

---

# Practical Bayesian Optimization for Model Fitting with Bayesian Adaptive Direct Search – Supplementary Material

---

**Luigi Acerbi\***

Center for Neural Science  
New York University  
luigi.acerbi@nyu.edu

**Wei Ji Ma**

Center for Neural Science & Dept. of Psychology  
New York University  
weijima@nyu.edu

In this Supplement, we expand on the definitions and implementations of Gaussian Processes (GPs) and Bayesian optimization in BADS (Section A); we give a full description of the BADS algorithm, including details omitted in the main text (Section B); we report further details of the benchmark procedure, such as the full list of tested algorithms and additional results (Section C); and, finally, we briefly discuss the numerical implementation (Section D).

## A Gaussian processes for Bayesian optimization in BADS

In this section, we describe definitions and additional specifications of the Gaussian process (GP) model used for Bayesian optimization (BO) in BADS. Specifically, this part expands on Sections 2.2 and 3.2 in the main text.

**GP posterior moments** We consider a GP based on a training set  $\mathbf{X}$  with  $n$  points, a vector of observed function values  $\mathbf{y}$ , and GP mean function  $m(\mathbf{x})$  and GP covariance or kernel function  $k(\mathbf{x}, \mathbf{x}')$ , with i.i.d. Gaussian observation noise  $\sigma^2 > 0$ . The GP posterior latent marginal conditional mean  $\mu$  and variance  $s^2$  are available in closed form at a chosen point as

$$\begin{aligned}\mu(\mathbf{x}) &\equiv \mu(\mathbf{x}; \{\mathbf{X}, \mathbf{y}\}, \boldsymbol{\theta}) = \mathbf{k}(\mathbf{x})^\top (\mathbf{K} + \sigma^2 \mathbf{I}_n)^{-1} (\mathbf{y} - m(\mathbf{x})) \\ s^2(\mathbf{x}) &\equiv s^2(\mathbf{x}; \{\mathbf{X}, \mathbf{y}\}, \boldsymbol{\theta}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x})^\top (\mathbf{K} + \sigma^2 \mathbf{I}_n)^{-1} \mathbf{k}(\mathbf{x})\end{aligned}\tag{S1}$$

where  $\mathbf{K}_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ , for  $1 \leq i, j \leq n$ , is the kernel matrix,  $\mathbf{k}(\mathbf{x}) \equiv (k(\mathbf{x}, \mathbf{x}^{(1)}), \dots, k(\mathbf{x}, \mathbf{x}^{(n)}))^\top$  is the  $n$ -dimensional column vector of cross-covariances, and  $\boldsymbol{\theta}$  is the vector of GP hyperparameters.

### A.1 Covariance functions

Besides the automatic relevance determination (ARD) *rational quadratic* (RQ) kernel described in the main text (and BADS default), we also considered the common *squared exponential* (SE) kernel

$$k_{\text{SE}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp \left\{ -\frac{1}{2} r^2(\mathbf{x}, \mathbf{x}') \right\}, \quad \text{with } r^2(\mathbf{x}, \mathbf{x}') = \sum_{d=1}^D \frac{1}{\ell_d^2} (x_d - x'_d)^2, \tag{S2}$$

and the ARD *Matérn 5/2* kernel [1],

$$k_{\text{M52}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \left[ 1 + \sqrt{5r^2(\mathbf{x}, \mathbf{x}') + \frac{5}{3}r^2(\mathbf{x}, \mathbf{x}')} \right] \exp \left\{ -\sqrt{5r^2(\mathbf{x}, \mathbf{x}')} \right\}, \tag{S3}$$

where  $\sigma_f^2$  is the signal variance, and  $\ell_1, \dots, \ell_D$  are the kernel length scales along each coordinate. Note that the RQ kernel tends to the SE kernel for  $\alpha \rightarrow \infty$ .

The Matérn 5/2 kernel has become a more common choice for Bayesian *global* optimization because it is only twice-differentiable [1], whereas the SE and RQ kernels are infinitely differentiable – a

---

\*Current address: Département des neurosciences fondamentales, Université de Genève, CMU, 1 rue Michel-Servet, 1206 Genève, Switzerland. E-mail: luigi.acerbi@gmail.com.

stronger assumption of smoothness which may cause extrapolation issues. However, this is less of a problem for a local interpolating approximation (as in BADS) than it is for a global approach, and in fact we find the RQ kernel to work well empirically (see main text).

**Composite periodic kernels** We allow the user to specify one or more *periodic* (equivalently, circular) coordinate dimensions  $P \subseteq \{1, \dots, D\}$ , which is a feature of some models in computational neuroscience (e.g., the preferred orientation of a neuron, as in the ‘neuronal selectivity’ problem set [2] of the CCN17 benchmark; see Section 4.3 in the main text). For a chosen base stationary covariance function  $k_0$  (e.g., RQ, SE,  $M_{5/2}$ ), we define the composite ARD periodic kernel as

$$k_{\text{PER}}(\mathbf{x}, \mathbf{x}'; k_0, P) = k_0(t(\mathbf{x}), t(\mathbf{x}')), \quad \text{with} \quad \begin{cases} [t(\mathbf{x})]_d = x_d & \text{if } d \notin P \\ [t(\mathbf{x})]_d = \sin\left(\frac{\pi x_d}{L_d}\right) & \text{if } d \in P \\ [t(\mathbf{x})]_{d+|P|} = \cos\left(\frac{\pi x_d}{L_d}\right) & \text{if } d \in P \end{cases} \quad (\text{S4})$$

for  $1 \leq d \leq D$ , where  $L_d$  is the period in the  $d$ -th coordinate dimension, and the length scale  $\ell_d$  of  $k_0$  is shared between  $(d, d+|P|)$  pairs when  $d \in P$ . In BADS, the period is determined by the provided hard bounds as  $L_d = \text{UB}_d - \text{LB}_d$  (where the hard bounds are required to be finite).

