Explanation of Mathematical Methods Behind the First Half of the Project

AUTHOR
Emma Bowen

Introduction

What mathematical methods will I explain in this document?

- Gaussian processes
- Bayesian History Matching
- Kriging
- Expected Improvement
- Augmented Expected Improvement
- Knowledge Gradient

Context

We are building on the paper "Using history matching to speed up management strategy evaluation grid searches". This paper is looking to find the Harvest Control Rule parametrised by F_{target} and $B_{trigger}$ that maximises the catch whilst keeping the risk below 0.05 for a single-stock fishery. The paper does not consider fleet dynamics. We have an objective function in the paper which combines the risk and the catch and so this is the function we want to maximise to get our maximal catch with the constraint of keeping the risk below 0.05.

To do this, the paper takes a Bayesian History Matching (BHM) approach. Firstly, we sample our first eight points which are spaced evenly throughout the sample space to get some initial data. Then, for each round we do the following:

- We set up or update the Gaussian Process (GP) to model the risk and the GP to model the catch
- We can then use the risk as a threshold so that we only consider the values of F_{target} and $B_{trigger}$ that have risk below 0.05
- We use the GP which is modelling the catch to get the value for the catch at every point in the sample space which will have some uncertainty
- We use BHM to remove any points that are implausible (that have a low probability of being higher than the current best catch)

We repeat this process until there is only one plausible point left and then we will accept this as being the F_{target} and $B_{trigger}$ that maximise the catch whilst keeping the risk below 0.05.

Now, we will look at the mathematics behind the methods above and also at the acquisition functions of Expected Improvement, Augmented Expected Improvement and Knowledge Gradient which I added to the original code.

Gaussian Processes

A Gaussian Process F has a mean function μ_0 and a covariance function $\text{cov}_0(x_i, x_j)$. We can then evaluate the covariance function $\text{cov}_0(x_i, x_j)$ for every pair x_i, x_j where $j, i \in \{1, \ldots, k\}$ to find the covariance matrix $\Sigma_{1:k}$. Then, F is a probability distribution over our objective function f with the property that, for any given collection of points $x_1, \ldots x_m$, the marginal probability distribution on $(F(x_1), \ldots, F(x_m))$ is given by(Frazier 2018):

$$(F(x_1),\ldots,F(x_m)) \sim N((\mu_0(x_1),\ldots,\mu(x_m)),\Sigma_{1:k})$$

We choose a covariance function such that inputs that have nearby points that have been evaluated have a more certain output than points that are further away from the points that have been evaluated(Spence 2025). This is equivalent to saying that if for some x, x', x'' in the design space we have $\|x - x'\| < \|x - x''\|$ for some norm $\|\cdot\|$, then $\text{cov}_0(x, x') > \text{cov}_0(x, x'')$(Frazier 2018).

From the original paper, we are using the mean function m_1 for the catch and the mean function m_2 for the risk:

$$m_1(\phi) = \beta_{1,0} + \beta_{1,1}(\ln(\phi_1 + 0.1)) + \beta_{1,2}(\ln(\phi_1 + 0.1))^2 + \beta_{1,3}(\ln(\phi_1 + 0.1))^3 + \beta_{1,4}(\phi_2 \ln(\phi_1 + 0.1)) + \beta_{1,5}\phi_2$$

$$m_2(\phi) = \beta_{2,0} + \beta_{2,1}\phi_1 + \beta_{2,2}\phi_2 + \beta_{2,3}\phi_1\phi_2$$

and our covariance function is the variance σ_i^2 (which is acting as a scalar) times the Ornstein-Uhlenbeck correlation function(Spence 2025):

$$r_i(\phi, \phi', \delta_i) = \exp\left(-\frac{|\phi_1 - \phi_1'|}{\delta_{i,1}} - \frac{|\phi_2 - \phi_2'|}{\delta_{i,2}}\right)$$

