

# Chronikis User Manual\*

Kevin S. Van Horn  
Adobe Inc.

July 19, 2019

Chronikis (kroh-NEE-kees) is a special-purpose language for creating time-series models. It comes with a compiler **chronikisc**, and an R package **chronikis** that contains utilities for compiling Chronikis programs as well as estimating and forecasting with the compiled time-series models.

The name “Chronikis” is derived from the phrase χρονική σειρά (*chronikí seirá*), which means “time series” in Greek.

*This initial release is still missing a number of functions and distributions that the language ought to have; the focus was on implementing enough that all of the models in the **compiler/Acceptance** subdirectory could be compiled.*

## 1 Installation

1. Install needed prerequisites.
  - (a) Install R if necessary.
  - (b) Install the R package rstan.
  - (c) Install The Haskell Tool Stack.
2. Clone or download this git repository and `cd` to the root.
3. `cd compiler`
4. `stack install`

You may run into issues with the Haskell package `hmatrix-gsl`. This requires the GNU Scientific Library and the `pkg-config` utility. On Mac OSX with Homebrew you can obtain these by issuing the commands

```
brew install gsl
brew install pkg-config
```
5. Fire up R and install the package **chronikis** using these commands:

```
setwd(path_to_cloned_git_repository)
install.packages('chronikis_0.2.0.tar.gz', repos=NULL)
```

---

\*©2019, Adobe Inc. This document is licensed to you under the Apache License, Version 2.0. You may obtain a copy of the license at <http://www.apache.org/licenses/LICENSE-2.0>.

6. Add `~/local/bin` to your bash PATH variable.
7. Add `~/local/bin` to your R PATH variable. (This is where `chronikisc` got installed.) To do this, open your `~/Renvirom` file (create it if it doesn't exist) and add the line  
`PATH=${PATH}:~/local/bin`  
 If there is already a line setting PATH, add `~/local/bin` to the end.

## 2 Using Chronikis with R

In the following, suppose that

- you have written a Chronikis program that begins with the line  

```
def main(q_scale, h_scale: real{0.0, }) =
```

 and saved it to `"my_model.cks"`;
- `ytrain` is the time series on which you will train the model, and for which you will forecast new values; and
- you want to use values of 50 and 10 (resp.) for model parameters `q_scale` and `h_scale`.

Here is what the process of compiling and running the model then looks like:

```
library(rstan)
library(chronikis)
sm <- cksCompile("my_model.cks", "createSSMs")
source("my_model.R") # file was created by cksCompile

# Model training
margs <- mdlArgs(q_scale=50, h_scale=10)
sma <- setArgs(sm, margs)
fit <- hmc_estimate(ytrain, sma)
# fit <- vb_estimate(ytrain, sma)
# fit <- map_estimate(ytrain, sma)
post <- posterior_sample(fit)
npost <- 100
# It gets slow if you use more than 100 posterior draws
models0 <- createSSMs(margs, post, npost)
filtered0 <- filter_models(ytrain, models0)

# Model checking
lltrain <- averagedLL(models0, ytrain)
# Compare yrep[,i], 1 <= i <= npost, to ytrain;
# are they visually similar?
# Can also compute various summary statistics T()
```

```

# and check that T(ytrain) is within range of values
# T(yrep[,i]).
yrep <- forecast_sample(models0, length(ytrain), 1)
# Check that smoothed time series looks reasonable.
s <- smoothed_ts(ytrain, filtered0)
ysmooth <- average_normals(s$means, s$stddevs, 0.1, TRUE)
# Get one-step predictive residuals.
res <- lapply(filtered0, residuals, type='raw', sd=FALSE)

# Forecasting
models <- update_models(filtered = filtered0)
nsteps <- 20
alpha <- 0.10
# 90% predictive intervals
fc <- forecast_intervals(models, nsteps, alpha, TRUE)
# fc$mean[t] is the predictive mean t steps in the future
# (fc$lower[t], fc$upper[t]) is the 90% predictive interval
# t steps in the future.
ndpm <- 200 # Number of predictive draws per model
ymc <- forecast_sample(models, nsteps, ndpm)
# ymc is a nsteps x (npost * ndpm) matrix of predictive draws
# that can be used for Monte Carlo analysis
minmc <- matrixStats::colMins(ymc[11:20, ])
# minmc is a posterior predictive sample of
# min(y[n+11], ..., y[n+20]), where n = length(ytrain)
pred <- quantile(minmc, c(0.05, 0.95))
# pred is 90% predictive interval for
# min(y[n+11], ..., y[n+20])

```

Use `help(package=chronikis)` to see documentation on all of the functions provided in the R package. The directory `compiler/Acceptance` contains examples of Chronikis programs.