## A.2 Construction of the training set

We construct the training set  $\mathbf{X}$  according to a simple *subset-of-data* [3] local GP approximation. Points are added to the training set sorted by their  $\ell$ -scaled distance  $r^2$  from the incumbent  $\mathbf{x}_k$ . The training set contains a minimum of  $n_{\min} = 50$  points (if available in the cache of all points evaluated so far), and then up to  $10 \times D$  additional points with  $r \leq 3\rho(\alpha)$ , where  $\rho(\alpha)$  is a *radius function* that depends on the decay of the kernel. For a given stationary kernel of the form  $k(\mathbf{x}, \mathbf{x}') = k(r^2(\mathbf{x}, \mathbf{x}'))$ , we define  $\rho$  as the distance such that  $k(2\rho^2) \equiv 1/(\sigma_f^2 e)$ . We have then

$$\rho_{\text{SE}} = 1, \quad \rho_{M52} \approx 0.92, \quad \text{and} \quad \rho_{\text{RQ}}(\alpha) = \sqrt{\alpha(e^{1/\alpha} - 1)}, \quad (\text{S5})$$

where for example  $\rho_{\text{RQ}}(1) \approx 1.31$ , and  $\lim_{\alpha \rightarrow \infty} \rho_{\text{RQ}}(\alpha) = 1$ .

## A.3 Treatment of hyperparameters

We fit the GP hyperparameters by maximizing their posterior probability (MAP),  $p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}) \propto p(\boldsymbol{\theta}, \mathbf{X}, \mathbf{y})$ , which, thanks to the Gaussian likelihood, is available in closed form as [4]

$$\ln p(\mathbf{y}, \mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2} \ln |\mathbf{K} + \sigma^2 \mathbf{I}_n| - \frac{1}{2} \mathbf{y}^\top (\mathbf{K} + \sigma^2 \mathbf{I}_n)^{-1} \mathbf{y} + \ln p_{\text{hyp}}(\boldsymbol{\theta}) + \text{const}, \quad (\text{S6})$$

where  $\mathbf{I}_n$  is the identity matrix in dimension  $n$  (the number of points in the training set), and  $p_{\text{hyp}}(\boldsymbol{\theta})$  is the prior over hyperparameters, described in the following.

**Hyperparameter prior** We adopt an approximate *empirical Bayes* approach by defining the prior based on the data in the training set, that is  $p_{\text{hyp}} = p_{\text{hyp}}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y})$ . Empirical Bayes can be intended as a quick, heuristic approximation to a proper but more expensive hierarchical Bayesian approach. We assume independent priors for each hyperparameter, with bounded (truncated) distributions. Hyperparameter priors and hard bounds are reported in Table S1. In BADS, we include an observation noise parameter  $\sigma > 0$  also for deterministic objectives  $f$ , merely for the purpose of fitting the GP, since it has been shown to yield several advantages [5]. In particular, we assume a prior such that  $\sigma$  decreases as a function of the poll size  $\Delta_k^{\text{poll}}$ , as the optimization ‘zooms in’ to smaller scales. Another distinctive choice for BADS is that we set the mean for the GP mean equal to the 90-th percentile of the observed values in the current training set  $\mathbf{y}$ , which encourages the exploration to remain local.

**Hyperparameter optimization** We optimize Eq. S6 with a gradient-based optimizer (see Section D), providing the analytical gradient to the algorithm. We start the optimization from the previous hyperparameter values  $\boldsymbol{\theta}_{\text{prev}}$ . If the optimization seems to be stuck in a high-noise mode, or we find an unusually low value for the GP mean  $m$ , we attempt a second fit starting from a draw from the prior averaged with  $\boldsymbol{\theta}_{\text{prev}}$ . If the optimization fails due to numerical issues, we keep the previous value of

Hyperparameter	Prior	Bounds
GP kernel		
Length scales $\ell_d$	$\ln \ell_d \sim \mathcal{N}_T\left(\frac{1}{2}(\ln r_{\max} + \ln r_{\min}), \frac{1}{4}(\ln r_{\max} - \ln r_{\min})^2\right)$	$[\Delta_{\min}^{\text{poll}}, L_d]$
Signal variability $\sigma_f$	$\ln \sigma_f \sim \mathcal{N}_T(\ln \text{SD}(\mathbf{y}), 2^2)$	$[10^{-3}, 10^9]$
RQ kernel shape $\alpha$	$\ln \alpha \sim \mathcal{N}_T(1, 1)$	$[-5, 5]$
GP observation noise $\sigma$		
deterministic $f$	$\sigma_{\text{est}} = \sqrt{10^{-3} \Delta_k^{\text{poll}}}$	$[4 \cdot 10^{-4}, 150]$
noisy $f$	$\sigma_{\text{est}} = 1$ (or user-provided estimate)	
GP mean $m$	$m \sim \mathcal{N}\left(\mathbf{Q}_{0.9}(\mathbf{y}), \frac{1}{5^2}(\mathbf{Q}_{0.9}(\mathbf{y}) - \mathbf{Q}_{0.5}(\mathbf{y}))^2\right)$	$(-\infty, \infty)$

Table S1: **GP hyperparameter priors.** Empirical Bayes priors and bounds for GP hyperparameters.  $\mathcal{N}(\mu, \sigma^2)$  denotes the normal pdf with mean  $\mu$  and variance  $\sigma^2$ , and  $\mathcal{N}_T(\cdot, \cdot)$  the *truncated* normal, defined within the bounds specified in the last column.  $r_{\max}$  and  $r_{\min}$  are the maximum (resp., minimum) distance between any two points in the training set;  $\Delta_{\min}^{\text{poll}}$  is the minimum poll size (default  $10^{-6}$ );  $L_d$  is the parameter range ( $\text{UB}_d - \text{LB}_d$ ), for  $1 \leq d \leq D$ ;  $\text{SD}(\cdot)$  denotes the standard deviation of a set of elements;  $\Delta_k^{\text{poll}}$  is the poll size parameter at the current iteration  $k$ ;  $\mathbf{Q}_q(\cdot)$  denotes the  $q$ -th quantile of a set of elements ( $\mathbf{Q}_{0.5}$  is the median).

the hyperparameters. We refit the hyperparameters every  $2D$  to  $5D$  function evaluations; more often earlier in the optimization, and whenever the current GP is particularly inaccurate at predicting new points. We test accuracy on newly evaluated points via a Shapiro-Wilk normality test on the residuals [6],  $z^{(i)} = (y^{(i)} - \mu(\mathbf{x}^{(i)})) / \sqrt{s^2(\mathbf{x}^{(i)}) + \sigma^2}$  (assumed independent, in first approximation), and flag the approximation as inaccurate if  $p < 10^{-6}$ .

