Surrogates 7020

Chapter 7: Bayesian Optimization

Dr. Alex Bledar Konomi

Department of Mathematical Sciences
University of Cincinnati

Model Based Design

- In this chapter, the goal is to demonstrate how Gaussian process (GP) surrogate modeling can assist in optimizing a black box objective function.
- A function about which one knows little – one opaque to the optimizer – and that can only be probed through expensive evaluation.
- We view optimization as an example of sequential design or active learning (6.2).
- Recently Bayesian Optimization (BO) has caught on, especially in the machine learning (ML) community, and likely that'll stick in part because it's punchier than the alternatives.
- BO terminology refers primarily to decision criteria for sequential selection and model updating.
- Modern ML vernacular prefers acquisition functions and GP learning.

Statistics and non-statistics optimization

- Models deployed to assist in optimization can be both statistical and non-statistical, however the latter often have strikingly similar statistical analogs.
- Potential for modern nonparametric statistical surrogate modeling in this context is just recently being recognized by communities for which optimization is bread-and-butter: mathematical programming, statistics, ML, and more. Optimization has played a vital role in stats and ML for decades.
- All those "L-BFGS-B" searches (Byrd et al. 1995) from optim in earlier chapters, say to find MLEs or optimal designs, are cases in point.
- It's intriguing to wonder whether statistical thinking might have something to give back to the optimization world, as it were.

Bayesian Optimization (BO)

- The methodology is simple: train a GP on function evaluations obtained so far; minimize the fitted surrogate predictive mean surface of the GP to select the next location for evaluation; repeat.
- This is an instance of Algorithm 6.1 in 6.2 where the criterion $J(\boldsymbol{x})$ in Step 3 is based on GP predictive mean $\mu_n(\boldsymbol{x}) = E(\boldsymbol{Y}(x)|\boldsymbol{D}_n)$ provided in Eq. (5.2).
- Although Step 3 deploys its own inner-optimization, minimizing $\mu_n(\boldsymbol{x})$ is comparatively easy since it doesn't involve evaluating a computationally intensive, and potentially noisy, blackbox.
- I shall refer to this "mean criterion" as the EY heuristic for surrogate-assisted (Bayesian) optimization (BO).

Bayesian Optimization (BO)

Before we continue, let's be clear about the problem. Whereas the RSM literature (Chapter 3) orients toward *maxima*, BO and math programming favor minimization, which I shall adopt for our discussions here. Specifically, we wish to find

$$\boldsymbol{x}^* = \arg\min_{x \in \mathcal{X}} f(\boldsymbol{x})$$

where $\boldsymbol{X}$ is usually a hyperrectangle, a bounding box, or another simply constrained region. We don't have access to derivative evaluations for $f(\boldsymbol{x})$, nor do we necessarily want them (or want to approximate them) because that could represent additional substantial computational expense. As such, methods described here fall under the class of derivative-free optimization.

Bayesian Optimization (BO)

- All we get to do is evaluate $f(\boldsymbol{x})$, which for now is presumed to be deterministic.
- Generalizations will come after introducing main concepts, including simple extensions for the noisy case.
- The literature targets scenarios where $f(\boldsymbol{x})$ is expensive to evaluate (in terms of computing time, say), but otherwise is well-behaved: continuous, relatively smooth, only real-valued inputs, etc.
- Again, these are relaxable modulo suitable surrogate and/or kernel structure. Several appropriate choices are introduced in later chapters.

best objective value (BOV)

- In empirical comparisons we typically track the *best objective value* (BOV) found as a function of the number of blackbox evaluations.
- In many applied contexts it's more common to have a fixed evaluation budget than it is to enjoy the luxury of running to convergence.
- Nevertheless, monitoring progress plays a key role when deciding if further expensive evaluations may be required.