### 3 Running the Compiler on its Own

The compiler translates a Chronikis program into a Stan program and some R code to facilitate using the results of model estimation. The general syntax for calling the compiler is

```
chronikisc < cksfname --stan stanfname --R rfname
```

where

- *cksfname* is the path to the file containing the Chronikis program; we suggest using a `.cks` extension.

- *stanfname* is the destination path for the Stan output; this should have a *.stan* extension.
- *rfname* is the destination path for the R output; this should have a *.R* extension.

If you want to see some examples of the output produced, the directory `compiler/Acceptance/Reference` contains the results of running `chronikisc` on the *.cks* files in `compiler/Acceptance`.

## 4 Language Definition

### 4.1 Overview

A Chronikis program defines a parameterized distribution over time series (infinite sequences of real numeric values.) The general form is

```
def main ( knownParameters ) = TSDExpression
```

where *knownParameters* is a comma-separated list of variables with their types, and *TSDExpression* is an expression that denotes a probability distribution over time series.

Here are some examples for *knownParameters*:

- `N: int{1,}, mu: real[N], sigma: real{0.0,}`  
This declares *N* to be a positive integer, *mu* to be a real-valued vector of length *N*, and *sigma* to be a nonnegative real value.
- `rho: real{0.0, 1.0}, sigma_a, sigma_h: real{0.0,}[3]`  
This declares *rho* to be a real number between 0 and 1, and *sigma\_h* and *sigma\_a* to be length-3 vectors of nonnegative real values.

*N*, *mu*, etc. are called *known* parameters because they are already known when the model is created, rather than being inferred from training data. The range constraints (e.g. `{0.0,}`, `{0.0,1.0}`) on known parameters are checked when a model is trained.

A *TSDExpression* can have one of three forms:

- `variable = defExpr ; TSDExpression1`  
This defines the *variable* to have the value *defExpr* within *TSDExpression*<sub>1</sub>.
- `variable ~ distrExpression ; TSDExpression1`  
This defines the *variable* as a latent (unobserved) variable having the probability distribution *distrExpression*, with the time-series distribution *TSDExpression*<sub>1</sub> being conditional on the value of the *variable*. Typically *variable* will be a *fitted* parameter, i.e. a parameter to be inferred from training data, with *distrExpr* being the prior distribution for *variable*.
- A function call (including use of the binary operator `+`) that returns a distribution over time series. Some examples:

- `wn(sigma)` is a white noise process with mean 0 and variance `sigma` squared. That is, it corresponds to independent normal distributions for each time step.
- `qp(7.0, ell, 6, 0.0, sigma_p) + constp(mu0, sigma0)` is a distribution over periodic patterns of period 7, with the distribution having smoothness parameter `ell`, and scale parameter `sigma_p`, and the repeating pattern having an unknown mean that is given a `normal(mu0,sigma0)` prior.

The above is a recursive definition, so in practice a Chronikis program will have one or more variable definitions and/or variable draws, followed by a function call that returns a time-series distribution.

## 4.2 Examples

(To be written. For now, look in the directory `compiler/Acceptance`.)

## 4.3 Types

Chronikis programs are strongly and statically typed, but the only place you provide types are in the parameter list of `main`. The compiler infers all the remaining types.

A type has four attributes:

- its *element type*, which is `int` or `real`;
- its *shape*, which is a list of nonnegative integers;
- whether or not it is a *time series*, indicated with `$`; and
- whether or not it is a *distribution*, indicated with `~`.

The attributes apply in the order given above; thus,

- “distribution over time series of real vectors of length `n`” is a valid type (`real[n]$~`), but
- “time series of distributions over real vectors of length `n`” (`real[n]~$`) is *not* valid, and
- “length-`n` vector of distributions over univariate time series” (`real$~[n]`) is *not* valid.