#### A.4 Acquisition functions

Besides the *GP lower confidence bound* (LCB) metric [7] described in the main text (and default in BADS), we consider two other choices that are available in closed form using Eq. S1 for the GP predictive mean and variance.

**Probability of improvement (PI)** This strategy maximizes the probability of improving over the current best minimum  $y_{\text{best}}$  [8]. For consistency with the main text, we define here the *negative* PI,

$$a_{\text{PI}}(\mathbf{x}; \{\mathbf{X}_n, \mathbf{y}_n\}, \boldsymbol{\theta}) = -\Phi(\gamma(\mathbf{x})), \quad \gamma(\mathbf{x}) = \frac{y_{\text{best}} - \xi - \mu(\mathbf{x})}{s(\mathbf{x})} \quad (\text{S7})$$

where  $\xi \geq 0$  is an optional trade-off parameter to promote exploration, and  $\Phi(\cdot)$  is the cumulative distribution function of the standard normal.  $a_{\text{PI}}$  is known to excessively favor exploitation over exploration, and it is difficult to find a correct setting for  $\xi$  to offset this tendency [9].

**Expected improvement (EI)** We then consider the popular predicted improvement criterion [1, 10, 11]. The expected improvement over the current best minimum  $y_{\text{best}}$  (with an offset  $\xi \geq 0$ ) is defined as  $\mathbb{E}[\max\{y_{\text{best}} - y, 0\}]$ . For consistency with the main text we consider the *negative* EI, which can be computed in closed form as

$$a_{\text{EI}}(\mathbf{x}; \{\mathbf{X}, \mathbf{y}\}, \boldsymbol{\theta}) = -s(\mathbf{x}) [\gamma(\mathbf{x})\Phi(\gamma(\mathbf{x})) + \mathcal{N}(\gamma(\mathbf{x}))] \quad (\text{S8})$$

where  $\mathcal{N}(\cdot)$  is the standard normal pdf.

## B The BADS algorithm

We report here extended details of the BADS algorithm, and how the various steps of the MADS framework are implemented (expanding on Sections 3.1 and 3.3 of the main text). Main features of the algorithm are summarized in Table S2. Refer also to Algorithm 1 in the main text.

Feature	Description (defaults)
Surrogate model	GP
Hyperparameter treatment	optimization
GP training set size $n_{\max}$	70 ( $D = 2$ ), 250 ( $D = 20$ ) (min 200 for noisy problems)
POLL directions generation	LTMADS with GP rescaling
SEARCH set generation	Two-step ES algorithm with search matrix $\Sigma$
SEARCH evals. ( $n_{\text{search}}$ )	$\max\{D, 3 + \lfloor D/2 \rfloor\}$
Aquisition function	LCB
Supported constraints	None, bound, and non-bound via a barrier function $c$
Initial mesh size	$\Delta_0^{\text{mesh}} = 2^{-10}$ , $\Delta_k^{\text{poll}} = 1$
Implementation	badS (MATLAB)

Table S2: **Summary of features of BADS.**

### B.1 Problem definition and initialization

BADS solves the optimization problem

$$\begin{aligned} f_{\min} &= \min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) \quad \text{with } \mathcal{X} \subseteq \mathbb{R}^D \\ (\text{optional}) \quad c(\mathbf{x}) &\leq 0 \end{aligned} \tag{S9}$$

where  $\mathcal{X}$  is defined by pairs of hard bound constraints for each coordinate,  $\text{LB}_d \leq x_d \leq \text{UB}_d$  for  $1 \leq d \leq D$ , and we allow  $\text{LB}_d \in \mathbb{R} \cup \{-\infty\}$  and similarly  $\text{UB}_d \in \mathbb{R} \cup \{\infty\}$ . We also consider optional non-bound constraints specified via a *barrier* function  $c : \mathcal{X} \rightarrow \mathbb{R}$  that returns constraint violations. We only consider solutions such that  $c$  is zero or less.

**Algorithm input** The algorithm takes as input a starting point  $\mathbf{x}_0 \in \mathcal{X}$ ; vectors of *hard* lower/upper bounds  $\text{LB}$ ,  $\text{UB}$ ; optional vectors of *plausible* lower/upper bounds  $\text{PLB}$ ,  $\text{PUB}$ ; and an optional barrier function  $c$ . We require that, if specified,  $c(\mathbf{x}_0) \leq 0$ ; and for each dimension  $1 \leq d \leq D$ ,  $\text{LB}_d \leq (\mathbf{x}_0)_d \leq \text{UB}_d$  and  $\text{LB}_d \leq \text{PLB}_d < \text{PUB}_d \leq \text{UB}_d$ . Plausible bounds identify a region in parameter space where most solutions are expected to lie, which in practice we usually think of as the region where starting points for the algorithm would be drawn from. Hard upper/lower bounds can be infinite, but plausible bounds need to be finite. As an exception to the above bound ordering, the user can specify that a variable is *fixed* by setting  $(\mathbf{x}_0)_d = \text{LB}_d = \text{UB}_d = \text{PLB}_d = \text{PUB}_d$ . Fixed variables become constants, and BADS runs on an optimization problem with reduced dimensionality. The user can also specify circular or periodic dimensions (such as angles), which change the definition of the GP kernel as per Section A.1. The user can specify whether the objective  $f$  is deterministic or noisy (stochastic), and in the latter case provide a coarse estimate of the noise (see Section B.5).

