Calibration

RSMs and Computer Experiments: Part 5

Robert B. Gramacy (rbg@vt.edu : http://bobby.gramacy.com)
Department of Statistics, Virginia Tech

Goals

Many scientific phenomena are studied via mathematical (i.e., computer) models and field experiments, simultaneously.

- Real experiments are expensive, and for this and other reasons (ethics, lack of materials/infrastructure, etc.), limited configurations can be entertained.

Computer simulations are lots cheaper, but usually not so cheap as to allow infinite exploration of the configuration space(s).

- Plus the simulations usually idealize reality (i.e., they are biased)
- and can involve more "knobs", or tuning parameters, than can be controlled (or even known) in the field.

So the goal here is to build an apparatus that can harmonize the two data types

- for the purpose of learning about/predicting the "real" process,
- or possibly optimizing some aspect of it.

Calibration

Notation

There are three processes involved:

1. the *real* process R that dictates the dynamics of the phenomena being studied;
2. the field F where a physical experiment observing R takes place;
3. and a computer model M implementing/solving a mathematical model that idealizes the real system.

Let $y^F(x)$ denote a field observation under conditions x, and $y^R(x)$ denote the real output under condition x.

R and F are assumed to be related as follows.

$$y^F(x) = y^R(x) + \varepsilon, \qquad \text{where } \varepsilon \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_\varepsilon^2)$$

- We may have a small number of N_F observations in the field at different x locations, often under replication.

Computer model

Let $y^M(x, u)$ denote the (deterministic) output of a computer model run under conditions x and tuning or calibration parameters u.

- These u are any aspect of the computer model which cannot be controlled in F and are unknown in R.
  - They may simply be an artificial aspect of a computer implementation, like a mesh size.
  - Or they might have real physical meaning (like the acceleration due to gravity) which is not known (precisely enough) to be recorded in the field experiment.

Some people make a big deal about the difference between the two, calling the former a **tuning** parameter, and the latter a **calibration** parameter.

- We won't generally make such a distinction.

Kennedy and O'Hagan

Kennedy and O'Hagan (2001) proposed a Bayesian framework for coupling M and F, hereafter KOH.

KOH represent a real process R as

- the computer model output at the best setting of the calibration parameters, u^*,
- plus a discrepancy term acknowledging that there can be systematic disagreement between the model and the truth.

$$\begin{aligned} & y^R(x) = y^M(x, u^*) + b(x), \\ \text{so that} \quad & y^F(x) = y^M(x, u^*) + b(x) + \varepsilon \end{aligned}$$

- The discrepancy is $b(x)$, and although we may casually refer to it as "bias", the actual bias (which is a property of M not R) would actually work out to

$$-b(x) = y^M(x, u^*) - y^R(x).$$

Unknowns/Bayes

The unknowns are u^*, σ_ε^2, and the discrepancy $b(\cdot)$.

- KOH propose a GP prior for $b(\cdot)$.

Known information or restrictions on the u-values can be specified via a prior $p(u)$,

- or otherwise a default/uniform prior can be used.
- Reference priors are typical for σ_ε^2.

KOH emphasized Bayesian inference,

- particularly averaging over the trade-off between calibration values u and discrepancies $b(\cdot)$.

Emulation

If evaluating the computer model is fast, then inference (Bayesian or otherwise) is made rather straightforward using residuals between computer model outputs and field observations

- at the N_F field locations X^F

$$y^F(X^F) - y^M(X^F, u)$$

- which can be computed at will for any u (Higdon, et al., 2004).

If evaluating the computer model is expensive or otherwise indirectly available,

- an emulator $\hat{y}^M(\cdot, \cdot)$ can be fit to N_M simulations of M run over a design of (x, u) inputs.
- KOH recommend a GP prior for y^M, as usual.

Joint inference

Rather than performing inference for y^M separately, using just the N_M runs, as is typical of a computer experiment in isolation,

- they recommend joint (Bayesian) inference with $b(\cdot)$, u^*, and σ^2_ε
- using *both* field observations and runs of the computer model.

From a Bayesian perspective this is the coherent thing to do:

- infer all unknowns jointly given all data.

It is also potentially *practical* when M is slow, giving small N_M.

- In that setting, even a small number of N_F field observations could be highly informative about $\hat{y}^M$,
  - in addition to informing on $\hat{b}$.

Woah a minute …

But this approach is fraught with computational challenges.

Coupled $b(\cdot)$ and $y^M(\cdot, \cdot)$ lead to parameter/process identification, confounding and MCMC mixing issues.

- Plus, why should $\hat{y}^M$ worry about anything other than y^M?

Emulation already demands substantial computational effort in (modestly) large N_M contexts, i.e., when applied in isolation.

- Coupling compounds the issue.

Modularization

Liu, Bayarri and Berger (2009), hard-core but pragmatic Bayesians, proposed going "back to basics" by

- fitting the emulator $\hat{y}^M(\cdot, \cdot)$ independently, using only the N_M simulations.

They gave it a fancy name: **modularization**.

- Then the went through several other (non computer experiments) examples where Bayesians had gone off the deep end, and showed modularization helps protect against pathologies.

Inference for the rest of the KOH calibration apparatus is still joint,

- for all parameters given $\hat{y}^M$ and field data y^F.

Being practical

We're going to use the modular approach,

- and we're not going to do any MCMC.

Things work great when you maximize, and if you are worried about posterior uncertainty you can usually back that out

- or Bootstrap.

I'm not saying the Bayesian (modularized) approach is bad,

- just that its a little too involved for us right now,
- and it doesn't scale well to some of the bigger (large N_M) examples I've come across in my own work.

Acceleration due to gravity

Ok, enough of the high-level stuff, lets see how this works …

We wish to predict the amount of time it takes for a wiffle ball to hit the ground when dropped at a certain height.