We can set up Gaussian Processes (GPs) in R using the DiceKriging package as below:

```
gp_cat <- km(~.^2,design=runs[,c("Ftarget","Btrigger")],estim.method="MLE",
             response = res_cat,nugget=1e-12*var(res_cat),covtype = "exp")

gp_risk <- km(~.^2,design=runs[,c("Ftarget","Btrigger")],estim.method="MLE",
              response = res_risk,nugget=1e-12*var(res_risk),covtype = "exp")
```

where `covtype` tells us the type of covariance function we are using and `estim.method` tells us how to estimate unknown parameters(Roustant 2025). Here, `nugget` is a small value being used to ensure we have an invertible matrix and telling the GP that our objective function is deterministic because it is so small(Roustant 2025). If our objective function had noise, `nugget` would instead be used to add that noise along the diagonal of the covariance matrix to encode that noise into our GP(MacKay et al. 1998).

The `design` variable gives the input parameters of the evaluated points we have so far (Roustant 2025). Here, `estim.method` is set to `"MLE"` which means we are using the maximum likelihood estimate to get the hyperparameters of μ_0 and cov_0 for our GPs. We can do this as follows. Firstly, we let the vector η represent the hyperparameters that give us μ_0 and cov_0. Then, given the observations $f(x_{1:n}) = (f(x_1),\ldots,f(x_n))$, we calculate the likelihood of these observations under the prior given η which is denoted as $p(f(x_{1:n})|\eta)$. This likelihood can be modelled by a multivariate normal(Frazier 2018). Lastly, we set $\hat{\eta}$ to the value that maximizes this likelihood:

$$\hat{\eta} = argmax_\eta p(f(x_{1:n})|\eta)$$

We use a GP to emulate the objective function because it is much cheaper to evaluate compared to our objective function. If we let F and f be as in the Gaussian Processes section, then we can calculate $F(x)$ for any x in the design space as our estimate of $f(x)$ based on our current beliefs. This is true even for the evaluated points $x_1,\ldots,x_n$ as the emulator is fitted to these points(Spence 2025).

In the code, we predict the log(catch) and log(risk) at every point in the design space using the code below:

```
pred_risk_g <- predict(gp_risk,newdata=gridd,type="SK")
pred_cat_g <- predict(gp_cat,newdata=gridd,type="SK")
```

and then we can find the probability that the log(risk) is ≤ 0.05 by using:

```
prisk <- pnorm(log(0.05),pred_risk_g$mean,pred_risk_g$sd+1e-12)
```

where we include `+1e-12` in the above to prevent the variance becoming negative due to imprecision when calculating `pred_risk_g$sd`. This would encounter an error from the pnorm function which would stop the code(Roustant 2025).

We can then exponentiate these results where needed. We are building our GPs with log of the values we want because this helps us generate better predictions(Huang et al. 2006).

We have been able to visualise this process using the code below. Firstly, we can look at the GP for risk after the first round of `case_study8.R`:

```
library(DiceView)
library(rgl)
```

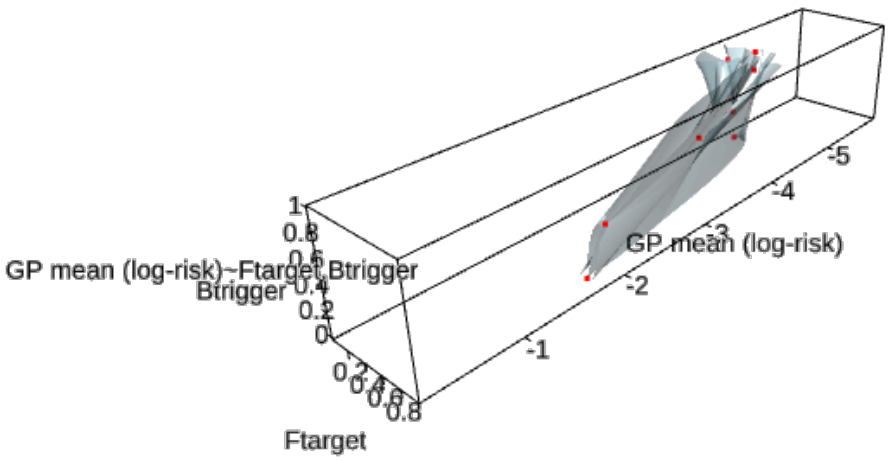
```
This build of rgl does not include OpenGL functions.  Use
 rglwidget() to display results, e.g. via options(rgl.printRglwidget = TRUE).
```

```
Xlim <- rbind(
  apply(gp_risk@X, 2, min),
  apply(gp_risk@X, 2, max)
)


sectionview3d(
  gp_risk,
  dim = 2,
  Xlim = Xlim,
  Xlab = c("Ftarget", "Btrigger"),
  ylab = "GP mean (log-risk)",
  npoints = c(100, 100),
  col = "lightblue",
  scale = TRUE,
  col_points = "red",
  add = FALSE
)

rgl::aspect3d("iso")
rgl::view3d(theta = 40, phi = 25, zoom = 0.8)

rglwidget()
```