A running example

```
f <- function(X)
 {
  if(is.null(nrow(X))) X <- matrix(X, nrow=1)
  m <- 8.6928
  s <- 2.4269
  x1 <- 4*X[,1] - 2
  x2 <- 4*X[,2] - 2
  a <- 1 + (x1 + x2 + 1)^2 *
    (19 - 14*x1 + 3*x1^2 - 14*x2 + 6*x1*x2 + 3*x2^2)
  b <- 30 + (2*x1 - 3*x2)^2 *
    (18 - 32*x1 + 12*x1^2 + 48*x2 - 36*x1*x2 + 27*x2^2)
  f <- log(a*b)
  f <- (f - m)/s
  return(f)
 }
```

A running example

Begin with a small space-filling Latin hypercube sample (LHS; 4.1) seed design in 2d.

```
library(lhs)
ninit <- 12
X <- randomLHS(ninit, 2)
y <- f(X)
```

Next fit a separable GP to those data, with a small nugget for jitter.

```
library(laGP)
da <- darg(list(mle=TRUE, max=0.5), randomLHS(1000, 2))
gpi <- newGPsep(X, y, d=da$start, g=1e-6, dK=TRUE)
mleGPsep(gpi, param="d", tmin=da$min, tmax=da$max, ab=da$ab)$msg
```

A running example

Now the predictive mean surface (like f, through the evaluations it's trained on) may have many local minima, but let's punt for now on the ideal of global optimization of EY – of the so-called "inner loop" – and see where we get with a search initialized at the current best value.

```
obj.mean <- function(x, gpi)
  predGPsep(gpi, matrix(x, nrow=1), lite=TRUE)$mean

m <- which.min(y)
opt <- optim(X[m,], obj.mean, lower=0, upper=1, method="L-BFGS-B", gpi=gpi)
opt$par
```

```
plot(X[1:ninit,], xlab="x1", ylab="x2", xlim=c(0,1), ylim=c(0,1))
arrows(X[m,1], X[m,2], opt$par[1], opt$par[2], length=0.1)
```

A running example

Code below wraps what we've been doing above into a while loop with a simple check on convergence in order to "break out". If two outputs in a row are sufficiently close, within a tolerance $1e-4$, then stop. That's quite crude, but sufficient for illustrative purposes.

```
while(1) {
  m <- which.min(y)
  opt <- optim(X[m,], obj.mean, lower=0, upper=1,
    method="L-BFGS-B", gpi=gpi)
  ynew <- f(opt$par)
  if(abs(ynew - y[length(y)]) < 1e-4) break
  updateGPsep(gpi, matrix(opt$par, nrow=1), ynew)
  mle <- mleGPsep(gpi, param="d", tmin=da$min, tmax=da$max, ab=da$ab)
  X <- rbind(X, opt$par)
  y <- c(y, ynew)
}
deleteGPsep(gpi)
```

bov: best objective value

```
bov <- function(y, end=length(y))
 {
  prog <- rep(min(y), end)
  prog[1:min(end, length(y))] <- y[1:min(end, length(y))]
  for(i in 2:end)
    if(is.na(prog[i]) || prog[i] > prog[i-1]) prog[i] <- prog[i-1]
  return(prog)
 }

######################

plot(prog, type="l", col="gray", xlab="n: blackbox evaluations",
  ylab="best objective value")
abline(v=ninit, lty=2)
legend("topright", "seed LHS", lty=2, bty="n")
```

Comparison to optim function in R

How about our favorite optimization library: optim using "L-BFGS-B"? Working *optim* in as a comparator requires a slight tweak.