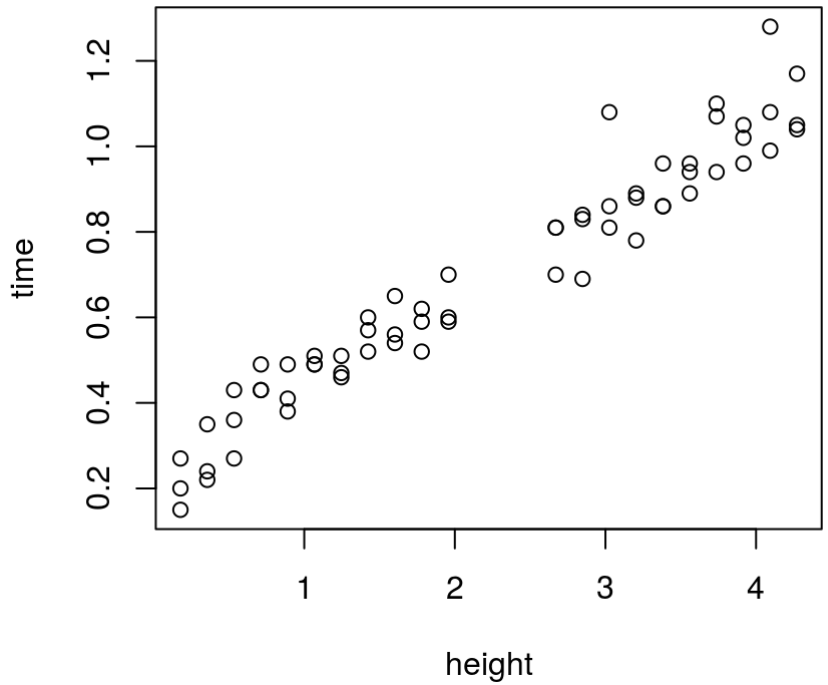
So we perform a physical experiment:

- Drop a ball from different heights and measure how long it takes to hit the ground.
- We have data on times from three replicates of 21 heights.
- These comprise the field data y_F with $N_F = 63$.

(Many thanks to Derek Bingham and Jason Loeppky for this example.)

Visualizing the field data

```
ball <- read.csv("ball.csv")
plot(ball, xlab="height", ylab="time")
```



Staight-fit the field data

One option is, of course, to fit the field data with a GP, be done and go home.

```
library(laGP)
field.fit <- newGP(as.matrix(ball$height), ball$time, d=0.1,
    g=var(ball$time)/10, dK=TRUE)
mle <- jmleGP(field.fit, drange=c(eps, 10), grange=c(eps, var(ball$time)),
    dab=c(3/2, 8))
```

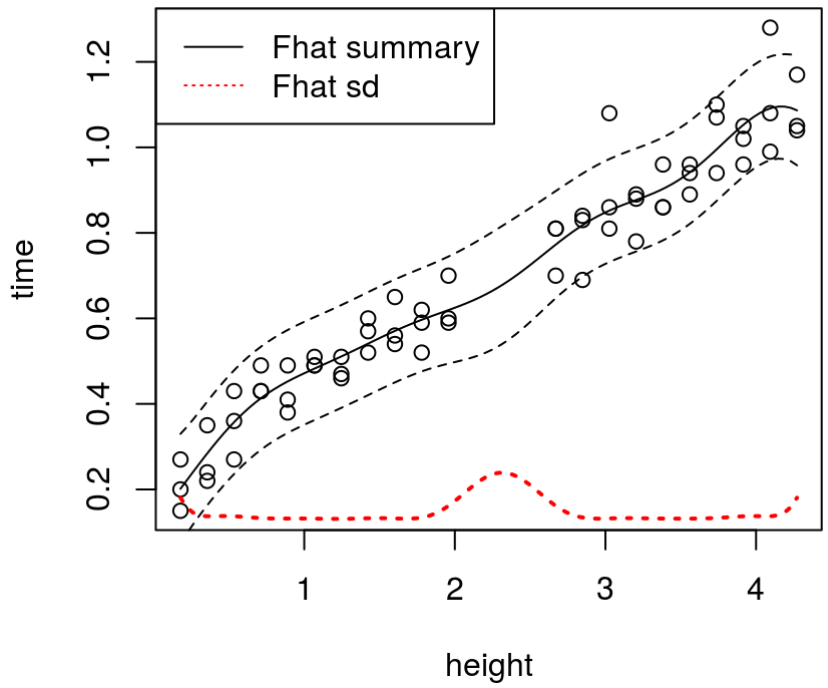
Now make a prediction using this $\hat{y}^F$ on a grid,

- constructing some coded height inputs for later.

```
hr <- range(ball$height)
hs <- seq(0, 1, length=100)
heights <- hs*diff(hr)+hr[1]
p <- predGP(field.fit, as.matrix(heights), lite=TRUE)
```

The result is too wigly, and involves high uncertainty in the gap.

```
plot(ball, xlab="height", ylab="time"); lines(heights, p$mean)
lines(heights, qnorm(0.05, p$mean, sqrt(p$s2)), lty=2)
lines(heights, qnorm(0.95, p$mean, sqrt(p$s2)), lty=2)
lines(heights, 10*sqrt(p$s2)-0.6, col=2, lty=3, lwd=2)
legend("topleft", c("Fhat summary", "Fhat sd"), lty=c(1,3), col=c(1,2))
```



Mathematical model

Perhaps by coupling with "known physics" we can mitigate that effect.

What does "Physics 101" say?

- The time t to drop a distance h for gravity g is given by

$$t = \sqrt{2h/g}.$$

Somewhat realistically, we don't know the value of g for the location where the balls were dropped.

- So gravity is our calibration parameter.
- And of course there are other unknowns, like the air resistance on the ball – which will interact differentially with height/terminal velocity.
    - (I.e., the model is biased/there is potential to improve upon it.)

Computer model

Consider the following computer implementation of our mathematical model using coded inputs in $[0, 1]^2$

- for height (x), taking the range from the observed field data,
- and gravity (u), restricting to $g \in [6, 14]$, equivalently defining a (uniform) prior.

```
timedrop <- function(x, u, hr, gr)
 {
    g <- diff(gr)*u + gr[1]
    h <- diff(hr)*x + hr[1]
    return(sqrt(2*h/g))
 }
```

Computer model design

Now lets fit the computer model on a maximin LHS in 2d of size 21.