The middle plane is the mean of the GP, whereas the bottom and top planes represent the lower and upper bounds of the 95% confidence interval respectively. The planes meet at the evaluated points, which are the red dots on the diagram. The scales are odd due to the re-scaling of F_{target} and $B_{trigger}$ and fitting our GPs to log of the values we want throughout the code which helps to keep the GP stable(Huang et al. 2006).

We can then also do this for the GP for catch:

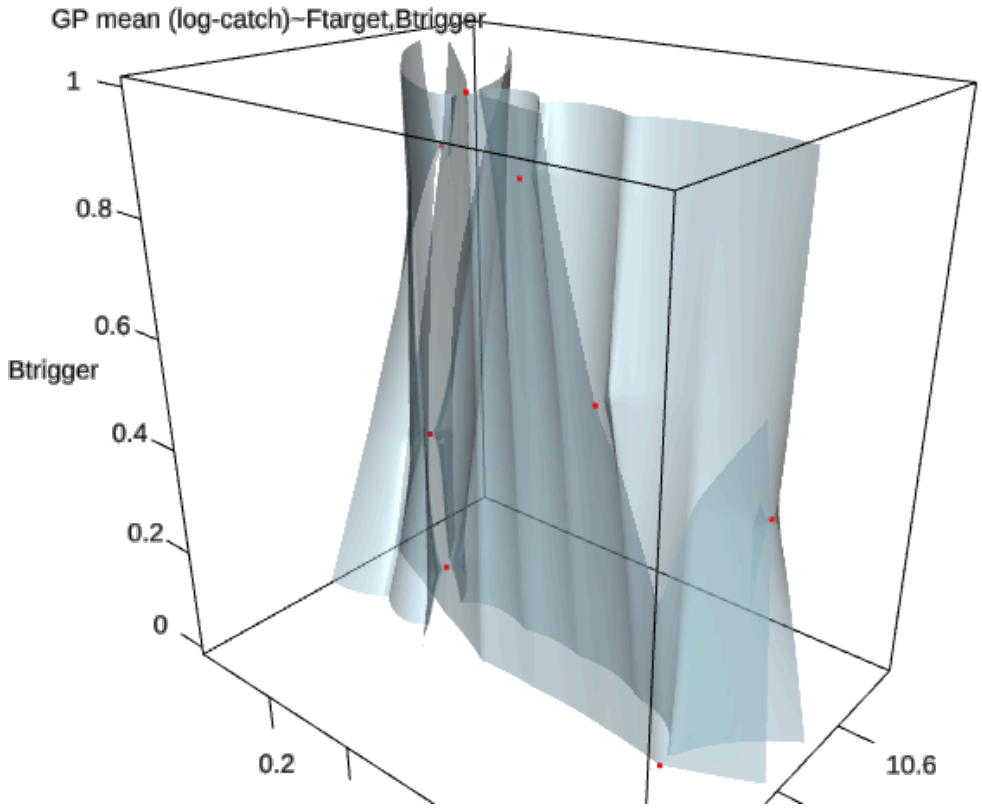
```
library(DiceView)
library(rgl)

Xlim <- rbind(
  apply(gp_cat@X, 2, min),
  apply(gp_cat@X, 2, max)
)


sectionview3d(
  gp_cat,
  dim = 2,
  Xlim = Xlim,
  Xlab = c("Ftarget", "Btrigger"),
  ylab = "GP mean (log-catch)",
  npoints = c(100, 100),
  col = "lightblue",
  scale = TRUE,
  col_points = "red",
  add = FALSE
)

rgl::aspect3d("iso")
rgl::view3d(theta = 40, phi = 25, zoom = 0.8)

rglwidget()
```



We'll now explore how these GPs can be used to estimate the values of functions in the Kriging section.

