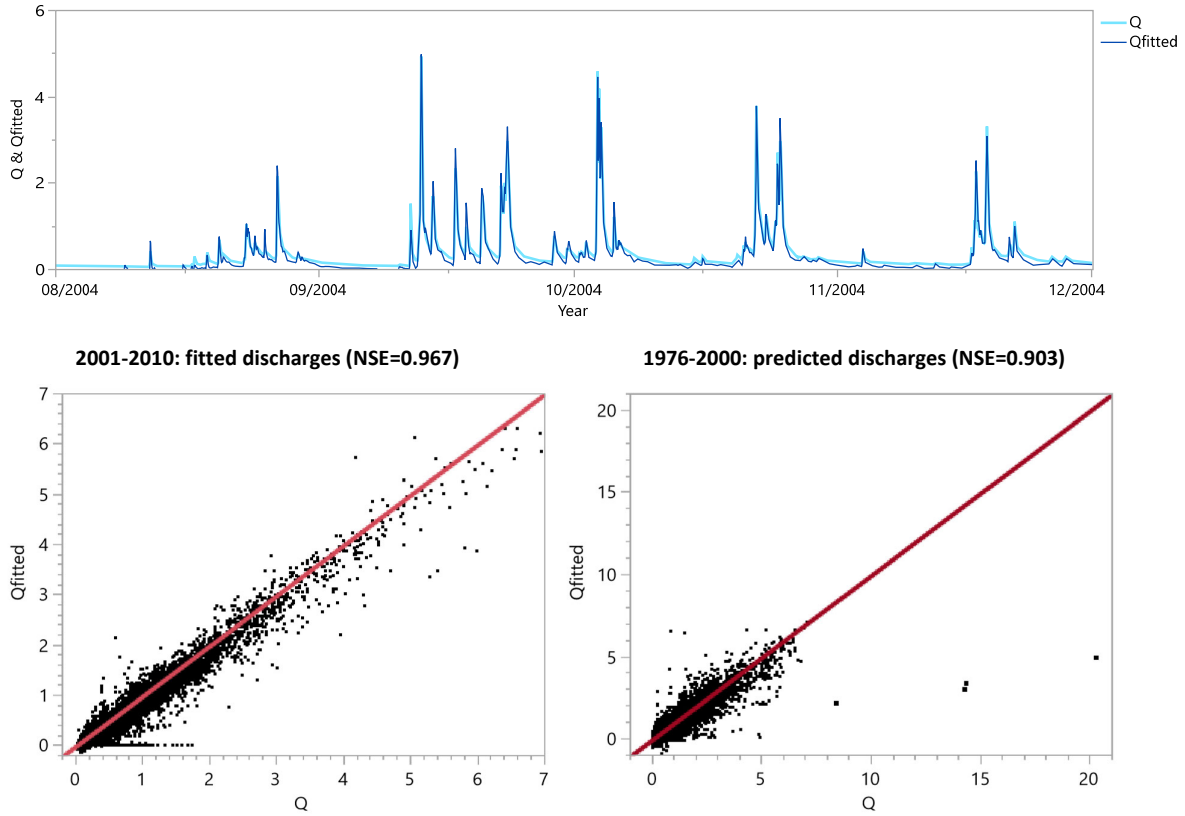

Figure xxf19. Split-sample cross-validation test for ERRA fits to 2001-2010 streamflows at Hafren. Top panel shows example time series of measured and predicted streamflows for four months in late 2004 (in-sample fits). Bottom left panel shows measured and fitted discharges during 2001-2010 (calibration period); bottom right panel shows measured and predicted discharges during 1976-2000 (out-of-sample predictions). 1:1 lines are shown in red. Out-of-sample predictions clearly underestimate the August 1977 flood of record.

If one instead fits the convolution model to the last 10 years of data, and tests it against the preceding 25 years of data, one obtains the results shown in Figure xxf19. Again the predicted time series reproduces most of the dynamics in the observed discharges in late 2004 (top panel), perhaps unsurprisingly because these are in-sample fitted values. The bottom left panel shows that the fitted streamflows during 2001-2010 correspond closely to the measured values, and the bottom right panel shows that out-of-sample predicted values from 1976-2000 are generally well reproduced. However the August 1977 flood of record is underestimated by roughly a factor of 4.

It should be noted that "predictions" in this simple demonstration case are not purely independent predictions, because this particular convolution model uses lagged discharges as an indicator of catchment wetness, in order to determine how sensitive streamflow is to precipitation. Nonetheless the lagged discharge information is not itself used as a baseline for estimating future discharge, as would be typical in one-step-ahead prediction models. For purely independent predictions, other measures like antecedent precipitation could be used instead as inputs.

