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
- K-Means Clustering

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 select 8 plausible points to sample in the next round

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.

General theory

Gaussian Processes

A Gaussian Process F has a mean function μ_0 and a covariance function $\mathrm{cov}_0(x_i, x_j)$. We can then evaluate the covariance function $\mathrm{cov}_0(x_i, x_j)$ for every pair x_i, x_j where $i, j \in \{1, \ldots, m\}$ to find the covariance matrix $\Sigma_{1:m}$. 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 $F(x_{1:m}) = (F(x_1), \ldots, F(x_m))$ is given by (Frazier 2018):

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

where

$$\mu_0(x_{1:m}) = (\mu_0(x_1), \ldots, \mu_0(x_m))$$

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 $\mathrm{cov}_0(x, x') > \mathrm{cov}_0(x, x'')$ (Frazier 2018).

We use a GP to emulate the objective function because it is much cheaper to evaluate than 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 $x_1, \ldots, x_m$ as the emulator is fitted to these points (Spence 2025).

Maximum Likelihood Estimation

When using GPs to emulate our objective function, we need to be able to estimate the coefficients of the objective function using the data we gain from evaluating our points $x_1, \ldots, x_m$. - FIND CITATION, JUSTIFY

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 $f(x_{1:m}) = (f(x_1), \ldots, f(x_m))$, we calculate the likelihood of these observations under the prior given η which is denoted as $p(f(x_{1:m})|\eta)$. This likelihood can be modelled by a multivariate normal - WHY IS THIS THE SAME AS EQ 1 (Frazier 2018). Lastly, we set $\hat{\eta}$ to the value that maximizes this likelihood:

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

Kriging

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

$$F(x_{1:k}) \sim N(\mu_0(x_{1:k}), \Sigma_{1:k}) \tag{2}$$

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 2. Then, we can compute the conditional distribution of $F(x_{n+1})$ given $f(x_{1:n})$ using Bayes' rule:

$$F(x_{n+1})|f(x_{1:n}) \sim N(\mu_n(x_{n+1}), \sigma_n^2(x_{n+1})) \tag{3}$$

where:

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

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

where:

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

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.

Bayesian History Matching

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 use Bayes' Theorem 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:

$$P(f(x) > a|f(x_1), \ldots, f(x_l)) = \int_a^\infty P(f(x)|f(x_1), \ldots, f(x_l))df(x) < \varepsilon \tag{4}$$

for a small $\varepsilon > 0$ 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.

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 $f_n^* = max\{f(x_1), \ldots, f(x_n)\}$ (Frazier 2018). 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 (Frazier 2018).

There is another expression for $EI_n(x_{n+1})$:

$$EI_n(x_{n+1}) = [(\mu_n(x_{n+1}) - f_n^* \cdot \Phi(Z)) + (\sigma_n(x_{n+1}) \cdot \phi(Z))]^+ \tag{5}$$

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 5 can be gained from Equation 2 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 as 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:

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

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).

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 3 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):

$$KG_n(x) := E_n[\mu^*_{n+1} - \mu^*_n | x_{n+1} = x] \tag{6}$$

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).

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 6 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):

$$\mu_{n+1}(x) = \mu_n(x) + \frac{\mathrm{cov}_n(x_{n+1}, x)}{\mathrm{var}_n(x_{n+1}) + \sigma^2_{\mathrm{obs}}}(F(x_{n+1}) - \mu_n(x_{n+1})) \tag{7}$$

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 $\mathrm{cov}_n(x_{n+1}, x)$ and the standard deviation σ^2_{obs}.

Kmeans process for selecting multiple points

We have now been able to determine which points are possible based on the probability that their catch is higher than the current maximum catch (using Bayesian History Matching) and the probability that their risk is less than 0.05. These points will be called the Possible Space, PS. We have then assigned a value to each point in PS using one of the acquistion functions above. Now, we want to decide which 8 points are best to evaluate next.

