arXiv:1705.04405v1 [stat.ML] 11 May 2017

Practical Bayesian Optimization for Model Fitting with Bayesian Adaptive Direct Search

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

Abstract

Computational models in fields such as computational neuroscience are often evaluated via stochastic simulation or numerical approximation. Fitting these models implies a difficult optimization problem over complex, possibly noisy parameter landscapes. Bayesian optimization (BO) has been successfully applied to solving expensive black-box problems in engineering and machine learning. Here we explore whether BO can be applied as a general tool for model fitting. First, we present a novel BO algorithm, Bayesian adaptive direct search (BADS), that achieves competitive performance with an affordable computational overhead for the running time of typical models. We then perform an extensive benchmark of BADS vs. many common and state-of-the-art nonconvex, derivative-free optimizers on a set of model-fitting problems with real data and models from six studies in behavioral, cognitive, and computational neuroscience. With default settings, BADS consistently finds comparable or better solutions than other methods, showing great promise for BO, and BADS in particular, as a general model-fitting tool.

1 Introduction

Many complex, nonlinear computational models in fields such as behaviorial, cognitive, and computational neuroscience cannot be evaluated analytically, but require moderately expensive numerical approximations or simulations. In these cases, finding the maximum-likelihood (ML) solution – for parameter estimation, or model selection – requires the costly exploration of a rough or noisy nonconvex landscape, in which gradients are often unavailable to guide the search.

Here we consider the problem of finding the (global) optimum $\boldsymbol{x}^* = \text{argmin}_{\boldsymbol{x}\in\mathcal{X}}\mathbb{E}\left[f(\boldsymbol{x})\right]$ of a possibly noisy *objective* f over a (bounded) domain $\mathcal{X} \subseteq \mathbb{R}^D$, where the function f can be intended as the (negative) log likelihood of a parameter vector $\boldsymbol{x}$ for a given dataset and model, but is generally a *black box*. With many derivative-free optimization algorithms available to the researcher [1], it is unclear which one should be chosen. Crucially, an inadequate optimizer can hinder progress, limit the complexity of the models that can be fit, and even cast doubt on the reliability of one's findings.

Bayesian optimization (BO) is a state-of-the-art machine learning framework for optimizing expensive and possibly noisy black-box functions [2, 3, 4]. This makes it an ideal candidate for solving difficult model-fitting problems. Yet there are several obstacles to a widespread usage of BO as a general tool for model fitting. First, traditional BO methods target *very* costly problems, such as hyperparameter tuning [5], whereas evaluating a typical behavioral model might only have a moderate computational cost (e.g., 0.1-10 s per evaluation). This implies major differences in what is considered an acceptable algorithmic overhead, and in the maximum number of allowed function evaluations (e.g., hundreds vs. thousands). Second, it is unclear how BO methods would fare in this regime against commonly used and state-of-the-art, non-Bayesian optimizers. Finally, BO might be perceived by non-practitioners as an advanced tool that requires specific technical knowledge to be implemented or tuned.

*Alternative email address: luigi.acerbi@gmail.com.

We address these issues by developing a novel BO algorithm, Bayesian Adaptive Direct Search (BADS), that achieves competitive performance at a small computational cost. We tested BADS, together with a wide array of commonly used optimizers, on a novel benchmark set of model-fitting problems with real data and models drawn from studies in cognitive, behaviorial and computational neuroscience. Finally, we make BADS available as a free MATLAB package with the same user interface as existing optimizers and that can be used out-of-the-box with no tuning.[2]

BADS combines the mesh adaptive direct search (MADS) framework [6] (Section 2.1) with a BO search performed via a local Gaussian process (GP) surrogate (Section 2.2), implemented via a number of heuristics for efficiency (Section 3). BADS proves to be highly competitive on both artificial functions and real-world model-fitting problems (Section 4), showing that BO, and BADS in particular, is a viable general tool for model fitting in computational neuroscience and related fields.

Related work There is a large literature about (Bayesian) optimization of expensive, possibly stochastic, computer simulations, mostly used in machine learning [3, 4, 5] or engineering (known as *kriging-based* optimization) [7, 8, 9]. Recent work has combined MADS with treed GP models for constrained optimization (TGP-MADS [9]). Crucially, these methods have large overheads and may require problem-specific tuning, making them impractical as a generic tool for model fitting.

2 Optimization frameworks

2.1 Mesh adaptive direct search (MADS)

The MADS algorithm is a directional direct search framework for nonlinear optimization [6, 10]. Briefly, MADS seeks to improve the current solution by testing points in the neighborhood of the current point (the *incumbent*), by moving one step in each direction on an iteration-dependent mesh. In addition, the MADS framework can incorporate in the optimization any arbitrary search strategy which proposes additional test points that lie on the mesh.

MADS defines the current mesh at the k-th iteration as $M_k = \bigcup_{\boldsymbol{x} \in S_k} \left\{ \boldsymbol{x} + \Delta_k^{\text{mesh}} \mathbf{D} \boldsymbol{z} : \boldsymbol{z} \in \mathbb{N}^D \right\}$, where $S_k \subset \mathbb{R}^n$ is the set of all points evaluated since the start of the iteration, $\Delta_k^{\text{mesh}} \in \mathbb{R}_+$ is the *mesh size*, and $\mathbf{D}$ is a fixed matrix in $\mathbb{R}^{D \times n_{\mathbf{D}}}$ whose $n_{\mathbf{D}}$ columns represent viable search directions. We choose $\mathbf{D} = [\mathbf{I}_D, -\mathbf{I}_D]$, where $\mathbf{I}_D$ is the identity matrix in dimension D.