- Comparable to the field data size.

```
library(lhs)
XU <- maximinLHS(21, 2)   ## we're going to want to randomize over this
gr <- c(6, 14)
ym <- timedrop(XU[,1], XU[,2], hr, gr)
```

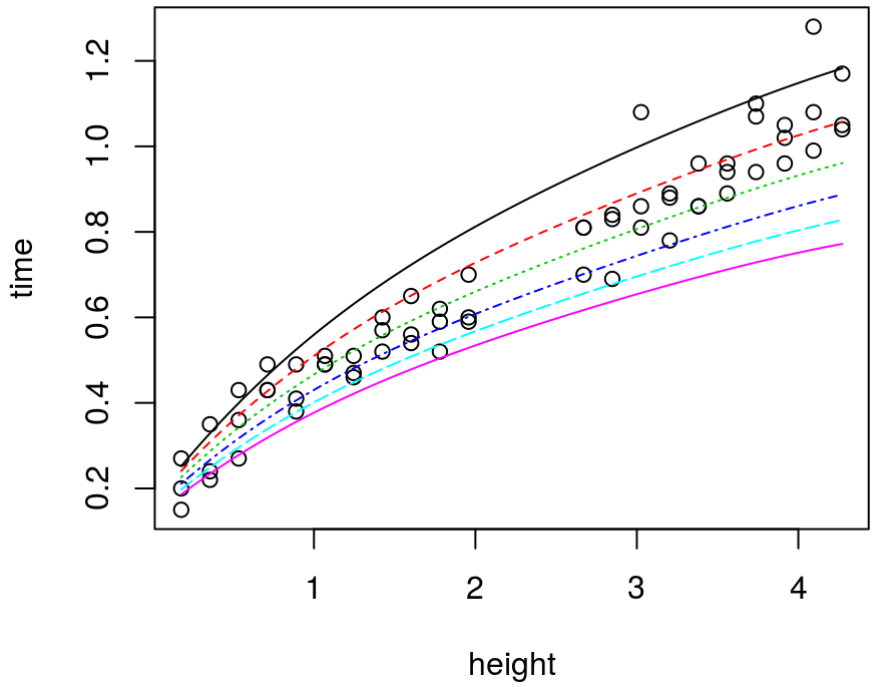
Now lets train a GP on those realizations.

```
ymhat <- newGPsep(XU, ym, d=0.1, g=1e-7, dK=TRUE)
mle <- mleGPsep(ymhat, tmax=10)
```

Lets visualize our computer model output over a range of heights, for particular choices of g.

Some better than others, but possibly all biased.

```
us <- seq(0, 1, length=6)
XX <- expand.grid(hs, us)
pmhat <- predGPsep(ymhat, XX, lite=TRUE)
plot(ball); matlines(heights, matrix(pmhat$m, ncol=length(us)))
```



Modularized calibration

The modularized apparatus calibrates u via the discrepancy between emulated computer model output and field data runs.

```
bhat.fit <- function(X, Y, Ym, da, ga, clean=TRUE)
  {
    bhat <- newGPsep(X, Y-Ym, d=da$start, g=ga$start, dK=TRUE)
    if(ga$mle) cmle <- jmleGPsep(bhat, drange=c(da$min, da$max),
            grange=c(ga$min, ga$max), dab=da$ab, gab=ga$ab)
    else cmle <- mleGPsep(bhat, tmin=da$min, tmax=da$max, ab=da$ab)
    cmle$nll <- - llikGPsep(bhat, dab=da$ab, gab=ga$ab)
    if(clean) deleteGPsep(bhat)
    else cmle$gp <- bhat
    return(cmle)
  }
```

- $\hat{b}$ log likelihood measures goodness-of-fit.
- Observe that `bhat.fit` combines $\hat{b}$ and $\hat{\sigma}^2_\varepsilon$ fits.

An objective to optimize

Now we need to create an objective that we can optimize, over coded gravity u-values, to find the best setting $\hat{u}$ estimating the unknown u^*.

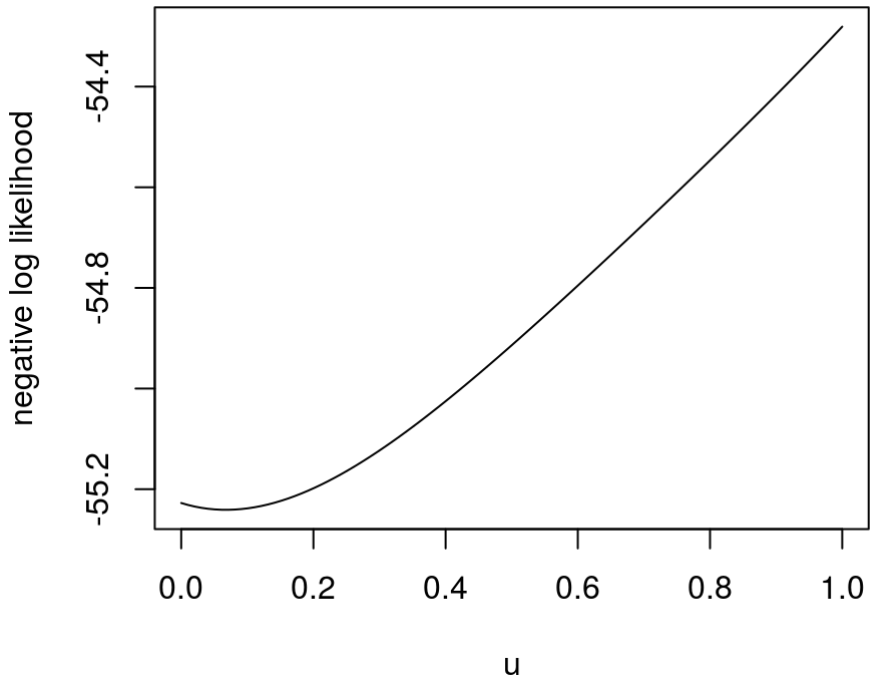
```
calib <- function(u, X, Y, ymhat, da, ga, clean=TRUE)
  {
    Xu <- cbind(X, matrix(rep(u, nrow(X)), ncol=length(u), byrow=TRUE))
    Ym <- predGPsep(ymhat, Xu, lite=TRUE)$mean
    cmle <- bhat.fit(X, Y, Ym, da, ga, clean=clean)
    return(cmle)
  }
```