Kriging

Kriging is a Bayesian statistical method for modelling functions(Frazier 2018). Let f be the objective function. We want to focus on the values $F(x_1), \dots F(x_k)$ where $X := \{x_1, \dots, x_k\}$ is the finite design space. We can represent these values as the vector $F(x_{1:k}) := (F(x_1), \dots, F(x_k))$. This vector is unknown, but we can assume it was drawn at random from a multivariate normal distribution(Frazier 2018).

We let $\mu_0(x_{1:k})$ be the mean vector which we construct by evaluating μ_0 at each point in X. To encode the belief that as the inputs change slightly, the function only changes slightly, we choose cov_0 such that any points x_i, x_j that are close together in X have a large positive correlation(Frazier 2018). Then, we can model $F(x_{1:k})$ as below:

$$F(x_{1:k}) \sim N(\mu_0(x_{1:k}), \Sigma_{1:k}) \tag{1}$$

where

$$\mu_0(x_{1:k}) = (\mu_0(x_1), \dots, \mu_0(x_k))$$

and N is the normal distribution(Frazier 2018). Now, if we have evaluated n points such that we have $f(x_{1:n})$ and want to evaluate x_{n+1} we let $k = n + 1$ in Equation 1. Then, we can compute the conditional distribution of $F(x_{n+1})$ given $f(x_{1:n})$ using Bayes' rule:

$$F(x_{n+1})|f(x_{1:n}) \sim N(\mu_n(x_{n+1}), \sigma_n^2(x_{n+1})) \tag{2}$$

where:

$$\mu_n(x_{n+1}) = \text{cov}_0(x_{n+1}, x_{1:n})(\Sigma_{1:n})^{-1}(f(x_{1:n}) - \mu_0(x_{1:n})) + \mu_0(x_{n+1})$$

$$\sigma_n^2(x_{n+1}) = \text{cov}_0(x_{n+1}, x_{n+1}) - \text{cov}_0(x_{n+1}, x_{1:n})(\Sigma_{1:n})^{-1}\,\text{cov}_0(x_{1:n}, x_{n+1})$$

where:

$$\text{cov}_0(x_{n+1}, x_{1:n}) = (\text{cov}_0(x_{n+1}, x_1), \dots, \text{cov}_0(x_{n+1}, x_n))$$

This conditional distribution $F(x_{n+1})|f(x_{1:n})$ is called the posterior probability distribution for x_{n+1}. We can calculate this distribution for every point in the design space X. This results in a new GP F_n with a mean vector and covariance kernel that depend on the location of the unevaluated points, the locations of the evaluated points $x_{1:n}$, and their values $f(x_{1:n})$(Frazier 2018). So, we can update our GP every round based on the new points we have evaluated.

Our process for updating GPs can be seen in the code below:

```
gp_risk <- km(~.^2, design = runs[, c("Ftarget", "Btrigger")], estim.method = "MLE",
              response = res_risk, nugget = 1e-12 * var(res_risk), covtype = "exp")
```

This looks the same as before, but the `runs` variable includes all of our evaluated points and so we are adding the points that have been newly evaluated this round. Hence, we can do the calculations above to update the GP with a new mean vector and covariance kernel for our next round.

Bayesian History Matching

Once the objective function f has been evaluated at eight points for the first round, Bayesian history matching speeds up the process by removing any points that are implausible(Spence 2025). This is done by using Bayes' Theorem to update our beliefs about the value of every unevaluated point based on the values of the points we have evaluated so far(Spence 2025). The general process is described below.

