A Simple Conjugate Emulator: Theory and Programming (v. 0.92)

Jonathan Rougier[1]
Department of Mathematics
University of Bristol, UK

February 26, 2007

[1]Department of Mathematics, University Walk, Bristol BS8 1TW, U.K. Email J.C.Rougier@bristol.ac.uk. This document was written while I was a University Fellow at Duke University, Durham NC, U.S.A, working at the Statistical and Applied Mathematical Sciences Institute (SAMSI), http://www.samsi.info

Abstract

This is a self-contained document that gives the statistical theory and the programming for a Normal Inverse Gamma emulator of a stochastic process (e.g., a complex function, a spatio-temporal field). It includes code for the statistical computing environment R (R Development Core Team, 2004), embedded using the `noweb` tool for literate programming (Ramsey, 1994). This code includes adaptations for large problems, and several examples.

To cite this document, please use: "J.C. Rougier (2007), *A Simple Conjugate Emulator: Theory and Programming (v. 0.92)*, unpublished, on-line, `http://www.maths.bris.ac.uk/~mazjcr/SGP.pdf`". The R code is available in `SGP.R`.

Contents

1 Background

1.1 The Normal Inverse Gamma distribution

A brief summary of the Normal Inverse Gamma. I use square brackets to denote vector concatenation. If

$$[x, y] \mid \tau \sim \mathrm{N}_{k_1+k_2}(m, \tau V) \tag{1a}$$

$$\tau \sim \mathrm{IG}(a, d) \tag{1b}$$

where x and y are k_1- and k_2-vectors, respectively, then $\{[x, y], \tau\}$ has a Normal Inverse Gamma distribution, $\{[x, y], \tau\} \sim \mathrm{NIG}_{k_1+k_2}(m, V, a, d)$. The convenient Inverse Gamma parameterisation is

$$\pi(\tau; a, d) = \frac{(d/2)^{a/2}}{\Gamma(a/2)} \tau^{-(1+a/2)} \exp\{-(d/2)\tau^{-1}\} \tag{2}$$

with mean and variance

$$\mathrm{E}(\tau) = \frac{d}{a-2} \quad \text{and} \quad \mathrm{Var}(\tau) = \frac{2d^2}{(a-2)^2(a-4)} \tag{3}$$

provided that $a > 2$ and $a > 4$, respectively. Here d is the scale parameter and a is the shape parameter.

If we update by y, then

$$\pi(x, \tau \mid y) = \pi(x \mid y, \tau)\, \pi(\tau \mid y). \tag{4}$$

The first distribution is a standard conditional Normal update, and retains the general form $x \mid \{y, \tau\} \sim \mathrm{N}_{k_1}(m_{1\cdot2}, \tau V_{1\cdot2})$, where

$$m_{1\cdot2} \triangleq m_1 + V_{12}(V_{22})^{-1}(y - m_2) \tag{5a}$$

$$V_{1\cdot2} \triangleq V_{11} - V_{12}(V_{22})^{-1}(V_{12})^T, \tag{5b}$$

the usual Normal conditional updates. The updated distribution for τ remains Inverse Gamma:

$$\begin{aligned}
\pi(\tau \mid y) &\propto \pi(y \mid \tau) \times \pi(\tau) \\
&\propto |\tau V_{22}|^{-1/2} \exp\{-(1/2)q\tau^{-1}\} \times \tau^{-(1+a/2)} \exp\{-(d/2)\tau^{-1}\} \\
&\propto \tau^{-(1+\{a+k_2\}/2)} \exp\{-(\{d+q\}/2)\tau^{-1}\} \\
&\sim \mathrm{IG}(a + k_2, d + q)
\end{aligned} \tag{6}$$

where $q \triangleq (y - m_2)^T(V_{22})^{-1}(y - m_2)$. Thus the updated distribution is still Inverse Gamma:

$$\{x, \tau\} \mid y \sim \mathrm{IG}_{k_1}(m_{1\cdot2}, V_{1\cdot2}, a + k_2, d + q). \tag{7}$$

If $\{x, \tau\} \sim \mathrm{NIG}_k(m, V, a, d)$ then integrating out τ gives the unconditional mean and variance

$$\mathrm{E}(x) = m \quad \text{and} \quad \mathrm{Var}(x) = \mathrm{E}(\tau) \times V = \frac{d}{a-2} V. \tag{8}$$

For the density function, we have

$$\begin{aligned} \pi(x) &\propto [(d+q)/2]^{-(a+k)/2} \\ &\propto \Big[1 + a^{-1}(x-m)^T\{(d/a)V\}^{-1}(x-m)\Big]^{-(a+k)/2} \end{aligned} \tag{9}$$

which is the density function of the Multivariate Student-t distribution,

$$x \sim \mathrm{T}_k\Big(m, (d/a)V, a\Big), \tag{10}$$

where a is the degrees of freedom.

1.2 Conjugate SGP emulator

Consider a computer model with model-input v, producing a field of outputs indexed by s. Each individual output can be represented as $f(s, v)$, or $f(x)$ for simplicity, where $x = [s, v]$. For kriging, we would have $v = \emptyset$. Note that there is no requirement that s and v form a rectangular layout, with a full set of v for every s and a full set of s for every v, although this can make the calculations more efficient; see section 8. It can happen, for example, that some outputs (values for s) are only accessible at certain inputs (values for v).

The NIG approach is suitable when all of the outputs are of the same type (e.g., sea surface temperatures); in this case, s usually indexes location and/or time. While it is possible, in principle, to include in s an indication of *type* (e.g., $0 =$ sea surface temperature and $1 =$ humidity), differences in units and scales are hard to incorporate into the NIG framework, unless prior information is very strong. There is a generalisation for multiple types, the Matrix Normal Inverse Wishart, which will be described elsewhere.

The conjugate emulator for $f(x)$ with hyperparemeters $\{m, V, a, d\}$ has the prior form

$$f(x) \mid \beta, \tau \sim \mathrm{GP}\Big(g(x)^T\beta, \tau \times r(x, x')\Big) \tag{11a}$$

$$\beta \mid \tau \sim \mathrm{N}_q(m, \tau V) \tag{11b}$$

$$\tau \sim \mathrm{IG}(a, d) \tag{11c}$$

where $\mathrm{GP}(\cdot,\cdot)$ denotes a Gaussian process with specified mean and variance functions, $g(\cdot)$ is a q-vector of specified regressor functions, and $r(\cdot,\cdot)$ is a specified variance function. Writing $u(x) \triangleq f(x) - g(x)^T\beta$, so that

$$f(x) \equiv g(x)^T\beta + u(x), \tag{12}$$

$u(\cdot)$ is a Gaussian process, $u(x) \perp\!\!\!\perp \beta$, $\mathrm{E}(u(x)) = 0$ and $\mathrm{Cov}(u(x), u(x')) = \tau \times r(x, x')$.