It should also be kept in mind that ERRA's intended purpose is to quantify RRDs and NRFs as measures of the coupling between precipitation and streamflow. Convolution models like those estimated by ERRA can

be useful indicators of rainfall-runoff coupling, but not a complete model of everything that controls streamflow (which is what goodness-of-fit statistics and plots like Figures xxf17-xxf19 reflect).

13. Baseline adjustment with parameter fq

In some catchments, baseflows vary seasonally in response to variations in evapotranspiration. One example is the Nantahala River, previously introduced in Figure xxf09. As Figure xxf20 below illustrates, Nantahala baseflows are distinctly higher in winter than in summer.

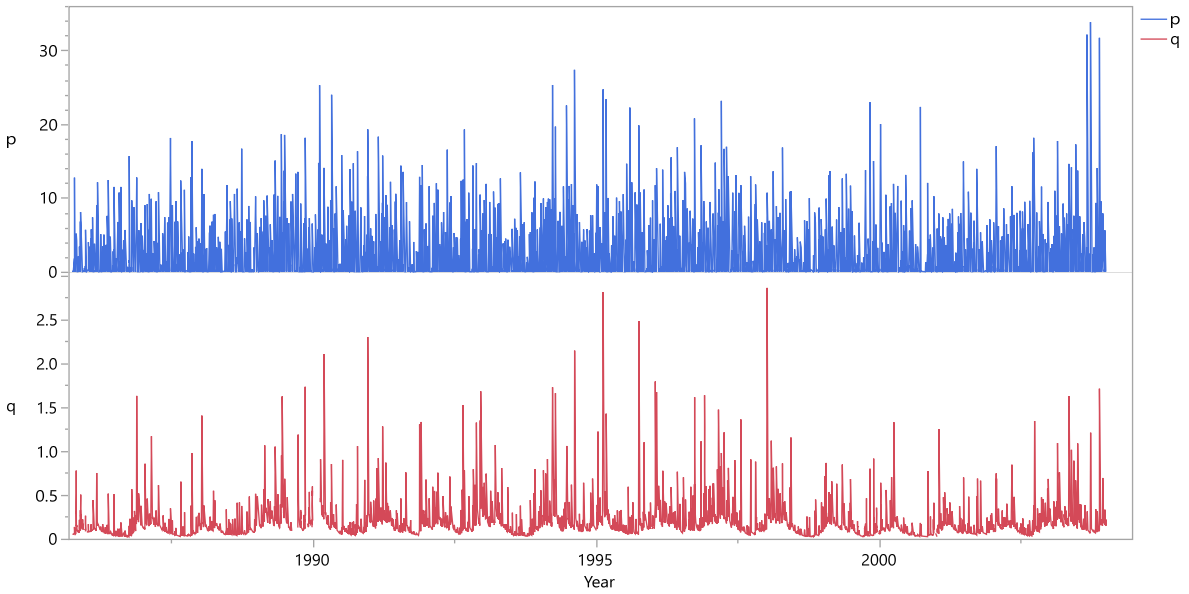

Figure xxf20. Hourly precipitation and streamflow from 1985 through 2003 at Nantahala River near Rainbow Springs, NC (134 km²). Seasonality in baseflow is evident in the lower panel.

This seasonal variation in baseflows can distort ERRA estimates of rainfall-runoff relationships. To illustrate this phenomenon, we first divide the precipitation record according to four ranges of antecedent wetness:

```
breakpts <- c(0.1, 0.2, 0.4)
antQparams <- list(crit = list(q) ,
                   crit_label = c("antQ") ,
                   crit_lag = 1 ,
                   breakpts = list(breakpts) ,
                   pct_breakpts = c(FALSE) ,
                   thresh = 0.0 ,
                   by_bin = c(FALSE)
) # end split_params
```

And then estimate the nonlinear, nonstationary coupling between precipitation and streamflow as follows:

```
zz <- ERRA(p=p, q=q, xknots=c(4, 20), xknot_type="even", m=200, h=3,
split_params=antQparams)
```

The weighted average RRDs thus obtained are plotted in the left panel of Figure xxf21 below. One can see that the RRDs do not converge at long lags, as one would expect, and the RRD for the lowest antecedent wetness remains persistently below zero by many times its standard error, which is clearly nonphysical.

What has happened here is that the seasonal variation in baseflow has confounded the relationship between antecedent wetness and runoff response.

The option `fq` is designed to minimize the effects of confounding by long-term baseflow variations that are unrelated to runoff response on the timescales measured by ERRA. The option `fq` (for "filter quantile") implements quantile baseline filtering. If `fq` is greater than its default value of zero (and less than its maximum value of 1), ERRA calculates the running `fq`'th quantile of the discharge time series, centered over a window four times as wide as the maximum lag `m`. ERRA then subtracts that running quantile filter from the discharge time series. For applications that have been tried so far, `fq` values on the order of 0.1 or so have proven to be effective.