For sampling only one point next, this would be very simple as you would take the point with the highest value assigned by the acquisition function (Frazier 2018). However, we want to sample 8 points next so that we continue the pattern set up in the original paper and we want a good trade-off between exploration and exploitation (Spence 2025),(**batchspreadinoutjustification?**).

So, we use the `kmeans` function which is part of the stats package in R (which is automatically loaded into an R session). This function uses the algorithm from Hartigan and Wong, 1979 by default(Hartigan and Wong 2018), (RDocumentation 2025). The k-means clustering method is the most commonly used due to its simplicity compared to other clustering algorithms(Kodinariya, Makwana, et al. 2013.)

This algoritm creates k clusters (groups of points) such that the points within each cluster have the sum of squares to the centre of their cluster smaller than it would be to the centre of any other cluster (RDocumentation 2025). It starts by defining k centroids which should be placed as much as possible far away from each other (Kodinariya, Makwana, et al. 2013). Then, we take each point in the space and associate it to the nearest centroid. We stop when every point has been assigned to a centroid (Kodinariya, Makwana, et al. 2013). At this point, we re-calculate k new centroids as the centers of the clusters created by the previous step. This may result in some points changing clusters (Hartigan and Wong 2018). We repeat this process until no points change clusters (Hartigan and Wong 2018),(Kodinariya, Makwana, et al. 2013).

However, before the algorithm can start, we must specify how many clusters we want (Kodinariya, Makwana, et al. 2013). This can be difficult in many cases (Kodinariya, Makwana, et al. 2013). In our case, it is relatively simple as we know how many points we want to sample next and so we set this to be the number of clusters. We then run the algorithm on PS to form the clusters. Then, we pick the point with the highest value of the acquisition function from each cluster to sample in our next round. This allows us to search for viable points by looking in the Possible Space but also to keep the points we are going to sample spread out so that we can balance exploitation and exploration more effectively (Azimi, Fern, and Fern 2010).

Application of theory in my project

Set up the Gaussian Processes

To maintain the notation from the Spence 2025 paper, we use m_1 for the mean function of the catch GP and m_2 for the mean function of the risk GP (Spence 2025):

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

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

where all of the above $\beta_{s,t}$ for $s \in \{1,2\}$ and $t \in \{1,2,3,4,5\}$ are coefficients to be found through maximum likelihood estimation and:

$$\phi_1 = \frac{F_{target} - 0.1}{0.4} \quad \text{and} \quad \phi_2 = \frac{B_{trigger} - 110000}{90000}$$

where we rescale for numerical stability in the GP (Spence 2025).

Our covariance function c for both GPs is the variance σ_i^2 (which is acting as a scalar) times the Ornstein-Uhlenbeck correlation function (Spence 2025):

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

where $\delta_{i,1}$ and $\delta_{i,2}$ are the length scales for each of ϕ_1 and ϕ_2 respectively (**williams2006gaussian?**).

We need to sample our first round of eight points before setting up the GPs so that we have enough data to estimate all of the coefficients in our GPs (Jones, Schonlau, and Welch 1998). Note that until we have sampled sixteen points, we set the prior of the catch GP to be the same as the risk GP, $m_2(\phi)$ (Spence 2025). This is because we need to estimate the coefficients for the mean function for the catch, risk and the length scales for the covariance function c (Jones, Schonlau, and Welch 1998),(Roustant, Ginsbourger, and Deville 2012). For mathematical stability, these functions cannot have more than eight coefficients between them in our first round as we are only sampling eight points per round due to computation limitations encountered at the time of the Spence 2025 paper (Jones, Schonlau, and Welch 1998),(Spence 2025). However, after our first round we reset the mean function for the catch GP to $m_1(\phi)$ (Spence 2025).

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 $m_1(\phi), m_2(\phi)$ and c for our GPs (Roustant 2025),(Roustant, Ginsbourger, and Deville 2012).