Suppose we have an ensemble of evaluations of $f(\cdot)$ at the n settings given by the rows of X; write this ensemble as $(f; X)$. Consider some arbitrary collection of 'new' x-values, Z. Denote by f_z the vector $f(Z)$, likewise for u_z. The vector $[\beta, u_z, f] \,|\, \tau$ is jointly normal, with conditional mean and variance

$$\begin{pmatrix} m \\ \mathbf{0} \\ Gm \end{pmatrix} \quad \text{and} \quad \tau \times \begin{pmatrix} V & \mathbf{0} & VG^T \\ \cdot & R_z & R_{zx} \\ \cdot & \cdot & GVG^T + R_x \end{pmatrix} \tag{13}$$

where G is the regressor matrix with $G_i \triangleq g(X_i)$, R_x is the matrix $r(X_i, X_j)$, likewise for R_z and R_{xz}. When we update $\{[\beta, u_z], \tau\}$ by $(f; X)$ we have the Normal Inverse Gamma form where $[\beta, u_z] \mid \{f, \tau\}$ is Normal, and $\tau \mid f$ is Inverse Gamma. As f_z is a specified linear combination of β and u_z, the predictive distribution for $f_z \mid \tau$ is Normal. When τ is integrated out the predictive distribution for f_z is Multivariate Student-t, parameterised by a mean vector, a scale matrix, and a scalar 'degrees of freedom'.

1.3 Outline of the code

The calculation of the predictive distribution of f_z proceeds in two stages. First, a container is constructed from the initial values of the hyperparameters and the ensemble $(f; X)$. This does all the once-only calculations that are required for any choice of Z. Then this container is combined with information about a particular choice of Z to make a particular set of predictions. This set of predictions is represented, in its most general terms, as the three parameters of the Multivariate Student-t distribution.

Here is the top-level code-chunk.

3 ⟨*3⟩≡
```
#### create SGP emulator ####

## Do not edit this file, which is created by 'notangle SGP.nw > SGP.R'.
## Edit SGP.nw instead.
```
⟨*set up SGP* 5⟩

⟨*predict with SGP* 10⟩
⟨*example* 13b⟩
⟨*sample from the emulator* 17⟩
⟨*emulator diagnostics* 19⟩
⟨*example Fit* 22c⟩
⟨*example IR* 37⟩

```
#### option to run examples if debug == TRUE

debug <- getOption('debug')
if (is.logical(debug) && debug) {
  eval(SGPexample)
  eval(SGPexampleFit)
  eval(SGPexampleEB)
  eval(SGPexampleMix)
  eval(SGPexampleIR)
}
```

2 Setting up the emulator container

It's simplest to represent the emulator as a list with class `c("SGP", "list")`. Note that the function `ir` is discussed in section 8, along with `SMW update hyperparameters`.

5 ⟨*set up SGP* 5⟩≡

```
#### set up emulator using prior hyperparameters and ensemble

"setUpSGP" <- function(g, r, m, V, a, d, f = NULL, X = NULL, ir = NULL)
{
  ⟨sort out verbose 6a⟩
  ⟨unpack a list 6b⟩
  ## package into a list: initial objects

  robj <- list(g = g, r = r, m = m, V = V, a = a, d = d,
    f = f, X = X, ir = ir)

  ⟨check the arguments 6c⟩
  if (is.null(ir)) {

    ⟨update hyperparameters 8⟩
    ## add new objects

    robj <- c(robj, list(Q = Q, P1 = P1, P2 = P2))

  } else {

    ⟨SMW update hyperparameters 34⟩
    ## add new objects

    robj$ir <- ir # might have been redefined
    robj <- c(robj, list(iS = iS, P1 = P1, P2 = P2))

  }

  ## add updated hyperparameters and return

  robj <- c(robj, list(EB = EB, SB = SB, anew = anew, dnew = dnew))
  class(robj) <- c("SGP", class(robj))
  return(robj)
}
```

This gets used more than once.

6a ⟨*sort out verbose* 6a⟩≡
```
verbose <- getOption('verbose')
verbose <- is.logical(verbose) && verbose
```

Sometimes it is convenient to pass in a list instead of individual values; for safety, ensure that this is from a proper `SGP` object. Note that `f` and `X` do not get picked up.

6b ⟨*unpack a list* 6b⟩≡
```
## unpack list argument

if (inherits(g, "SGP")) {
  SGP <- g
  for(obj in c("g", "r", "m", "V", "a", "d", "ir"))
    assign(obj, SGP[[obj]])
  rm(SGP)
}
```

2.1 Checking the arguments

In `setUpSGP` we have the option not to pass `f, X`, which will cause prediction according to the prior.

6c ⟨*check the arguments* 6c⟩≡
```
## check the arguments

stopifnot(is.vector(m), is.matrix(V), length(a) == 1, length(d) == 1)
q <- length(m)
stopifnot(identical(dim(V), c(q, q)))

## chance for a quick exit

if (is.null(f) || is.null(X)) {

  class(robj) <- c("SGP", class(robj))
  return(robj)

} else {

  ⟨check not-NULL X 7⟩
}
```

The following snippet can be re-used later on in `predictSGP`. Note that there is no checking of `X` itself, likewise `Z` below. These can be any objects: the important thing is that the expressions `g(X)` and `r(X, Z)` evaluate correctly. `f` is only checked for consistency with `X`.

7 ⟨*check not-NULL X* 7⟩≡
```
stopifnot(is.function(g), is.function(r))

G <- try(g(X))
if (inherits(G, "try-error"))
  stop("Cannot evaluate 'g' with argument 'X'")
stopifnot(is.matrix(G), ncol(G) == q, !any(is.na(G)))

n <- nrow(G)
stopifnot(is.vector(f), length(f) == n)

Rx <- try(r(X, X))
if (inherits(Rx, "try-error"))
  stop("Cannot evaluate 'r' with argument 'X, X'")
stopifnot(is.matrix(Rx), identical(dim(Rx), c(n, n)))
```

Note that to make `r` more efficient, it can be defined to take a single argument in the case where a variance matrix is required, which can then exploit the fact that `r(X, X)` is symmetric. This would involve modifying the code to evaluate `r(X)` (above) and `r(Z)` (below).

2.2 Updating the hyperparameters

The central object is the inverse of the variance matrix of $(f; X)$. This variance is (suppressing the dependence on X)

$$\mathrm{Var}\left(f \mid \tau\right) = \tau\{GVG^T + R_x\} \equiv \tau S, \tag{14}$$

say, where S is the scale matrix. The updated conditional mean and variance of β are

$$\mathrm{E}(\beta \mid f, \tau) = m + VG^T S^{-1}(f - Gm) \tag{15a}$$
$$\mathrm{Var}(\beta \mid f, \tau) = \tau\{V - VG^T S^{-1}(VG^T)^T\}, \tag{15b}$$

applying the Normal conditioning formulae from (13).

For this type of calculation I like to combine `chol` with `backsolve` and `crossprod`: it's clean and efficient, and it ensures that all scale matrices are (almost) sure to be non-negative definite and symmetric. So this code will always run if there is enough memory to compute the Choleski decomposition of S. Larger calculations, where `chol(S)` might be a problem, are discussed in section 8.