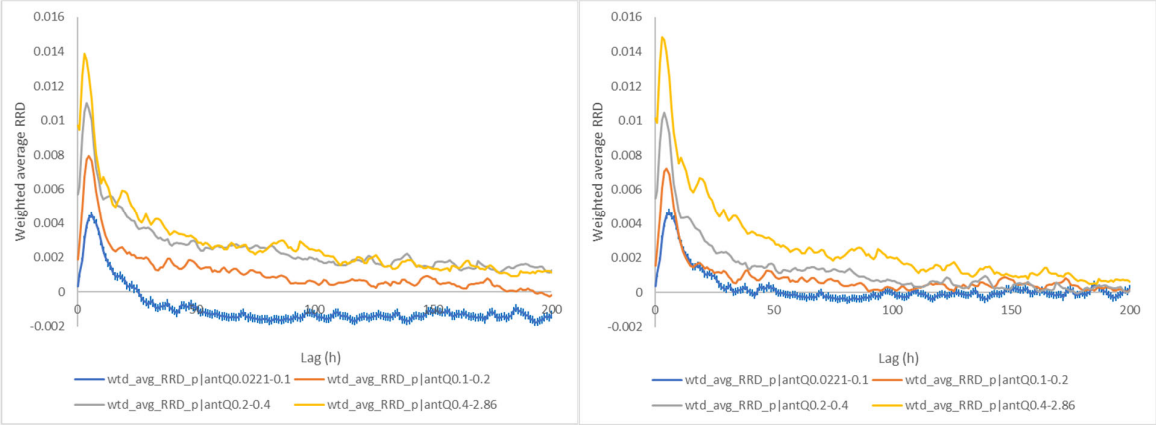

Figure xxf21. Weighted average RRDs for four different ranges of antecedent wetness (indicated by different colors), without (left panel) and with (right panel) baseline adjustment by setting fq=0.1. Error bars on blue curve (lowest antecedent wetness) are 1 standard error.

For the example shown here, we can invoke this running quantile filter as follows:

```
zz <- ERRA(p=p, q=q, xknots=c(4, 20), xknot_type="even", m=200, h=3, fq=0.1,
split_params=antQparams)
```

The results from this call to ERRA are shown in the right-hand panel of Figure xxf21. One can see that the negative bias at low antecedent wetness (the blue curve) is mostly eliminated, and the RRDs for the different antecedent wetness levels converge toward one another at long lags, as expected.

14. The ERRA function call

```
ERRA <- function(p ,
        q ,
        wt = rep(1, NROW(as.matrix(p))) ,
        m = 60 ,
        nk = 0 ,
        nu = 0.0 ,
        fq = 0 ,
        p_label = NULL ,
        split_params = NULL ,
        xknots = NULL ,
        xknot_type = NULL ,
        show_top_xknot = FALSE ,
        Qfilter = rep(1, NROW(as.matrix(p))) ,
        Qavgd = TRUE ,
        dt = 1 ,
        agg = 1 ,
        h = NULL ,
        ARprob = 0.05 ,
        ARlim = 0.2 ,
        max.AR = 8 ,
        complete.RRD = FALSE ,
        verbose = TRUE ,
        robust = FALSE ,
        max.chunk = 1e8 ,
        low.data.warn = 16 ,
        low.data.fail = 8 ,
        ser.corr.warn = 0.99)
```

15. Inputs to ERRA

p vector of precipitation rates (or a matrix or data frame, if there is more than one precipitation driver), evenly spaced in time
if p is a matrix or data frame, p is treated as ncol(p) simultaneous time series of inputs to the catchment
if p is supplied as a time series, it will be coerced to a vector or matrix, as appropriate

q vector of discharge (in same units, and over the same time intervals, as p)
if q is supplied as a time series, it will be coerced to a vector

wt optional vector of point weights

m maximum lag for RRD (default=60), in number of time steps (number of lag steps will be m+1 to include lag of zero)

nk number of knots in a piecewise linear broken stick representation of runoff response (RRD and NRF) as a function of lag. Must be an integer greater than 2 and less than or equal to m+1, or must be zero (the default).
If nk>2 and nk<=m+1, nk knots will be created at lags of 0, 1, and a geometric progression (or as close to geometric as it can be, given that the lags are integers) between lags 1 and m.
If nk<=2, nk>m+1, or nk==0 (the default), the broken-stick approach is not used, and instead the IRF is evaluated for m+1 lags between 0 and m.
Note that analogous broken-stick representations can also be used to estimate how runoff response changes as a function of precipitation intensity. The knots for that broken-stick model are called "xknots" (see below) to distinguish them from the knots defined here.

nu fractional weight between 0 and 1 for Tikhonov-Phillips regularization (0 is the default: no regularization)

fq filter quantile, for which values of 0<=fq<1 are valid; anything else is a fatal error. If fq>0, a running quantile filter is calculated and subtracted from q, to remove drift or seasonal patterns that could otherwise generate artifacts in the RRD. The running quantile filter is the fq'th quantile of a moving window of 4m+1 time steps, centered around the time in question. If fq==0.5, the fast running median algorithm is used; otherwise the somewhat slower moving quantile algorithm is used. If fq==0 (the default), no filtering is performed.