Some examples:

- `real` is the type of real scalars.
- `real[n]` is the type of vectors of length `n`.
- `real[m,n]` is the type of `m`-by-`n` matrices.

- `real~` is the type of distributions over `real` values; an example is `normal(mu, sigma)`.
- `real[k,p]$` is the type of `k`-by-`p` matrix-valued time series.
- `real$~` is the type of distributions over univariate time series.

Currently, variables and parameters cannot have types that are time series or distributions, but we can construct expressions having such types.

## 4.4 Data Functions

In the following, a *signature* gives argument types in parentheses, and the corresponding return type after a colon. Italicized variables such as *m* or *n* indicate that there is one such signature for every combination of nonnegative values of the variables.

- `(+)` (binary operator). Signatures:
  - `(int, int): int`.
  - `(real, real): real`.
  - `(real, real[n]): real[n]`.  
Add a scalar to each element of a vector.
  - `(real[n], real): real[n]`.  
Add a scalar to each element of a vector.
  - `(real, real[m, n]): real[m, n]`.  
Add a scalar to each element of a matrix.
  - `(real[m, n], real): real[m, n]`.  
Add a scalar to each element of a matrix.
- `(-)` (unary operator). Negation. Signatures:
  - `(int): int`.
  - `(real): real`.
  - `(real[n]): real[n]`.  
Negate each element of a vector.
  - `(real[m, n]): real[m, n]`.  
Negate each element of a matrix.
- `(-)` (binary operator). Signatures:
  - `(int, int): int`.
  - `(real, real): real`.
  - `(real, real[n]): real[n]`.  
Subtract each element of a vector from a scalar.

- $(\text{real}[n], \text{real}): \text{real}[n]$ .  
Subtract a scalar from each element of a vector.
- $(\text{real}, \text{real}[m, n]): \text{real}[m, n]$ .  
Subtract each element of a matrix from a scalar.
- $(\text{real}[m, n], \text{real}): \text{real}[m, n]$ .  
Subtract a scalar from each element of a matrix.
- $(*)$  (binary operator). Signatures:
  - $(\text{int}, \text{int}): \text{int}$ .
  - $(\text{real}, \text{real}): \text{real}$ .
  - $(\text{real}, \text{real}[n]): \text{real}[n]$ .  
Multiply each element of a vector by a scalar.
  - $(\text{real}[n], \text{real}): \text{real}[n]$ .  
Multiply each element of a vector by a scalar.
  - $(\text{real}, \text{real}[m, n]): \text{real}[m, n]$ .  
Multiply each element of a matrix by a scalar.
  - $(\text{real}[m, n], \text{real}): \text{real}[m, n]$ .  
Multiply each element of a matrix by a scalar.
  - $(\text{real}, \text{real}[m, n, p]): \text{real}[m, n, p]$ .  
Multiply each element of a 3D array by a scalar.
  - $(\text{real}[m, n, p], \text{real}): \text{real}[m, n, p]$ .  
Multiply each element of a 3D array by a scalar.
- $(/)$  (binary operator). Signatures:
  - $(\text{real}, \text{real}): \text{real}$ .
  - $(\text{real}, \text{real}[n]): \text{real}[n]$ .  
Divide a scalar by each element of a vector.
  - $(\text{real}[n], \text{real}): \text{real}[n]$ .  
Divide each element of a vector by a scalar.
  - $(\text{real}, \text{real}[m, n]): \text{real}[m, n]$ .  
Divide a scalar by each element of a matrix.
  - $(\text{real}[m, n], \text{real}): \text{real}[m, n]$ .  
Divide each element of a matrix by a scalar.
- $(^)$  (binary operator). Exponentiation. Signatures:
  - $(\text{real}, \text{int}): \text{real}$ .
- $([])$  Array indexing, 1-based. Signatures:
  - $(\text{real}[n], \text{int}): \text{real}$ .  
 $a[i]$  is element  $i$  of vector  $a$ .