A note on pivoting. The `chol` function in R has the option to using pivoting, which is a very good idea for $n \times n$ matrices. With pivoting, $(QP)^T QP = S$, where P is a permutation matrix that re-orders the columns of Q. If `Q <- chol(S, pivot = TRUE)` then setting `pivot <- attr(Q, "pivot"); oo <- order(pivot)`, we have, for arbitrary conformable matrix A,

$$\begin{aligned} AP &= \texttt{A[, oo]} \\ P^T A &= \texttt{A[oo,]} \\ AP^T &= \texttt{A[, pivot]} \\ PA &= \texttt{A[pivot,]} \end{aligned}$$

(don't forget to include `drop = FALSE` in practice). The following calculation uses

$$P_1 \triangleq (QP)^{-T} GV = Q^{-T} PGV \tag{16a}$$

$$P_2 \triangleq (QP)^{-T}(f - Gm) = Q^{-T} P(f - Gm), \tag{16b}$$

so that

$$\mathrm{E}(\beta \mid f, \tau) = m + VG^T (QP)^{-1} (QP)^{-T} (f - Gm) = m + (P_1)^T P_2 \tag{16c}$$

$$\mathrm{Var}(\beta \mid f, \tau) = \tau \{V - (P_1)^T P_1\} \tag{16d}$$

$$(f - Gm)^T \Sigma^{-1} (f - Gm) = (P_2)^T P_2. \tag{16e}$$

8 ⟨*update hyperparameters* 8⟩≡
```
## update the B mean and scale

Q <- chol(crossprod(tcrossprod(chol(V), G)) + Rx, pivot = TRUE) # chol(S)
pivot <- attr(Q, "pivot")

PP <- backsolve(Q, cbind(G %*% V, f - G %*% m)[pivot, , drop = FALSE],
  transpose = TRUE)
P1 <- PP[, 1:q, drop = FALSE]
P2 <- PP[, q + 1]
rm(PP)

EB <- m + drop(crossprod(P1, P2))
SB <- V - crossprod(P1)
```

```
## update a and d

anew <- a + n
dnew <- d + drop(crossprod(P2))
```

3 Prediction at new inputs

The `predictSGP` function can return either the parameters of the Multivariate Student-t, or the mean and variance directly. I've added a `fromPrior` argument, so that updating does not preclude examination of the prior predictions.

A new variance function can be provided, `r2`. This is because sometimes we want to predict a version of f with a smoother residual, which will affect R_{zx} and R_z, below.

10 ⟨*predict with SGP* 10⟩≡

```
"predictSGP" <- function(SGP, Z, type = c("pars", "MV"),
  fromPrior = FALSE, r2 = r)
{
  stopifnot(inherits(SGP, "SGP"), is.logical(fromPrior))
  type <- match.arg(type)
  ⟨sort out verbose 6a⟩
  ## unpack basic objects

  for (obj in c("g", "r", "f", "X"))
    assign(obj, SGP[[obj]])

  if (!identical(r, r2))
    r <- r2

  ⟨predict with the prior? 11a⟩
  ## predicting from the updated distribution

  if (is.null(SGP$ir)) {

    ⟨update the residual 12⟩
  } else {

    ⟨SMW update the residual 35a⟩
  }

  ⟨Conditional mean and scale of fz 13a⟩
  ⟨wrap up predict 11b⟩
}
```

If `X` is `NULL` then we have not updated the hyperparameters, so we predict with the prior. The snippet `check not-NULL X` will create `G` and `Rx`. The snippet `wrap up predict` can be used here and at the end.

11a ⟨*predict with the prior?* 11a⟩≡

```
## option to predict with the prior

if (fromPrior || is.null(f) || is.null(X)) {

  if (verbose)
    warning("Predicting from the prior")

  for (obj in c("m", "V", "a", "d"))
    assign(obj, SGP[[obj]])

  X <- Z
  f <- rep(0, nrow(X))
  q <- length(m)

  ⟨check not-NULL X 7⟩
  ## do prediction with prior

  EfZ <- drop(G %*% m)
  SfZ <- crossprod(tcrossprod(chol(V), G)) + Rx
  anew <- a
  dnew <- d

  ⟨wrap up predict 11b⟩
}
```

This snippet gets used twice.

11b ⟨*wrap up predict* 11b⟩≡

```
## package and return

if (type == "pars")
  return(list(E = EfZ, S = (dnew / anew) * SfZ, df = anew))
else if (type == "MV")
  return(list(M = EfZ, V = (dnew / (anew - 2)) * SfZ))
else stop("Never get here.")
```

3.1 Updating the residual

We need to fill out the rest of the Normal distribution $[\beta, u_z] \mid \{f, \tau\}$. We need the updated conditional mean and variance of the residual u_z, and the updated conditional covariance of β and the residual. These come directly from (13).

$$\mathrm{E}\,(u_z \mid f, \tau) = R_{zx} S^{-1} (f - Gm) \tag{17a}$$
$$\mathrm{Var}\,(u_z \mid f, \tau) = \tau\,\{R_z - R_{zx} S^{-1} (R_{zx})^T\} \tag{17b}$$
$$\mathrm{Cov}\,(\beta, u_z \mid f, \tau) = -\tau\,\{V G^T S^{-1} (R_{zx})^T\} \tag{17c}$$

Objects `Q`, `P1`, and `P2` will be used again here, along with $P_3 \triangleq (QP)^{-T} (R_{zx})^T = Q^{-T} P (R_{zx})^T$, remembering that `Q` has been computed with pivoting.

12 ⟨*update the residual* 12⟩≡
```
## unload new objects

for (obj in c("Q", "P1", "P2"))
  assign(obj, SGP[[obj]])

## compute mean and scale of uz

Rz <- r(Z, Z)
Rxz <- r(X, Z)

pivot <- attr(Q, "pivot")
P3 <- backsolve(Q, Rxz[pivot, , drop = FALSE], transpose = TRUE)
EuZ <- drop(crossprod(P3, P2))
SuZ <- Rz - crossprod(P3)
SZB <- -crossprod(P3, P1)
```

3.2 Conditional predictive mean and scale

Our prediction is at inputs Z. The vector f_z is the specified linear combination

$$f_z \equiv \begin{bmatrix} G, I \end{bmatrix} \begin{pmatrix} \beta \\ u_z \end{pmatrix}, \tag{18}$$

where now $G_i \triangleq g(Z_i)$. We have computed the mean vector and scale matrix of $[\beta, u_z] \mid \{f, \tau\}$, so now we can find the mean vector and scale matrix of $f_z \mid \{f, \tau\}$.

Note that if the rows of `Z` are named, then `EfZ` will inherit those names.

13a ⟨*Conditional mean and scale of fz* 13a⟩≡
```
## Conditional mean and scale of fz

for (obj in c("EB", "SB", "anew", "dnew"))
  assign(obj, SGP[[obj]])

G <- try(g(Z))
if (inherits(G, "try-error"))
  stop("Problem with evaluating 'g' at 'Z', check 'Z'")

EfZ <- drop(G %*% EB) + EuZ
SfZ <- crossprod(tcrossprod(chol(SB), G)) + SuZ
tmp <- tcrossprod(SZB, G)
SfZ <- SfZ + tmp + t(tmp)
```

4 Simple 1D example

Here's a simple example, written as an expression to be evaluated with `eval(SGPexample)`.