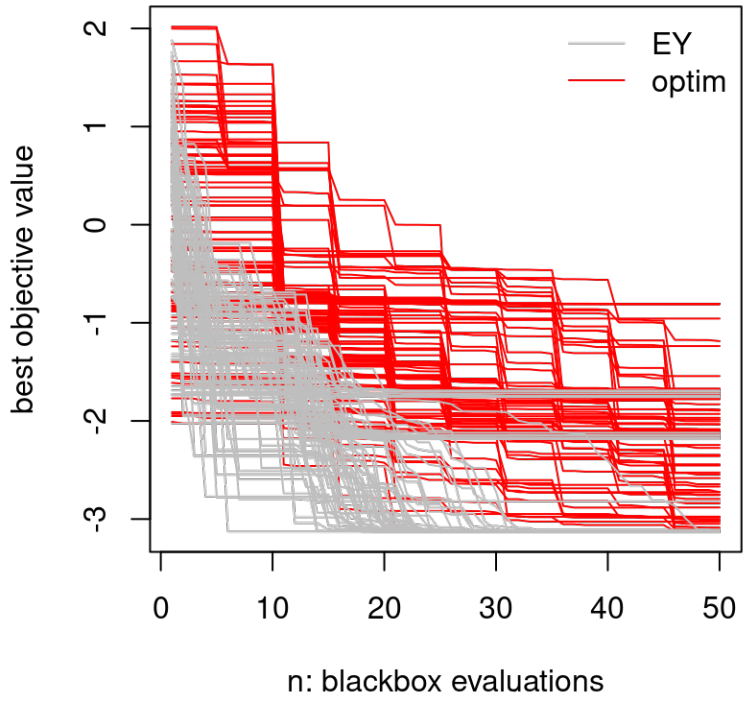


Figure: Multiple BOV progress paths under random reinitialization using BO and "L-BFGS-B" (comparator under random restarts).

Construct an optimization function

```
optim.surr <- function(f, m, ninit, end, tol=1e-4)
 {
  ## initialization
  X <- randomLHS(ninit, m)
  y <- f(X)
  da <- darg(list(mle=TRUE, max=0.5), randomLHS(1000, m))
  gpi <- newGPsep(X, y, d=da$start, g=1e-6, dK=TRUE)
  mleGPsep(gpi, param="d", tmin=da$min, tmax=da$max, ab=da$ab)

  ## optimization loop
  for(i in (ninit+1):end) {
    m <- which.min(y)
    opt <- optim(X[m,], obj.mean, lower=0, upper=1,
      method="L-BFGS-B", gpi=gpi)
    ynew <- f(opt$par)
    if(abs(ynew - y[length(y)]) < tol) break
    updateGPsep(gpi, matrix(opt$par, nrow=1), ynew)
    mleGPsep(gpi, param="d", tmin=da$min, tmax=da$max, ab=da$ab)
    X <- rbind(X, opt$par)
    y <- c(y, ynew)
  }

  ## clean up and return
  deleteGPsep(gpi)
```

Simulation Study

```
reps <- 100
end <- 50
prog <- matrix(NA, nrow=reps, ncol=end)
for(r in 1:reps) {
  os <- optim.surr(f, 2, ninit, end)
  prog[r,] <- bov(os$y, end)
}
#######
matplot(t(prog), type="l", col="gray", lty=1,
  xlab="n: blackbox evaluations",  ylab="best objective value")
abline(v=ninit, lty=2)
legend("topright", "seed LHS", lty=2, bty="n")
```

Comparison to "L-BFGS-B": Simulation Study

```
fprime <- function(x)
 {
  ynew <- f(x)
  y <<- c(y, ynew)
  return(ynew)
}
########################
prog.optim <- matrix(NA, nrow=reps, ncol=end)
for(r in 1:reps) {
  y <- c()
  os <- optim(runif(2), fprime, lower=0, upper=1, method="L-BFGS-B")
  prog.optim[r,] <- bov(y, end)
}

########################
matplot(t(prog.optim), type="l", col="red", lty=1,
  xlab="n: blackbox evaluations", ylab="best objective value")
matlines(t(prog), type="l", col="gray", lty=1)
legend("topright", c("EY", "optim"), col=c("gray", "red"), lty=1, bty="n")
```

Something to think

Notice that we're not actually doing statistics, because at no point is uncertainty being taken into account. Only predictive means are used; predictive variance doesn't factor in. One way to incorporate uncertainty is through the lower confidence bound (LCB) heuristic (Srinivas et al. 2009). LCB is a simple linear combination between mean and standard deviation:

$$\alpha_{LCB}(\boldsymbol{x}) = -\mu_n(\boldsymbol{x}) + \beta_n \sigma_n(\boldsymbol{x}),$$

and

$$\boldsymbol{x}_{n+1} = \underset{\boldsymbol{x}}{\operatorname{argmax}}\, \alpha_{LCB}(\boldsymbol{x})$$

LCB introduces a sequence of tuning parameters β_n , targeting a balance between exploration and exploitation as iterations of optimization progress. Larger β_n lead to more conservative searches, until in the limit $\mu_n(\boldsymbol{x})$ is ignored and acquisitions reduce to ALM (6.2.1).