**Transformation of variables and constraints** Problem variables whose hard bounds are strictly positive and  $\text{UB}_d \geq 10 \cdot \text{LB}_d$  are automatically converted to log space for all internal calculations of the algorithm. All variables are also linearly rescaled to the standardized box  $[-1, 1]^D$  such that the box bounds correspond to  $[\text{PLB}, \text{PUB}]$  in the original space. BADS converts points back to the original coordinate space when calling the target function  $f$  or the barrier function  $c$ , and at the end of the optimization. BADS never violates constraints, by removing from the POLL and SEARCH sets points that violate either bound or non-bound constraints ( $c(\mathbf{x}) > 0$ ). During the SEARCH stage, we project candidate points that violate a bound constraint to the closest mesh point within the bounds. We assume that  $c(\cdot)$ , if provided, is known and inexpensive to evaluate.

**Objective scaling** We assume that the scale of interest for differences in the objective (and the scale of other features, such as noise in the proximity of the solution) is of order  $\sim 1$ , and that differences in the objective less than  $10^{-3}$  are negligible. For this reason, BADS is *not* invariant to arbitrary rescalings of the objective  $f$ . This assumption does not limit the actual values taken by the objective across the optimization. If the objective  $f$  is the log likelihood of a dataset and model (e.g., summed over trials), these assumptions are generally satisfied. They would not be if, for example, one were to feed to BADS the *average* log likelihood per trial, instead of the total (summed) log likelihood. In cases in which  $f$  has an unusual scale, we recommend to rescale the objective such that the magnitude of differences of interest becomes of order  $\sim 1$ .

**Initialization** We initialize  $\Delta_0^{\text{poll}} = 1$  and  $\Delta_0^{\text{mesh}} = 2^{-10}$  (in standardized space). The initial design comprises of the provided starting point  $\mathbf{x}_0$  and  $n_{\text{init}} = D$  additional points chosen via a low-discrepancy Sobol quasirandom sequence [12] in the standardized box, and forced to be on the mesh grid. If the user does not specify whether  $f$  is deterministic or stochastic, the algorithm assesses it by performing two consecutive evaluations at  $\mathbf{x}_0$ . For all practical purposes, a function is deemed noisy if the two evaluations at  $\mathbf{x}_0$  differ more than  $1.5 \cdot 10^{-11}$ .<sup>1</sup>

## B.2 SEARCH stage

In BADS we perform an aggressive SEARCH stage in which, in practice, we keep evaluating candidate points until we fail for  $n_{\text{search}}$  consecutive steps to find a *sufficient* improvement in function value, with  $n_{\text{search}} = \max\{D, \lfloor 3 + D/2 \rfloor\}$ ; and only then we switch to the POLL stage. At any iteration  $k$ , we define an improvement *sufficient* if  $f_{\text{prev}} - f_{\text{new}} \geq (\Delta_k^{\text{poll}})^{3/2}$ , where  $\Delta_k^{\text{poll}}$  is the poll size.

In each SEARCH step we choose the final candidate point to evaluate,  $\mathbf{x}_{\text{search}}$ , by performing a fast, approximate optimization of the chosen acquisition function in the neighborhood of the incumbent  $\mathbf{x}_k$ , using a two-step evolutionary heuristic inspired by CMA-ES [13]. This local search is governed by a *search covariance matrix*  $\Sigma$ , and works as follows.

**Local search via two-step evolutionary strategy** We draw a first generation of candidates  $\mathbf{s}_1^{(i)} \sim \mathcal{N}(\mathbf{x}_k, (\Delta_k^{\text{poll}})^2 \Sigma)$  for  $1 \leq i \leq n_{\text{search}}$ , where we project each point onto the closest mesh point (see Section 2.1 in the main text);  $\Sigma$  is a search covariance matrix with unit trace,<sup>2</sup> and  $n_{\text{search}} = 2^{11}$  by default. For each candidate point, we assign a number of offsprings inversely proportionally to the square root of its ranking according to  $a(\mathbf{s}_1^{(i)})$ , for a total of  $n_{\text{search}}$  offsprings [13]. We then draw a second generation  $\mathbf{s}_{\text{II}}^{(i)} \sim \mathcal{N}(\mathbf{s}_1^{(\pi_i)}, \lambda^2 (\Delta_k^{\text{poll}})^2 \Sigma)$  and project it onto the mesh grid, where  $\pi_i$  is the index of the parent of the  $i$ -th candidate in the 2nd generation, and  $0 < \lambda \leq 1$  is a zooming factor (we choose  $\lambda = 1/4$ ). Finally, we pick  $\mathbf{x}_{\text{search}} = \arg \min_i a(\mathbf{s}_{\text{II}}^{(i)})$ . At each step, we remove candidate points that violate non-bound constraints ( $c(\mathbf{x}) > 0$ ), and we project candidate points that fall outside hard bounds to the closest mesh point inside the bounds.

**Hedge search** The search covariance matrix can be constructed in several ways. Across SEARCH steps we use both a diagonal matrix  $\Sigma_{\ell}$  with diagonal  $(\ell_1^2/|\ell|^2, \dots, \ell_D^2/|\ell|^2)$ , and a matrix  $\Sigma_{\text{WCM}}$  proportional to the weighted covariance matrix of points in  $\mathbf{X}$  (each point weighted according to a function of its ranking in terms of objective values  $y_i$ , see [13]). At each step, we compute the probability of choosing  $\Sigma_s$ , with  $s \in \{\ell, \text{WCM}\}$ , according to a *hedging* strategy taken from the Exp3 HEDGE algorithm [14],

$$p_s = \frac{e^{\beta_H g_s}}{\sum_{s'} e^{\beta_H g_{s'}}} (1 - \gamma_H n_{\Sigma}) + \gamma_H \quad (\text{S10})$$

where  $\beta_H = 1$ ,  $\gamma_H = 0.125$ ,  $n_{\Sigma} = 2$  is the number of considered search matrices, and  $g_s$  is a running estimate of the reward for option  $s$ . The running estimate is updated each SEARCH step as

$$g_s^{\text{new}} = \alpha_H g_s^{\text{old}} + \frac{\Delta f_s}{p_s \Delta_k^{\text{poll}}} \quad (\text{S11})$$

where  $\alpha_H = 0.1^{1/(2D)}$  is a decay factor, and  $\Delta f_s$  is the improvement in objective of the  $s$ -th strategy (0 if  $s$  was not chosen in the current SEARCH step). This method allows us to switch between searching along coordinate axes ( $\Sigma_{\ell}$ ), and following an approximation of the local curvature around the incumbent ( $\Sigma_{\text{WCM}}$ ), according to their track record of cumulative improvement.

