# Multifidelity Paths and Trees

Dr TP Prescott*

May 5, 2022

## 1 Preliminaries

### 1.1 Bayesian Inference

Let $f(\cdot \mid \theta)$ denote a high-fidelity model of the data-generating process that has produced data, $y_{\text{obs}}$. The likelihood of the parameters under this model is denoted by $\mathscr{L}(\theta) = f(y_{\text{obs}} \mid \theta)$. For Bayesian inference, we also specify a prior distribution, $\pi(\cdot)$, on the parameter values. The likelihood allows us to update the prior in the context of data into the *posterior*,

$$\mathscr{P}(\theta \mid y_{\text{obs}}) = \frac{\mathscr{L}(\theta)\pi(\theta)}{\mathscr{Z}},$$

where $\mathscr{Z} = \int \mathscr{L}(\theta)\pi(\theta)\,\mathrm{d}\theta$ is a normalisation constant, ensuring that the posterior, $\mathscr{P}(\theta \mid y_{\text{obs}})$, is a probability distribution. Finally, given an arbitrary integrable function $G : \theta \mapsto \mathbb{R}$ defined on the parameter space, we denote the *posterior expectation* of $G$ by

$$\mathbf{E}(G \mid y_{\text{obs}}) = \int G(\theta)\mathscr{P}(\theta \mid y_{\text{obs}})\,\mathrm{d}\theta.$$

However, we assume that the likelihood function, $\mathscr{L}(\theta)$, cannot be computed, and therefore that the derived quantities are intractable.

*The Alan Turing Institute, London NW1, England.

## 1.2 Likelihood-Free Bayes

Given $\theta$, the likelihood-free setting replaces calculation of $\mathcal{L}(\theta)$ with a stochastic estimate, $\omega(y)$, which we will term a *likelihood-free weighting*. This is a random variable, defined on the probability space of simulations of the model, $y \sim f(\cdot \mid \theta)$, under the given parameter. The *simulation-based approximate likelihood* is defined as the conditional expectation,

$$L(\theta) = \mathbf{E}(\omega \mid \theta) = \int \omega(y) f(y \mid \theta) \, \mathrm{d}y,$$

of the stochastic likelihood-free weighting. The simulation-based approximate likelihood induces a *simulation-based approximate posterior*,

$$P(\theta \mid y_{\text{obs}}) = \frac{L(\theta)\pi(\theta)}{Z},$$

with normalisation constant $Z = \int L(\theta)\pi(\theta) \, \mathrm{d}\theta$. The distribution $P$ induces a *simulation-based approximate posterior expectation* of the arbitrary function $G$, denoted

$$\mathbf{E}_P(G \mid y_{\text{obs}}) = \int G(\theta) P(\theta) \, \mathrm{d}\theta.$$

The *fidelity* of likelihood-free inference is defined by how well $\mathbf{E}_P(G \mid y_{\text{obs}})$ approximates the true posterior, $\mathbf{E}(G \mid y_{\text{obs}})$, and relies on the function $L(\theta)$ being approximately proportional to $\mathcal{L}(\theta)$.

### 1.2.1 ABC

For example, the approximate Bayesian computation version of the likelihood-free weighting is defined as

$$\omega_{\text{ABC}}(y) = \mathbf{1}(y \in \Omega(y_{\text{obs}}))$$

for an appropriately defined neighbourhood of the observed data, $\Omega(y_{\text{obs}})$. The corresponding simulation-based approximate likelihood is

$$L_{\text{ABC}}(\theta) = \mathbf{P}(y \in \Omega(y_{\text{obs}}) \mid \theta),$$

which is approximately proportional to $\mathscr{L}(\theta)$ for an appropriately chosen neighbourhood, $\Omega$. The corresponding simulation-based approximate posterior induced by $L_{\text{ABC}}$ is

$$P_{\text{ABC}}(\theta \mid y_{\text{obs}}) = \frac{L_{\text{ABC}}(\theta)\pi(\theta)}{Z_{\text{ABC}}},$$

for the normalisation constant $Z_{\text{ABC}} = \int L_{\text{ABC}}(\theta)\pi(\theta)\,\mathrm{d}\theta$.