Let f be the objective function and x be a point in the sample space. We begin with some uncertainty about $f(x)$(Spence 2025). However, we can make probabilistic statements such as:

$$P(f(x) > a) = \int_a^{\infty} P(f(x))df(x)$$

Once we evaluate another point x' where $x' \neq x$, we are able to improve our integral to:

$$P(f(x) > a|f(x')) = \int_a^{\infty} P(f(x)|f(x'))df(x)$$

We now let $a = max\{f(x_1), \ldots, f(x_l)\}$ where l is the number of points we have evaluated so far. For the first round, $l = 8$ but as the rounds increase we make sure to include all previous points of the objective function that have been evaluated.(Spence 2025) We remove the point x if:

$$P(f(x) > a|f(x_1), \ldots, f(x_l)) = \int_a^{\infty} P(f(x)|f(x_1), \ldots, f(x_l))df(x) < \varepsilon$$

using $\varepsilon = 0.00001$ until no plausible points remain. Then, the optimum will be x^* such that:

$$f(x^*) = max\{f(x_1), \ldots, f(x_l)\}$$

as our index l counts the number of points we have evaluated throughout the whole simulation.

In our case, x^* is the F_{target} and $B_{trigger}$ that will give the highest catch whilst following the precautionary principle(Spence 2025).

This process is written up in the code as the below:

```
#Finds the probability that log(risk) <= log(0.05)
prisk <- pnorm(log(0.05),pred_risk1_g$mean,pred_risk1_g$sd+1e-12)

#Finds the current maximum catch which also has risk < 0.05
#Note that we have exponentiated for this part
current_max <- max(runs$C_long[runs$risk3_long < 0.05])

#Finds the probability that the log(catch)<= log(current_max) for
#all points in the design space
pcat <- pnorm(log(current_max), pred_cat_g$mean, pred_cat_g$sd + 1e-12)

#Check which points have P(log(catch) >= log(curent_max)) > 0.00001
#and have P(log(risk) <= log(0.05)) > 0.00001.
#These are our plausible points that remain.
possible <- (apply(cbind((1 - pcat), prisk), 1, min) > eps)
```

Expected Improvement

The first type of acquisition function we will look at is Expected Improvement (EI). Suppose we have sampled the points $x_1, \ldots, x_n$ and observe the values $f(x_{1:n})$. Then, if we were to return a solution at this point, bearing in mind we observe the objective function f without noise and we can only return points we have already evaluated, we would return $f_n^* = max\{f(x_1), \ldots, f(x_n)\}$. Imagine we then consider evaluating another point x_{n+1} to get $f(x_{n+1})$. We can then define the Expected Improvement as:

$$EI_n(x_{n+1}) := E[F(x_{n+1})|f(x_{1:n}) - f_n^*]^+$$

where $[F(x_{n+1}) - f_n^*]^+$ is the positive part of $[F(x_{n+1}) - f_n^*]$. This acquisition function is relatively easy to optimise and many different methods have been developed for doing this.

In our case, we want to sample multiple points at the same time, up to 8 points each round. This method of EI can be coded up as the function:

```r
expected_improvement <- function(mu, sigma, y_best, xi = 0.05, task = "max",
                                 pred_risk, eps = 1e-4)
{
  #Setting up EI vector
  ei <- numeric(length(mu))
  #Determining points with P(risk<=0.05)>0.00001
  safe_points <- pred_risk >= eps

  # Only calculate EI if there are safe points and the task is a valid option
  if (any(safe_points)) {
    if (task == "min")
        imp <- y_best - mu[safe_points] - xi
    if (task == "max")
        imp <- mu[safe_points] - y_best - xi
    else
        stop('task must be "min" or "max"')

    Z <- imp / sigma[safe_points]
    #Only calculate EI for safe points
    ei[safe_points] <- imp * pnorm(Z) + sigma[safe_points] * dnorm(Z)
    ei[safe_points][sigma[safe_points] == 0.0] <- 0.0
  }

  return(ei)
}
```

where `mu` is the mean of the GP for log(catch) and `sigma` is the standard deviation of the GP for log(catch). Then we also have `y_best` which is the current best log(catch), `xi` which is a parameter balancing exploration and exploitation. We set `pred_risk = prisk` and `eps = 1e-4` and these are the same as before. This gives us the formula:

$$EI_n(x_{n+1}) = [(\mu_n(x_{n+1}) - f_n^* \cdot \Phi(Z)) + (\sigma_n(x_{n+1}) \cdot \phi(Z))]^+ \tag{3}$$

where again the notation $[\cdot]^+$ means the positive part and where:

$$Z = \frac{\mu_n(x_{n+1}) - f_n^*}{\sigma_n(x_{n+1})}$$

Equation 3 can be gained from Equation 1 by setting $k = n + 1$ and then studying the distribution of $F(x_{n+1}) - f_n^*$. However, we can also consider it as a version of Equation (15) from (Jones, Schonlau, and Welch 1998) where we first flip the signs are we are focused on the maximisation case and then set $f_{min} = f_n^*, \hat{y} = \mu_n(x)$ and $s = \sigma_n(x)$.

Augmented Expected Improvement

This was included to help make the method perform better for noisy functions which will make it more generally applicable(Huang et al. 2006). To deal with these noisy observations, a change was proposed to standard EI function as detailed below. This change seems mostly to have been justified by empirical performance(Letham et al. 2018).

By adjusting Equation (12) found in (Huang et al. 2006) to our own notation, we get that:

$$AEI_n(x_{n+1}) = E_n[F(x_{n+1})|f(x_{1:n}) - f_{eb}^*]^+ \left(1 - \frac{\sigma_{obs}}{\sqrt{\sigma_n^2(x_{n+1}) + \sigma_{obs}^2}}\right)$$

where σ_{obs} is the standard deviation of the noise variable set by the user and $\sigma_n(x)$ is the standard deviation of GP F at the n^{th} iteration, as used beforehand. We have also changed f_n^* to f_{eb}^* which is the highest predicted mean at any sampled point so far so that we take into account that the uncertainty in our observations could cause a large spike(Huang et al. 2006).

As our situation has no noise, in our code we are still using f_n^*(Spence 2025). Thus, we code up the equation:

$$AEI_n(x_{n+1}) = EI_n(x_{n+1}) \left(1 - \frac{\sigma_{obs}}{\sqrt{\sigma_n^2(x_{n+1}) + \sigma_{obs}^2}}\right)$$

in the code below:

```
augmented_expected_improvement <- function(mu, sigma, y_best, xi = 0.05,
                                           task = "max", pred_risk, eps = 1e-4,
                                           noise_var = 0)
{
  ei <- numeric(length(mu))
  safe_points <- pred_risk >= eps

  # Only calculate AEI for safe points
  if (any(safe_points))
        if (task == "min")
            imp <- y_best - mu[safe_points] - xi
        if (task == "max")
            imp <- mu[safe_points] - y_best - xi
        else
            stop('task must be "min" or "max"')

        Z <- imp / sigma[safe_points]
        ei[safe_points] <- imp * pnorm(Z) + sigma[safe_points] * dnorm(Z)
        ei[safe_points][sigma[safe_points] == 0.0] <- 0.0

        # Augmentation factor to handle noise
        # When noise_var = 0 it reduces to standard EI
        augmentation_factor <- 1 - sqrt(noise_var / (noise_var + sigma^2))
        aei <- ei * augmentation_factor

  # Giving points with prob(risk <= 0.05) < 0.0001 an expected
  # improvement of zero so we avoid them
  aei <- ifelse(pred_risk < eps, 0, aei)

  return(aei)
}
```

where `noise_var` corresponds to σ_{obs}^2 and `sigma` corresponds to $\sigma_n(\cdot)$.

Knowledge Gradient

We remove the assumption of EI that we have to return a pre-evaluated point as our best point(Frazier 2018). This allows us to do some different computations to the ones in EI. We also now start by saying that the solution we would choose if we have to stop sampling after n points would be the point in the design space with the largest $\mu_n(\cdot)$ value, where $\mu_n(\cdot)$ is the mean vector of the posterior probability distribution after n iterations. We call this maximum x_n^* and then can say that $F(x_n^*)$ is random under the posterior distribution and has the mean vector after sampling $f(x_{1:n})$ of:

$$\mu_n^* := \mu_n(x_n^*) = max_x \mu_n(x)$$

where x is any point in the sample space(Frazier 2018).