Since its in 1d, lets evaluate it on a u-grid.

```
u <- seq(0, 1, length=100)
unll <- rep(NA, length(u))
X <- as.matrix((ball$height - hr[1])/diff(hr))
da <- darg(list(mle=TRUE), expand.grid(X[,1], u))
ga <- garg(list(mle=TRUE), ball$time)
for(i in 1:length(u)) unll[i] <- calib(u[i], X, ball$time, ymhat, da, ga)$nll
```

Likelihood surface for u

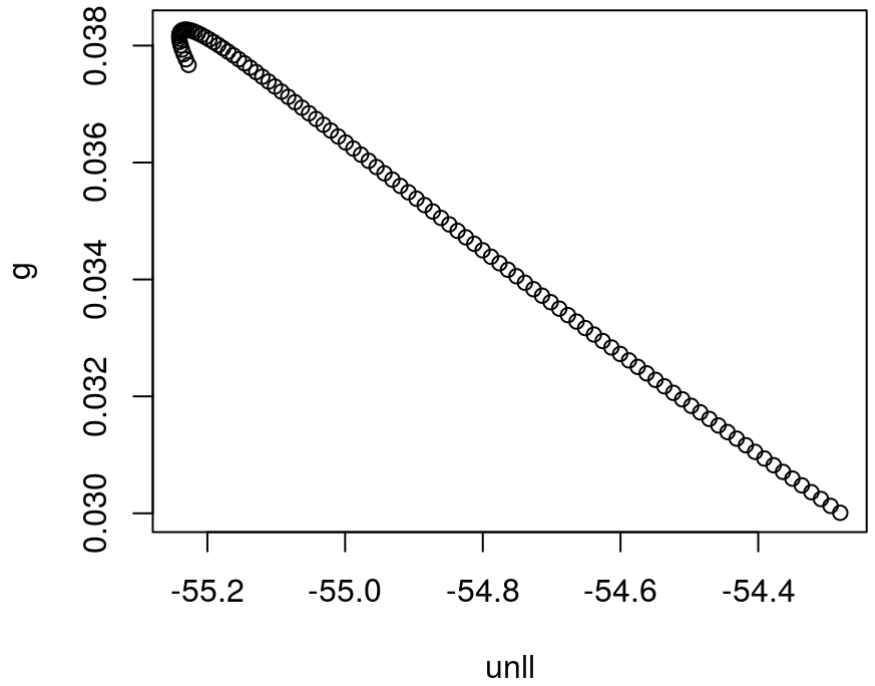
```
plot(u, unll, type="l", xlab="u", ylab="negative log likelihood")
```



- The surface *can* be "noisy". (Why?)

Because the estimated nuggets (i.e., the signal-to-noise ratio) are not smooth over the u-values that are tried.

```
g <- unll
for(i in 1:length(u)) g[i] <- calib(u[i], X, ball$time, ymhat, da, ga)$g
plot(unll, g)
```



Simple fix

This is why folks like the fully Bayesian version.

But there is a simple fix in our setup:

- choose the best u (and its nugget) from the grid, ignoring the "spikes",
- and then re-do the search holding the nugget fixed.

On a grid ….

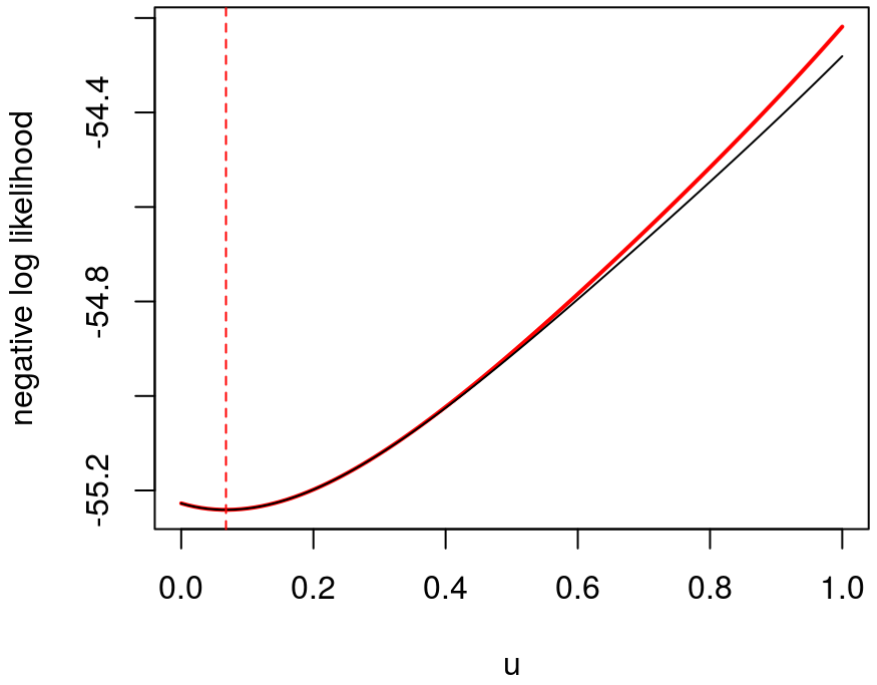
```
unll2 <- unll
ga$mle <- FALSE; ga$start <- g[which.min(unll)]
for(i in 1:length(u)) unll2[i] <- calib(u[i], X, ball$time, ymhat, da, ga)$nll
```

… or via ia `optimize`.

```
obj <- function(x, X, Y, ymhat, da, ga)
    calib(x, X, Y, ymhat, da, ga)$nll
soln <- optimize(obj, lower=0, upper=1, X=X, Y=ball$time,
    ymhat=ymhat, da=da, ga=ga)
```

Fixed

```
plot(u, unll2, type="l", xlab="u", col=2, lwd=2, ylab="negative log likelihood")
lines(u, unll, col=1)
uhat <- soln$minimum
abline(v=uhat, col=2, lty=2)
```



What are we doing?

Before we complete the example it could help to characterize the inferential procedure mathematically.

- We maximized *something*; it ought to be possible (and possibly helpful) to specify (or more accurately, back out) that criteria.

It all flows from the emulator $\hat{y}^M$ we fit earlier, and little notation …

- Let $\hat{Y}^M_{N_F}|u = \hat{y}^M(X^F_{N_F}, u)$ denote a vector of N_F emulated output y-values at the X_{N_F} locations obtained under a setting u of the calibration parameter(s).
- Then, let the computer model *residual* $\hat{Y}^{B|u}_{N_F} = Y^F_{N_F} - \hat{Y}^M_{N_F}|u$ denote the N_F-vector of fitted discrepancies.