## 1.3 Likelihood-Free Monte Carlo

Consider a weighted Monte Carlo sample produced by Algorithm 1.

**Algorithm 1** Stop condition can be computational budget exceeded, or $N$ sufficiently large, etc.

Initialise $N = 0$.
**repeat**
  Increment $N \leftarrow N + 1$.
  Generate $\theta \sim \pi(\cdot)$.
  Simulate $y \sim f(\cdot \mid \theta)$.
  Store weight $w_N \leftarrow \omega(y)$.
  Store $\theta_N = \theta$.
**until** stop condition met.
**return** $\{(w_n, \theta_n)\}_{n=1}^{N}$.

For any integrable function $G : \theta \mapsto \mathbb{R}$, the ratio

$$\hat{G}_N = \frac{\sum_{n=1}^{N} w_n G(\theta_n)}{\sum_{m=1}^{N} w_m} \approx \mathbf{E}_P(G \mid y_{\text{obs}}) \tag{1}$$

approximates the simulation-based approximate posterior expectation of $G$. The estimate, $\hat{G}_N$, is consistent as $N \to \infty$, in the sense that $\hat{G}_N \to \mathbf{E}_P(G \mid y_{\text{obs}})$ approaches the approximate posterior expectation. In general, there remains bias versus the true posterior expectation, $\mathbf{E}(G \mid y_{\text{obs}})$, as $N \to \infty$.

# 2 Multifidelity Inference

Likelihood-free inference as described above relies on repeated simulation of the model of interest, once per iteration in Algorithm 1. If the *cost* of generating $y \sim$

$f(\cdot \mid \theta)$ is prohibitively expensive, this should be required as little as possible.[1] In this section, we allow for alternative, approximate models to be simulated in place of $f(\cdot \mid \theta)$. The key assumption here is that these *low-fidelity* models are significantly less expensive to simulate, at the cost of accuracy. Below, we show how to incorporate such models into likelihood-free inference to produce weighted Monte Carlo samples. As with the weighted Monte Carlo samples produced by Algorithm 1, these samples will be used to estimate approximate posterior expectations, $\mathbf{E}_P(G \mid y_{\text{obs}})$, for the simulation-based approximate posterior $P(\theta \mid y_{\text{obs}})$ induced by the high-fidelity model, $f(\cdot \mid \theta)$, and its associated likelihood-free weighting $\omega(y)$.

## 2.1 Multifidelity Sequential Simulation

Suppose that there exists an ordered list, $f_{\text{lo}} = (f^{(1)}, \ldots, f^{(K)})$ of $K$ *low-fidelity* approximations of the *high-fidelity* model, $f^{(K+1)} = f$. We assume for simplicity that each low fidelity model has the same parameter space as the high-fidelity model, and that simulations $y^{(k)} \sim f^{(k)}(\cdot \mid \theta)$ of each low-fidelity model take values in the same output space as simulations $y \sim f(\cdot \mid \theta)$. This assumption can be relaxed with little consequence beyond notational faff.

In the original setting, each iteration of Algorithm 1 depends on a draw from the probability space of parameter–simulation pairs, $(\theta, y)$, with corresponding density $\pi(\theta) f(y \mid \theta)$. Using the additional $K$ low-fidelity models in $f_{\text{lo}}$, we can extend this probability space to parameter–simulation tuples $z^{(K+1)} = (\theta, y^{(1)}, \ldots, y^{(K)}, y^{(K+1)})$. Assuming that the simulations in $z^{(K+1)}$ are produced in sequence, we also denote the partially completed sequence by $z^{(k)} = (\theta, y^{(1)}, \ldots, y^{(k)})$, for $k = 1, \ldots, K$, writing $z^{(0)} = (\theta)$ where no simulations have been completed. The marginal density of each $z^{(k)}$ is given by the recursive relationship

$$g^{(k)}(z^{(k)}) = \begin{cases} f^{(k)}(y^{(k)} \mid z^{(k-1)}) g^{(k-1)}(z^{(k-1)}) & k = 1, \ldots, K+1, \\ \pi(\theta) & k = 0. \end{cases}$$

Note that we have allowed that simulations $y^{(k)} \sim f^{(k)}(\cdot \mid z^{(k-1)})$ of the $k^{\text{th}}$ model, for $k = 2, \ldots, K+1$, may depend not only on $\theta$ but also all preceding simulations

[1] Here, we will take "cost" to be elapsed time, but there may be other useful measurements of cost.

in the sequence, $y^{(1)}$ to $y^{(k-1)}$. This is known as *coupling*; it is not necessary but is useful enough to justify the notational faff.