Defining Improvement

- In the mid 1990s, Matthias Schonlau (1997) was working on his dissertation, which basically revisited Mockus' Bayesian optimization idea from a GP and computer experiments perspective.
- He came up with a heuristic called expected improvement (EI), which is the basis of the so-called efficient global optimization (EGO) algorithm.
- This distinction is subtle: one is the sequential design criterion (EI), and the other is its repeated application toward minimizing a blackbox function (EGO). In the literature, you'll see the overall method referred to by both names/acronyms.
- Schonlau's key insight was that predictive uncertainty is underutilized by surrogate frameworks for optimization

Expected Improvement

Schonlau defined potential for improvement over fnmin at an input location $\boldsymbol{x}$ as

$$I(\boldsymbol{x}) = \max 0, f^n_{min} - Y(\boldsymbol{x}).$$

$I(x)$ **is a random variable**. It measures the amount by which a response $Y(\boldsymbol{x})$ could be below the BOV obtained so far. **Here** $Y(\boldsymbol{x})$ **is shorthand for** $Y(\boldsymbol{x})|\boldsymbol{D}_n$, the predictive distribution obtained from a fitted model. If $Y(\boldsymbol{x})|\boldsymbol{D}_n$ has nonzero probability of taking on any value on the real line, as it does under a Gaussian predictive distribution, then $I(\boldsymbol{x})$ has nonzero probability of being positive for any $\boldsymbol{x}$.

Probability of Improvement (PI)

- One option is to convert it into a probability. The probability of improvement (PI) criterion is $PI(\boldsymbol{x}) = P(I(\boldsymbol{x}) > 0 | \boldsymbol{D}_n)$, which is equivalent to $P(Y(\boldsymbol{x}) < f^n_{min} | \boldsymbol{D}_n)$.
- Maximizing PI is sensible, but could result in very small steps.
- The most probable input $\boldsymbol{x}^* = \max_{\boldsymbol{x}} PI(\boldsymbol{x})$ may not hold the greatest potential for large improvement, which is important when considering the tacit goal of minimizing the number of expensive blackbox evaluations.
- Instead, **maximizing expected improvement (EI), $EI(\boldsymbol{x}) = EI(\boldsymbol{x}) | \boldsymbol{D}_n$, more squarely targets potential for large improvement.**

Calculate PI and/or EI via MC

The easiest way to calculate PI or EI, where "easy" means agnostic to the form of $Y(\boldsymbol{x})|\boldsymbol{D}_n$, is through MC approximation. Draw $y^{(t)} \sim Y(\boldsymbol{x})|\boldsymbol{D}_n$, for $t = 1, \ldots, T$, from their posterior predictive distribution, and average

$$PI(\boldsymbol{x}) \approx \frac{1}{T} \sum_{t=1}^{T} I_{y^{(t)}>0}$$

or

$$EI(\boldsymbol{x}) \approx \frac{1}{T} \sum_{t=1}^{T} \max\{0, f_{min}^n - y^{(t)}\}.$$

In the limit as $T \longrightarrow \infty$ these approximations become exact. This approach works no matter what the distribution of $Y(\boldsymbol{x})$ is, so long as you can simulate from it. With fully Bayesian response surface methods leveraging Markov chain Monte Carlo (MCMC) posterior sampling, say, such approximation may represent the only viable option.

Calculate PI in GP

However if $Y(\boldsymbol{x})|\boldsymbol{D}_n$ is Gaussian, as it's under the predictive equations of a GP surrogate conditional on a particular set of hyperparameters, both have a convenient closed form. PI involves a standard Gaussian CDF (Φ) evaluation, as readily calculated with built-in functions in R.

$$PI(\boldsymbol{x}) = \Phi(\frac{f^n_{min} - \mu_n(\boldsymbol{x})}{\sigma_n(\boldsymbol{x})})$$

Transparent in the formula above is that both predictive mean and uncertainty factor into the calculation.

Calculate EI in GP

Deriving EI takes a little more work, but nothing a student of calculus couldn't do using substitution and integration by parts. Details are in an appendix of Schonlau's thesis. We will simply quote the final result here.

$$EI(\boldsymbol{x}) = (f^n_{min} - \mu_n(\boldsymbol{x}))\Phi(\frac{f^n_{min} - \mu_n(\boldsymbol{x})}{\sigma_n(\boldsymbol{x})}) + \sigma_n(\boldsymbol{x})\phi(\frac{f^n_{min} - \mu_n(\boldsymbol{x})}{\sigma_n(\boldsymbol{x})}), \tag{1}$$

where ϕ is the standard Gaussian PDF. Notice how EI contains PI as a component in a larger expression. "One half" of EI is PI multiplied (or weighted) by the amount by which the predictive mean is below f^n_{min}.