Given these quantities, the *quality* of a particular u can be measured by the implied joint probability density of observing $Y^F_{N_f}$ at the inputs $X^F_{N_F}$,

- under our model for the discrepancies $\hat{Y}^{B|u}_{N_F}$.

Joint probability via GP likelihood

$b(\cdot)$ is our model for $\hat{Y}_{N_F}^{B|u}$, and thus fully specifies that joint probability.

- The best-fitting GP regression $\hat{b}(\cdot)$ trained on data $D_{N_F}^B(u) = (X_{N_F}^F, \hat{Y}_{N_F}^{B|u})$, viewed as a function of u, defines a likelihood for u.

Values of u which lead to a higher likelihood, i.e., a higher probability of observing $Y_{N_F}^F$ via those discrepancies, are preferred.

So in symbols we have following optimization problem:

$$\hat{u} = \arg\max_u \left\{ p(u) \left[\max_{\theta_b} p_b(\theta_b | D_{N_F}^B(u)) \right] \right\}.$$

- Recall that $p(u)$ is a (possibly uniform) prior for u,
- and $p_b(\theta_b | \cdots)$ denotes the (marginal) likelihood defined by the GP prior for $b(\cdot)$, via hyperparameters θ_b, including lengthscales and nugget.

Getting $\hat{b}$

Lets run back through some of the calculations to get out the estimated bias (`gpi` reference) with the $\hat{u}$ value we found.

- Provide `clean=FALSE` to `bhat`:

```
bhat <- calib(uhat, X, ball$time, ymhat, da, ga, clean=FALSE)
```

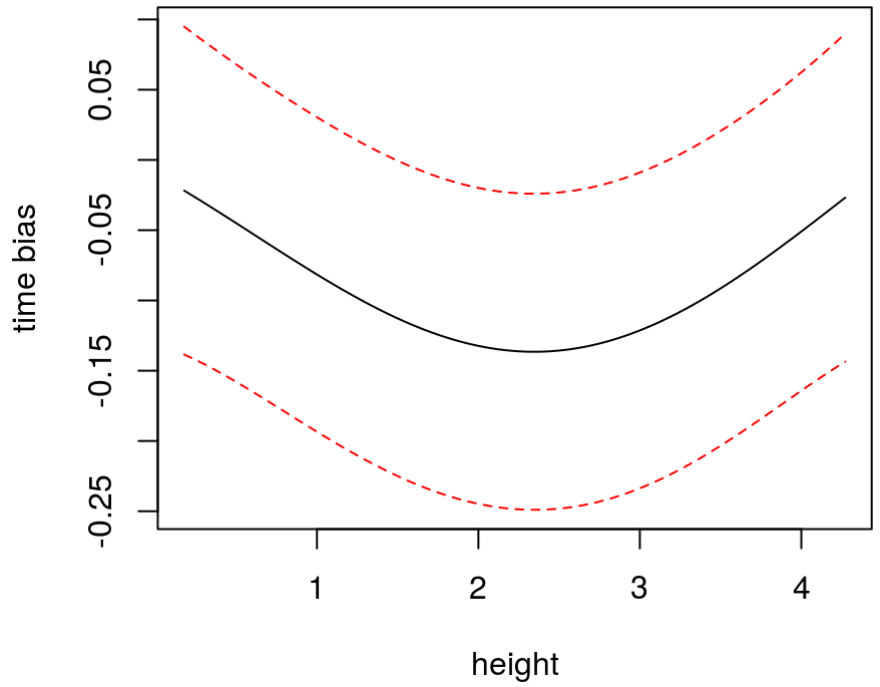
Then we may obtain predictions over our heights grid, with full covariance for later.

```
p <- predGPsep(bhat$gp, as.matrix(hs))
mb <- p$mean
q1b <- qnorm(0.95, mb, sqrt(diag(p$Sigma)))
q2b <- qnorm(0.05, mb, sqrt(diag(p$Sigma)))
```

Visualizing the discrepancy

The bias straddles zero.

```
plot(heights, mb, type="l", xlab="height", ylab="time bias",
  ylim=range(c(q1b, q2b)))
lines(heights, q1b, col=2, lty=2); lines(heights, q2b, col=2, lt=2)
```



Adjusting $\hat{y}^M$

First, obtain the prediction from the emulator, with full covariance structure.

```
pmhat <- predGPsep(ymhat, cbind(hs, uhat))
```

Now, for full propagation of uncertainty, lets combine sample paths from both emulator and bias processes.

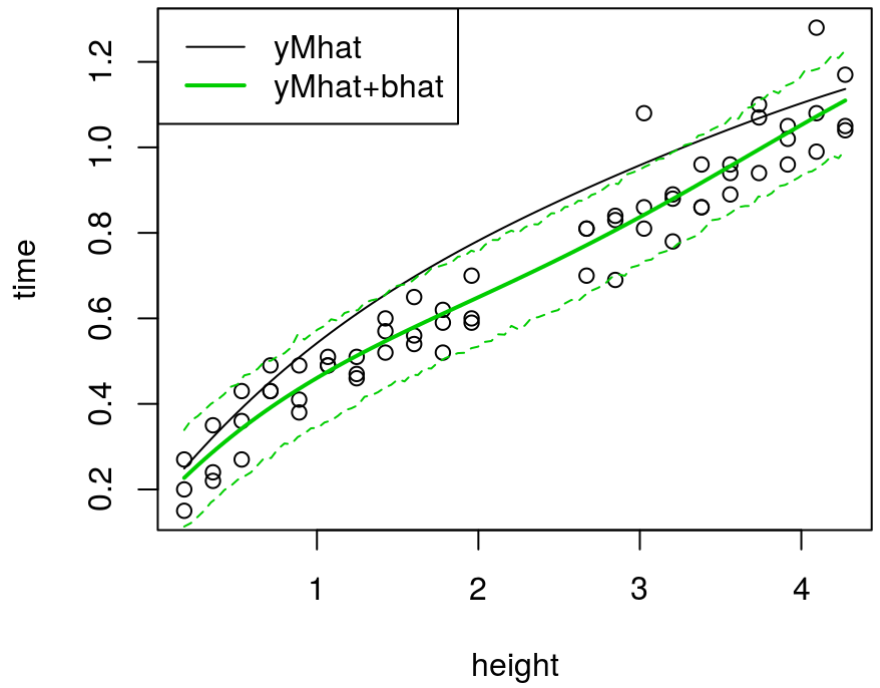
```
library(mvtnorm)
Ym <- rmvnorm(1000, pmhat$mean, pmhat$Sigma)
Yb <- rmvnorm(1000, p$mea, p$Sigma)
Yc <- Ym + Yb
```