p_label an optional vector of string labels for the columns of the p matrix. For example, if p has two columns corresponding to rain and snow, and p_label = c("rain", "snow"), then outputs will also have those labels to identify responses to those two variables. If p_label is not NULL, it must have as many entries as there are columns in p. If p_label is NULL (the default) or if it has the wrong number of entries, then any column names of the p matrix will be used instead. If those column names do not exist, then values of "p1", "p2" etc. will be assigned (or simply "p", if p is a vector or single-column matrix).

split_params a list containing all the parameters needed for splitting by precipitation time steps. These are:

- **crit** a list of n_crit criterion variables, each of which is a vector of same length as p and q Note that precipitation itself should NOT be used as a splitting variable (although lagged values of precipitation could potentially be used). To estimate nonlinear responses to precipitation intensity, set xknots accordingly.
- **crit_label** a vector of n_crit strings that label the criterion variables
- **crit_lag** a vector of n_crit integers, indicating how many time steps each criterion should be lagged. Values <1 result in no lagging.
- **pct_breakpts** a boolean vector of length n_crit, indicating whether each criterion's breakpoints are given in the units of the criteria themselves (pct_breakpts==FALSE) or in percentiles (pct_breakpts==TRUE)
- **breakpts** a list of n_crit vectors of breakpoint values that will be used to divide the criterion variables into bins. Each vector must be in ascending order. For example, if the first variable is divided at values of 0.3 and 0.6, and the second variable is divided at percentiles of 70, 80, and 90 percent, then breakpts=list(c(0.3,0.6), c(70,80,90)) and pct_breakpts=c(FALSE, TRUE)
- **thresh** a vector of n_crit threshold values. Criterion values <= thresh are ignored in setting break points according to percentile ranks (that's if pct_breakpts==TRUE. thresh has no effect if pct_breakpts==FALSE). Thresholds can be helpful when a criterion variable has many trivially small values that should all go in the lowest bin, with the breakpoints being set within in the range of the criterion values that are not trivially small. Note that criterion values below thresh will still be used in splitting the data set, just not in setting the values of the breakpoints.
- **by_bin** a boolean vector of length n_crit, indicating whether each criterion's breakpoints are set separately within each bin of the previous criteria, or as one set of breakpoints for the whole data set. For example, if by_bin==TRUE, criterion3 is applied separately within each bin formed by criteria 1 and 2. That is, the bin limits for criterion3 are set according to the percentiles of criterion3 values among cases that conform to a given bin of criterion1 and 2 (rather than the percentiles of all cases, which are used if by_bin==FALSE). This is useful when criteria are interdependent, such that (for example) the smallest bin of criterion1 may have no values that also fit into the largest bin of criterion3. If pct_breakpts==FALSE for a criterion, by_bin has no effect on the binning of that criterion.

if split_params==NULL (the default), no splitting is performed

xknots a vector or matrix of knots for the piecewise linear approximation of the RRD's nonlinear dependence on p. xknots can be specified as fixed values or as percentiles of the p distribution (depending on the xknot_type flag -- see below). Values of p=0 are ignored when these percentiles are later converted to fixed values. If xknots is a matrix, it must have np*n_sets columns, and each column of xknots will be applied to the corresponding (split) columns of p. Unless xknot_type=="even", the xknots vector (or each column of the xknots matrix) must be sorted into strictly ascending order, with no duplicate values. If xknots is a vector or single-column matrix and p is a multi-column matrix, the same xknots will be applied to each column of p. xknots must be between (and not include) the minimum and maximum values of the corresponding column of p (including any splits). Each column of xknots must have the same

number of knots, although the xknot values themselves may differ.
If xknots == NULL, potential nonlinear responses to precipitation intensity are ignored and a single RRD is estimated for each column of p (and, potentially, each subset created by split_params).
If xknots != NULL, separate RRD's are estimated for each specified knot point (and also the maximum value) of each column of p (and, potentially, each subset created by split_params).
If xknots != NULL and xknot_type=="even", then xknots must consist of two (and only two) positive integers, specifying the number of xknots and the minimum number of data points between each pair of xknots (see xknot_type=="even", below).

xknot_type a flag indicating how the nonlinearity knots should be used. This can be abbreviated as the first letter of the string values below.

If xknot_type=="**values**", xknots are expressed as values of p. If xknots is a vector, the same xknot values will be applied across all precipitation columns (including any precipitation subsets).

If xknot_type=="**percentiles**", xknots are expressed as percentiles of the p distribution. If xknots is a vector, the same xknot percentiles (yielding different numerical values) will be applied to each precipitation column (including any precipitation subsets).