Each iteration of MADS comprises of two stages, a SEARCH stage and an optional POLL stage. The SEARCH stage evaluates a finite number of points proposed by a provided search strategy, with the only restriction that the tested points lie on the current mesh. The search strategy is intended to inject problem-specific information in the optimization. In BADS, we exploit the freedom of SEARCH to perform Bayesian optimization in the neighborhood of the incumbent (see Section 2.2 and 3.3). The POLL stage is performed if the SEARCH fails in finding a point with an improved objective value. POLL constructs a *poll set* of candidate points, P_k, defined as $P_k = \left\{ \boldsymbol{x}_k + \Delta_k^{\text{mesh}} \boldsymbol{v} : \boldsymbol{v} \in \mathbf{D}_k \right\}$, where $\boldsymbol{x}_k$ is the incumbent and $\mathbf{D}_k$ is the set of *polling directions* constructed by taking discrete linear combinations of the set of directions $\mathbf{D}$. The *poll size* parameter $\Delta_k^{\text{poll}} \geq \Delta_k^{\text{mesh}}$ defines the maximum length of poll displacement vectors $\Delta_k^{\text{mesh}} \boldsymbol{v}$, for $\boldsymbol{v} \in \mathbf{D}_k$ (typically, $\Delta_k^{\text{poll}} \approx \Delta_k^{\text{mesh}} ||\boldsymbol{v}||$). Points in the poll set can be evaluated in any order, and the POLL is opportunistic in that it can be stopped as soon as a better solution is found. The POLL stage ensures theoretical convergence to a local stationary point according to Clarke calculus for nonsmooth functions [6, 11].

If either SEARCH or POLL are a *success*, finding a mesh point with an improved objective value, the incumbent is updated and the mesh size remains the same or is multiplied by a factor $\tau > 1$. If neither SEARCH or POLL are successful, the incumbent does not move and the mesh size is divided by τ. The algorithm proceeds until a stopping criterion is met (e.g., maximum budget of function evaluations).

2.2 Bayesian optimization

The typical form of Bayesian optimization (BO) [2] builds a Gaussian process (GP) approximation of the objective f, which is used as a relatively inexpensive surrogate to guide the search towards regions that are promising (low GP mean) and/or unknown (high GP uncertainty), according to a rule, the acquisition function, that formalizes the exploitation-exploration trade-off.

[2] Code available at `https://github.com/lacerbi/bads`.

Gaussian processes GPs are a flexible class of models for specifying prior distributions over unknown functions $f : \mathcal{X} \subseteq \mathbb{R}^D \to \mathbb{R}$ [12]. GPs are specified by a mean function $m : \mathcal{X} \to \mathbb{R}$ and a positive definite covariance, or kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Given any finite collection of n points $\mathbf{X} = \left\{\boldsymbol{x}^{(i)} \in \mathcal{X}\right\}_{i=1}^{n}$, the value of f at these points is assumed to be jointly Gaussian with mean $(m(\boldsymbol{x}^{(1)}), \ldots, m(\boldsymbol{x}^{(n)}))^\top$ and covariance matrix $\mathbf{K}$, where $\mathbf{K}_{ij} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ for $1 \leq i, j \leq n$. We assume i.i.d. Gaussian observation noise such that f evaluated at $\boldsymbol{x}^{(i)}$ returns $y^{(i)} \sim \mathcal{N}\left(f(\boldsymbol{x}^{(i)}), \sigma^2\right)$, and $\boldsymbol{y} = (y^{(1)}, \ldots, y^{(n)})^\top$ is the vector of observed values. For a deterministic f, we still assume a small $\sigma > 0$ to improve numerical stability of the GP [13]. Conveniently, observation of such (noisy) function values will produce a GP posterior whose latent marginal conditional mean $\mu(\boldsymbol{x}; \{\mathbf{X}, \boldsymbol{y}\}, \boldsymbol{\theta})$ and variance $s^2(\boldsymbol{x}; \{\mathbf{X}, \boldsymbol{y}\}, \boldsymbol{\theta})$ at a given point are available in closed form (see Supplementary Material), where $\boldsymbol{\theta}$ is a hyperparameter vector for the mean, covariance, and likelihood. In the following, we omit the dependency of μ and s^2 from the data and GP parameters to reduce clutter.

Covariance functions Our main choice of stationary (translationally-invariant) covariance function is the automatic relevance determination (ARD) *rational quadratic* (RQ) kernel,

$$k_{\text{RQ}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \left[1 + \frac{1}{2\alpha} r^2(\boldsymbol{x}, \boldsymbol{x}')\right]^{-\alpha}, \qquad \text{with} \quad r^2(\boldsymbol{x}, \boldsymbol{x}') = \sum_{d=1}^{D} \frac{1}{\ell_d^2} (x_d - x_d')^2, \tag{1}$$

where σ_f^2 is the signal variance, $\ell_1, \ldots, \ell_D$ are the kernel length scales along each coordinate direction, and $\alpha > 0$ is the shape parameter. More common choices for Bayesian optimization include the *squared exponential* (SE) kernel [9] or the twice-differentiable ARD *Matérn 5/2* ($\text{M}_{5/2}$) kernel [5], but we found the RQ kernel to work best in combination with our method (see Section 4.2). We also consider *composite periodic kernels* for circular or periodic variables (see Supplementary Material).

Acquisition function For a given GP approximation of f, the *acquisition function*, $a : \mathcal{X} \to \mathbb{R}$, determines which point in $\mathcal{X}$ should be evaluated next via a proxy optimization $\boldsymbol{x}_{\text{next}} = \text{argmin}_{\boldsymbol{x}} a(\boldsymbol{x})$. We consider here the *GP lower confidence bound* (LCB) metric [14],

$$a_{\text{LCB}}(\boldsymbol{x}; \{\mathbf{X}, \boldsymbol{y}\}, \boldsymbol{\theta}) = \mu(\boldsymbol{x}) - \sqrt{\nu \beta_t s^2(\boldsymbol{x})}, \qquad \beta_t = 2 \ln\left(D t^2 \pi^2 / (6\delta)\right) \tag{2}$$

where $\nu > 0$ is a tunable parameter, t is the number of function evaluations so far, $\delta > 0$ is a probabilistic tolerance, and β_t is a learning rate chosen to minimize cumulative regret under certain assumptions. For BADS we use the recommended values $\nu = 0.2$ and $\delta = 0.1$ [14]. Another popular choice is the (negative) *expected improvement* (EI) over the current best function value [15], and an historical, less used metric is the (negative) *probability of improvement* (PI) [16].