- `(real[m,n], int): real[n]`.  
`a[i]` is row  $i$  of matrix  $a$  (as a vector).
- `(real[m,n], int, int): real`.  
`a[i,j]` is row  $i$ , column  $j$  of matrix  $a$ .
- etc. for higher-dimensional arrays.
- - `{}` Array enumeration. Signatures:
  - `(real[m,n], ..., real[m,n]): real[k,m,n]`.  
 $\{M_1, \dots, M_k\}$  is the 3D array  $a$  such that  $a[i] = M_i$  for  $1 \leq i \leq k$ .  
 It is required that  $k \geq 1$ .
  - `(real[m,n,p], ..., real[m,n,p]): real[k,m,n,p]`.  
 $\{A_1, \dots, A_k\}$  is the 4D array  $a$  such that  $a[i] = A_i$  for  $1 \leq i \leq k$ . It  
 is required that  $k \geq 1$ .
  - etc. for higher dimensions.
- `blocks4`. Create a matrix from four submatrices. Signatures:
  - `(real[m1,n1], real[m1,n2], real[m2,n1], real[m2,n2]): real[m,n]`  
 where  $m = m_1 + m_2$  and  $n = n_1 + n_2$ .  
`blocks4(A,B,C,D)` is the matrix
 
$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}.$$
  - `(real, real[n], real[m], real[m,n]): real[m+1,n+1]`.  
`blocks4(a,b,c,D) = blocks4(A,B,C,D)` where
    - \*  $A$  is the  $1 \times 1$  matrix created from scalar  $a$ ,
    - \*  $B$  is the  $1 \times n$  matrix created from vector  $b$ , and
    - \*  $C$  is the  $m \times 1$  matrix created from vector  $c$ .
  - `(real[m,n], real[m], real[n], real): real[m+1,n+1]`.  
`blocks4(A,b,c,d) = blocks4(A,B,C,D)` where
    - \*  $B$  is the  $m \times 1$  matrix created from vector  $b$ ,
    - \*  $C$  is the  $1 \times n$  matrix created from vector  $c$ ,
    - \*  $D$  is the  $1 \times 1$  matrix created from scalar  $d$ .
- `diag`. Create a diagonal or block-diagonal matrix. Signatures:
  - `(): real[0,0]`.  
`diag()` is the  $0 \times 0$  matrix.
  - `(real, ..., real): real[k,k]`.  
`diag(x1, ..., xk) = diag(vec(x1, ..., xk))` for  $k \geq 1$ .
  - `(real[n]): real[n,n]`.  
`diag(v)` is the diagonal matrix with  $v$  as its diagonal.



- $(\text{real}[m_1, n_1], \dots, \text{real}[m_k, n_k]): \text{real}[m, n]$   
 where  $m = m_1 + \dots + m_k$  and  $n = n_1 + \dots + n_k$  and  $k \geq 1$ .  
 $\text{diag}(M_1, \dots, M_k)$  is the block diagonal matrix having  $M_1, \dots, M_k$   
 as the blocks.
- $(\text{real}[k, m, n]): \text{real}[km, kn]$ .  
 $\text{diag}(A) = \text{diag}(A[1], \dots, A[k])$ .
- $(T_1, \dots, T_k): \text{real}[m, n]$   
 where each  $T_i$  is one of  $\text{real}$ ,  $\text{real}[n_i]$ ,  $\text{real}[m_i, n_i]$ , or  $\text{real}[p_i, m_i, n_i]$ ,  
 and  $m$  and  $n$  are computed from the shapes of the types  $T_i$ .  
 $\text{diag}(a_1, \dots, a_k) = \text{diag}(M_1, \dots, M_k)$  where  $M_i$  is
  - \*  $\text{diag}(a_i)$  if  $T_i$  is  $\text{real}$  or  $\text{real}[n_i]$  or  $\text{real}[p_i, m_i, n_i]$ ;
  - \*  $a_i$  if  $T_i$  is  $\text{real}[m_i, n_i]$ .
- **diag\_sqr**. Signatures: same as **diag**.  
 $\text{diag\_sqr}(x_1, \dots, x_k) = \text{diag}(\text{square}(x_1), \dots, \text{square}(x_k))$ .
- **div** (binary operator). Integer division. Signatures:
  - $(\text{int}, \text{int}): \text{int}$ .  
 It is required that  $a \geq 0$  and  $b > 0$  in the expression  $a \text{ div } b$ .
- **i2r**. Convert an **int** to a **real**. Signatures:
  - $(\text{int}): \text{real}$ .
- **mat11**. Create a  $1 \times 1$  matrix. Signatures:
  - $(\text{real}): \text{real}[1, 1]$ .
- **mat22**. Create a  $2 \times 2$  matrix. Signatures:
  - $(\text{real}, \text{real}, \text{real}, \text{real}): \text{real}[2, 2]$ .  
 $\text{mat22}(a, b, c, d)$  is the matrix
 