Extract quantiles from the combined sample paths.

```
q1c <- apply(Yc, 2, quantile, prob=0.05)
q2c <- apply(Yc, 2, quantile, prob=0.95)
```

Craziness!

```
plot(ball); lines(heights, pmhat$mean)
lines(heights, pmhat$mean + mb, col=3, lwd=2)
lines(heights, q1c, col=3, lty=2)
lines(heights, q2c, col=3, lty=2)
legend("topleft", c("yMhat", "yMhat+bhat"), col=c(1,3), lty=1, lwd=1:2)
```



What's happening here?

The calibration apparatus doesn't care about minimizing bias;

- rather maximizing the *likelihood* of the residual process

$$\hat{Y}^{B|u}_{N_F} = Y^F_{N_F} - \hat{Y}^M_{N_F}|u$$

- via $\hat{b}(\cdot)$ trained at those values.

When it chooses the $\hat{b}$ hyperparameters and $\hat{u}$, via large likelihood,

- it may actually be better to have a larger amplitude bias,
- preferring $\hat{u}$ that push $\hat{y}^M(\cdot, \hat{u})$ away from $y^R(\cdot)$ rather than toward it.

Interpreting calibration parameters

In particular, our estimate of the gravitational constant, $\hat{g}$ via $\hat{u}$

```
ghat <- uhat*diff(gr)+gr[1]
ghat
```

```
## [1] 6.54197
```

loses some of its *physical* interpretation (and its way too small).

We have to be satisfied with g being a "tuning" parameter rather than a primary quantity of interest.

If minimizing bias is really what we want, then some adjustments are needed. See

- Plumlee (2016): forcing the b to be orthogonal to y^M
- Wu & Tuo (2015): using least squares for b rather than a full GP.

Both sacrifice prediction for enhanced interpretation.

A supremely flexible model

The thing to keep in mind is that the calibration apparatus couples two highly flexible nonparametric GP models, linked by a tuning parameter u.

- It will find a way to use that flexibility to its advantage,
- especially when coping with a data-generating mechanism which may not be faithful to the GP modeling assumptions (and when is it ever?).

Authors looking for more flexible GP models have deliberately deployed similar tactics outside the calibration setting.

- Ba and Joseph (2012) coupled two GPs to deal with heteroskedasticity.
- Bornn, Shaddick and Zidek (2012) introduced a latent input dimension (e.g., a u) to gain nonstationary flexibility.

Surprisingly, the KOH framework nests these two options, yet precedes them by more than a decade.

Removing the bias

What happens when we remove some of that flexibility in the calibration context?

- I.e., forcing a zero bias and estimating zero-mean noise σ_ε^2.

Here is how you can accomplish that with `laGP`.

```
se2.fit <- function(X, Y, Ym, clean=TRUE)
  {
    gp <- newGP(X, Y-Ym, d = 0, g = 0)
    cmle <- list(nll=-llikGP(gp))
    if(clean) deleteGP(gp)
    else cmle$gp <- gp
    return(cmle)
  }
```

New no-bias calibration function

We need a slightly adjusted calibration objective.

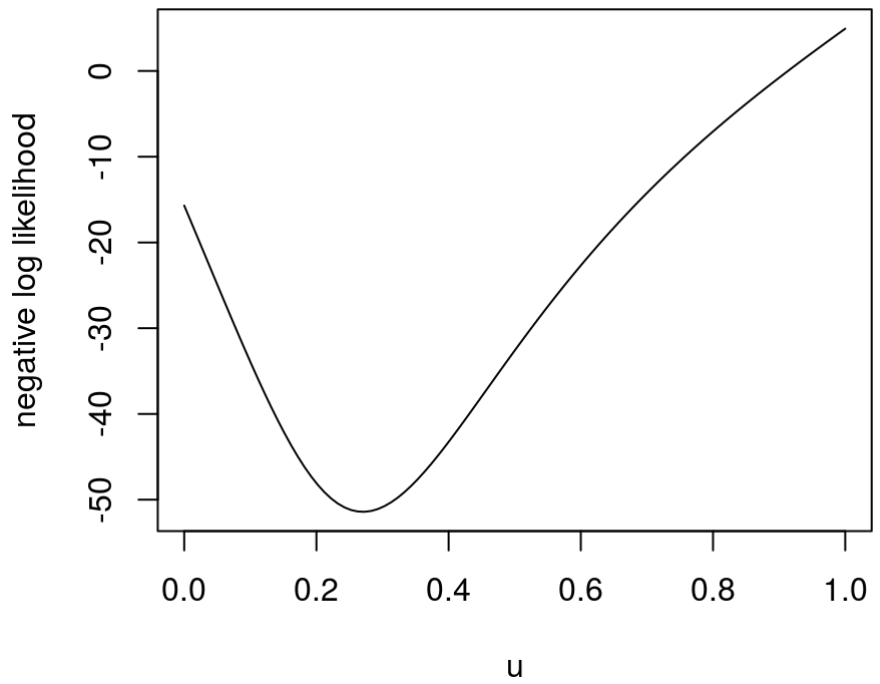
```
calib.nobias <- function(u, X, Y, ymhat, clean=TRUE)
  {
    Xu <- cbind(X, matrix(rep(u, nrow(X)), ncol=length(u), byrow=TRUE))
    Ym <- predGPsep(ymhat, Xu, lite=TRUE)$mean
    cmle <- se2.fit(X, Y, Ym, clean=clean)
    return(cmle)
  }
```

Again, since its in 1d, lets evaluate it on a u-grid.

```
unll.se2 <- rep(NA, length(u))
for(i in 1:length(u))
  unll.se2[i] <- calib.nobias(u[i], X, ball$time, ymhat)$nll
```

Unbiased $\hat{u}$

```
plot(u, unll.se2, type="l", xlab="u", ylab="negative log likelihood")
```



- Much bigger than before.