## B.3 POLL stage

We perform the POLL stage only after a SEARCH stage that did not produce a *sufficient* improvement after  $n_{\text{search}}$  steps. We incorporate the GP approximation in the POLL in two ways: when constructing the set of polling directions  $\mathbf{D}_k$ , and when choosing the polling order.

<sup>1</sup>Since this simple test might fail, users are encouraged to actively specify whether the function is noisy.

<sup>2</sup>Unit trace (sum of diagonal entries) for  $\Sigma$  implies that a draw  $\sim \mathcal{N}(0, \Sigma)$  has unit expected squared length.

**Set of polling directions** At the beginning of the POLL stage, we generate a preliminary set of directions  $\mathbf{D}'_k$  according to the random LTMADS algorithm [15]. We then transform it to a *rescaled* set  $\mathbf{D}_k$  based on the current GP kernel length scales: for  $\mathbf{v}' \in \mathbf{D}'_k$ , we define a rescaled vector  $\mathbf{v}$  with  $v_d \equiv v'_d \cdot \omega_d$ , for  $1 \leq d \leq D$ , and  $\omega_d \equiv \min\{\max\{10^{-6}, \Delta_k^{\text{mesh}}, \ell_d/\text{GM}(\ell)\}, \text{UB}_d - \text{LB}_d\}$ , where  $\text{GM}(\cdot)$  denotes the geometric mean, and we use  $\text{PLB}_d$  (resp.  $\text{PUB}_d$ ) whenever  $\text{UB}_d$  (resp.  $\text{LB}_d$ ) is unbounded. This construction of  $\mathbf{D}_k$  deviates from the standard MADS framework. However, since the applied rescaling is bounded, we could redefine the mesh parameters and the set of polling directions to accomodate our procedure (as long as we appropriately discretize  $\mathbf{D}_k$ ). We remove from the poll set points that violate constraints, if present.

**Polling order** Since the POLL is opportunistic, we evaluate points in the poll set starting from most promising, according to the ranking given by the chosen acquisition function [16].

#### B.4 Update and termination

If the SEARCH stage was successful in finding a sufficient improvement, we skip the POLL, move the incumbent and start a new iteration, without changing the mesh size (note that mesh expansion under a success is not required in the MADS framework [15]). If the POLL stage was executed, we verify if overall the iteration was successful or not, update the incumbent in case of success, and double (halven, in case of failure) the mesh size ( $\tau = 2$ ). If the optimization has been *stalling* (no sufficient improvement) for more than three iterations, we *accelerate* the mesh contraction by temporarily switching to  $\tau = 4$ .

The optimization stops when one of these conditions is met:

- the poll size  $\Delta_k^{\text{poll}}$  goes below a threshold  $\Delta_{\min}^{\text{poll}}$  (default  $10^{-6}$ );
- the maximum number of objective evaluations is reached (default  $500 \times D$ );
- the algorithm is *stalling*, that is there has no *sufficient* improvement of the objective  $f$ , for more than  $4 + \lfloor D/2 \rfloor$  iterations.

The algorithm returns the optimum  $\mathbf{x}_{\text{end}}$  (transformed back to original coordinates) that has the lowest objective value  $y_{\text{end}}$ . For a noisy objective, we return instead the stored point with the lowest quantile  $q_\beta$  across iterations, with  $\beta = 0.999$ ; see Section 3.4 in the main text. We also return the function value at the optimum,  $y_{\text{end}}$ , or, for a noisy objective, our estimate thereof (see below, Section B.5). See the online documentation for more information about the returned outputs.

#### B.5 Noisy objective

For noisy objectives, we change the behavior and default parameters of the algorithm to offset measurement uncertainty and allow for an accurate local approximation of  $f$ . First, we:

- double the minimum number of points added to the GP training set,  $n_{\min} = 100$ ;
- increase the total number of points (within radius  $\rho$ ) to at least 200, regardless of  $D$ ;
- increase the initial design set size to  $n_{\text{init}} = 20$  points;
- double the number of allowed stalled iterations before stopping.

**Uncertainty handling** The main difference with a deterministic objective is that, due to observation noise, we cannot simply use the output values  $y_i$  as ground truth in the SEARCH and POLL stages. Instead, we adopt a *plugin* approach [17] and replace  $y_i$  with the GP latent quantile function  $q_\beta$  [18] (see Eq. 3 in the main text). Moreover, we modify the MADS procedure by keeping an *incumbent set*  $\{\mathbf{x}_i\}_{i=1}^k$ , where  $\mathbf{x}_i$  is the incumbent at the end of the  $i$ -th iteration. At the end of each POLL stage, we re-evaluate  $q_\beta$  for all elements of the incumbent set, in light of the new points added to the cache which might change the GP prediction. We select as current (active) incumbent the point with lowest  $q_\beta(\mathbf{x}_i)$ . During optimization, we set  $\beta = 0.5$  (mean prediction only), which promotes exploration. For the last iteration, we instead use a conservative  $\beta_{\text{end}} = 0.999$  to select the optimum  $\mathbf{x}_{\text{end}}$  returned by the algorithm in a robust manner. For a noisy objective, instead of the noisy measurement  $y_{\text{end}}$ , we return either our best GP prediction  $\mu(\mathbf{x}_{\text{end}})$  and its uncertainty  $s(\mathbf{x}_{\text{end}})$ , or, more conservatively, an estimate of  $\mathbb{E}[f(\mathbf{x}_{\text{end}})]$  and its standard error, obtained by averaging  $N_{\text{final}}$  function evaluations

at  $\mathbf{x}_{\text{end}}$  (default  $N_{\text{final}} = 10$ ). The latter approach is a safer option to obtain an unbiased value of  $\mathbb{E}[f(\mathbf{x}_{\text{end}})]$ , since the GP approximation may occasionally fail or have substantial bias.