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")
```

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)
```

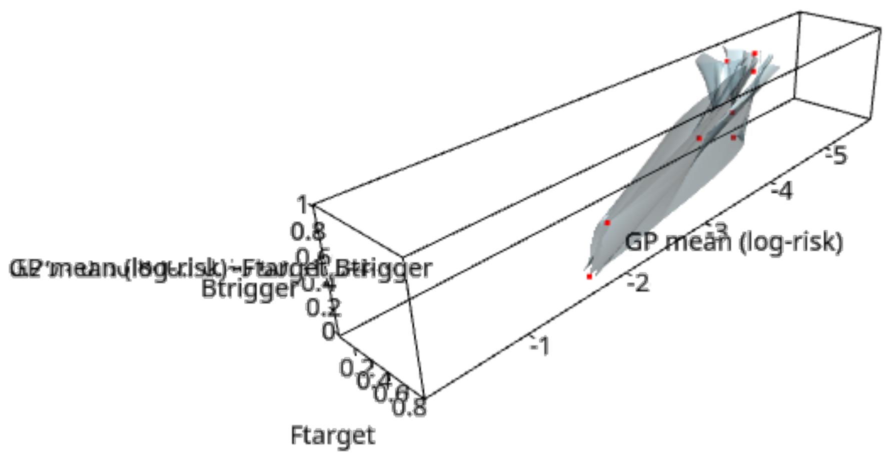
```
This build of rgl does not include OpenGL functions.  Use
 rglwidget() to display results, e.g. via options(rgl.printRglwidget = TRUE).
```

```
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:

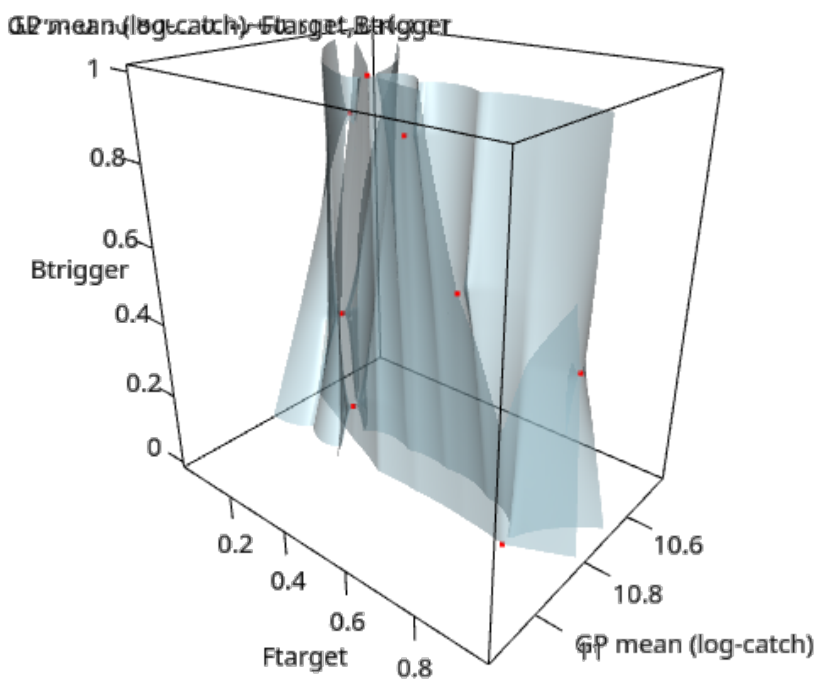
```r
library(DiceView)
library(rgl)

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()
```



Excluding implausible points

Firstly, we enforce the precautionary threshold of $P(log(risk) \leq 0.05)$ by calculating:

```r
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),(Roustant, Ginsbourger, and Deville 2012).

Then, Bayesian history matching speeds up the process by removing any points that are implausible according to Equation 4. 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, where we have set $\varepsilon = 0.0001$ (Spence 2025):

```
#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)
```

Deciding on next point to sample

The method we use here depends on which script we are running. We have three options.

Expected Improvement

We use the Equation 5 expression. This method of EI can be coded up as the function:

```
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.

Augmented Expected Improvement

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:

```r
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 $\sigma_n(\cdot)$ and the other arguments are as in EI.