EI conversation

Deriving EI takes a little more work, but nothing a student of calculus couldn't do using substitution and integration by parts. Details are in an appendix of Schonlau's thesis. We will simply quote the final result here.

$$EI(\boldsymbol{x}) = (f^n_{min} - \mu_n(\boldsymbol{x}))\Phi(\frac{f^n_{min} - \mu_n(\boldsymbol{x})}{\sigma_n(\boldsymbol{x})}) + \sigma_n(\boldsymbol{x})\phi(\frac{f^n_{min} - \mu_n(\boldsymbol{x})}{\sigma_n(\boldsymbol{x})}), \tag{2}$$

where ϕ is the standard Gaussian PDF. Notice how EI contains PI as a component in a larger expression. "One half" of EI is PI multiplied (or weighted) by the amount by which the predictive mean is below f^n_{min}.

Illustration of EI

As a first illustration of EI, code chunks below recreate an example and visuals presented in the first published/journal manuscript describing EI/EGO (Jones, Schonlau, and Welch 1998).

```
x <- c(1, 2, 3, 4, 12)
y <- c(0, -1.75, -2, -0.5, 5)
######################
gpi <- newGP(matrix(x, ncol=1), y, d=10, g=1e-8)
xx <- seq(0, 13, length=1000)
p <- predGP(gpi, matrix(xx, ncol=1), lite=TRUE)
####################
m <- which.min(y)
fmin <- y[m]
d <- fmin - p$mean
s <- sqrt(p$s2)
dn <- d/s
ei <- d*pnorm(dn) + s*dnorm(dn)
```

Illustration of EI

```
par(mfrow=c(1,2))
plot(x, y, pch=19, xlim=c(0,13), ylim=c(-4,9), main="predictive surface")
lines(xx, p$mean)
lines(xx, p$mean + 2*sqrt(p$s2), col=2, lty=2)
lines(xx, p$mean - 2*sqrt(p$s2), col=2, lty=2)
abline(h=fmin, col=3, lty=3)
legend("topleft", c("mean", "95% PI", "fmin"), lty=1:3,
  col=1:3, bty="n")
plot(xx, ei, type="l", col="blue", main="EI", xlab="x", ylim=c(0,0.15))
```

Illustration of EI

```
mm <- which.max(ei)
x <- c(x, xx[mm])
y <- c(y, p$mean[mm])
updateGP(gpi, matrix(xx[mm], ncol=1), p$mean[mm])
p <- predGP(gpi, matrix(xx, ncol=1), lite=TRUE)
deleteGP(gpi)
m <- which.min(y)
fmin <- y[m]
d <- fmin - p$mean
s <- sqrt(p$s2)
dn <- d/s
ei <- d*pnorm(dn) + s*dnorm(dn)
########################
par(mfrow=c(1,2))
plot(x, y, pch=19, xlim=c(0,13), ylim=c(-4,9), main="predictive surface")
lines(xx, p$mean)
lines(xx, p$mean + 2*sqrt(p$s2), col=2, lty=2)
lines(xx, p$mean - 2*sqrt(p$s2), col=2, lty=2)
abline(h=fmin, col=3, lty=3)
legend("topleft", c("mean", "95% PI", "fmin"), lty=1:3,
  col=1:3, bty="n")
plot(xx, ei, type="l", col="blue", main="EI", xlab="x", ylim=c(0,0.15))
```

EI as an objective function

```
EI <- function(gpi, x, fmin, pred=predGPsep)
 {
  if(is.null(nrow(x))) x <- matrix(x, nrow=1)
  p <- pred(gpi, x, lite=TRUE)
  d <- fmin - p$mean
  sigma <- sqrt(p$s2)
  dn <- d/sigma
  ei <- d*pnorm(dn) + sigma*dnorm(dn)
  return(ei)
 }

obj.EI <- function(x, fmin, gpi, pred=predGPsep)
  - EI(gpi, x, fmin, pred)
```