## 2.2 Multifidelity Trees

Algorithm 2 defines the function `MF_Tree`, which produces a random *multifidelity tree*, denoted $\tau^{(k)}$ for $k = 0, \ldots, K+1$, where $\tau^{(0)} = \texttt{MF\_Tree}(\,)$ and $\tau^{(k)} = \texttt{MF\_Tree}(z^{(k-1)})$ for $k > 0$. Slightly abusing notation, we denote the random output of `MF_Tree` by $T^{(0)} \sim \texttt{MF\_Tree}(\cdot)$ and $T^{(k)} \sim \texttt{MF\_Tree}(\cdot \mid z^{(k-1)})$.

In addition to the prior, $\pi$, and the models $f^{(k)}$ for $k = 1, \ldots, K+1$, Algorithm 2 requires a positive-valued function, $\mu$, defined on the space of (partial) multifidelity simulations, $z^{(k)}$. This function determines the Poisson distribution of $M$, the random number of child subtrees of $\tau^{(k)}$, conditional on $z^{(k)}$.

## 2.3 Multifidelity Likelihood-Free Weighting

Having assumed for simplicity that simulations $y^{(k)} \sim f^{(k)}(\cdot \mid z^{(k-1)})$ of each model take values in the same output space, we assume that the likelihood-free weighting $\omega(y)$, originally defined on the high-fidelity model output, can also be applied to the outputs $y^{(k)}$ from the low-fidelity models.[2] Then, for any multifidelity tree of the form

$$\tau^{(k)} = \left(z^{(k)},\, c^{(k)},\, \mu^{(k)},\, (\tau_m^{(k+1)})_{m=1}^{M}\right),$$

we define the recursive function

$$w(\tau^{(k)}) = \begin{cases} \frac{1}{\mu^{(0)}} \sum_{m=1}^{M} w(\tau_m^{(1)}), & k = 0, \\ \omega(y^{(k)}) + \frac{1}{\mu^{(k)}} \sum_{m=1}^{M} \left(w(\tau_m^{(k+1)}) - \omega(y^{(k)})\right), & 1 \le k \le K, \\ \omega(y^{(K+1)}), & k = K+1. \end{cases} \tag{2}$$

Note that, by definition, trees $\tau^{(K+1)} = \texttt{MF\_Tree}(z^{(K)})$ have no child subtrees. For any observed multifidelity tree $\tau^{(0)} = \texttt{MF\_Tree}(\,)$, the value of $w(\tau^{(0)})$ is referred to as its *multifidelity likelihood-free weighting*.

[2] In principle, each low-fidelity model may have its own likelihood-free weighting function $\omega^{(k)}(y^{(k)})$. We neglect this extension for notational simplicity.

**Algorithm 2** Recursive generation of a *multifidelity tree* is defined by the overloaded function `MF_Tree`. Each tree $\tau^{(k)}$ is associated with a realisation of $z^{(k)} = (\theta, y^{(1)}, \ldots, y^{(k)})$, produced conditional on $z^{(k-1)}$. The function `MF_Tree` assigns to this node a Poisson number, $M$, of child subtrees $\tau_m^{(k+1)}$, where $M$ has conditional mean $\mu(z^{(k)})$. Nodes at depth $K+1$ are realisations of the full multifidelity simulation, $z^{(K+1)} = (\theta, y^{(1)}, \ldots, y^{(K)}, y^{(K+1)})$, and have no child subtrees.