If xknot_type=="**cumsum**", xknots are expressed as percentiles of the cumulative sum of p (so if xknots=80, for example, the xknot will be the value of p for which all smaller p's sum to 80 percent of the sum of all p's). Thus these xknots will delimit fractions of the total input p. If xknots is a vector, the same cumulative sum percentiles (yielding different numerical values) will be applied to each precipitation column (including any precipitation subsets).

If xknot_type=="**sqsum**", sknots are calculated as percentiles of the cumulative sum of squares of p (so if xknots=20, for example, the xknot will be the value of p for which all smaller p's, squared and summed, add up to 20 percent of the sum of squared p's). Thus these xknots will delimit fractions of the total sum of squared inputs p^2. These will roughly approximate the corresponding fractions of the total leverage in the data set, if the distribution of p's is strongly skewed with a peak near zero and a long right tail. If xknots is a vector, the same percentiles of the cumulative sum of squares (yielding different numerical values) will be applied to each precipitation column (including any precipitation subsets).

If xknot_type=="**even**", then knots are spaced as evenly as possible across the range of p. If xknot_type=="even", then xknots must be a vector of length 2, with both values being positive integers (anything else is a fatal error). The first integer indicates the number of xknots to be used (not including the uppermost xknot at max(p) or the lowermost xknot at p=0). The second integer indicates the minimum number of valid points in each interval between xknots (which may require a somewhat uneven spacing of xknots). This second integer should be set large enough that the slope of the response (beta_prime) can be adequately constrained in each interval between knots, but kept small enough that the range of p spanned by any interval is not too wide (particularly at the upper tail of the distribution, where a wide range of p is spanned by relatively few data points. If xknots is a vector, the same parameters will be used to construct separate sets of xknot values for each precipitation column (including any precipitation subsets).

show_top_xknot boolean flag for whether the highest xknot (at the highest precipitation value in the input data) should be reported as output or not. Default is FALSE, because values at the highest

xknot tend to be unreliable, since they are controlled by correlations between discharge and relatively few, highly skewed precipitation values

xknots should be chosen so that (these are not checked):
(a) there are enough points in each interval between xknots, and sufficient variability in their values, to define the dependence of q on p,
(b) intervals between individual pairs of xknots do not span major changes in slope in the nonlinear relationship between q and p

Qfilter optional boolean vector (1 and 0, T and F, or TRUE and FALSE) indicating whether individual discharge time steps should be included in the analysis. This vector allows one to analyze subsets filtered according to discharge time. To analyze multiple subsets, change Qfilter and re-run this function.

Qavgd set this flag TRUE if q represents the average discharge over each sampling interval. Set it FALSE if q represents an instantaneous value at the end of each sampling interval. This affects the lag time that corresponds to each lag interval. It also affects the RRD and NRF values at lag zero (same-time-step response).

dt optional time step length, in whatever time units the user wants. If step length is not supplied, then RRD returns the runoff response distribution in fractions per time step rather than per unit time (i.e., the default value of dt is 1).

agg integer aggregation factor for time steps. If agg>1, p, and q, and splitting criteria will be combined by averaging in sets of agg time steps (e.g., agg=24 converts hourly data to daily averages). Non-integer values of agg will be rounded up. Values of agg<1 will be converted to 1. Default is agg=1 (no aggregation).

h integer order of autoregressive correction (non-negative integer). Higher-order AR corrections can, by the duality principle, be used to correct for moving average (MA) noise as well. If h==0, no correction is applied. If h==NULL (the default), the order of autoregressive correction will be determined iteratively, as described in nonlinIRF script (see K2022 for details).

ARprob significance threshold in automated AR order selection (see IRF script and K2022 for details). Default is 0.05.

ARlim threshold value of residual correlation coefficients in automated AR order selection (see IRF script and K2022 for details). Default is 0.2.

max.AR maximum order of AR correction that will be accepted in automatic AR order selection (see IRF script and K2022 for details). Default is 8.

complete.RRD flag for whether the number of lagtimes will be assumed to be sufficient to hold the complete RRD (TRUE), meaning that any RRD coefficients at longer lagtimes are trivially small, or whether this cannot be assumed (FALSE, the default). Complete.RRD=TRUE will yield smaller, and more accurately estimated, standard errors if the real-world RRD actually does converge to zero before the maximum lag is reached. But if this is not the case, complete.RRD=TRUE will force the

RRD to artifactually converge toward zero at the longest lag (with artificially small standard errors). Complete.RRD=TRUE should thus be invoked with caution.