$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$
- **negate**. Same as unary  $(-)$ .
- **sqr**. Square-root applied element-wise. Signatures:
  - $(\text{real}): \text{real}$ .
  - $(\text{real}[n]): \text{real}[n]$ .
  - $(\text{real}[m, m]): \text{real}[m, n]$ .
  - etc. for higher dimensions.
- **square**. Square each element. Signatures:
  - $(\text{real}): \text{real}$ .

- `(real[n]): real[n]`.
- `(real[m,n]): real[m,n]`.
- etc. for higher dimensions.
- **to\_matrix**. Convert a vector to a matrix. Signatures:
  - `(real[n]): real[n,1]`.
- **transp**. Matrix transpose. Signatures:
  - `(real[m,n]): real[n,m]`.
- **vec**. Create a vector. Signatures:
  - `(real,...,real): real[k]`.  
`vec(x1,...,xk)` is the vector of length  $k \geq 0$  with the indicated elements.
  - `(real[n1],...,real[nk]): real[n]`  
 where  $n = n_1 + \dots + n_k$  and  $k \geq 1$ .  
`vec(v1,...,vk)` is the result of appending vectors  $v_1$  through  $v_k$ .
  - `(T1,...,Tk): real[n]`  
 where each  $T_i$  is **real** or **real[n<sub>i</sub>]**, and  $n$  is computed from the shapes of the types  $T_i$ .  
`vec(a1,...,ak) = vec(v1,...,vk)` where  $v_i$  is
    - \* the length-one vector `vec(ai)` if  $T_i$  is **real**,
    - \*  $a_i$  if  $T_i$  is **real[n<sub>i</sub>]**.
- **vec0**. Create a vector of zeroes. Signatures:
  - `(int): real[n]`  
 where  $n$  is the argument.  
`vec0(n)` is the length- $n$  vector `vec(0,...,0)`.

## 4.5 Data Distributions

- **certainly**. Degenerate distribution that assigns probability 1 to its argument. An example usage would be
 

```
x ~ uniform(0.0,1.0);
y ~ uniform(0.0, 2.0);
certainly(x+y)
```

 which is the distribution of the sum of draws from a  $U(0,1)$  and a  $U(0,2)$  distribution. Signatures:
  - `(real): real~`.
  - `(real[n]): real[n]~`.
  - `(real[m,n]): real[m,n]~`.

- etc. for higher dimensions.
- **exponential\_m**. Exponential distribution parameterized by mean. Signatures:
  - (real): real~.  
`exponential_m( $\mu$ )` is the exponential distribution with mean  $\mu$ , which must be positive.
- **exponential\_mt**. Truncated exponential distribution parameterized by mean and upper bound. Signatures:
  - (real, real): real~.  
`exponential_mt( $\mu, u$ )` is the truncated exponential distribution with upper bound  $u$  and mean  $\mu$ . It is required that  $u > \mu > 0$ . Note that `exponential_mt( $\mu, u$ )` is *not* the same as taking `exponential_m( $\mu$ )` and truncating above at  $u$ , as the mean of that distribution is less than  $\mu$ .
- **exponential\_r**. Exponential distribution parameterized by rate. Signatures:
  - (real): real~.  
`exponential_r( $\theta$ ) = exponential_m(1/ $\theta$ )`. It is required that  $\theta > 0$ .
- **exponential\_rt**. Truncated exponential distribution parameterized by rate and upper bound. Signatures:
  - (real, real): real~.  
`exponential_rt( $\theta, u$ )` is the distribution `exponential_r( $\theta$ )` truncated above at  $u$ . Note that its mean is *not*  $1/\theta$ , due to the truncation. It is required that  $\theta > 0$  and  $u > 0$ .
- **half\_cauchy**. Cauchy distribution truncated below at 0. Signatures:
  - (real): real~.  
`half_cauchy( $s$ )` is the Cauchy distribution with scale parameter  $s > 0$ , truncated below at 0.
- **half\_normal**. 0-mean normal distribution truncated below at 0. Signatures:
  - (real): real~.  
`half_normal( $\sigma$ )` is the normal distribution with mean 0 and standard deviation  $\sigma > 0$ , truncated below at 0.
- **normal**. Univariate normal distribution. Signatures:

- `(real, real): real~`.  
`normal( $\mu, \sigma$ )` is the normal (Gaussian) distribution with mean  $\mu$  and standard deviation  $\sigma > 0$ .

- **uniform.** Uniform distribution. Signatures:

- `(real, real): real~`.  
`uniform( $l, u$ )` is the uniform distribution over the interval from  $l$  to  $u$ . It is required that  $l < u$ .

## 4.6 Time-series Distributions

In the following we write  $y_t$  for the value of a time series at time  $t$ . Note that time series start at  $t = 1$ , so when we refer to  $y_0$  below, this is a latent "one step before the first" value.

- **(+)** (binary op). Sum of time-series distributions. Signatures:

- `(real$, real$): real$~`.  
The distribution  $d_1 + d_2$  is equivalent to

$$\begin{aligned} v_1 &\sim d_1 \\ v_2 &\sim d_2 \\ y_t &= v_{1t} + v_{2t} \quad \text{for all } t \geq 1 \end{aligned}$$

- **accum.** Accumulate. Signatures:

- `(real$~, real, real): real$~`.  
The distribution `accum( $d, \mu, \sigma$ )` is equivalent to

$$\begin{aligned} \delta &\sim d \\ y_0 &\sim \text{normal}(\mu, \sigma) \\ y_t &= y_{t-1} + \delta_t \quad \text{for all } t \geq 1 \end{aligned}$$

Put another way, the time series of differences  $y_t - y_{t-1}$  has the distribution  $d$ . It is required that  $\sigma > 0$ .

- **ar1.** An AR(1) process. Signatures:

- `(real, real, real): real$~`.  
`ar1( $\phi, \sigma_q, \sigma_0$ )` is equivalent to

$$\begin{aligned} y_0 &\sim \text{normal}(0, \sigma_0) \\ y_t &\sim \text{normal}(\phi y_{t-1}, \sigma_q) \quad \text{for all } t \geq 1 \end{aligned}$$

If  $\sigma_0^2 = \sigma_q^2 / (1 - \phi^2)$  then this distribution is a stationary process with mean 0, standard deviation  $\sigma_0$ , and correlation  $\phi^k$  between  $y_t$  and  $y_{t+k}$ . It is required that  $0 < \phi < 1$ ,  $\sigma_q > 0$ , and  $\sigma_0 > 0$ .

- **constp**. Constant time series. Signatures:

– **(real, real): real\$~**.  
**constp**( $\mu, \sigma$ ) is equivalent to

$$\begin{aligned} y_0 &\sim \text{normal}(\mu, \sigma) \\ y_t &= y_0 \quad \text{for all } t \geq 1 \end{aligned}$$

It is required that  $\sigma > 0$ .

- **qp**. Quasi-periodic process. Signatures:

– **(real, real, int, real, real): real\$~**.

**qp**( $P, \ell, n, \rho, \sigma$ ) is a distribution over quasiperiodic time series. The arguments  $P$ ,  $\ell$ , and  $n$  must currently be numeric literals, e.g. **qp**(7.0, 0.8, 6,  $\rho, \sigma$ ). It is required that  $P > 0$ ,  $\ell > 0$ ,  $0 \leq n < P$ ,  $0 \leq \rho \leq 1$ , and  $\sigma > 0$ .