3 Bayesian adaptive direct search (BADS)

We describe here the main steps of BADS (Algorithm 1). Briefly, BADS alternates between a series of fast, local BO steps (the SEARCH stage of MADS) and, when the search repeatedly fails, a systematic, slower exploration of the mesh grid (POLL stage). See Supplementary Material for a full description.

3.1 Initial setup

Problem specification The algorithm is initialized by providing a starting point $\boldsymbol{x}_0$, vectors of *hard* lower/upper bounds `LB`, `UB`, and optional vectors of *plausible* lower/upper bounds `PLB`, `PUB`, with the requirement that for each dimension $1 \leq d \leq D$, $\texttt{LB}_d \leq \texttt{PLB}_d < \texttt{PUB}_d \leq \texttt{UB}_d$. Plausible bounds identify a region in parameter space where most solutions are expected to lie. Hard upper/lower bounds can be infinite, but plausible bounds need to be finite. Problem variables whose hard bounds are strictly positive and $\texttt{UB}_d \geq 10 \cdot \texttt{LB}_d$ are automatically converted to log space. All variables are then linearly rescaled to the standardized box $[-1, 1]^D$ such that the box bounds correspond to $[\texttt{PLB}, \texttt{PUB}]$ in the original space. BADS supports bound or no constraints, and optionally other constraints via a provided *barrier* function c (see Supplementary Material). The user can also specify circular or periodic dimensions (such as angles); and whether the objective f is deterministic or noisy (stochastic), and in the latter case provide a coarse estimate of the noise (see Section 3.4).

Initial design The initial design consists of the provided starting point $\boldsymbol{x}_0$ and $n_{\text{init}} = D$ additional points chosen via a space-filling quasi-random Sobol sequence [17] in the standardized box, and forced to lie 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 $\boldsymbol{x}_0$.

Algorithm 1 Bayesian Adaptive Direct Search

Input: objective function f, starting point $\boldsymbol{x}_\mathbf{0}$, hard bounds `LB`, `UB`, (*optional*: plausible bounds `PLB`, `PUB`, barrier function c, additional `options`)

Initialization: $\Delta_0^{\text{mesh}} \leftarrow 2^{-10}$, $\Delta_0^{\text{poll}} \leftarrow 1$, $k \leftarrow 0$, evaluate f on initial design ▷ Section 3.1
repeat
  (update GP approximation at any step; refit hyperparameters if necessary) ▷ Section 3.2
  for $1 \dots n_{\text{search}}$ **do** ▷ SEARCH stage, Section 3.3
    $\boldsymbol{x}_{\text{search}} \leftarrow$ SEARCHORACLE ▷ local Bayesian optimization step
    Evaluate f on $\boldsymbol{x}_{\text{search}}$, **if** improvement is *sufficient* **then break**
  if SEARCH is NOT *successful* **then** ▷ optional POLL stage, Section 3.3
    compute poll set P_k
    evaluate opportunistically f on P_k sorted by acquisition function
  if iteration k is *successful* **then**
    update incumbent $\boldsymbol{x}_{k+1}$
    if POLL was *successful* **then** $\Delta_k^{\text{mesh}} \leftarrow 2\Delta_k^{\text{mesh}}$, $\Delta_k^{\text{poll}} \leftarrow 2\Delta_k^{\text{poll}}$
  else
    $\Delta_k^{\text{mesh}} \leftarrow \frac{1}{2}\Delta_k^{\text{mesh}}$, $\Delta_k^{\text{poll}} \leftarrow \frac{1}{2}\Delta_k^{\text{poll}}$
  $k \leftarrow k + 1$
until `fevals` $>$ `MaxFunEvals` **or** $\Delta_k^{\text{poll}} < 10^{-6}$ **or** stalling ▷ stopping criteria
return $\boldsymbol{x}_{\text{end}} = \arg\min_k f(\boldsymbol{x}_k)$ (or $\boldsymbol{x}_{\text{end}} = \arg\min_k q_\beta(\boldsymbol{x}_k)$ for noisy objectives, Section 3.4)

3.2 GP model in BADS

The default GP model is specified by a constant mean function $m \in \mathbb{R}$, a smooth ARD RQ kernel (Eq. 1), and we use a_{LCB} (Eq. 2) as a default acquisition function.

Hyperparameters The default GP has hyperparameters $\boldsymbol{\theta} = (\ell_1, \dots, \ell_D, \sigma_f^2, \alpha, \sigma^2, m)$. We impose an empirical Bayes prior on the GP hyperparameters based on the current training set (see Supplementary Material), and select $\boldsymbol{\theta}$ via maximum a posteriori (MAP) estimation. We fit $\boldsymbol{\theta}$ via a gradient-based nonlinear optimizer, starting from either the previous value of $\boldsymbol{\theta}$ or a weighted draw from the prior, as a means to escape local optima. 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, according to a normality test on the residuals, $z^{(i)} = \left(y^{(i)} - \mu(\boldsymbol{x}^{(i)})\right) / \sqrt{s^2(\boldsymbol{x}^{(i)}) + \sigma^2}$ (assumed independent, in first approximation).

Training set The GP training set $\mathbf{X}$ consists of a subset of the points evaluated so far (the *cache*), selected to build a local approximation of the objective in the neighborhood of the incumbent $\boldsymbol{x}_k$. When $\mathbf{X}$ is rebuilt, points in the cache are sorted by their $\boldsymbol{\ell}$-scaled distance r^2 (Eq. 1) from $\boldsymbol{x}_k$. The closest $n_{\min} = 50$ points are automatically added to $\mathbf{X}$. Then, up to $10D$ additional points with $r \leq 3\rho(\alpha)$ are included in the set, where $\rho(\alpha) \gtrsim 1$ is a kernel-dependent radius function (see Supplementary Material). Newly evaluated points are added incrementally to the set, allowing for fast rank-one updates of the GP posterior. The training set is rebuilt any time the incumbent is moved.