verbose controls whether intermediate progress reports are printed (TRUE) or suppressed (FALSE). Default is TRUE

robust flag controlling whether robust estimation by Iteratively Reweighted Least Squares (IRLS) will be used. A special tweak is used to prevent the solution from collapsing if more than half of the values of the response variable (i.e., discharge) are the same (for example, in ephemeral streams with frequent discharges of zero). Instead of scaling the robustness weighting function by the median absolute residual, we scale it by the (0.5 + 0.5*(# identical y's)/(# total y's)) quantile of the absolute residuals, where "# identical y's" means the largest number of y's with the same identical value. This defaults to the median when we have no repeated y's, but also gives reasonable robust estimates when we have large numbers of y's with the same value. Default is FALSE

max.chunk maximum size, in bytes, of the largest piece of the design matrix (the matrix of p and its lags) that will be created at one time. Design matrices that would be larger than max.chunk will be created and processed in separate "chunks" to avoid triggering memory paging, which could substantially increase runtime. Keeping max.chunk relatively small (order 1e8 or 1e9) incurs only a small performance penalty, except in the case of robust estimation, where the need to iterate means that the solution will be faster (by about a factor of 2 or so) *if* one can keep the whole design matrix in one chunk without triggering memory paging. Default is 1e8.

low.data.fail failure threshold for minimum count of nonzero p values in any precipitation subset at any lag. A hard stop is triggered if min(n.nz) (see below for explanation of n.nz) is smaller than this. Default is 8.

low.data.warn warning threshold for minimum count of nonzero p values in any precipitation subset at any lag. If min(n.nz) (see below for explanation of n.nz) is smaller than this, a warning is issued but results will be returned. Default is 16.

ser.corr.warn warning threshold for lag-1 serial correlation of residuals. When this is exceeded, a warning is issued suggesting that time steps should be aggregated. Note that warning will only be issued if h is 0 or NULL; for other values of h this test would not have the same relevance. Warning is also suppressed when using broken-stick linear interpolation over lags (i.e., if nk>2). Default is 0.99.

q, p, and wt can contain missing values. Missing values of q create 1+h missing rows in the regression matrix. Missing values of p create m+1 missing rows (one for each lag) in the regression matrix.

Here is an example of a set of splitting criteria for split_params:

split_crit <- list(this instruction says that what's coming is a list

crit = list(T, q) , here we designate two variables we will use as splitting criteria: temperature and lagged discharge. That means, we divide precipitation time steps into groups according to temperature and then according to lagged discharge (see crit_lag) as a proxy measure of catchment wetness. Both variables must be present in the calling environment, and both must be of the same length as q.

crit_label = c("T", "lagQ") , these are labels so that we can later tell which group is which

crit_lag = c(0, 1) , this says we will lag the second criterion variable (q) by one time step, and won't lag the first criterion variable (T)

pct_breakpts = c(FALSE, TRUE) , this says that the breakpoints that will be used to divide T and lag(q) into different groups are expressed in raw values of T, and percentiles of the lag(q) distribution

breakpts = list(c(0, 2, 4, 6), c(80, 90, 95)) , the first vector says we will divide temperature into five groups: <0 degrees, 2-4 degrees, 4-6 degrees, and above 6 degrees, and we will divide lag(q) into four groups: the lowest 80 percent, the 80-90th percentiles, the 90-95th percentiles, and above the 95th percentile. Thus these breakpoints will construct a total of 20 different sets of precipitation time steps (the four lagged discharge ranges, nested within each of the five temperature ranges).

thresh = c(-999, 0) , this says that *if* we calculate breakpoints from percentiles, we evaluate those percentiles among values above the threshold. Thus, for example, the percentiles of lag(Q) will be determined using values of lag(Q) that are above zero. Another example could be if a splitting criterion is lagged precipitation, where a substantial fraction of values may be zero (and this fraction will change as the resolution of the time steps changes). Note that criterion values at or below the threshold will still be used in splitting the data set, just not in setting the values of the breakpoints. The thresh parameter will have no effect on the temperature breakpoints because they are not calculated from percentiles.

by_bin = c(TRUE, TRUE) , this vector specifies whether the criterion breakpoint percentiles are nested within the bin(s) of the previous criteria. Thus, for example, the second TRUE means that the 80th percentile breakpoint for lag(Q) is calculated separately for each of the temperature bins (the 80th percentile of precipitation that occurs below 0 degrees C, the 80th percentile that occurs between 0 and 2 degrees C, and so on). These may correspond to different absolute values of lag(Q), but they also guarantee that there will be *something* in each set of bins(!). If the second element were FALSE instead, then the lag(Q) breakpoints would be the same across all of the temperature bins. The by_bin parameter has no effect if pct_breakpts is FALSE (then the same absolute value of the breakpoint applies across all "parent" bins), and the first element of by_bin never has any effect (because the first criterion isn't nested within anything) but it must be specified anyhow, so that the by_bin vector is the correct length.