Our unbiased gravity estimate

```
obj.nobias <- function(x, X, Y, ymhat) calib.nobias(x, X, Y, ymhat)$nll
soln <- optimize(obj.nobias, lower=0, upper=1, X=X, Y=ball$time, ymhat=ymhat)
uhat.nobias <- soln$minimum
ghat.nobias <- uhat.nobias*diff(gr)+gr[1]
ghat.nobias
```

```
## [1] 8.162054
```

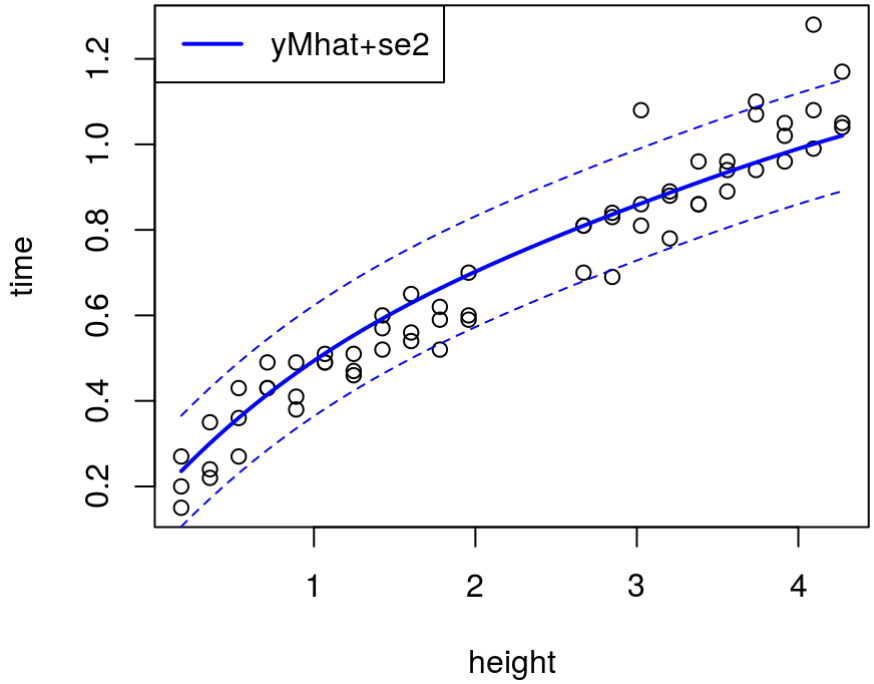
- Still too small, but perhaps it is compensating for some air resistance.

How do our predicted times look?

```
cmle.nobias <- calib.nobias(uhat.nobias, X, ball$time, ymhat, clean=FALSE)
se2.p <- predGP(cmle.nobias$gp, as.matrix(hs), lite=TRUE)
pmhat.nobias <- predGPsep(ymhat, cbind(hs, uhat.nobias), lite=TRUE)
q1nob <- qnorm(0.05, pmhat.nobias$mean, sqrt(pmhat.nobias$s2+se2.p$s2))
q2nob <- qnorm(0.95, pmhat.nobias$mean, sqrt(pmhat.nobias$s2+se2.p$s2))
```

Cleaner, but better? Maybe it under-predicts for higher balls?

```
plot(ball); lines(heights, pmhat.nobias$mean, col=4, lwd=2)
lines(heights, q1nob, col=4, lty=2)
lines(heights, q2nob, col=4, lty=2)
legend("topleft", c("yMhat+se2"), col=4, lty=1, lwd=2)
```



Cross-validation

When we have two models and we don't know which is best,

- set up a prediction exercise.

In what follows we collect some of the code above into stand-alone functions that can be called in a leave-one-out fashion,

- or in some other scheme.

Note that throughout we are conditioning on the computer model fit to the *full* LHS sample.

- The CV will be over the field data only.

Bias-calibrated prediction

Cutting-and-pasting from earlier code.

```
calib.pred <- function(XX, X, Y, ymhat, da, ga, T=1000)
 {
  g <- unll <- u <- seq(0,1, length(100))
  for(i in 1:length(u)) {
    cmle <- calib(u[i], X, Y, ymhat, da, ga)
    unll[i] <- cmle$nll; g[i] <- cmle$g
  }
  ga$mle <- FALSE; ga$start <- g[which.min(unll)]
  soln <- optimize(obj, lower=0, upper=1, X=X, Y=Y, ymhat=ymhat, da=da, ga=ga)
  bhat <- calib(soln$minimum, X, Y, ymhat, da, ga, clean=FALSE)
  p <- predGPsep(bhat$gp, XX)
  pmhat <- predGPsep(ymhat, cbind(XX, soln$minimum))
  Yc <- rmvnorm(T, pmhat$mean, pmhat$Sigma) + rmvnorm(T, p$mean, p$Sigma)
  mc <- pmhat$mean + p$mean; s2c <- apply(Yc, 2, var)
  q1c <- apply(Yc, 2, quantile, prob=0.05)
  q2c <- apply(Yc, 2, quantile, prob=0.95)
  deleteGPsep(bhat$gp)
  return(list(mean=mc, s2=s2c, q1=q1c, q2=q2c, uhat=soln$minimum))
 }
```

Leave-one-out (biased)

```
ga <- garg(list(mle=TRUE), ball$time)
uhat <- q1 <- q2 <- m <- s2 <- rep(NA, nrow(X))
for(i in 1:nrow(X)) {
    cp <- calib.pred(X[i,,drop=FALSE], X[-i,,drop=FALSE], ball$time[-i],
        ymhat, da, ga)
    m[i] <- cp$mean; s2[i] <- cp$s2
    q1[i] <- cp$q1; q2[i] <- cp$q2
    uhat[i] <- cp$uhat
}
```

What $\hat{u}$ values did we get?