**function** `MF_Tree( )`
  Generate $\theta \sim \pi(\cdot)$ with observed cost $c^{(0)}$.
  Denote $z^{(0)} = (\theta)$.
  Set $\mu^{(0)} = \mu(z^{(0)})$.
  Generate $M \sim \text{Poisson}(\mu^{(0)})$.
  **for** $m = 1, \ldots, M$ **do**
    Generate $\tau_m^{(1)} =$ `MF_Tree`$(z^{(0)})$
  **end for**
  **return** $\tau^{(0)} = (z^{(0)}, c^{(0)}, \mu^{(0)}, (\tau_m^{(1)})_{m=1}^M)$.
**end function**

**function** `MF_Tree`$(z^{(k-1)})$ for $k = 1, \ldots, K$
  Simulate $y^{(k)} \sim f^{(k)}(\cdot \mid z^{(k-1)})$ with observed cost $c^{(k)}$.
  Append $y^{(k)}$ to $z^{(k-1)}$ to give $z^{(k)}$.
  Set $\mu^{(k)} = \mu(z^{(k)})$.
  Generate $M \sim \text{Poisson}(\mu^{(k)})$.
  **for** $m = 1, \ldots, M$ **do**
    Generate $\tau_m^{(k+1)} =$ `MF_Tree`$(z^{(k)})$.
  **end for**
  **return** $\tau^{(k)} = (z^{(k)}, c^{(k)}, \mu^{(k)}, (\tau_m^{(k+1)})_{m=1}^M)$.
**end function**

**function** `MF_Tree`$(z^{(K)})$
  Simulate $y^{(K+1)} \sim f^{(K+1)}(\cdot \mid z^K)$ with observed cost $c^{(K+1)}$.
  Append $y^{(K+1)}$ to $z^{(K)}$ to give $z^{(K+1)}$.
  Return $\tau^{(K+1)} = (z^{(K+1)}, c^{(K+1)}, 0, ( ))$.
**end function**

**RESULT** The random multifidelity tree $T^{(0)} \sim \texttt{MF\_Tree}()$ induces a random multifidelity likelihood-free weight, $W = w(T^{(0)})$. Given $\theta$, the conditional expectation of $W$ is

$$\mathbf{E}(W \mid \theta) = \mathbf{E}(\omega(y) \mid \theta) = L(\theta),$$

equal to the high-fidelity simulation-based approximation to the likelihood.

**COROLLARY** For the weighted Monte Carlo sample produced by Algorithm 3, the ratio estimate $\bar{G}_N$ defined in Equation (1) is a consistent estimate of the high-fidelity simulation-based approximate posterior expectation $\mathbf{E}_P(G \mid y_{\text{obs}})$.

**Algorithm 3** Stop condition can be computational budget exceeded, or $N$ sufficiently large, etc.

Initialise $N = 0$.
**repeat**
  Increment $N \leftarrow N + 1$.
  Generate $\tau^{(0)} = \texttt{MF\_Tree}()$ of form $\tau^{(0)} = (z_0, c_0, \mu_0, (\tau_m^{(1)})_{m=1}^M)$.
  Store weight $w_N \leftarrow w(\tau^{(0)})$ for $w$ in Equation (2).
  Store $\theta_N \leftarrow \theta$ from $z_0 = (\theta)$.
**until** stop condition met.
**return** $\{(w_n, \theta_n)\}_{n=1}^N$.

## 2.4 Performance

The results above are true for any function, $\mu$, defining the conditional Poisson mean of $M$ in a call of $\texttt{MF\_Tree}()$ or $\texttt{MF\_Tree}(z^{(k-1)})$, so long as $\mu$ takes strictly positive values. We assume that the computational cost of an iteration of Algorithm 3 is dominated by generating $T^{(0)} \sim \texttt{MF\_Tree}(\cdot)$. The recursive function

$$C(\tau^{(k)}) = c^{(k)} + \sum_{m=1}^{M} C(\tau_m^{(k+1)}),$$

represents the observed cost of generating $\tau^{(k)} = \texttt{MF\_Tree}(z^{(k-1)})$, where we note that $C(\tau^{(K+1)}) = c^{(K+1)}$. It follows that the cost of a given iteration of Algorithm 3 is the cost, $C(\tau^{(0)})$, of the corresponding multifidelity tree, $\tau^{(0)} = \texttt{MF\_Tree}()$. The distribution of $C(T^{(0)})$ for $T^{(0)} \sim \texttt{MF\_Tree}(\cdot)$ thus depends on the function

$\mu$ through its influence on $M$.