Then, we imagine that we are now allowed to sample a new point x_{n+1}. We get a new posterior distribution at the point x which we can calculate using Equation 2 by replacing x_{n+1} with x and $x_{1:n}$ with $x_{1:n+1}$ to include our new observation. This will have the posterior mean function $\mu_{n+1}(\cdot)$ and the conditional expected value for $F(x_n^*)$ changes to be:

$$\mu_{n+1}^* := max_x \mu_{n+1}(x)$$

So, we can see that the increase in the conditional expected value of $F(x_n^*)$ by sampling the new point x_{n+1} is(Frazier 2018):

$$\mu_{n+1}^* - \mu_n^*$$

While this quantity is unknown before we sample x_{n+1} we can calculate it's expected value given our observations $x_1, \ldots, x_n$. The Knowledge Gradient for sampling at a new point x in the design space is defined as(Frazier 2018):

$$KG_n(x) := E_n[\mu_{n+1}^* - \mu_n^* | x_{n+1} = x] \tag{4}$$

where again E_n indicates the expectation taken under the posterior distribution at the n^{th} iteration. We would sample the point x with the largest $KG_n(x)$ as our next point(Frazier 2018).

We can see this definition in the code as:

```
kg[i] <- mean(max_after - mu_best)
```

where `max_after` is μ_{n+1}^* and `mu_best` is μ_n^*.

The easiest way to calculate the KG is via simulation. This can be done by simulating one possible value for $f(x_{n+1})$. Then, we calculate μ_{n+1}^* and subtract μ_n^*. We iterate this process many times so that we can find the average of $\mu_{n+1}^* - \mu_n^*$ and this allows us to estimate $KG_n(x)$(Frazier 2018). This process, or calculating Equation 4 directly from the properties of the normal distribution, both work well in discrete, low dimensional problems which is the situation we are in for the first half of the project(Frazier 2018).

Alternatively, we can calculate μ_{n+1} using the formula below(Ungredda, Pearce, and Branke 2022):

$$\mu_{n+1}(x) = \mu_n(x) + \frac{\text{cov}_n(x_{n+1}, x)}{\text{var}_n(x_{n+1}) + \sigma_{\text{obs}}^2}(F(x_{n+1}) - \mu_n(x_{n+1})) \tag{5}$$

where σ_{obs} is a noise variable which can be determined by the user(Ungredda, Pearce, and Branke 2022). From the GP for catch, we get $\text{cov}_n(x_{n+1}, x)$ and the standard deviation σ_{obs}^2.

Equation 5 is mirrored in the code as:

```
# nsim number of simulated observations from a normal
y_sim <- rnorm(nsim, mu_i, sigma_i)
# the covariance between every point in the grid and the point x_i
cov_xp_xi <- cov_grid[, i]
# denominator term in the KG update formula
denom <- var[i] + obs_noise_var
# For each simulated y, compute updated max(mu)
max_after <- numeric(nsim)

for (s in seq_len(nsim)) {
                # says what would the new maximum mu be if we observed this
                # simulated value at the point x_i
                # Evaluates the posterior mean for every point in the
                # space, updating other points based on closeness
                # to the evaluated point by using the covariance matrix
                mu_new <- mu + cov_xp_xi * (y_sim[s] - mu_i) / denom
                # takes this new maximum mu into a vector for later averaging
                max_after[s] <- max(mu_new)
            }
```

where `mu_new` is $\mu_{n+1}(x)$, `cov_xp_xi` is $\text{cov}_n(x_{n+1}, x)$, `y_sim[s]` is a simulated value drawn from F_{n+1} and `mu_i` is $\mu_n(x_{n+1})$. We can also see that `var[i]` is $\text{var}_n(x_{n+1})$ and `obs_noise_var` is σ_{obs}^2 which makes the denominators equivalent.