3.3 Implementation of the MADS framework

We initialize $\Delta_0^{\text{poll}} = 1$ and $\Delta_0^{\text{mesh}} = 2^{-10}$ (in standardized space). We use $\tau = 2$, and increase the mesh size only after a successful POLL. We skip the POLL after a successful SEARCH.

Search stage The SEARCH consists of up to $n_{\text{search}} = \max\{D, \lfloor 3 + D/2 \rfloor\}$ SEARCH steps. In each step we use a *search oracle*, based on a local BO with the current GP, to produce a search point $\boldsymbol{x}_{\text{search}}$ (see below). We evaluate $f(\boldsymbol{x}_{\text{search}})$ and add it to the cache. If the improvement in objective value is none or *insufficient*, that is less than $(\Delta_k^{\text{poll}})^{3/2}$, we continue searching, otherwise we call it a *success*.

Search oracle We choose $\boldsymbol{x}_{\text{search}}$ via a fast, approximate optimization inspired by CMA-ES [18]. We sample batches of points in the neighborhood of the incumbent $\boldsymbol{x}_k$, drawn $\sim \mathcal{N}(\boldsymbol{x}_{\text{s}}, \lambda^2(\Delta_k^{\text{poll}})^2\boldsymbol{\Sigma})$, where $\boldsymbol{x}_{\text{s}}$ is the current search focus, $\boldsymbol{\Sigma}$ a *search covariance matrix*, and $\lambda > 0$ a scaling factor, and we pick the point that optimizes the acquisition function (see Supplementary Material). Across

SEARCH steps we use both a diagonal matrix $\boldsymbol{\Sigma}_{\boldsymbol{\ell}}$ with diagonal $\left(\ell_1^2/|\boldsymbol{\ell}|^2, \ldots, \ell_D^2/|\boldsymbol{\ell}|^2\right)$, and a matrix $\boldsymbol{\Sigma}_{\text{WCM}}$ 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). We choose between $\boldsymbol{\Sigma}_{\boldsymbol{\ell}}$ and $\boldsymbol{\Sigma}_{\text{WCM}}$ probabilistically via a *hedge* strategy, based on their track record of cumulative improvement [19].

Poll stage 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. We generate $\mathbf{D}_k$ according to the random LTMADS algorithm [6], but then rescale each vector coordinate $1 \leq d \leq D$ proportionally to the GP length scale ℓ_d (see Supplementary Material). Second, since the POLL is opportunistic, we evaluate points in the poll set according to the ranking given by the acquisition function [9].

Stopping criteria We stop the optimization when the poll size Δ_k^{poll} goes below a threshold (default 10^{-6}); when reaching a maximum number of objective evaluations (default $500D$); or if there is no significant improvement of the objective for more than $4 + \lfloor D/2 \rfloor$ iterations. The algorithm returns the optimum $\boldsymbol{x}_{\text{end}}$ (transformed back to original coordinates) with the lowest objective value y_{end}.

3.4 Noisy objective

In case of a noisy objective, we assume for the noise a hyperprior $\ln \sigma \sim \mathcal{N}(\ln \sigma_{\text{est}}, 1)$, with σ_{est} a base noise magnitude (default $\sigma_{\text{est}} = 1$, but the user can provide an estimate). To account for additional uncertainty, we also make the following changes: double the minimum number of points added to the training set, $n_{\min} = 100$, and increase the maximum number to 200; increase the initial design to $n_{\text{init}} = 20$; and double the number of allowed stalled iterations before stopping.

Uncertainty handling Due to noise, we cannot simply use the output values y_i as ground truth in the SEARCH and POLL stages. Instead, we replace y_i with the GP latent quantile function [20]

$$q_\beta\left(\boldsymbol{x}; \{\mathbf{X}, \boldsymbol{y}\}, \boldsymbol{\theta}\right) \equiv q_\beta(\boldsymbol{x}) = \mu\left(\boldsymbol{x}\right) + \Phi^{-1}(\beta) s\left(\boldsymbol{x}\right), \qquad \beta \in [0.5, 1), \tag{3}$$

where $\Phi^{-1}(\cdot)$ is the quantile function of the standard normal (*plugin* approach [21]). Moreover, we modify the MADS procedure by keeping an *incumbent set* $\{\boldsymbol{x}_i\}_{i=1}^k$, where $\boldsymbol{x}_i$ is the incumbent at the end of the i-th iteration. At the end of each POLL we re-evaluate q_β for all elements of the incumbent set, in light of the new points added to the cache. We select as current (active) incumbent the point with lowest $q_\beta(\boldsymbol{x}_i)$. During optimization we set $\beta = 0.5$ (mean prediction only), which promotes exploration. We use a conservative $\beta_{\text{end}} = 0.999$ for the last iteration, to select the optimum $\boldsymbol{x}_{\text{end}}$ returned by the algorithm in a robust manner. Instead of y_{end}, we return either $\mu(\boldsymbol{x}_{\text{end}})$ or an unbiased estimate of $\mathbb{E}[f(\boldsymbol{x}_{\text{end}})]$ obtained by averaging multiple evaluations (see Supplementary Material).

4 Experiments

We tested BADS and many optimizers with implementation available in MATLAB (R2015b) on a large set of artificial and real optimization problems (see Supplementary Material for details).