EI as an objective function

```
EI <- function(gpi, x, fmin, pred=predGPsep)
 {
  if(is.null(nrow(x))) x <- matrix(x, nrow=1)
  p <- pred(gpi, x, lite=TRUE)
  d <- fmin - p$mean
  sigma <- sqrt(p$s2)
  dn <- d/sigma
  ei <- d*pnorm(dn) + sigma*dnorm(dn)
  return(ei)
 }

obj.EI <- function(x, fmin, gpi, pred=predGPsep)
  - EI(gpi, x, fmin, pred)
```

EI search function

```
eps <- sqrt(.Machine$double.eps) ## used lots below
EI.search <- function(X, y, gpi, pred=predGPsep, multi.start=5, tol=eps)
 {
  m <- which.min(y)
  fmin <- y[m]
  start <- matrix(X[m,], nrow=1)
  if(multi.start > 1)
    start <- rbind(start, randomLHS(multi.start - 1, ncol(X)))
  xnew <- matrix(NA, nrow=nrow(start), ncol=ncol(X)+1)
  for(i in 1:nrow(start)) {
    if(EI(gpi, start[i,], fmin) <= tol) { out <- list(value=-Inf); next }
    out <- optim(start[i,], obj.EI, method="L-BFGS-B",
      lower=0, upper=1, gpi=gpi, pred=pred, fmin=fmin)
    xnew[i,] <- c(out$par, -out$value)
  }
  solns <- data.frame(cbind(start, xnew))
  names(solns) <- c("s1", "s2", "x1", "x2", "val")
  solns <- solns[solns$val > tol,]
  return(solns)
}
```

EI search function

```
X <- randomLHS(ninit, 2)
y <- f(X)
gpi <- newGPsep(X, y, d=0.1, g=1e-6, dK=TRUE)
da <- darg(list(mle=TRUE, max=0.5), randomLHS(1000, 2))
####################
solns <- EI.search(X, y, gpi)
m <- which.max(solns$val)
maxei <- solns$val[m]
####################
plot(X, xlab="x1", ylab="x2", xlim=c(0,1), ylim=c(0,1))
arrows(solns$s1, solns$s2, solns$x1, solns$x2, length=0.1)
points(solns$x1[m], solns$x2[m], col=2, pch=20)
```

Next surch

```
xnew <- as.matrix(solns[m,3:4])
X <- rbind(X, xnew)
y <- c(y, f(xnew))
updateGPsep(gpi, xnew, y[length(y)])
mle <- mleGPsep(gpi, param="d", tmin=da$min, tmax=da$max, ab=da$ab)
###################
solns <- EI.search(X, y, gpi)
m <- which.max(solns$val)
maxei <- c(maxei, solns$val[m])
xnew <- as.matrix(solns[m,3:4])
X <- rbind(X, xnew)
y <- c(y, f(xnew))
updateGPsep(gpi, xnew, y[length(y)])
mle <- mleGPsep(gpi, param="d", tmin=da$min, tmax=da$max, ab=da$ab)
###################
plot(X, xlab="x1", ylab="x2", xlim=c(0,1), ylim=c(0,1))
arrows(solns$s1, solns$s2, solns$x1, solns$x2, length=0.1)
points(solns$x1[m], solns$x2[m], col=2, pch=20)
```

EI search function

```
xnew <- as.matrix(solns[m,3:4])
X <- rbind(X, xnew)
y <- c(y, f(xnew))
updateGPsep(gpi, xnew, y[length(y)])
mle <- mleGPsep(gpi, param="d", tmin=da$min, tmax=da$max, ab=da$ab)

for(i in nrow(X):end) {
  solns <- EI.search(X, y, gpi)
  m <- which.max(solns$val)
  maxei <- c(maxei, solns$val[m])
  xnew <- as.matrix(solns[m,3:4])
  ynew <- f(xnew)
  X <- rbind(X, xnew)
  y <- c(y, ynew)
  updateGPsep(gpi, xnew, y[length(y)])
  mle <- mleGPsep(gpi, param="d", tmin=da$min, tmax=da$max, ab=da$ab)
}
deleteGPsep(gpi)
```

Compute BOV of EI search

```
prog.ei <- bov(y)
par(mfrow=c(1,2))
plot(prog.ei, type="l", xlab="n: blackbox evaluations",
  ylab="EI best observed value")
abline(v=ninit, lty=2)
legend("topright", "seed LHS", lty=2)
plot(ninit:end, maxei, type="l",  xlim=c(1,end),
  xlab="n: blackbox evaluations", ylab="max EI")
abline(v=ninit, lty=2)
```

EI progress.