13b ⟨*example* 13b⟩≡
```
#### Simple example, run with 'eval(SGPexample)'

# source("SGP.R")

"SGPexample" <- expression({
  ⟨SGP 1D example 14⟩
})
```

14 ⟨*SGP 1D example* 14⟩≡

```
## Create a function on [0, 1]

foo <- function(x) 1 + (3/2) * x + exp(-x) * sin(3 * pi * x / 2)

## sample some points

n <- 5
x <- runif(n)
f <- foo(x)

## Set up an emulator

"g" <- function(x) {
  v <- 2*x - 1 # onto [-1, 1]
  cbind(1, v, v^2) # constant, linear, and quadratic terms
}

theta <- 0.33 / sqrt(-log(0.05)) # correlation length 0.33
"r" <- function(x, y) {
  x <- as.vector(x)
  y <- as.vector(y)
  exp(-(outer(x, y, "-")/ theta)^2)
}

myEm <- setUpSGP(
  g = g, r = r,
  m = c(1, 0, 0), V = diag(0.1, 3), a = 3, d = 1,
  f = f, X = matrix(x))

## make a prediction along a grid

grid <- seq(from = 0, to = 1, length = 101)
pp <- predictSGP(myEm, Z = matrix(grid), type = "pars")

## draw a picture?

if (interactive()) {

  ⟨draw picture 16⟩
  ## from the prior

  op <- par(ask = TRUE)
```

```
    drawPicture(myEm, grid = grid, fromPrior = TRUE,
      main = "Function, prior marginal 95% CI and regression line")
    lines(grid, foo(grid), lwd = 2)

    ## from the updated distribution

    drawPicture(myEm, grid = grid, fromPrior = FALSE,
      main = "Function, evaluations, updated marginal 95% CI and regression line")
    lines(grid, foo(grid), lwd = 2)

    par(op)
  }
```

16 ⟨*draw picture* 16⟩≡

```
## little function to draw picture

"drawPicture" <- function(SGP, grid, fromPrior = FALSE, main = "",
  alpha = 0.05)
{
  pp <- predictSGP(SGP, Z = matrix(grid), type = "pars",
    fromPrior = fromPrior)

  sd <- sqrt(diag(pp$S))
  z <- qt(c(alpha/2, 1 - alpha/2), df = pp$df)
  lims <- pp$E + outer(sd, z); colnames(lims) <- c("lower", "upper")

  mu <- if (fromPrior)
    drop(g(matrix(grid)) %*% SGP$m)
  else
    drop(g(matrix(grid)) %*% SGP$EB)

  plot(foo, type = "n", axes = FALSE, ylim = range(lims, mu),
    main = main, cex.main = 1)

  polygon(c(grid, rev(grid)), c(lims[, "lower"], rev(lims[, "upper"])),
    col = "grey", border = NA)

  axis(1)
  axis(2)

  if (!fromPrior)
    points(as.vector(SGP$X), SGP$f, pch = 16)
  lines(grid, mu, lty = 2)

  invisible(NULL)
}
```

5 Sampling from the emulator

It's useful to be able to generate samples from the emulator. For this we need to be able to sample from $\mathrm{T}_k(m, S, \delta)$. If $y \sim \mathrm{T}_k(m, S, \delta)$ then $y - m$ and $(x - m)/\sqrt{w/\delta}$ have the same distribution, where $x \sim \mathrm{N}_k(m, S)$, $w \sim \chi^2(\delta)$, and $x \perp\!\!\!\perp w$.

The `tol` argument is for checking `S` for singularity. The `drop` argument allows for the case `n = 1`.

17 ⟨*sample from the emulator* 17⟩≡

```
#### Sample from the emulator

## MV Student-t random variates

"mvrt" <- function(n = 1, m, S, delta, tol = 1e-6, drop = FALSE)
{
  k <- length(m)
  stopifnot(is.matrix(S), identical(dim(S), c(k, k)), length(delta) == 1)

  eS <- eigen(S, symmetric = TRUE)
  ev <- eS$values
  if (!all(ev >= -tol * abs(ev[1])))
      stop("'S' is not non-negative definite")
  X <- matrix(rnorm(k * n), n)

  X <- eS$vectors %*% diag(sqrt(pmax(ev, 0)), k) %*% t(X)
  chi <- rchisq(n, df = delta)
  X <- drop(m) + sweep(X, 2, sqrt(chi/delta), "/")

  nm <- names(m)
  if (is.null(nm) && !is.null(dn <- dimnames(S)))
      nm <- dn[[1]]
  dimnames(X) <- list(nm, NULL)
  if (n == 1 && drop)
    drop(X)
  else t(X)
}
```

Now we can provide a simple wrapper to sample from the emulator.

18a ⟨*sample from the emulator* 17⟩+≡

```
## wrapper for the emulator

"rsamSGP" <- function(SGP, Z, n = 1, drop = (n == 1), fromPrior = FALSE)
{
  stopifnot(inherits(SGP, "SGP"), length(n) == 1, n >= 1)
  pp <- predictSGP(SGP, Z = Z, type = "pars", fromPrior = fromPrior)
  mvrt(n = n, m = pp$E, S = pp$S, delta = pp$df, drop = drop)
}
```

Append an example.

18b ⟨*SGP 1D example* 14⟩+≡

```
## sample from the emulator

sam <- rsamSGP(myEm, Z = matrix(grid), n = 10)
if (interactive()) {
  matplot(grid, t(sam), add = TRUE, type = "l", lty = 1)
  if (!file.exists("SGPpicture.pdf"))
    dev.print(pdf, file = "SGPpicture.pdf")
}
```

6 Emulator diagnostics

This is the most important section of all!

6.1 Predictive density

The simplest diagnostic is the predictive density, which I usually compute as the negative logarithmic score,

$$\text{NLS} \triangleq -\ln \pi\left(f \mid m, V, a, d; X\right). \tag{19}$$

For this we need the log of the normalizing constant for $\mathrm{T}_k(m, S, \delta)$, which is the log of

$$|\delta\pi S|^{-1/2} \frac{\Gamma\big((\delta+k)/2\big)}{\Gamma(\delta/2)}. \tag{20}$$

19 ⟨*emulator diagnostics* 19⟩≡
```
#### Emulator diagnostics

⟨predictive density 20⟩
⟨leave-one-out 22a⟩
⟨one-step-ahead 24⟩
```

The predictive density can be evaluated over the whole ensemble, or over a new set of observations, $(f_z; Z)$. The default is to compute the NLS.

20 ⟨*predictive density* 20⟩≡
```
## neg log score

"nlsSGP" <- function(SGP, fz = NULL, Z = NULL)
{
  if (is.null(fz) || is.null(Z)) {

    ⟨check for ensemble 21b⟩
    fz <- SGP$f
    Z <- SGP$X
    fromPrior <- TRUE

  } else fromPrior <- FALSE

  ## get prior predictive distribution

  pp <- predictSGP(SGP, Z = Z, type = "pars", fromPrior = fromPrior)
  k <- length(pp$E)
  delta <- pp$df

  ## compute negative of [1 + ...]^{...}

  Q <- try(chol(pp$S))
  if (inherits(Q, "try-error")) {
    warning("Not positive definite scale matrix")
    return(NA)
  }
  q <- drop(crossprod(backsolve(Q, fz - pp$E, transpose  = TRUE)))
  robj <- ((delta + k)/2) * log(1 + q / delta)

  ## add neg normalising constant

  robj <- robj + sum(log(diag(Q))) + (k/2) * log(delta * pi) -
    lgamma((delta + k) / 2) + lgamma(delta / 2)
  robj
}
```

Append an example.

21a ⟨*SGP 1D example* 14⟩+≡
```
## Compute the negative logarithmic score

nls <- nlsSGP(myEm)
cat(sprintf("Negative log score is %.2f\n", nls))
```

Still to do: Provide `E` and `Var` attributes for `nls`, for calibration. Here's a useful snippet.

21b ⟨*check for ensemble* 21b⟩≡
```
## check for ensemble

stopifnot(inherits(SGP, "SGP"))
if (is.null(SGP$f) || is.null(SGP$X))
  stop("Cannot proceed without an ensemble!")
```

6.2 Subset diagnostics

This is a matter of going through the ensemble, computing the appropriate score, and returning the result.

Append an example.