4.1 Design of the benchmark

Algorithms Besides BADS, we tested 15 optimization algorithms, including popular choices such as Nelder-Mead (`fminsearch` [22]), several constrained nonlinear optimizers in the `fmincon` function (default *interior-point* [23], *sequential quadratic programming* `sqp` [24], and *active-set* `actset` [25]), genetic algorithms (`ga` [26]), random search (`randsearch`) as a baseline [27]; and also less-known state-of-the-art methods for nonconvex derivative-free optimization [1], such as Multilevel Coordinate Search (`mcs` [28]) and CMA-ES [18, 29] (`cmaes`, in different flavors). For noisy objectives, we included algorithms that explicitly handle uncertainty, such as `snobfit` [30] and *noisy* CMA-ES [31]. For all algorithms, including BADS, we used default settings (no fine-tuning).

Problem sets First, we considered a standard benchmark set of artificial, noiseless functions (BBOB09 [32], 24 functions) in dimensions $D \in \{3, 6, 10, 15\}$, for a total of 96 test functions. We also created 'noisy' versions of the same set. Second, we collected model-fitting problems from six published or ongoing studies in cognitive and computational neuroscience (CCN17). The objectives of the CCN17 set are negative log likelihood functions of an input parameter vector, for specified datasets and models, and can be deterministic or stochastic. For each study in the CCN17 set we asked its authors for six different real datasets (i.e., subjects or neurons), divided between one or two main models of interest; collecting a total of 36 test functions with $D \in \{6, 9, 10, 12, 13\}$.

Procedure We ran 50 independent runs of each algorithm on each test function, with randomized starting points and a budget of $500 \times D$ function evaluations ($200 \times D$ for noisy problems). If an algorithm terminated before depleting the budget, it was restarted from a new random point. We consider a run *successful* if the current best (or returned, for noisy problems) function value is within a given *error tolerance* $\varepsilon > 0$ from the true optimum $f_{\min}$ (or our best estimate thereof). For noiseless problems, we compute the fraction of successful runs as a function of number of objective evaluations, averaged over datasets/functions and over $\varepsilon \in [0.01, 10]$ (log spaced). This is a realistic range for ε, as differences in log likelihood below 0.01 are irrelevant for model selection; an acceptable tolerance is $\varepsilon \sim 0.5$ (a difference in *deviance*, the metric used for AIC or BIC, less than 1); larger ε associate with coarse solutions, but errors larger than 10 would induce excessive biases in model selection. For noisy problems, what matters most is the solution $\boldsymbol{x}_{\text{end}}$ that the algorithm *actually* returns. Thus, we plot the fraction of successful runs at $200 \times D$ function evaluations as a function of ε, for $\varepsilon \in [0.1, 10]$ (noise makes higher precisions moot), and averaged over datasets/functions. In all plots we omit error bars for clarity (standard errors would be about the size of the line markers or less).

4.2 Results on artificial functions (BBOB09)

The BBOB09 noiseless set [32] comprises of 24 functions divided in 5 groups with different properties: separable; low or moderate conditioning; unimodal with high conditioning; multi-modal with adequate / with weak global structure. First, we use this benchmark to show the performance of different configurations for BADS. Note that we selected the default configuration (RQ kernel, a_{LCB}) and other algorithmic details by testing on a different benchmark set (see Supplementary Material). Fig 1 (left) shows aggregate results across all noiseless functions with $D \in \{3, 6, 10, 15\}$, for alternative choices of kernels and acquisition functions (only a subset is shown, such as the popular $\text{M}_{5/2}$, EI combination), or by altering other features (such as setting $n_{\text{search}} = 1$, or fixing the search covariance matrix to $\boldsymbol{\Sigma}_{\boldsymbol{\ell}}$ *or* $\boldsymbol{\Sigma}_{\text{WCM}}$). Almost all changes from the default configuration worsen performance.

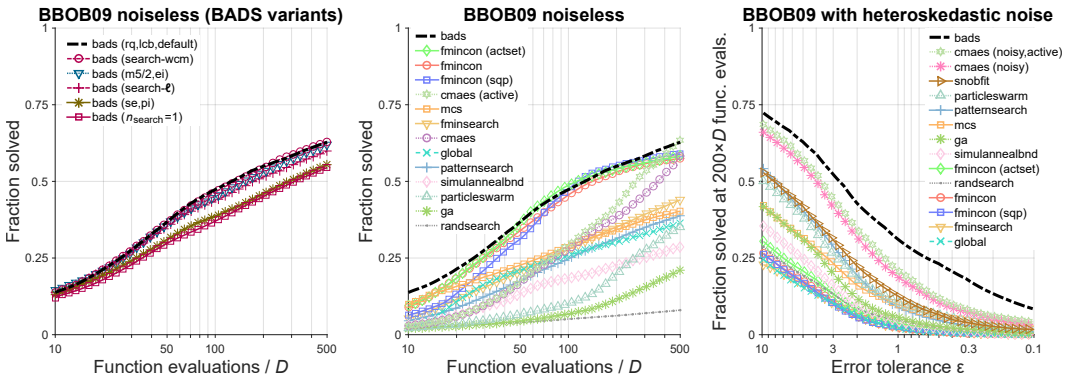

Figure 1: **Artificial test functions (BBOB09).** *Left & middle*: Noiseless functions. Fraction of successful runs ($\varepsilon \in [0.01, 10]$) vs. # function evaluations per # dimensions, for $D \in \{3, 6, 10, 15\}$ (96 test functions); for different BADS configurations (*left*) and all algorithms (*middle*). *Right*: Heteroskedastic noise. Fraction of successful runs at $200 \times D$ objective evaluations vs. tolerance ε.

Noiseless functions We then compared BADS to other algorithms (Fig 1 middle). Depending on the number of function evaluations, the best optimizers are BADS, methods of the `fmincon` family, and, for large budget of function evaluations, CMA-ES with *active* update of the covariance matrix.