**Noise estimate** The user can optionally provide a noise estimate  $\sigma_{\text{est}}$  which is used to set the mean of the hyperprior over the observation noise  $\sigma$  (see Table S1). We recommend to set  $\sigma_{\text{est}}$  to the standard deviation of the noisy objective in the proximity of a good solution. If the problem has tunable precision (e.g., number of samples for log likelihoods evaluated via Monte Carlo), we recommend to set it, compatibly with computational cost, such that the standard deviation of noisy evaluations in the neighborhood of a good solution is of order 1.

## C Benchmark

We tested the performance of BADS on a large set of artificial and real problems and compared it with that of many optimization methods with implementation available in MATLAB (R2015b, R2017a).<sup>3</sup> We include here details that expand on Section 4.1 of the main text.

### C.1 Algorithms

Package	Algorithm	Source	Ref.	Noise	Global
badS	Bayesian Adaptive Direct Search	GitHub page <sup>4</sup>	This	✓	≈
fminsearchbnd	Nelder-Mead (fminsearch) w/ bounded domain	File Exchange <sup>5</sup>	[19]	✗	✗
cmaes	Covariance Matrix Adaptation Evolution Strategy	Author's website <sup>6</sup>	[13]	✗	≈
— (active)	CMA-ES with active covariance adaptation	—	[20]	✗	≈
— (noise)	CMA-ES with uncertainty handling	—	[21]	✓	≈
mcs	Multilevel Coordinate Search	Author's website <sup>7</sup>	[22]	✗	✓
snobfit	Stable Noisy Optimization by Branch and FIT	Author's website <sup>8</sup>	[23]	✓	✓
global	GLOBAL	Author's website <sup>9</sup>	[24]	✗	✓
randsearch	Random search	GitHub page <sup>10</sup>	[25]	✗	✓
fmincon	Interior point (interior-point, default)	Opt. Toolbox	[26]	✗	✗
— (sqp)	Sequential quadratic programming	—	[27]	✗	✗
— (active-set)	Active-set	—	[28]	✗	✗
patternsearch	Pattern search	Global Opt. Toolbox	[29]	✗	✗
ga	Genetic algorithms	Global Opt. Toolbox	[30]	✗	≈
particleswarm	Particle swarm	Global Opt. Toolbox	[31]	✗	≈
simulannealbnd	Simulated annealing w/ bounded domain	Global Opt. Toolbox	[32]	✗	≈
bayesopt	Vanilla Bayesian optimization	Stats. & ML Toolbox	[1]	✓	✓

Table S3: **Tested algorithms.** *Top:* Freely available algorithms. *Bottom:* Algorithms in MATLAB's Optimization, Global Optimization, and Statistics and Machine Learning toolboxes. For all algorithms we note whether they explicitly deal with noisy objectives (*noise* column), and whether they are local or global algorithms (*global* column). Global methods (✓) potentially search the full space, whereas local algorithms (✗) can only find a local optimum, and need a multi-start strategy. We denote with (≈) *semi-local* algorithms with intermediate behavior – semi-local algorithms might be able to escape local minima, but still need a multi-start strategy.

The list of tested algorithms is reported in Table S3. For all methods, we used their default options unless stated otherwise. For BADS, CMA-ES, and bayesopt, we activated their *uncertainty handling* option when dealing with noisy problems (for CMA-ES, see [21]). For noisy problems of the CCN17 set, within the fmincon family, we only tested the best representative method (active-set), since we found that these methods perform comparably to random search on noisy problems (see Fig S1

<sup>3</sup>MATLAB's bayesopt optimizer was tested on version R2017a, since it is not available for R2015b.

<sup>4</sup><https://github.com/lacerbi/bads>

<sup>5</sup><https://www.mathworks.com/matlabcentral/fileexchange/8277-fminsearchbnd--fminsearchcon>.

<sup>6</sup>[https://www.lri.fr/~hansen/cmaes\\_inmatlab.html](https://www.lri.fr/~hansen/cmaes_inmatlab.html)

<sup>7</sup><https://www.mat.univie.ac.at/~neum/software/mcs/>

<sup>8</sup><http://www.mat.univie.ac.at/~neum/software/snobfit/>

<sup>9</sup>[http://www.inf.u-szeged.hu/~csendes/index\\_en.html](http://www.inf.u-szeged.hu/~csendes/index_en.html)

<sup>10</sup><https://github.com/lacerbi/neurobench/tree/master/matlab/algorithms>

right, and Fig 1, right panel, in the main text). For the combinatorial game-playing problem subset in the CCN17 test set, we used the settings of MCS provided by the authors as per the original study [33]. We note that we developed algorithmic details and internal settings of BADS by testing it on the CEC14 test set for expensive optimization [34] and on other model-fitting problems which differ from the test problems presented in this benchmark. For bayesopt, we allowed up to 300 training points for the GP, restarting the BO algorithm from scratch with a different initial design every 300 BO iterations (until the total budget of function evaluations was exhausted). The choice of 300 iterations already produced a large average algorithmic overhead of  $\sim 8$  s per function evaluation. As acquisition function, we used the default EI-per-second [1], except for problems for which the computational cost is constant across all parameter space, for which we used the simple EI. All algorithms in Table S3 accept *hard* bound constraints lb, ub, which were provided with the BBOB09 set and with the original studies in the CCN17 set. For all studies in the CCN17 set we also asked the original authors to provide *plausible* lower/upper bounds plb, pub for each parameter, which we would use for all problems in the set (if not available, we used the hard bounds instead). For all algorithms, plausible bounds were used to generate starting points. We also used plausible bounds (or their range) as inputs for algorithms that allow the user to provide additional information to guide the search, e.g. the length scale of the covariance matrix in CMA-ES, the initialization box for MCS, and plausible bounds in BADS.

## C.2 Procedure

For all problems and algorithms, for the purpose of our benchmark, we first transformed the problem variables according to the mapping described in ‘Transformation of variables and constraints’ (Section B.1). In particular, this transformation maps the plausible region to the  $[-1, 1]^D$  hypercube, and transforms to log space positive variables that span more than one order of magnitude. This way, all methods dealt with the same standardized domains. Starting points during each optimization run were drawn uniformly randomly from inside the box of provided plausible bounds.