Now, we have everything needed to compute Equation 4 . We can write the whole process in code as below:

```
knowledge_gradient_sim <- function(mu, sigma, model, obs_noise_var = 0,
                                   nsim = 100, pred_risk, eps = 1e-4)
{
    X_pred <- dat[, c("Ftrgt", "Btrigger")]
    cov_grid <- cov_exp(X_pred, X_pred, theta = model@covariance@range.val,
                        sigma2 = model@covariance@sd2)
    m <- length(mu)
    var <- sigma^2
    # Current best mean catch
    mu_best <- max(mu)
    kg <- numeric(m)

        # Loop over all candidate points
        for (i in seq_len(m)) {
            # Set KG to 0 for points where risk is too high
            if (pred_risk[i] < eps) {
                kg[i] <- 0
                next
            }

            mu_i <- mu[i]
            sigma_i <- sqrt(var[i] + obs_noise_var)
            # nsim simulated observations - using rnorm to sample from a
            # normal distribution is ok here as we are running a GP
            # to model the catch, which is assumed to be normally
            # distributed
            y_sim <- rnorm(nsim, mu_i, sigma_i)
            # the covariance between every point in the grid and
            # the point x_i
            cov_xp_xi <- cov_grid[, i]
            # denominator term in the KG update formula
            denom <- var[i] + obs_noise_var
            # For each simulated y, compute updated max(mu)
            max_after <- numeric(nsim)
            for (s in seq_len(nsim)) {
                # says what would the new maximum mu be if we observed this
                # simulated value at the point x_i by implementing
                # the standard formula for this in GP posterior updating
                # This evaluates the posterior mean for every point in the
                # space, updating other points based on closeness
                # to the evaluated point by using the cov matrix
                mu_new <- mu + cov_xp_xi * (y_sim[s] - mu_i) / denom
                # takes this new maximum mu into a vector for later
                # averaging
                max_after[s] <- max(mu_new)
            }
            # Computes expected increase in the maximum posterior mean, mu
            # higher values mean we're getting better knowledge as to where
            # and what the maximum catch could be
            kg[i] <- mean(max_after - mu_best)
        }
    # Returns the vector of knowledge gradient values for each point in
    # the design space
    kg
}
```

All the arguments of the function are as before with a few exceptions. `obs_noise_var` is the new name for the noise variable set by the user; `theta` is the length scale of our GP determining how quickly the objective function changes as

$Ftarget$ and $Btrigger$ change; and `nsim` tells the code how many times to estimate the value of $f(x_{n+1})$ using $F(x_{n+1})$ before we calculate $KG_n(x)$(Roustant 2025).

References

Frazier, Peter. 2018. “A Tutorial on Bayesian Optimization.” https://doi.org/10.48550/arXiv.1807.02811.

Huang, D., Theodore Allen, William Notz, and Ning Zheng. 2006. “Global Optimization of Stochastic BlackBox Systems via Sequential Kriging Meta-Models.” *Journal of Global Optimization* 34: 441–66. https://doi.org/10.1007/s10898-005-2454-3.

Jones, Donald, Matthias Schonlau, and William Welch. 1998. “Efficient Global Optimization of Expensive Black-Box Functions.” *Journal of Global Optimization* 13: 455–92. https://doi.org/10.1023/A:1008306431147.

Letham, Benjamin, Brian Karrer, Guilherme Ottoni, and Eytan Bakshy. 2018. “Constrained Bayesian Optimization with Noisy Experiments.” https://arxiv.org/abs/1706.07094.

MacKay, David JC et al. 1998. “Introduction to Gaussian Processes.” *NATO ASI Series F Computer and Systems Sciences* 168: 133–66.

Roustant, Olivier. 2025. https://www.rdocumentation.org/packages/DiceKriging/versions/1.6.1.

Spence, Michael A. 2025. “Using History Matching to Speed up Management Strategy Evaluation Grid Searches.” *Canadian Journal of Fisheries and Aquatic Sciences* 82: 1–14. https://doi.org/10.1139/cjfas-2024-0191.

Ungredda, Juan, Michael Pearce, and Juergen Branke. 2022. “Efficient Computation of the Knowledge Gradient for Bayesian Optimization.” https://arxiv.org/abs/2209.15367.