Noisy functions We produce noisy versions of the BBOB09 set by adding i.i.d. Gaussian observation noise at each function evaluation, $y^{(i)} = f(\boldsymbol{x}^{(i)}) + \sigma(\boldsymbol{x}^{(i)})\eta^{(i)}$, with $\eta^{(i)} \sim \mathcal{N}(0, 1)$. We consider a variant with moderate *homoskedastic* (constant) noise ($\sigma = 1$), and a variant with *heteroskedastic* noise with $\sigma(\boldsymbol{x}) = 1 + 0.1 \times (f(\boldsymbol{x}) - f_{\min})$, which follows the observation that variability generally increases for solutions away from the optimum. For many functions in the BBOB09 set, this heteroskedastic noise can become substantial ($\sigma \gg 10$) away from the optimum. Fig 1 (right) shows aggregate results for the heteroskedastic set (homoskedastic results are similar). BADS outperforms all other optimizers, with CMA-ES (*active*, with or without the *noisy* option) coming second.

Notably, BADS performs well even on problems with non-stationary (location-dependent) features, such as heteroskedastic noise, thanks to its local GP approximation.

4.3 Results on real model-fitting problems (CCN17)

The objectives of the CCN17 set are deterministic (e.g., computed via numerical approximation) for three studies (Fig 2), and noisy (e.g., evaluated via simulation) for the other three (Fig 3).

The algorithmic cost of BADS is ~ 0.03 s to 0.15 s per function evaluation, depending on D. This time is mostly spent refitting the GP hyperparameters. Since BADS is the only method to have a non-negligible *overhead*, defined as $100\% \times$ (*total optimization time* / *total function time* -1), for deterministic problems we also plot the effective performance of BADS by accounting for the extra cost per function evaluation. For stochastic problems we cannot compute effective performance as easily, but there we found small overheads ($< 5\%$), due to more costly evaluations (more than 1 s).

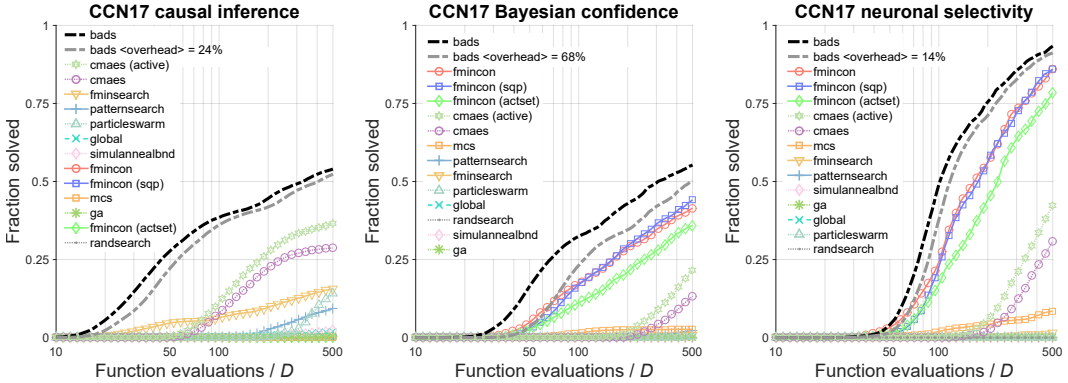

Figure 2: **Real model-fitting problems (CCN17, deterministic).** Fraction of successful runs ($\varepsilon \in [0.01, 10]$) vs. # function evaluations per # dimensions. *Left*: Causal inference in visuo-vestibular perception [33] (6 subjects, $D = 10$). *Middle*: Bayesian confidence in perceptual categorization [34] (6 subjects, $D = 13$). *Right*: Neural model of orientation selectivity [35] (6 neurons, $D = 12$).

Causal inference in visuo-vestibular perception Causal inference (CI) in perception is the process whereby the brain decides whether to integrate or segregate multisensory cues that could arise from the same or from different sources [36]. This study investigates CI in visuo-vestibular heading perception across tasks and under different levels of visual reliability, via a factorial model comparison [33]. For our benchmark we fit three subjects with a Bayesian CI model ($D = 10$), and another three with a fixed-criterion CI model ($D = 10$) that disregards visual reliability. Both models include heading-dependent likelihoods and marginalization of the decision variable over the latent space of noisy sensory measurements $(x_{\text{vis}}, x_{\text{vest}})$, solved via nested numerical integration in 1-D and 2-D.

Bayesian confidence in perceptual categorization This study investigates the *Bayesian confidence hypothesis* that subjective judgments of confidence are directly related to the posterior probability the observer assigns to a learnt perceptual category [34] (e.g., whether the orientation of a drifting Gabor patch belongs to a 'narrow' or to a 'wide' category). For our benchmark we fit six subjects to the 'Ultrastrong' Bayesian confidence model ($D = 13$), which uses the same mapping between posterior probability and confidence across two tasks with different distributions of stimuli. This model includes a latent noisy decision variable, marginalized over via 1-D numerical integration.

Neural model of orientation selectivity The authors of this study explore the origins of diversity of neuronal orientation selectivity in visual cortex via novel stimuli (orientation mixtures) and modeling [35]. We fit the responses of five V1 and one V2 cells with the authors' neuronal model ($D = 12$) that combines effects of filtering, suppression, and response nonlinearity [35]. The model has one circular parameter, the preferred direction of motion of the neuron. The model is analytical but still computationally expensive due to large datasets and a cascade of several nonlinear operations.

Word recognition memory This study models a word recognition task in which subjects rated their confidence that a presented word was in a previously studied list [37] (data from [38]). We consider six subjects divided between two normative models, the 'Retrieving Effectively from Memory' model [39] ($D = 9$) and a similar, novel model[3] ($D = 6$). Both models use Monte Carlo methods to draw random samples from a large space of latent noisy memories, yielding a stochastic log likelihood.

[3]Unpublished; upcoming work from Aspen H. Yoo and Wei Ji Ma.