21c ⟨*SGP 1D example* 14⟩+≡
```
## compute subset diagnostics

loo <- looSGP(myEm)
cat("Leave-one-out diagnostic:\n")
print(loo)

osa <- osaSGP(myEm)
cat("One-step-ahead diagnostic:\n")
print(osa)

stopifnot(all.equal.numeric(nls, sum(osa)))
```

6.2.1 Leave one evaluation out at a time

We can compute the leave-one-out score for each evaluation

$$S_{-i} \triangleq -\ln \pi_{-i}(f_i; X_i) \tag{21}$$

where $\pi_{-i}(\cdot)$ denotes the predictive density after updating by all but the ith evaluation.

22a ⟨*leave-one-out* 22a⟩≡

```
## Leave-one-out

"looSGP" <- function(SGP)
{
  ⟨check for ensemble 21b⟩
  loo <- lapply(seq(along = SGP$f), function(i) {
    em <- setUpSGP(SGP, f = SGP$f[-i], X = SGP$X[-i, , drop = FALSE])
    nlsSGP(em, fz = SGP$f[i], Z = SGP$X[i, , drop = FALSE])
  })

  unlist(loo)
}
```

From `looSGP` we can also compute the cross-validation score

$$\text{CVS} \triangleq \sum_{i=1}^{n} S_{-i}. \tag{22}$$

If it is decided to tune the prior hyperparameters, then this might be a better optimand than the NLS, although the latter would be the usual choice within an Empirical Bayes approach, or when mixing over different priors.

22b ⟨*leave-one-out* 22a⟩+≡

```
## cross-validation score

"cvsSGP" <- function(SGP) sum(looSGP(SGP))
```

Here is some example code to optimise the choice of correlation length.

22c ⟨*example Fit* 22c⟩≡

```
#### fit correlation length by x-validation

"SGPexampleFit" <- expression({
  ⟨SGPexampleFit 23⟩
})
```

23 ⟨*SGPexampleFit* 23⟩≡

```
## make objective function of correlation length

"optimand" <- function(theta, type = c("CVS", "NLS")) {

  type <- match.arg(type)

  ## r inherits local theta

  "r" <- function(x, y) {
    x <- as.vector(x)
    y <- as.vector(y)
    exp(-(outer(x, y, "-")/ theta)^2)
  }

  em <- setUpSGP(g = myEm$g, r = r, m = myEm$m, V = myEm$V,
    a = myEm$a, d = myEm$d, f = myEm$f, X = myEm$X)

  if (type == "CVS")
    cvsSGP(em)
  else if (type == "NLS")
    nlsSGP(em)
  else stop("Never get here.")
}

## call optimize

opt <- optimize(f = optimand,
  lower = 0.1 / sqrt(-log(0.05)),  # correlation length 0.1
  upper = 1.0 / sqrt(-log(0.05)),  # correlation length 1.0
  type = "CVS")

cat(sprintf("Optimal value is theta = %.2f; corr length = %.2f\n",
  opt$minimum, opt$minimum * sqrt(-log(0.05))))

## Note that this solution is likely to be on the upper boundary

if (interactive()) {
  op <- par(ask = TRUE)
  theta <- opt$minimum
  myEmOpt <- setUpSGP(myEm, f = myEm$f, X = myEm$X) # r() picks up new theta
  drawPicture(myEmOpt, grid = grid, fromPrior = FALSE,
    main = "Function, evaluations, updated marginal 95% CI and regression line\nAfte
    lines(grid, foo(grid), lwd = 2)
    par(op)
  }

  ## Let this be a cautionary tale about the difficulty of interpreting
  ## the regression coefficients when the correlation length is fitted!
```

6.2.2 Add one evaluation at a time

The prequential score is

$$S_i \triangleq -\ln \pi_i(f_i; X_i) \tag{23}$$

where $\pi_i(\cdot)$ is the predictive density after updating by evaluations $1, \dots, i-1$. This is less expensive to compute than the leave-one-out score. Maybe it can be computed sequentially, by growing the scale matrix of $f_{1:i}$ using the partitioned matrix inverse formula, but not without losing the solidity of the Choleski approach.

24 ⟨*one-step-ahead* 24⟩≡

```
  ## One-step-ahead

  "osaSGP" <- function(SGP)
  {
   ⟨check for ensemble 21b⟩
    n <- nrow(SGP$X)
    osa <- vector("list", n)

    for (i in 1:n) {

      if (i == 1)
        em <- setUpSGP(SGP)

      osa[[i]] <- nlsSGP(em, fz = SGP$f[i], Z = SGP$X[i, , drop = FALSE])

      if (i < n)
        em <- setUpSGP(SGP, f = SGP$f[1:i], X = SGP$X[1:i, , drop=FALSE])
    }

    unlist(osa)
  }
```

Note that the sum of all n one-step-ahead diagnostics equals the negative logarithmic score of the ensemble, i.e.

$$\sum_{i=1}^{n} - \ln \pi_i(f_i; X_i) = - \ln \pi_0(f; X) \equiv \text{NLS}. \tag{24}$$

This is checked in `SGPexample`.

7 Transformations

Sometimes the original random field is unlikely to conform to a Normal process until after some non-linear transformation. For simplicity, suppose that we are dealing with a strictly positive field, for which we think a member of the Box and Cox (1964) family of transformations, indexed by λ, is appropriate:

$$f^{(\lambda)} \triangleq \begin{cases} (f^{\lambda} - 1)/\lambda & \lambda \neq 0 \\ \ln f & \lambda = 0. \end{cases} \tag{25}$$

One approach is to make an explicit choice for λ up-front, and then proceed as though that value is the correct one (which is implicitly what we do if we decide not to transform, for which $\lambda = 1$). Of course, this approach does not preclude discovering through diagnostic testing that the choice is poor, and then refining it in some way. Another approach is to make the choice part of the analysis. This is the one I consider here.

7.1 Choosing a good plug-in

A simple treatment is to select the value for λ that maximises the marginal density of the ensemble, $(f; X)$. This would be the Empirical Bayes approach. Like choosing the correlation length, there are all sorts of murky identification problems here, so caution is advised. For the marginal density on the original scale,

$$\begin{aligned} \pi\left(f \mid \lambda; X\right) &= \pi\left(f^{(\lambda)} \mid \lambda; X\right) \, |J_\lambda| \\ &= |J_\lambda| \iint \pi\left(f^{(\lambda)} \mid \beta, \tau, \lambda; X\right) \, \pi\left(\beta, \tau \mid \lambda\right) \, \mathrm{d}\beta \, \mathrm{d}\tau \end{aligned} \tag{26a}$$

where

$$|J_\lambda| \triangleq \prod_{i=1}^{n} df_i^{(\lambda)}/df_i = \prod_{i=1}^{n} f_i^{\lambda - 1}. \tag{26b}$$

Note the presence of λ in the prior for the parameters $\{\beta, \tau\}$. Different values for λ change the scale of the data $f^{(\lambda)}$, and consequently they change the interpretation of the parameters, and the prior. This point, recognised in Box and Cox, is discussed in Pericchi (1981).