) This closing parenthesis marks the end of the list

ERRA optionally divides each precipitation time series into n_sets=(length(breakpts[[1]])+1) * (length(breakpts[[2]])+1) * ... interleaved/overlapping subsets by dividing the criteria variables split_params$crit at the stated breakpoint values (if pct_breakpts==FALSE) or at the stated percentile ranks (if pct_breakpts==TRUE). It then creates runoff response distributions for each of these subsets. Note that the subsets must be analyzed jointly, each with its own RRD coefficients, because the lagged precipitation values from each subset must be interleaved (since filtering by precipitation time produces diagonal stripes in the design matrix for each subset, and thus discharge can depend on the overlapping effects of precipitation from different subsets at different lags).

If xknots is not NULL, nonlinearities in the discharge response to precipitation are analyzed by calculating the runoff response distributions for subsets of each precipitation time series between each successive pair of knot values precipitation (xknots), by calling the nonlinIRF routine. In this case, the number of columns is *multiplied* by the number of xknots (or number of rows in the xknots matrix) plus one.

This procedure can fail or give nonsense results if the breakpoints and/or xknots are set in incompatible ways. There must be sufficient cases remaining in each bin delimited by the breakpoints that the regression routine still has enough to work with in each column of the matrix. The routine reports the minimum number of non-zero values of p in any column (min(n.nz)), which gives a clue about whether the breakpoints and xknots are leaving unreasonably few values to work with.

Multiple precipitation time series can be analyzed as inputs to the same discharge, although if they are too strongly correlated, it may not be possible to distinguish their effects from one another. Multiple precipitation time series are supplied by specifying p as a matrix of np columns, rather than a vector or single-column matrix. If any element of the splitting parameter "crit" is a matrix, it must be np columns wide, and each column will be used to split the corresponding column of the precipitation matrix. Any element of crit that is a vector or a one-column matrix will be applied to all columns of the precipitation matrix jointly.

The number of input precipitation series is np=NCOL(p). The number of subsets these series are divided into is n_sets. The number of knot points will be nxk=NROW(knots)+1 unless xknot_type="even", in which case the number of knot points will be nxk=xknots[1]+1. Each subset of each precipitation series will have m+1 columns representing lag times 0..m. The AR method (for autoregressive noise correction) will also add h more lagtimes, plus h more columns for lagged discharge (where h is the order of the AR correction). Thus the total number of columns will be np*n_sets*nxk*(1+m+h) + h, plus one column for the constant term.

p and q can have missing values (NA), but the time steps must be continuous and evenly spaced (and must of course be the same for both p and q).
Missing values of q create a missing row in the regression matrix.
Missing values of p create multiple missing rows in the design matrix (diagonals corresponding to each lag).
Each criterion (each element of crit) must be same length as vectors of p and q. Each can have missing values. Any missing criterion values cause multiple rows in the design matrix to be missing (diagonals corresponding to each lag).

16. Outputs from ERRA (linear analyses)

ERRA returns a list as output, with the following objects for linear analyses (i.e., if xknots==NULL), where **np** is the number of input precipitation time series, and **n_sets** is the number of different sets that these are split into, via split_params:

RRD data table of runoff response distributions evaluated for each subset of each precipitation variable (matrix with np*n_sets and m+1 rows, corresponding to lags 0 through m), and their standard errors. The RRD table will have one RRD column and one se (standard error) column for each P subset (if P is subsetted for nonstationarity analyses)

criteria a data table documenting the splitting criteria, including:

lwr lower limit of each criterion in each bin (matrix with np*n_sets rows and n_crit columns)

upr upper limit of each criterion in each bin (matrix with np*n_sets rows and n_crit columns)

binmean average of each criterion in each bin (matrix with np*n_sets rows and n_crit columns)

peakstats a data table containing the splitting criteria and the following statistics for each RRD:

- **set_label** vector of character string labels for each subset
- **peakht** peak of RRD by quadratic fitting to uppermost 20% of RRD
- **peakht_se** standard error
- **tpeak** time of peak by quadratic fitting to uppermost 20% of RRD
- **tpeak_se** standard error
- **width** width of peak, as full width at half maximum
- **width_se** standard error
- **rc** runoff coefficient (integral of RRD, approximating cumulative discharge per unit precipitation, over the analyzed range of lag times)
- **rc_se** standard error

17. Outputs from ERRA (nonlinear analyses)

ERRA returns a list as output, with the following objects for nonlinear analyses (i.e., if xknots != NULL), where **np** is the number of input precipitation time series, **n_sets** is the number of different sets that these are split into, via split_params, and **nxk** is the number of different xknots, which will equal the number of specified knot points plus one for the maximum p value. Note that the number of xknots and broken-stick segments may be nxk or nxk-1, depending on whether the option show_top_xknot is TRUE or FALSE.