Smaller values of $\mu$ reduce the computational cost of generating $T^{(0)} \sim \texttt{MF\_Tree}(\cdot)$, by reducing the number of calls of $\texttt{MF\_Tree}(z_m^{(k)})$ within the inner loop of any call of $\texttt{MF\_Tree}(z^{(k-1)})$. However, small values for $\mu$ incur less high quality Monte Carlo samples, in the sense of increasing the mean squared error (MSE) of $\bar{G}_N$ as an estimate of $\bar{G} = \mathbf{E}_P(G \mid y_{\text{obs}})$. We can derive the leading order behaviour of the MSE as a function of the total computational cost, $C_{\text{tot}}$, of producing a sample from Algorithm 3 as

$$\text{MSE} = \frac{\mathbf{E}((\Delta(T)w(T))^2)\mathbf{E}(C(T))}{\mathbf{E}(w(T))^2}\frac{1}{C_{\text{tot}}} + O(C_{\text{tot}}^{-2})$$

for the random multifidelity tree $T \sim \texttt{MF\_Tree}(\cdot)$. Here $w$ and $C$ are given above, and $\Delta(\tau) = G(\theta) - \bar{G}$ for the approximate posterior expectation $\bar{G} = \mathbf{E}_P(G \mid y_{\text{obs}})$. We write

$$\mathcal{J}[\mu] = \mathbf{E}((\Delta(T)w(T))^2)\mathbf{E}(C(T))$$

as a functional of $\mu$, and note that $\mathbf{E}(w(T)) = Z$ is equal to the normalisation constant, independently of $\mu$. Thus, minimising $\mathcal{J}[\mu]$ will optimise the performance of Algorithm 3, in the sense of producing a Monte Carlo sample with the smallest MSE for a given computational budget.[3]

The functional $\mathcal{J}[\mu]$ is, in general, difficult to work with. In the following section we will define a data-driven cost functional $J[\mu]$ to work with instead.

# 3 Multifidelity Implementation

## 3.1 Cost functionals

### 3.1.1 Simulation cost functional

Consider any one multifidelity tree

$$\tau^{(0)} = \left(z^{(0)}, c^{(0)}, \mu^{(0)}, \left(\tau_m^{(1)}\right)_{m=1}^{M}\right) = \texttt{MF\_Tree}(\,)$$

[3] Equivalently, producing a Monte Carlo sample with a given MSE as cheaply as possible.

produced for an iteration of Algorithm 3. We assume that a baseline mean function, $\mu$, is used for the call of `MF_Tree()`. For any proposed alternative mean function, $\mu'$, we can define the recursive function

$$C(\tau^{(k)}; \mu') = c^{(k)} + \frac{\mu'(z^{(k)})}{\mu^{(k)}} \sum_{m=1}^{M} C(\tau_m^{(k+1)}; \mu'),$$
$$C(\tau^{(K+1)}; \mu') = c^{(K+1)}.$$

The function $C(\tau^{(k)}; \mu')$ is a data-driven estimate of the expected cost of producing $\tau^{(k)} = \texttt{MF\_Tree}(z^{k-1})$, using $\mu'$ as the mean function in place of $\mu$.

Now consider the the set of multifidelity trees $\tau_n^{(0)}$, where $n = 1, \ldots N$, produced by $N$ independent calls of `MF_Tree()` in Algorithm 3. We define the data-driven simulation cost functional[4]

$$C[\mu'] = \frac{1}{N} \sum_{n=1}^{N} C(\tau_n; \mu')$$

as the average value of $C(\tau_n; \mu')$ for the sample of independent observations, $\tau_n^{(0)}$, of the random multifidelity tree, $T^{(0)} \sim \texttt{MF\_Tree}(\cdot)$.