To specify the family of priors over λ we pick some reference value λ_1, and make a choice

$$\pi\,(\beta, \tau \mid \lambda = \lambda_1) \propto \pi_1(\beta, \tau). \tag{27}$$

Relying on the smoothness of our transformation, we can write

$$f^{(\lambda)} \approx a_\lambda + \ell_\lambda\, f^{(1)}, \tag{28}$$

for some $(\alpha_\lambda, \ell_\lambda)$, where ℓ_λ controls the scale of the transformation. Now β will be on the same scale as $f^{(\lambda)}$, and τ on the same scale as $(f^{(\lambda)})^2$. Making the transformation to the λ_1 prior,

$$\pi\,(\beta, \tau \mid \lambda) \propto \pi_1\Big(\beta/\ell_\lambda,\ \tau/(\ell_\lambda)^2\Big)\,(\ell_\lambda)^{-(q+2)}. \tag{29}$$

In our case we may take $\lambda_1 = 1$, which is the inference in its original units, for which we have

$$\pi_1(\beta, \tau) = \mathrm{NIG}_q(m, V, a, d)\,. \tag{30}$$

After substituting and rearranging,

$$\pi\,(\beta, \tau \mid \lambda) = \mathrm{NIG}_q\Big(m\ell_\lambda, V, a, d\ell_\lambda^2\Big)\,, \tag{31}$$

which, happily, is exactly what we would expect. The only thing left to resolve is the choice of ℓ_λ. Box and Cox make the pragmatic suggestion

$$\ell_\lambda = |J_\lambda|^{1/n} = \Big(\prod_{i=1}^{n} f_i\Big)^{(\lambda-1)/n}. \tag{32}$$

This is the harmonic mean of the gradients at the observed values of f. This can be criticised for making the prior depend upon the ensemble, but in an Empirical Bayes approach we are already guilty of that particular crime, so this does not seem to be such an issue.

In our code, we can minimise the NLS to choose λ. This can all be done ‘outside’ the emulator, as demonstrated by the following example.

26 $\langle *\ 3\rangle{+}{\equiv}$

```
#### choose non-linear transformation to plug-in, run with
#### 'eval(SGPexampleEB)'

"SGPexampleEB" <- expression({

  ⟨SGP example EB 27⟩
})
```

27 ⟨*SGP example EB* 27⟩≡
```
## create a Box-Cox function

"BoxCox" <- function(x, lambda = 1, tol = 1e-3) {
  if (abs(lambda) < tol)
    log(x)
  else (x^lambda - 1) / lambda
}

## wrapper to transform SGP with a value for lambda

"BoxCoxSGP" <- function(SGP, lambda)
{
  stopifnot(inherits(SGP, "SGP"))

  f <- SGP$f
  logJ <- (lambda - 1) * sum(log(f))
  ell <- exp(logJ / length(f))

  f <- BoxCox(f, lambda)

  em <- setUpSGP(g = SGP$g, r = SGP$r, m = SGP$m * ell, V = SGP$V, a = SGP$a,
    d = SGP$d * ell^2, f = f, X = SGP$X, ir = SGP$ir)

  nls <- nlsSGP(em) - logJ
  attr(em, "BoxCox") <- list(lambda = lambda, nls = nls)

  em
}

## objective function, starting from emulator Em0

"nlsLam" <- function(lambda) {
  em <- BoxCoxSGP(Em0, lambda = lambda)
  attr(em, "BoxCox")$nls
}

## minimise on lambda in [-3, 3]

if (!exists("myEm", inherits = FALSE))
  stop("Do 'eval(SGPexample)' first")

Em0 <- myEm
```

```
opt <- optimize(f = nlsLam, interval = c(-3, 3))

## But more fun to draw the picture

if (interactive()) {
  op <- par(ask = TRUE)
  lambda <- seq(from = -3, to = 3, by = 0.5)
  y <- sapply(lambda, nlsLam)
  plot(lambda, -y, type = "b",
    main = "Log marginal density of the ensemble",
    xlab = "Values of lambda in Box-Cox transformation", ylab = "")
  abline(v = opt$minimum, lty = 2)
  par(op)
}
```

Including an offset. Note that it would be very straightforward to include an offset in the transformation: the only thing that needs to be modified is the calculation of $|J_\lambda|$, which also affects ℓ_λ.

Back-transforming. When we predict f_z we want to do it on the original scale. Although there are standard methods for inferring the mean and variance of f_z from that of $f_z^{(\lambda)}$, it is probably easiest to do this by sampling $f_z^{(\lambda)}$, transforming back to the original scale, and then summarising. Also, this avoids problems where the back-transformation creates a process without a finite second moment, although we would have to take care how this was summarised. This is addressed by Oliveira *et al.* (1997).

29 ⟨*SGP example EB* 27⟩+≡

```
## invert Box Cox

"iBoxCox" <- function(y, lambda = 1, tol = 1e-3)
{
  if (abs(lambda) < tol)
    exp(y)
  else (lambda * y + 1)^(1/lambda)
}

## sample on transformed scale (reuse snippet)

lambda <- opt$minimum
EmEB <- BoxCoxSGP(Em0, lambda = lambda)

sam <- rsamSGP(EmEB, Z = matrix(grid), n = 50)
sam <- iBoxCox(sam, lambda = lambda)

if (interactive()) {
  op <- par(ask = TRUE)
  matplot(grid, t(sam), type = "l",
    main = "Random samples after Box-Cox transformation", xlab = "x",
    ylab = "foo(x)", lty = 1)
  lines(grid, foo(grid), lwd = 2)
  points(drop(Em0$X), Em0$f, pch = 16)
  par(op)
}
```

7.2 Mixing over transformations

A better—I think—treatment is not to choose and plug-in one particular value for λ, but to attach a prior to λ, and updating our choice of λ using the ensemble $(f; X)$. This accounts for uncertainty in the choice of transformation when making predictions.

It's simplest to use a discrete set of candidate values for λ, say λ_i, for $i = 1, \ldots, m$. Then we specify a prior $\pi\left(\lambda = \lambda_i\right) = p_i$. The updated probabilities are then

$$\pi\left(\lambda_i \mid f; X\right) \propto \pi\left(f \mid \lambda_i; X\right) \, p_i \tag{33}$$

normalised to sum to 1. Again, sampling is probably the best way to summarise the predictions. If we want a sample of size N, we can draw from a multinomial $\mathrm{M}_m(p', N)$, where p' are the updated probabilities: this tells us how many of each λ_i to include. But I think this introduces an extra source of variation into the estimates based on on the sample. So I would just draw about $\lfloor N p_i' + 0.5 \rfloor$ of each λ_i, where $\lfloor x \rfloor$ is the largest integer not more than x.

30 ⟨* 3⟩+≡

```
#### use mixture of transformations, run with 'eval(SGPexampleMix)'

"SGPexampleMix" <- expression({

  ⟨SGP example Mix 31⟩
})
```

31 ⟨*SGP example Mix* 31⟩≡

```
## check we have run SGPexampleEB

if (!exists("BoxCox", envir = .GlobalEnv, mode = "function"))
  stop("Must do 'eval(SGPexampleEB)' first")

## set up a prior for lambda

lambda <- seq(from = -3, to = 3, by = 0.5)
plam <- rep(1, length(lambda)) # doesn't need to be normalised

## create a list of emulators, with BoxCox attribute

if (!exists("myEm", inherits = FALSE))
  stop("Do 'eval(SGPexample)' first")

Em0 <- myEm
emList <- lapply(lambda, function(lam) BoxCoxSGP(Em0, lambda = lam))

## updated probabilities

plamNew <- plam * sapply(emList, function(em) exp(-attr(em, "BoxCox")$nls))
plamNew <- plamNew / sum(plamNew)

## summary by sampling

N <- 50
ni <- floor(N * plamNew + 0.5) # or use round()

sam <- matrix(NA, 0, length(grid))
for (i in which(ni > 0)) {
  em <- emList[[i]]
  lam <- attr(em, "BoxCox")$lambda
  sami <- rsamSGP(em, Z = matrix(grid), n = ni[i])
  sam <- rbind(sam, iBoxCox(sami, lambda = lam))
}

## colour plot by lambda

if (interactive()) {
  op <- par(ask = TRUE)
  matplot(grid, t(sam), type = "l",
    main = "Random samples from Box-Cox mixture", xlab = "x",
    ylab = "foo(x)", col = 1 + rep(1:sum(ni > 0), ni[ni > 0]), lty = 1)
  lines(grid, foo(grid), lwd = 2)
  points(drop(Em0$X), Em0$f, pch = 16)

  nonZeros <- paste("p(lam = ", lambda, ") = ",
    round(100 * plamNew), "%", sep = "")[ni > 0]
  legend(x = "bottomright", legend = nonZeros,
    col = 1 + 1:sum(ni > 0), lty = 1, bty = "n")

  par(op)
}
```