For deterministic problems, during each optimization run we kept track of the best (lowest) function value  $y_{\text{best}}^t$  found so far after  $t$  function evaluations. We define the *immediate regret* (or error) at time  $t$  as  $y_{\text{best}}^t - y_{\text{min}}$ , where  $y_{\text{min}}$  is the true minimum or our best estimate thereof, and we use the error to judge whether the run is a success at step  $t$  (error less than a given tolerance  $\varepsilon$ ). For problems in the BBOB09 set (both noiseless and noisy variants), we know the ground truth  $y_{\text{min}}$ . For problems in the CCN17 set, we do not know  $y_{\text{min}}$ , and we *define* it as the minimum function value found across all optimization runs of all algorithms ( $\approx 3.75 \cdot 10^5 \times D$  function evaluations per noiseless problem), with the rationale that it would be hard to beat this computational effort. We report the *effective* performance of an algorithm with non-negligible fractional overhead  $o > 0$  by plotting at step  $t \times o$  its performance at step  $t$ , which corresponds to a shift of the performance curve when  $t$  is plotted in log scale (Fig 2 in the main text).<sup>11</sup>

For noisy problems, we care about the true function value(s) at the point(s) returned by the algorithm, since, due to noise, it is possible for an algorithm to visit a neighborhood of the solution during the course of the optimization but then return another point. For each noisy optimization run, we allowed each algorithm to return up to three solutions, obtained either from multiple sub-runs, or from additional outputs available from the algorithm, such as with MCS, or with population-based methods (CMA-ES, ga, and particleswarm). If more than three candidate solutions were available, we gave precedence to the main output of the algorithm, and then we took the two additional solutions with lowest observed function value. We limited the number of candidates per optimization run to allow for a fair comparison between methods, since some methods only return one point and others potentially hundreds (e.g., ga) – under the assumption that evaluating the true value of the log likelihood for a given candidate would be costly. For the combinatorial game-playing problem subset in the CCN17 set, we increased the number of allowed solutions per run to 10 to match the strategy used in the original study [33]. For noisy problems in the CCN17 set, we estimated the log likelihood at each provided candidate solution via 200 function evaluations, and took the final estimate with lowest average.

<sup>11</sup>We did not apply this correction when plotting the results of vanilla BO (bayesopt), since the algorithm’s performance is already abysmal even without accounting for the substantial overhead.



For plotting, we determined ranking of the algorithms in the legend proportionally to the overall performance (area under the curve), across iterations (deterministic problems) or across error tolerances (noisy problems.)

### C.3 Alternative benchmark parameters

In our benchmark, we made some relatively arbitrary choices to assess algorithmic performance, such as the range of tolerances  $\varepsilon$  or the number of function evaluations. We show here that our findings are robust to variations in these parameters, by plotting results from the BBOB09 set with a few key changes (see Fig 1 in the main text for comparison). First, we restrict the error tolerance range for deterministic functions to  $\varepsilon \in [0.1, 1]$  instead of the wider range  $\varepsilon \in [0.01, 10]$  used in the main text (Fig S1 left and middle). This narrower range covers realistic practical requirements for model selection. Second, we reran the BBOB09 noisy benchmark, allowing  $500 \times D$  functions evaluation, as opposed to  $200 \times D$  in the main text (Fig S1 right). Our main conclusions do not change, in that BADS performs on par with or better than other algorithms.

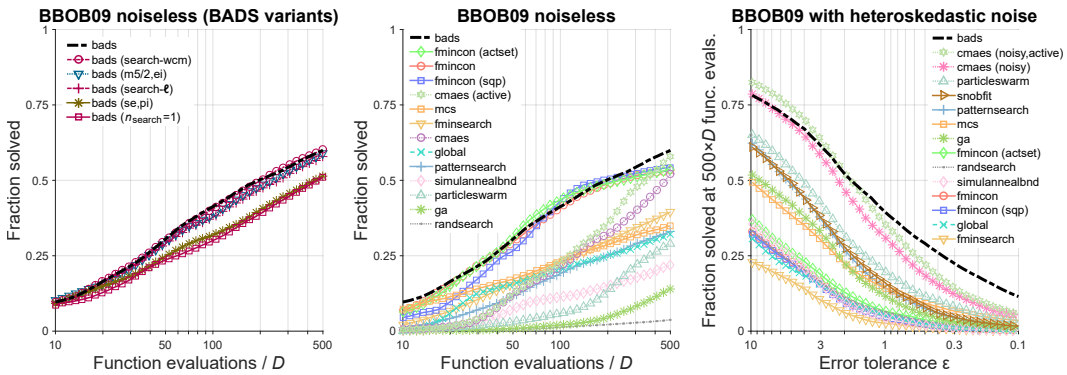


Figure S1: **Artificial test functions (BBOB09)**. Same as Fig 1 in the main text, but with with alternative benchmark parameters (in bold). *Left & middle*: Noiseless functions. Fraction of successful runs ( $\varepsilon \in [0.1, 1]$ ) vs. # function evaluations per # dimensions, for  $D \in \{3, 6, 10, 15\}$  (96 test functions); for different BADS configurations (*left*) and all algorithms (*middle*). *Right*: Heteroskedastic noise. Fraction of successful runs at  $500 \times D$  objective evaluations vs. tolerance  $\varepsilon$ .

## D Numerical implementation

BADS is currently freely available as a MATLAB toolbox, `bads` (a Python version is planned).

The basic design of `bads` is simplicity and accessibility for the non-expert end user. First, we adopted an interface that resembles that of other common MATLAB optimizers, such as `fminsearch` or `fmincon`. Second, `bads` is *plug-and-play*, with no requirements for installation of additional toolboxes or compiling C/C++ code via mex files, which usually requires specific expertise. Third, `bads` hides most of its complexity under the hood, providing the standard user with thoroughly tested default options that need no tweaking.

For the expert user or developer, `bads` has a modular design, such that POLL set generation, the SEARCH oracle, acquisition functions (separately for SEARCH and POLL), and initial design can be freely selected from a large list (under development), and new options are easy to add.