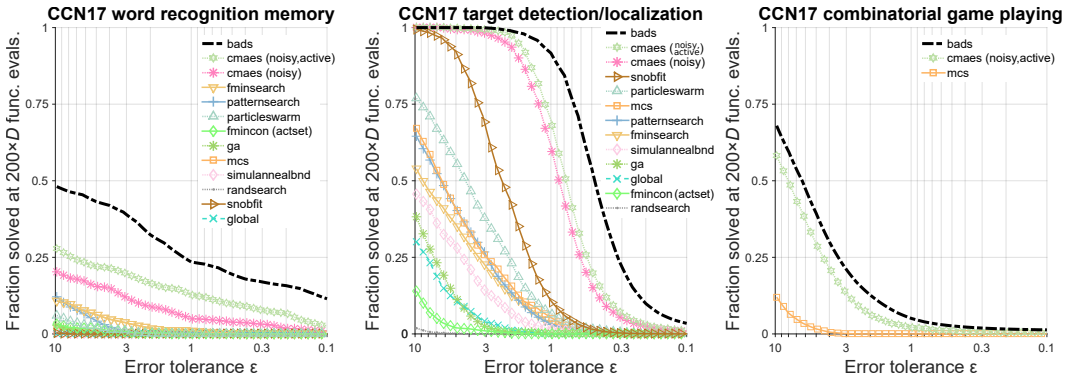

Figure 3: **Real model-fitting problems (CCN17, noisy).** Fraction of successful runs at $200 \times D$ objective evaluations vs. tolerance ε. *Left*: Confidence in word recognition memory [37] (6 subjects, $D = 6, 9$). *Middle*: Target detection and localization [41] (6 subjects, $D = 6$). *Right*: Combinatorial board game playing [43] (6 subjects, $D = 10$).

Target detection and localization This study looks at differences in observers' decision making strategies in target detection ('was the target present?') and localization ('which one was the target?') with displays of $2, 3, 4$, or 6 oriented Gabor patches.[4] Here we fit six subjects with a previously derived ideal observer model [40, 41] ($D = 6$) with variable-precision noise [42], assuming shared parameters between detection and localization. The log likelihood is evaluated via simulation due to marginalization over latent noisy measurements of stimuli orientations with variable precision.

Combinatorial board game playing This study analyzes people's strategies in a four-in-a-row game played on a 4-by-9 board against human opponents ([43], Experiment 1). We fit the data of six players with the *main* model ($D = 10$), which is based on a Best-First exploration of a decision tree guided by a feature-based value heuristic. The model also includes feature dropping, value noise, and lapses, to better capture human variability. Model evaluation is computationally expensive due to the construction and evaluation of trees of future board states, and achieved via *inverse binomial sampling*, an unbiased stochastic estimator of the log likelihood [43]. Due to prohibitive computational costs, here we only test three algorithms: BADS, MCS (the method used in the paper), and CMA-ES.

In all problems, BADS consistently performs on par with or outperforms all other tested optimizers, even when accounting for its extra algorithmic cost. The second best algorithm is either some flavor of CMA-ES or, for some deterministic problems, a member of the `fmincon` family. Crucially, their ranking across problems is inconsistent, with both CMA-ES and `fmincon` performing occasionally quite poorly (e.g., `fmincon` does poorly in the *causal inference* set because of small fluctuations in the log likelihood landscape caused by coarse numerical integration).

5 Conclusions

We have developed a novel BO method and an associated toolbox, BADS, with the goal of fitting moderately expensive computational models out-of-the-box. We have shown on real model-fitting problems that BADS outperforms widely used and state-of-the-art methods for nonconvex, derivative-free optimization. Our results demonstrate that a hybrid Bayesian approach to optimization can be beneficial beyond the domain of very costly black-box functions, in line with recent advancements in probabilistic numerics [44]. Like other surrogate-based methods, the performance of BADS is linked to its ability to obtain a fast approximation of the objective, which generally deteriorates in high dimensions, or for functions with pathological structure (often improvable via reparameterization). From our tests, we recommend BADS, paired with some multi-start optimization strategy, for models with up to ~ 15 variables, a noisy or jagged log likelihood landscape, and when algorithmic overhead is $\lesssim 75\%$ (e.g., model evaluation $\gtrsim 0.1$ s). Future work with BADS will focus on testing alternative statistical surrogates instead of GPs [45]; combining it with a smart multi-start method for global optimization; providing support for tunable precision of noisy observations [20]; improving the numerical implementation; and recasting some of its heuristics in terms of approximate inference.

[4] Unpublished; upcoming work from Andra Mihali and Wei Ji Ma.

References

[1] Rios, L. M. & Sahinidis, N. V. (2013) Derivative-free optimization: A review of algorithms and comparison of software implementations. *Journal of Global Optimization* **56**, 1247–1293.

[2] Jones, D. R., Schonlau, M., & Welch, W. J. (1998) Efficient global optimization of expensive black-box functions. *Journal of Global Optimization* **13**, 455–492.

[3] Brochu, E., Cora, V. M., & De Freitas, N. (2010) A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

[4] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & de Freitas, N. (2016) Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE* **104**, 148–175.

[5] Snoek, J., Larochelle, H., & Adams, R. P. (2012) Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems* **24**, 2951–2959.

[6] Audet, C. & Dennis Jr, J. E. (2006) Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on optimization* **17**, 188–217.

[7] Taddy, M. A., Lee, H. K., Gray, G. A., & Griffin, J. D. (2009) Bayesian guided pattern search for robust local optimization. *Technometrics* **51**, 389–401.

[8] Picheny, V. & Ginsbourger, D. (2014) Noisy kriging-based optimization methods: A unified implementation within the DiceOptim package. *Computational Statistics & Data Analysis* **71**, 1035–1053.

[9] 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.

[10] Audet, C., Custódio, A., & Dennis Jr, J. E. (2008) Erratum: Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization* **18**, 1501–1503.

[11] Clarke, F. H. (1983) *Optimization and Nonsmooth Analysis*. (John Wiley & Sons, New York).

[12] Rasmussen, C. & Williams, C. K. I. (2006) *Gaussian Processes for Machine Learning*. (MIT Press).