Note that we can also use mixing to generalise the choice of correlation length. Who knows what kind of bizarre identification problem will arise if we try simultaneously to mix over correlation length and transformation, with regressors? I would guess that this would be even more tricky if we use an Empirical Bayes approach to choose a plug-in. In this case, interpretation of the statistical parameters is not helpful, and the statistical model ends up having a purely instrumental role in predicting f_z from the ensemble $(f; X)$.

8 Large numbers of observations

Sooner or later you are going to want to compute a problem with $n = 4000$ evaluations (or observations), and you are going to find that it takes a long time to compute R_x, and the Choleski decomposition of the scale matrix S.

8.1 Re-expressing the update

In subsection 2.2 and subsection 3.1 I gave the simple update for $[\beta, u_z] | \{y, \tau\}$: see (15) and (17). This is not the most insightful form of these two expressions. We can use the Sherman-Morrison-Woodbury formula (see, e.g., Brookes, 2005, under Identities),

$$(A^{-1} + BC^{-1}B^T)^{-1} = A - AB(C + B^T AB)^{-1}B^T A \tag{34}$$

to write

$$\mathrm{Var}\,(\beta \mid f, \tau) = \tau\,\{V^{-1} + G^T R_x^{-1} G\}^{-1}; \tag{35a}$$

applying the same formula to S gives

$$S^{-1} = R_x^{-1} - R_x^{-1} G \{V^{-1} + G^T R_x^{-1} G\}^{-1} G^T R_x^{-1}. \tag{35b}$$

We can see that the limit $V^{-1} \to \mathbf{0}$, total prior ignorance about the regression coefficients, is well-defined in these two expressions if R_x is non-singular and $n \geq q$. We can also see the advantages that would come from a diagonal R_x, which would be called a 'nugget' in spatial statistics. With a diagonal R_x we do not have to invert a general $n \times n$ variance matrix, which is an $O(n^3)$ operation for the Choleski decomposition.

8.2 Working with the inverse variance function

In (35), the inverse of $r(\cdot, \cdot)$ can be a primitive function. The easiest route is to provide an explicit function `ir`, which computes the matrix inverse of $r(X_i, X_j)$. If $r(\cdot, \cdot)$ is separable and X has some kind of rectangular structure, then this can be very beneficial: see subsection 8.3 and the example in subsection 8.4.

Here is the update using the SMW approach. In order to handle large n, I sacrifice precise symmetry by not decomposing variance objects R_x^{-1} and S^{-1}, both $n \times n$. Define Q as the Choleski factorisation giving

$$Q^T Q \equiv V^{-1} + G^T R_x^{-1} G \tag{36}$$

which is $q \times q$. Then

$$P_1 \triangleq V^T G \quad \text{and} \quad P_2 \triangleq S^{-1}(f - Gm) \tag{37}$$

are both useful below and in `SMW update the residual`.

If no function `ir` is supplied, e.g., by passing the argument `ir = NA`, then one is provided. Note that `ir` takes a single argument, as it is always used to compute a variance matrix.

34 ⟨*SMW update hyperparameters* 34⟩≡

```
## handle non-null ir

if (!is.null(ir)) {

  if (!is.function(ir)) {
    if (verbose)
      warning("Providing default 'ir' function")
    "ir" <- function(X) chol2inv(chol(r(X, X)))
  }
}

## update hyperparameters using SMW approach

iRx <- try(ir(X))
if (inherits(iRx, "try-error"))
  stop("Cannot evaluate 'ir' with argument 'X'")
stopifnot(is.matrix(iRx), identical(dim(iRx), c(n, n)))

Q <- chol(chol2inv(chol(V)) + crossprod(G, iRx %*% G)) # no decomp of iRx
SB <- chol2inv(Q)

iS <- iRx - crossprod(backsolve(Q, crossprod(G, iRx), transpose = TRUE))
P1 <- tcrossprod(V, G)
P2 <- drop(iS %*% (f - G %*% m))

EB <- m + drop(P1 %*% P2)

anew <- a + n
dnew <- d + drop(crossprod(f - G %*% m, P2))
```

Here is the additional code for predictSGP, with

$$P_3 \triangleq S^{-1} R_{xz}. \tag{38}$$

35a ⟨*SMW update the residual* 35a⟩≡

```
## unload new objects

for (obj in c("iS", "P1", "P2"))
  assign(obj, SGP[[obj]])

## compute mean and variance of uz

Rz <- r(Z, Z)
Rxz <- r(X, Z)

EuZ <- drop(crossprod(Rxz, P2))
P3 <- iS %*% Rxz
SuZ <- Rz - crossprod(Rxz, P3) # no decomp of iS
SZB <- -crossprod(P3, t(P1))
```

Append an example of a SMW emulator.

35b ⟨*SGP 1D example* 14⟩+≡

```
#### try out the SMW emulator

myEm1 <- setUpSGP(
  g = g, r = r,
  m = c(1, 0, 0), V = diag(0.1, 3), a = 3, d = 1,
  f = f, X = matrix(x), ir = NA) # trigger SMW with non-null ir

pp1 <- predictSGP(myEm1, Z = matrix(grid))

stopifnot(all.equal(pp, pp1)) # tough test!

if (interactive()) {
  op <- par(ask = TRUE)
  drawPicture(myEm1, grid = grid,
  main = "SMW Emulator (different updating equations, same result)")
  par(op)
}
```

8.3 Exploiting rectangular structure

This is an outline of how an `ir` function can be used to speed up the calculations in situations where there is a rectangular structure in the rows of X, i.e. f can be arranged as a table, with each extent corresponding to one element of a partition of the columns of X. In a computer experiment, if we collect the same set of outputs for every evaluation, then we have a table of index variables by model inputs.

Often we will be able to partition the index variables as well. As a simple rule to determine an ordering across the index variables, I usually picture a methodical man with a measuring device. So in the ordering of the three index variables $(\text{type}, \text{location}, \text{time})$, type varies fastest, then location, and finally time. As already discussed, the NIG is not very suitable for multiple types. If f were rectangular in location and time, then location would go down the rows, and time along the columns. This follows the convention in R that the first extent varies fastest. If this was a computer experiment with multiple evaluations, and we collected same outputs for every evaluations, then the inputs could make up the third extent (depth?).