- \* The marginal distribution for  $y_t$  is **normal**(0,  $\sigma$ ). Additionally, the periodic pattern itself is centered around 0.
- \*  $P$  is the period; it does not need to be an integer.
- \*  $\ell$  gives a smoothness length scale;  $y_t$  and  $y_{t+k}$  are positively correlated for  $|k|$  less than about  $\ell P/4$ .
- \* Nonzero values for  $\rho$  allow the periodic pattern to change somewhat from one period to another. Specifically, if  $P$  is an integer then  $y_t$  and  $y_{t+P}$  have a correlation coefficient of  $\phi^P$ , where  $\phi = \sqrt{1 - \rho^2}$ .
- \*  $d$  is the minimum required degrees of freedom for the periodic pattern. This is an implementation wart that will be removed in a later release of Chronikis; for now you can just set it to the minimum of 10 and  $P - 1$ .
- \* This time-series distribution closely approximates a one-dimensional Gaussian process with covariance function

$$\kappa(x) = \sigma^2 \phi^x \exp \left( -2 \left( \frac{\sin(\pi x)}{\ell} \right)^2 \right).$$

- **rw**. Random-walk process. Signatures:

– **(real, real, real): real\$~**.  
**rw**( $\mu_0, \sigma_0, \sigma_q$ ) is equivalent to

$$\begin{aligned} y_0 &\sim \text{normal}(\mu_0, \sigma_0) \\ y_t &\sim \text{normal}(y_{t-1}, \sigma_q) \quad \text{for all } t \geq 1 \end{aligned}$$

It is required that  $\sigma_0 > 0$  and  $\sigma_q > 0$ .

- **ssm**. General form for creating a linear state-space model. Signatures:

- `(real[m], real, real[m,m], real[m,m], real[m], real[m,m]): real$~`.  
`ssm(z, h, T, Q, a0, P0)` specifies the following linear state-space model:

$$\begin{aligned}\alpha_0 &\sim \text{mvnormal}(a_0, P_0) \\ \alpha_t &\sim \text{mvnormal}(\alpha_{t-1}, Q) \quad \text{for all } t \geq 1 \\ y_t &\sim \text{normal}(z' \alpha_t, \sigma) \quad \text{for all } t \geq 1 \\ \sigma &= \sqrt{h}\end{aligned}$$

where `mvnormal`( $\mu, \Sigma$ ) is the multivariate normal distribution with mean vector  $\mu$  and covariance matrix  $\Sigma$ . It is required that  $h > 0$  and  $T$ ,  $Q$ , and  $P$  be nonnegative definite matrices.

- **wn.** White noise. Signatures:

- `(real): real$~`.  
`wn( $\sigma$ )` is equivalent to

$$y_t \sim \text{normal}(0, \sigma) \quad \text{for all } t \geq 1$$

It is required that  $\sigma > 0$ .

## 4.7 Formal Syntax

In the following,

- “`*`” means *zero* or more instances separated by commas,
- “`+`” means *one* or more instances separated by commas,
- “`?`” means optional (zero or one instance),
- “`|`” separates alternatives, and
- text in typewriter font is literal text.

Here is the grammar:



The following are tokens, so there is no whitespace within them:

$$\begin{aligned}
 \textit{IntegerLit} &:= \textit{Digits} \\
 \textit{RealLit} &:= \textit{Digits} \textit{Fraction?Exponent?} \\
 &\quad (\textit{must have at least one of Fraction or Exponent}) \\
 \textit{Fraction} &:= . \textit{Digits} \\
 \textit{Exponent} &:= \textit{ExpChar} \textit{Sign? Digits} \\
 \textit{ExpChar} &:= \textbf{E} \mid \textbf{e} \\
 \textit{Sign} &:= + \mid - \\
 \textit{Digits} &:= \textit{Digit} + \\
 \textit{Digit} &:= 0 \mid \dots \mid 9 \\
 \textit{Variable} &:= \textit{Identifier} \\
 \textit{FctName} &:= \textit{Identifier} \\
 \textit{Identifier} &:= \textit{IdentChar0} \textit{IdentChar*} \\
 \textit{IdentChar0} &:= \textbf{a} \mid \dots \mid \textbf{z} \mid \textbf{A} \mid \dots \mid \textbf{Z} \\
 \textit{IdentChar} &:= \textit{IdentChar0} \mid \textit{Digit} \mid \_
 \end{aligned}$$

Note that there is no automatic promotion of integers to reals.