Knowledge Gradient

Equation 7 is mirrored in the code as:

```r
# 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^2_{obs} which makes the denominators equivalent.

Now, we have everything needed to compute Equation 6 . 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
            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 averaging
                max_after[s] <- max(mu_new)
            }
            # Computes expected increase in the maximum posterior mean
            kg[i] <- mean(max_after - mu_best)
        }
    # Returns the vector of KG values for each point in the design space
    kg
}
```

where the KG for each point is calculated by:

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

where max_after is μ^*_{n+1} and mu_best is μ^*_n.

All the arguments of the function are as in EI 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).

Kmeans process

To then make sure that the next points we sample are spread out across the sample space, we use the Kmeans process as coded up below (Azimi, Fern, and Fern 2010):

```
# Select next points

# cand is our Possible Space
 if (nrow(cand) <= 8) {
    # Here, we take all of the points as there are less than the
    # amount we want to sample left
   next_points <- cand[order(-cand$ei), c("Ftarget", "Btrigger")]
 } else {
    # We order the points based on the value given
    # by the acquisition function, from highest to lowest here
   top_candidates <- cand[order(-cand$ei), ][1:nrow(cand), ]
   # We need to set a seed to make the points that kmeans
    # picks as the first centroids reproducible
   set.seed(123)
   # We find the clusters
   km_result <- kmeans(top_candidates[, c("Ftarget", "Btrigger")], centers = 8)
   # We add the cluster column to our table
   top_candidates$cluster <- km_result$cluster
   # We pick the point with the highest value from the
   # acquisition function from each cluster and assemble these
   # points into a vector
   next_points <- do.call(rbind,
                  lapply(split(top_candidates, top_candidates$cluster),
                         function(df) {
     df[which.max(df$ei), c("Ftarget", "Btrigger", "ei")]
   }))
 }
```

Then, at the beginning of our next round we sample the eight points given in next_points.

Updating our GPS

In our second round, we need to update our GPs with new data (Spence 2025). 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 from the Kriging section to update the GP with a new mean vector and covariance kernel for our next round.

For the catch GP, we move on to using a new prior because by the time we create this GP we have sampled 16 points(Spence 2025). Here is the catch GP for the second round and onwards:

```
gp_cat <- km(~I(log(Ftarget+0.1)^2)+I(log(Ftarget+0.1))+ I(log(Ftarget+0.1)^3) +
                I(Btrigger) + I(Btrigger * log(Ftarget+0.1)),
                design=runs[,c("Ftarget","Btrigger")],estim.method="MLE",
                response = res_cat,nugget=1e-12*var(res_cat),covtype = "exp")
```

Now, we repeat the full process described in the Application of theory in my project section until there are no points with a positive KG. Then, the optimal point is the x^* that has the highest catch out of the precautionary points.

References

Azimi, Javad, Alan Fern, and Xiaoli Fern. 2010. "Batch Bayesian Optimization via Simulation Matching." In *Advances in Neural Information Processing Systems*, edited by J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta. Vol. 23. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2010/file/e702e51da2c0f5be4dd354bb3e295d37-Paper.pdf.

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

Hartigan, J. A., and M. A. Wong. 2018. "A k-Means Clustering Algorithm." *Journal of the Royal Statistical Society Series C: Applied Statistics* 28 (1): 100–108. https://doi.org/10.2307/2346830.

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.

Kodinariya, Trupti M, Prashant R Makwana, et al. 2013. "Review on Determining Number of Cluster in k-Means Clustering." *International Journal* 1 (6): 90–95.

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.

RDocumentation. 2025. https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/kmeans.

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

Roustant, Olivier, David Ginsbourger, and Yves Deville. 2012. "DiceKriging, DiceOptim: Two r Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization." *Journal of Statistical Software* 51 (1): 1–55. https://doi.org/10.18637/jss.v051.i01.

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.