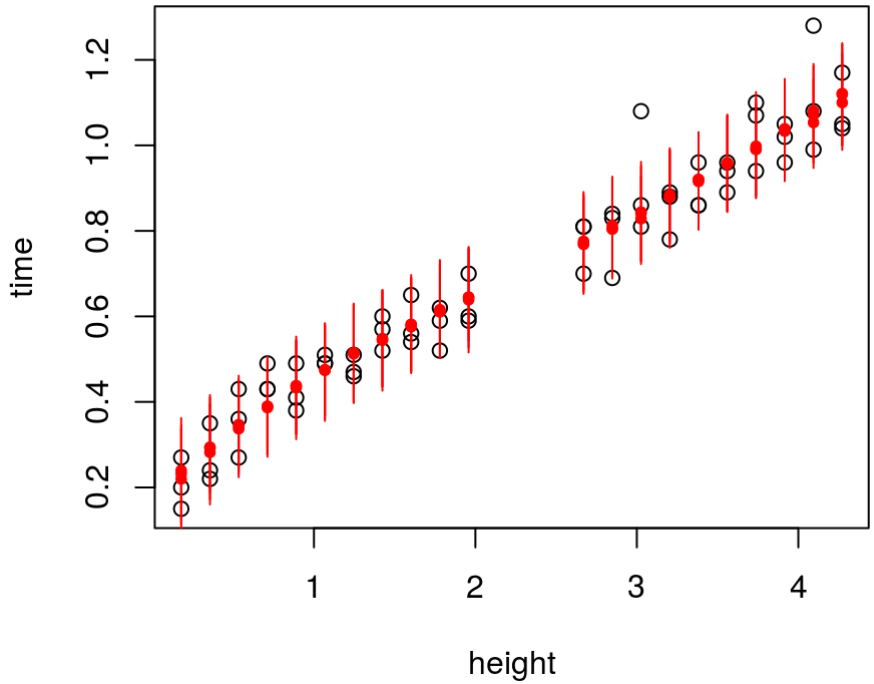
```
summary(uhat)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.01137 0.05899 0.06666 0.06833 0.07721 0.14232
```

- Some are really small.
- This is a **jackknife** sampling distribution; a precursor to the **bootstrap**.

How did we do?

```
plot(ball)
points(ball$height, m, col=2, pch=20)
segments(ball$height, q1, ball$height, q2, col=2)
```



Leave-one-out (un-biased)

Again, cutting-and-pasting from earlier code.

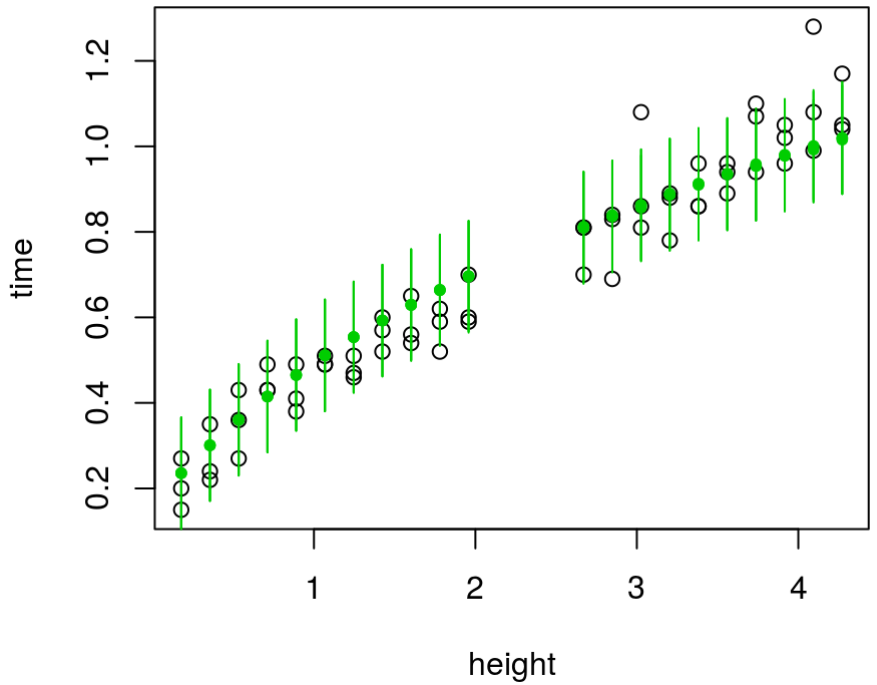
```
calib.nobias.pred <- function(XX, X, Y, ymhat)
 {
    soln <- optimize(obj.nobias, lower=0, upper=1, X=X, Y=Y, ymhat=ymhat)
    bhat <- calib.nobias(soln$minimum, X, Y, ymhat, clean=FALSE)
    p <- predGP(bhat$gp, XX, lite=TRUE)
    pmhat <- predGPsep(ymhat, cbind(XX, soln$minimum), lite=TRUE)
    mc <- pmhat$mean + p$mean; s2c <- pmhat$s2 + p$s2
    q1c <- qnorm(0.05, mc, sqrt(s2c)); q2c <- qnorm(0.95, mc, sqrt(s2c))
  deleteGP(bhat$gp)
    return(list(mean=mc, s2=s2c, q1=q1c, q2=q2c, uhat=soln$minimum))
 }
```

Then leave-one-out prediction.

```
q1nb <- q2nb <- mnb <- s2nb <- rep(NA, nrow(X))
for(i in 1:nrow(X)) {
    cp <- calib.nobias.pred(X[i,,drop=FALSE], X[-i,,drop=FALSE],
        ball$time[-i], ymhat)
    mnb[i] <- cp$mean; s2nb[i] <- cp$s2; q1nb[i] <- cp$q1; q2nb[i] <- cp$q2
}
```

Doesn't look as good

```
plot(ball)
points(ball$height, mnb, col=3, pch=20)
segments(ball$height, q1nb, ball$height, q2nb, col=3)
```



Calculating scores

Comparison by proper scoring (Gneiting & Raftery, 2007; Eq (27)):

```
b <- mean(- (ball$time - m)^2/s2 - log(s2))
nb <- mean(- (ball$time - mnb)^2/s2nb - log(s2nb))
scores <- c(biased=b, unbiased=nb)
scores
```

```
##   biased unbiased
## 4.194271 3.963874
```

- Higher is better: biased wins!

Don't forget that the computer experiment design was random (LHS),

- So these results have a distribution which we can explore via Monte Carlo.