**GP implementation** We based our GP implementation in MATLAB on the GPML Toolbox [35] (v3.6), modified for increased efficiency of some algorithmic steps, such as computation of gradients,<sup>12</sup> and we added specific functionalities. We optimize the GP hyperparameters with `fmincon` in MATLAB (if the Optimization Toolbox is available), or otherwise via a the `minimize` function provided with the GPML package, modified to support bound constraints.

<sup>12</sup>We note that version 4.0 of the GPML toolbox was released while BADS was in development. GPML v4.0 solved efficiency issues of previous versions, and might be supported in future versions of BADS.

## Supplementary references

- [1] Snoek, J., Larochelle, H., & Adams, R. P. (2012) Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems* **24**, 2951–2959.
- [2] Goris, R. L., Simoncelli, E. P., & Movshon, J. A. (2015) Origin and function of tuning diversity in macaque visual cortex. *Neuron* **88**, 819–831.
- [3] Quiñero Candela, J., Rasmussen, C. E., & Williams, C. K. (2007) Approximation methods for Gaussian process regression. *Large-scale kernel machines* pp. 203–224.
- [4] Rasmussen, C. & Williams, C. K. I. (2006) *Gaussian Processes for Machine Learning*. (MIT Press).
- [5] Gramacy, R. B. & Lee, H. K. (2012) Cases for the nugget in modeling computer experiments. *Statistics and Computing* **22**, 713–722.
- [6] Royston, J. (1982) An extension of Shapiro and Wilk’s W test for normality to large samples. *Applied Statistics* pp. 115–124.
- [7] Srinivas, N., Krause, A., Seeger, M., & Kakade, S. M. (2010) Gaussian process optimization in the bandit setting: No regret and experimental design. *ICML-10* pp. 1015–1022.
- [8] Kushner, H. J. (1964) A new method of locating the maximum point of an arbitrary multippeak curve in the presence of noise. *Journal of Basic Engineering* **86**, 97–106.
- [9] Lizotte, D. J. (2008) Ph.D. thesis (University of Alberta).
- [10] Mockus, J., Tiesis, V., & Zilinskas, A. (1978) in *Towards Global Optimisation*. (North-Holland Amsterdam), pp. 117–129.
- [11] Jones, D. R., Schonlau, M., & Welch, W. J. (1998) Efficient global optimization of expensive black-box functions. *Journal of Global Optimization* **13**, 455–492.
- [12] Bratley, P. & Fox, B. L. (1988) Algorithm 659: Implementing Sobol’s quasirandom sequence generator. *ACM Transactions on Mathematical Software (TOMS)* **14**, 88–100.
- [13] Hansen, N., Müller, S. D., & Koumoutsakos, P. (2003) Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation* **11**, 1–18.
- [14] Hoffman, M. D., Brochu, E., & de Freitas, N. (2011) Portfolio allocation for Bayesian optimization. *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence* pp. 327–336.
- [15] Audet, C. & Dennis Jr, J. E. (2006) Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on optimization* **17**, 188–217.
- [16] Gramacy, R. B. & Le Digabel, S. (2015) The mesh adaptive direct search algorithm with treed Gaussian process surrogates. *Pacific Journal of Optimization* **11**, 419–447.
- [17] Picheny, V., Wagner, T., & Ginsbourger, D. (2013) A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization* **48**, 607–626.
- [18] Picheny, V., Ginsbourger, D., Richet, Y., & Caplin, G. (2013) Quantile-based optimization of noisy computer experiments with tunable precision. *Technometrics* **55**, 2–13.
- [19] Lagarias, J. C., Reeds, J. A., Wright, M. H., & Wright, P. E. (1998) Convergence properties of the Nelder–Mead simplex method in low dimensions. *SIAM Journal on Optimization* **9**, 112–147.
- [20] Jastrebski, G. A. & Arnold, D. V. (2006) Improving evolution strategies through active covariance matrix adaptation. *IEEE Congress on Evolutionary Computation (CEC 2006)*. pp. 2814–2821.
- [21] Hansen, N., Niederberger, A. S., Guzzella, L., & Koumoutsakos, P. (2009) A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation* **13**, 180–197.
- [22] Huyer, W. & Neumaier, A. (1999) Global optimization by multilevel coordinate search. *Journal of Global Optimization* **14**, 331–355.
- [23] Huyer, W. & Neumaier, A. (2008) SNOBFIT—stable noisy optimization by branch and fit. *ACM Transactions on Mathematical Software (TOMS)* **35**, 9.
- [24] Csendes, T., Pál, L., Sendin, J. O. H., & Banga, J. R. (2008) The GLOBAL optimization method revisited. *Optimization Letters* **2**, 445–454.
- [25] Bergstra, J. & Bengio, Y. (2012) Random search for hyper-parameter optimization. *Journal of Machine Learning Research* **13**, 281–305.
- [26] Waltz, R. A., Morales, J. L., Nocedal, J., & Orban, D. (2006) An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming* **107**, 391–408.
- [27] Nocedal, J. & Wright, S. (2006) *Numerical Optimization*, Springer Series in Operations Research. (Springer Verlag), 2nd edition.

- [28] Gill, P. E., Murray, W., & Wright, M. H. (1981) *Practical Optimization*. (Academic press).
- [29] Kolda, T. G., Lewis, R. M., & Torczon, V. (2003) Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review* **45**, 385–482.
- [30] Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization & Machine Learning*. (Addison-Wesley).
- [31] Eberhart, R. & Kennedy, J. (1995) A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995 (MHS'95)*. pp. 39–43.
- [32] Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., et al. (1983) Optimization by simulated annealing. *Science* **220**, 671–680.
- [33] van Opheusden, B., Bnaya, Z., Galbiati, G., & Ma, W. J. (2016) Do people think like computers? *International Conference on Computers and Games* pp. 212–224.
- [34] Liang, J., Qu, B., & Suganthan, P. (2013) Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization.
- [35] Rasmussen, C. E. & Nickisch, H. (2010) Gaussian processes for machine learning (GPML) toolbox. *Journal of Machine Learning Research* **11**, 3011–3015.