### 3.1.2 MSE cost functional

Under the same assumptions as above, we define the recursive function

$$V(\tau^{(k)}; \mu') = \left(\frac{1}{\mu^{(k)}}\right)^2 \left( \sum_{m_1 \neq m_2} \left(w(\tau_{m_1}^{(k+1)}) - \omega(y^{(k-1)})\right) \left(w(\tau_{m_2}^{(k+1)}) - \omega(y^{(k-1)})\right) + \frac{\mu^{(k)}}{\mu'(z^{(k)})} \sum_{m=1}^{M} V(\tau_m^{(k+1)}; \mu') \right),$$

[4] $C$ is doing a lot of work here!

valid for $k = 2, 3, \ldots, K$, with the 'boundary conditions',

$$V(\tau^{(0)}; \mu') = \left(\frac{1}{\mu^{(0)}}\right)^2 \left( \sum_{m_1 \neq m_2} w(\tau_{m_1}^{(1)}) w(\tau_{m_2}^{(1)}) + \frac{\mu^{(0)}}{\mu'(z^{(0)})} \sum_{m=1}^{M} V(\tau_m^{(1)}; \mu'), \right)$$

$$V(\tau^{(1)}; \mu') = \left(\frac{1}{\mu^{(1)}}\right)^2 \left( \sum_{m_1 \neq m_2} w(\tau_{m_1}^{(2)}) w(\tau_{m_2}^{(2)}) + \frac{\mu^{(1)}}{\mu'(z^{(1)})} \sum_{m=1}^{M} V(\tau_m^{(2)}; \mu'), \right)$$

$$V(\tau^{(K+1)}; \mu') = \left(\omega(y^{(K+1)}) - \omega(y^{(K)})\right)^2.$$

Note that if a given tree $\tau^{(k)}$ has no child subtrees, then $V(\tau^{(k)}; \mu') = 0$.

Now consider the set of multifidelity trees $\tau_n^{(0)}$, where $n = 1, \ldots, N$, produced by $N$ independent calls of `MF_Tree()` in Algorithm 3. We define the data-driven MSE cost functional

$$V[\mu'] = \frac{1}{N} \sum_{n=1}^{N} \left(G(\theta_n) - \bar{G}_N\right)^2 V(\tau_n^{(0)}; \mu'),$$

where $\theta_n$ is the parameter associated with $\tau_n^{(0)}$ and where $\bar{G}_N$ is the Monte Carlo posterior estimate.

### 3.1.3 Performance cost functional

In summary, if we take the multifidelity trees produced by a batch run of Algorithm 3, then we can construct a cost functional for a new mean function $\mu'$, equal to

$$J[\mu'] = C[\mu']V[\mu'].$$

This cost functional can now be used to update the mean function. In the following, we will construct a mean function as a recurrent neural network, and train it on simulation data, based on minimising this cost function.

## 3.2 Recurrent Neural Network $\mu$

Layers $X$, $F$ and $G$ define the dynamic system:

- $x^{(0)} = X(\theta)$
- $x^{(k)} = F(x^{(k-1)}, y^{(k)})$ for $k = 1, \ldots, K$.

- $\mu(z^{(k)}) = G(x^{(k)})$, where we use a positive activation function such as `softplus` to ensure $\mu > 0$.

The details (i.e. sizes) of these layers will be application specific.

We train these layers by minimising a regularised cost function

$$J[\mu] + \gamma R[\mu]$$

derived from an existing batch of multifidelity simulations produced by Algorithm 3. Here, $J[\mu]$ is the performance cost functional derived in the previous subsection, which trades MSE against simulation cost. The regulariser functional, $R[\mu]$, needs to be chosen to ensure that $\mu$ does not take excessively small values. For example, we can use $L^2$ weight regularisation of the output layer $G : x^{(k)} \mapsto \mu(z^{(k)})$. If for a given positive $\beta$ the layer $G$ uses the `softplus` activation function,

$$\sigma_\beta(t) = \frac{1}{\beta} \log\left(1 + \exp(\beta t)\right),$$

then this regularisation can ensure a prior on $\mu$ to take values the order of $(\log 2)/\beta$. Note that the strength of the regularisation is parametrised by the free hyperparameter $\gamma \geq 0$.