If I then chose to have a separable $r(\cdot, \cdot)$, then (ignoring model inputs for simplicity)

$$r(x_{ij}, x_{i'j'}) = \sigma^{\text{loc}}_{ii'}\, \sigma^{\text{time}}_{jj'} \tag{39}$$

It follows that under my ordering

$$R = \Sigma^{\text{time}} \otimes \Sigma^{\text{loc}} \tag{40}$$

(note the reversal of the order here: location varies fastest so it goes last), where '$\otimes$' denotes the Kronecker product (see, e.g., Brookes, 2005, under Matrix Relations). This decomposition allows us to compute the inverse of R as the product

$$R^{-1} = (\Sigma^{\text{time}})^{-1} \otimes (\Sigma^{\text{loc}})^{-1}. \tag{41}$$

This is *much* faster than computing R and then inverting. If there were also model inputs to account for, we would tack them onto the final extent of the table, and their inverse variance matrix onto the front of R^{-1}.

Along the same lines, if R_x is sparse, there might be an advantage to using a sparse matrix package like `SparseM` to help compute R_x^{-1} (Koenker and Ng, 2007). Sparsity in R_x can be achieved by using correlation functions with thresholds, so that the correlation between two points X_i and X_j for which the threshold is exceeded on at least on component is zero. Note, however, that this sparsity does not carry over into the rest of the calculation: even if R_x^{-1} was sparse, $V^{-1} + G^T R_x^{-1} G$ would not be.

8.4 A more complicated example

Here is an example with both `r` and `ir` functions. For `ir` I provide the option to use the `SparseM` package. This is actually slower for this example (perhaps I could have programmed it better), because of the overhead of creating the sparse matrices. My view is that using `SparseM` might make an infeasibly large calculation into a do-able one, but it is not going to speed up a calculation that is already do-able.

The data are indexed by location and time; `X` and `Z` are passed as calls to `expand.grid`, e.g.,

```
X <- call("expand.grid", locs = ..., times = ...)
```

This means that the full dataframe for `X` can be unpacked with `eval(X)`, but also that, prior to unpacking, the location and time values can be extracted with `X$locs` and `X$times`.

37 ⟨*example IR* 37⟩≡
```
#### A more complicated example using ir, run with 'eval(SGPexampleIR)'

"SGPexampleIR" <- expression({

  ⟨SGP example IR 38⟩
})
```

38 ⟨*SGP example IR* 38⟩≡

```
## spherical stationary isotropic correlation function
## with correlation length a (corr is zero beyond a).

"spherical" <- function(h, a) {
  dd <- dim(h)
  h <- pmin(a, abs(h))
  robj <- 1 - (3/2) * (h / a) + (1/2) * (h / a)^3
  dim(robj) <- dd
  robj
}

## correlation function can handle product structure in X

clen <- list("locs" = 10, "times" = 1)

"r" <- function(x, y) {

  if (is.call(x) && x[[1]] == "expand.grid" && identical(y, x)) {

    Vlocs <- spherical(outer(x$locs, x$locs, "-"), a = clen$locs)
    Vtimes <- spherical(outer(x$times, x$times, "-"), a = clen$times)

    return(kronecker(Vtimes, Vlocs))
  }

  if (is.call(x)) x <- eval(x)
  if (is.call(y)) y <- eval(y)

  Vlocs <- spherical(outer(x$locs, y$locs, "-"), a = clen$locs)
  Vtimes <- spherical(outer(x$times, y$times, "-"), a = clen$times)

  return(Vtimes * Vlocs)
}

## Inverse correlation function insists on product structure

## Pass in SparseM flag through getOption()

"ir" <- function(x) {

  useSparseM <- getOption('useSparseM')
  useSparseM <- is.logical(useSparseM) && useSparseM
  if (is.call(x) && x[[1]] == "expand.grid") {

    Vlocs <- spherical(outer(x$locs, x$locs, "-"), a= clen$locs)
    Vtimes <- spherical(outer(x$times, x$times, "-"), a = clen$times)

    if (useSparseM) {

      stopifnot(require(SparseM))

      Vlocs <- as.matrix.csr(Vlocs)   # could be more
      Vtimes <- as.matrix.csr(Vtimes) # sophisticated here!

      return(as.matrix(solve(Vtimes) %x% solve(Vlocs)))

    } else {

      return(chol2inv(chol(Vtimes)) %x% chol2inv(chol(Vlocs)))
    }

  } else stop('Unexpected argument')
}

## make up some data and build an emulator

X <- call("expand.grid",
       locs = 1:30,
       times = seq(from = 0, to = 5, by = 1/12))

tmp <- eval(X)
f <- tmp$locs + 0.5 * (tmp$locs - mean(tmp$locs))^2 +
  tmp$locs * sin(2 * pi * tmp$times)
rm(tmp)

"g" <- function(X) {
  X <- eval(X)
  cbind(1, X$locs, X$times, X$locs * X$times, X$locs^2)
}

cat("system.time() for different treatments ...\n")

op <- options(useSparseM = NULL)
```

```
cat("Default, no ir:\n")
print(system.time(
  myEm2 <- setUpSGP(g = g, r = r, m = rep(0, 5), V = diag(1, 5),
    a = 3, d = 1, f = f, X = X)
))

cat("With default ir:\n")
print(system.time(
  myEm3 <- setUpSGP(g = g, r = r, m = rep(0, 5), V = diag(1, 5),
    a = 3, d = 1, f = f, X = X, ir = NA)
))

cat("With specified ir:\n")
print(system.time(
  myEm4 <- setUpSGP(g = g, r = r, m = rep(0, 5), V = diag(1, 5),
    a = 3, d = 1, f = f, X = X, ir = ir)
))

if (require(SparseM)) {
  cat("With specified ir, and SparseM:\n")
  options(useSparseM = TRUE)
  print(system.time(
    myEm5 <- setUpSGP(g = g, r = r, m = rep(0, 5), V = diag(1, 5),
      a = 3, d = 1, f = f, X = X, ir = NA)
  ))
}

options(op)

## Check they all give the same predictions

cat("Checking the answers are all the same ...\n")

Z <- call("expand.grid",
       locs = 1:3 + 0.5,
       times = c(2, 4, 6))

pp2 <- predictSGP(myEm2, Z = Z)
pp3 <- predictSGP(myEm3, Z = Z); stopifnot(all.equal(pp2, pp3))
pp4 <- predictSGP(myEm4, Z = Z); stopifnot(all.equal(pp3, pp4))
if (exists("myEm5", envir = .GlobalEnv, mode = "list")) {
  pp5 <- predictSGP(myEm5, Z = Z); stopifnot(all.equal(pp4, pp5))
}
```

References

G.E.P. Box and D.R. Cox, 1964. An analysis of transformations. *Journal of the Royal Statistical Society, Series B*, **26**, 211–243. with discussion, pp. 244–252. 25, 26

M. Brookes, 2005. The Matrix Reference Manual. On-line, `http://www.ee.ic.ac.uk/hp/staff/dmb/matrix/intro.html`. 33, 36

Roger Koenker and Pin Ng, 2007. SparseM: Sparse Linear Algebra. R package version 0.72, on-line, `http://www.econ.uiuc.edu/~roger/research/sparse/sparse.html`. 36

V. De Oliveira, B. Kedem, and D.A. Short, 1997. Bayesian prediction of transformed gaussian random fields. *Journal of the American Statistical Association*, **92**, 1422–1433. 29

L.R. Pericchi, 1981. A Bayesian approach to transformations to normality. *Biometrika*, **68**(1), 35–43. 26

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2004. ISBN 3-900051-00-3, `http://www.R-project.org`. 1

N. Ramsey, 1994. Literate programming simplified. *IEEE Software*, **11**(5), 97–105. See `http://www.eecs.harvard.edu/~nr/noweb/`. 1