NRF data table of Nonlinear Response Functions that express the effect of p on q, averaged over each broken-stick segment between adjacent xknots, and their standard errors. Matrix of np*n_sets*nxk (or nxk-1) columns and m+1 rows, corresponding to lags 0 through m). In contrast to the RRD, the NRF expresses the incremental effect of an additional time step of precipitation at a given rate, rather than the effect of an additional unit of volume of precipitation. So, for example, the NRF for a precipitation intensity of p=20 mm/hr expresses how much an additional rainfall input at 20 mm/hr (over the length of time dt) will raise

discharge. By contrast, the RRD would express this same increase in discharge, per unit of precipitation (rather than for the 20mm/hr all together).

The NRF table will have one NRF column and one se (standard error) column for each P subset and each broken-stick segment.

For nonlinear analyses, the NRF is used because it shows the shape of the nonlinearity more intuitively than the RRD. For example, in a linear system, the NRF will increase linearly with p, whereas RRD would be unchanged across the range of p (within uncertainty). And for a quadratically nonlinear system, the NRF will be a parabolic function of p, whereas RRD would increase linearly with p.

knot_NRF data table of NRF values evaluated at each nonlinearity knot (i.e., each xknot value). Will have one NRF column and one se column for each p subset and each xknot.

NOTE: knot_NRF's will typically be noisier than regular NRF's (averaged over each broken-stick segment). Thus for most purposes, NRF's will be preferable to knot_NRF's.

wtd_avg_RRD data table of weighted average RRDs, weighted by input p (matrix of np*n_sets columns and m+1 rows) and their standard errors. Will have one RRD column and one se column for each p subset.

peakstats a data table containing the splitting criteria and the following statistics for each NRF:

- **setwm_label** vector of character string labels for each subset
- **wtd_meanp** weighted mean precipitation within broken-stick segment
- **pvol** precipitation volume per time step
- **tpeak** time of peak by quadratic fitting to uppermost 20% of NRF
- **tpeak_se** standard error
- **NRF_peakht** peak height of NRF function
- **NRF_peakht_se** standard error
- **width** width of peak, as full width at half maximum
- **width_se** standard error
- **rc** runoff coefficient (integral of NRF divided by pvol, approximating cumulative discharge over the analyzed range of lag times that is attributable to each unit of precipitation input at precipitation intensities corresponding to each knot)
- **rc_se** standard error
- **pvol** precipitation volume per time step
- **rsum** runoff volume (integral of NRF, approximating cumulative discharge over the analyzed range of lag times that is attributable to one time unit of precipitation input at precipitation intensity of wtd_meanp). If P and Q are in mm/h, for example, rsum is also in mm/h, but note that this is mm of *Q* per hour of *P* (at the stated intensity).
- **rsum_se** standard error
- **n.nz** number of nonzero precipitation time steps in this broken-stick segment

knot_peakstats a data table containing the same elements as peakstats, but evaluated at each xknot instead of over each broken-stick segment

avgRRD_peakstats a data table containing the peakstats of the weighted average RRD

criteria a data table documenting the splitting criteria, including:
- **setwm_label** vector of character string labels for each subset
- **wtd_meanp** weighted mean precipitation within broken-stick segment

18. Outputs from ERRA (both linear and nonlinear analyses)

options a string containing values of frequently used ERRA options, to simplify and standardize file naming (options that are at their default values are not shown). The options string includes:
- **splitby** introduces labels for any splitting criteria that were used (see peakstats table, or column headings in NRF, RRD, or wtd_avg_RRD, for breakpoints between splitting bins)
- **m=** maximum lag
- **nk=** number of knots in broken-stick interpolation (if used)
- **agg=** aggregation (number of time steps)
- **h=** order of ARMA correction
- **nu=** regularization parameter (if used)
- **fq=** filter quantile (if quantile baseline correction is used)
- **nlin** indicates that nonlinear precipitation intensity effects were measured (see NRF column headings or peakstats table for broken-stick breakpoints)
- **robust** indicates that robust estimation was performed by IRLS

lagtime vector of lag times

Kbb covariance matrices for coefficients

sets matrix of length n and width np with integer values (1..n_sets*nxk) indicating which set each time step belongs to, for each precipitation source

set_label set labels including preciptation IDs

n length of original p and q series

n.eff effective sample size, accounting for uneven weighting

n.nz vector of nonzero values (with nonzero weight) in each column of p matrix

h order of AR correction that was applied

phi AR coefficients

resid residuals

resid.acf autocorrelation function of residuals

resid.pacf partial autocorrelation function of residuals

Qcomp data table comparing observed and fitted values of Q (fitted values are not shown at time steps for which predictors including lagged P are missing). Columns include:

- **timestep** sequence number of time steps
- **time** time since start of time series (in time units): timestep*dt
- **weight** weight of each time step in fitting procedure. Where Qfilter==FALSE, weight will be zero and time step will be excluded from fitting, thus enabling a test of out-of-sample fitting skill
- **P** vector or matrix of precipitation inputs
- **Q** vector of measured discharge
- **Qfitted** vector of fitted discharges
- **Qresidual** vector of residuals: fitted-measured discharges