EI progress in terms of BOV (left) and maximal EI used for acquisition (right).

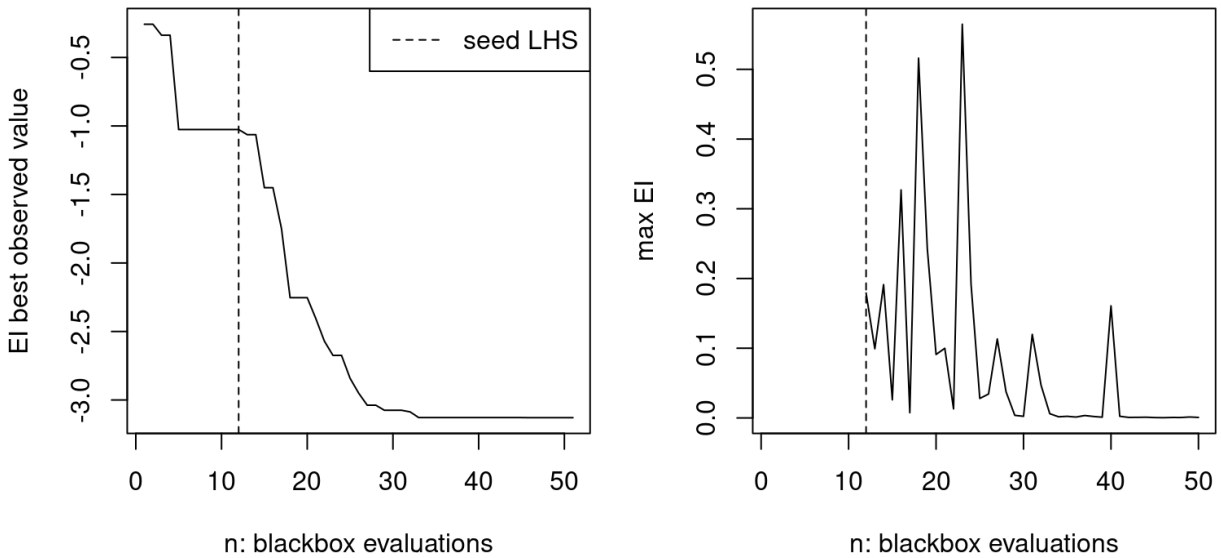


Figure: EI progress in terms of BOV (left) and maximal EI used for acquisition (right).

Compute BOV of EI search

```
optim.EI <- function(f, ninit, end)
 {
  ## initialization
  X <- randomLHS(ninit, 2)
  y <- f(X)
  gpi <- newGPsep(X, y, d=0.1, g=1e-6, dK=TRUE)
  da <- darg(list(mle=TRUE, max=0.5), randomLHS(1000, 2))
  mleGPsep(gpi, param="d", tmin=da$min, tmax=da$max, ab=da$ab)

  ## optimization loop of sequential acquisitions
  maxei <- c()
  for(i in (ninit+1):end) {
    solns <- EI.search(X, y, gpi)
    m <- which.max(solns$val)
    maxei <- c(maxei, solns$val[m])
    xnew <- as.matrix(solns[m,3:4])
    ynew <- f(xnew)
    updateGPsep(gpi, xnew, ynew)
    mleGPsep(gpi, param="d", tmin=da$min, tmax=da$max, ab=da$ab)
    X <- rbind(X, xnew)
    y <- c(y, ynew)
  }

  ## clean up and return
```

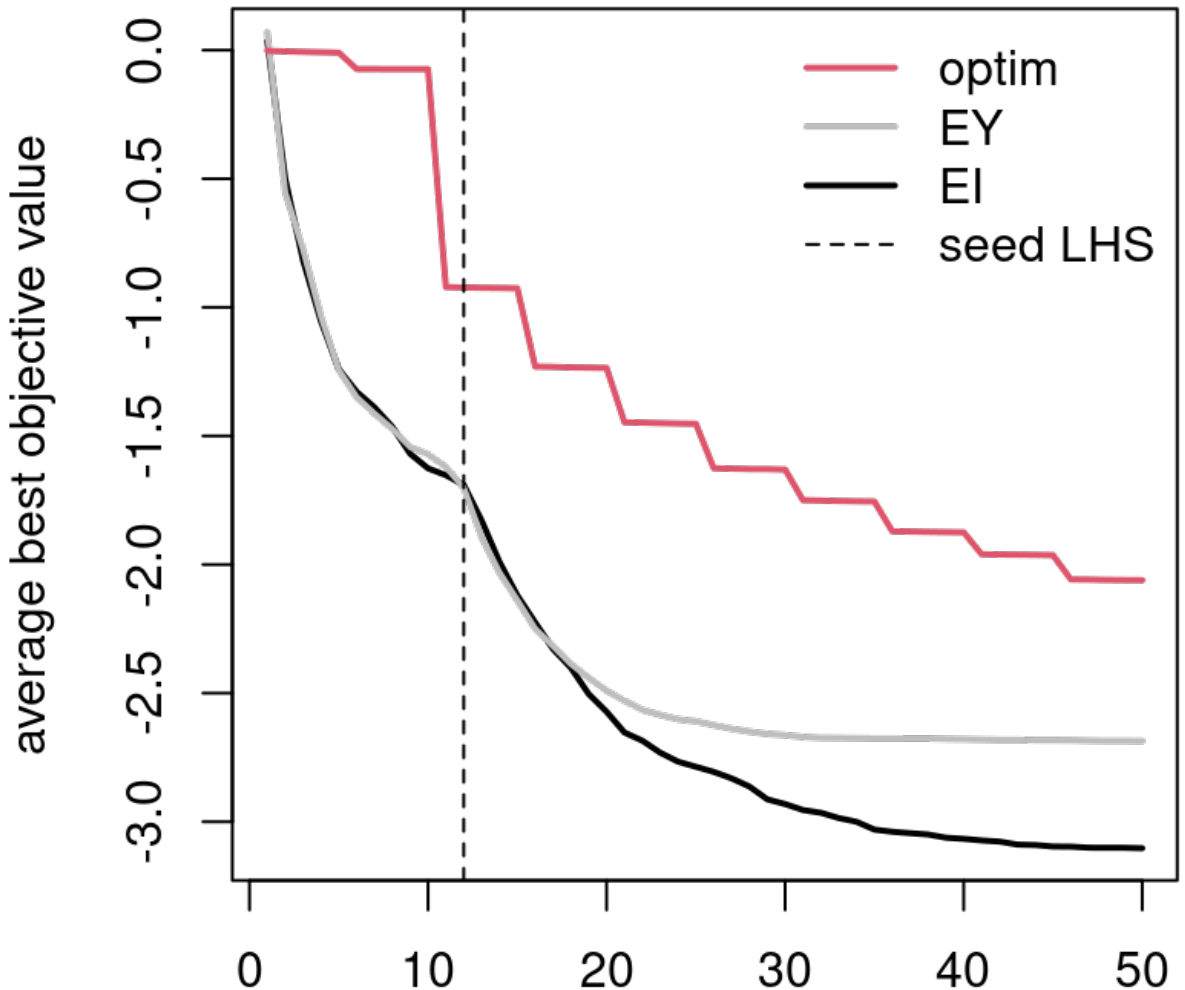
Compute BOV of EI search

```
reps <- 100
prog.ei <- matrix(NA, nrow=reps, ncol=end)
for(r in 1:reps) {
  os <- optim.EI(f, ninit, end)
  prog.ei[r,] <- bov(os$y)
}

plot(colMeans(prog.ei), col=1, lwd=2, type="l",
  xlab="n: blackbox evaluations", ylab="average best objective value")
lines(colMeans(prog), col="gray", lwd=2)
lines(colMeans(prog.optim, na.rm=TRUE), col=2, lwd=2)
abline(v=ninit, lty=2)
legend("topright", c("optim", "EY", "EI", "seed LHS"),
  col=c(2, "gray", 1, 1), lwd=c(2,2,2,1), lty=c(1,1,1,2),
  bty="n")
```

Average BOV progress

Average BOV progress for the three comparators entertained so far.



Boxplots optimum value

Boxplots summarizing the distribution of progress after the final acquisition.