[13] Gramacy, R. B. & Lee, H. K. (2012) Cases for the nugget in modeling computer experiments. *Statistics and Computing* **22**, 713–722.

[14] 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.

[15] Mockus, J., Tiesis, V., & Zilinskas, A. (1978) in *Towards Global Optimisation*. (North-Holland Amsterdam), pp. 117–129.

[16] Kushner, H. J. (1964) A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering* **86**, 97–106.

[17] Bratley, P. & Fox, B. L. (1988) Algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software (TOMS)* **14**, 88–100.

[18] 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.

[19] 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.

[20] Picheny, V., Ginsbourger, D., Richet, Y., & Caplin, G. (2013) Quantile-based optimization of noisy computer experiments with tunable precision. *Technometrics* **55**, 2–13.

[21] Picheny, V., Wagner, T., & Ginsbourger, D. (2013) A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization* **48**, 607–626.

[22] 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.

[23] 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.

[24] Nocedal, J. & Wright, S. (2006) *Numerical Optimization*, Springer Series in Operations Research. (Springer Verlag), 2nd edition.

[25] Gill, P. E., Murray, W., & Wright, M. H. (1981) *Practical Optimization*. (Academic press).

[26] Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization & Machine Learning*. (Addison-Wesley).

[27] Bergstra, J. & Bengio, Y. (2012) Random search for hyper-parameter optimization. *Journal of Machine Learning Research* **13**, 281–305.

[28] Huyer, W. & Neumaier, A. (1999) Global optimization by multilevel coordinate search. *Journal of Global Optimization* **14**, 331–355.

[29] 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.

[30] Csendes, T., Pál, L., Sendin, J. O. H., & Banga, J. R. (2008) The GLOBAL optimization method revisited. *Optimization Letters* **2**, 445–454.

[31] 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.

[32] Hansen, N., Finck, S., Ros, R., & Auger, A. (2009) Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions.

[33] Acerbi, L., Dokka, K., Angelaki, D. E., & Ma, W. J. (2017) Bayesian comparison of causal inference models in multisensory perception: Analysis of explicit and implicit strategies in visuo-vestibular heading perception.

[34] Adler, W. T. & Ma, W. J. (2016) The computations underlying human confidence reports are probabilistic, but not bayesian. *bioRxiv preprint bioRxiv:093203*.

[35] Goris, R. L., Simoncelli, E. P., & Movshon, J. A. (2015) Origin and function of tuning diversity in macaque visual cortex. *Neuron* **88**, 819–831.

[36] Körding, K. P., Beierholm, U., Ma, W. J., Quartz, S., Tenenbaum, J. B., & Shams, L. (2007) Causal inference in multisensory perception. *PLoS One* **2**, e943.

[37] van den Berg, R., Yoo, A. H., & Ma, W. J. (2017) Fechner's law in metacognition: A quantitative model of visual working memory confidence. *Psychological Review* **124**, 197–214.

[38] Mickes, L., Wixted, J. T., & Wais, P. E. (2007) A direct test of the unequal-variance signal detection model of recognition memory. *Psychonomic Bulletin & Review* **14**, 858–865.

[39] Shiffrin, R. M. & Steyvers, M. (1997) A model for recognition memory: REM—retrieving effectively from memory. *Psychonomic Bulletin & Review* **4**, 145–166.

[40] Ma, W. J., Navalpakkam, V., Beck, J. M., van Den Berg, R., & Pouget, A. (2011) Behavior and neural basis of near-optimal visual search. *Nature Neuroscience* **14**, 783–790.

[41] Mazyar, H., van den Berg, R., & Ma, W. J. (2012) Does precision decrease with set size? *J Vis* **12**, 1–10.

[42] van den Berg, R., Shin, H., Chou, W.-C., George, R., & Ma, W. J. (2012) Variability in encoding precision accounts for visual short-term memory limitations. *Proc Natl Acad Sci U S A* **109**, 8780–8785.

[43] 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.

[44] Hennig, P., Osborne, M. A., & Girolami, M. (2015) Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society A* **471**, 20150142.

[45] Talgorn, B., Le Digabel, S., & Kokkolaras, M. (2015) Statistical surrogate formulations for simulation-based design optimization. *Journal of Mechanical Design* **137**, 021405–1–021405–18.

[46] Quiñonero Candela, J., Rasmussen, C. E., & Williams, C. K. (2007) Approximation methods for Gaussian process regression. *Large-scale kernel machines* pp. 203–224.

[47] Royston, J. (1982) An extension of Shapiro and Wilk's W test for normality to large samples. *Applied Statistics* pp. 115–124.

[48] Lizotte, D. J. (2008) Ph.D. thesis (University of Alberta).

[49] 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.

[50] Huyer, W. & Neumaier, A. (2008) SNOBFIT–stable noisy optimization by branch and fit. *ACM Transactions on Mathematical Software (TOMS)* **35**, 9.

[51] 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.

[52] 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.

[53] Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., et al. (1983) Optimization by simulated annealing. *Science* **220**, 671–680.

[54] Rasmussen, C. E. & Nickisch, H. (2010) Gaussian processes for machine learning (GPML) toolbox. *Journal of Machine Learning Research* **11**, 3011–3015.

Supplementary Material

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 $\boldsymbol{y}$, and GP mean function $m(\boldsymbol{x})$ and GP covariance or kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$, with i.i.d. Gaussian observation noise $\sigma^2 > 0$. The GP posterior latent marginal conditional mean μ and variance s^2 are available in closed form at a chosen point as

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

where $\mathbf{K}_{ij} = k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$, for $1 \leq i, j \leq n$, is the kernel matrix, $\boldsymbol{k}(\boldsymbol{x}) \equiv (k(\boldsymbol{x}, \boldsymbol{x}^{(1)}), \ldots, k(\boldsymbol{x}, \boldsymbol{x}^{(n)}))^\top$ 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

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

and the ARD *Matérn 5/2* kernel [5],

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

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

The Matérn 5/2 kernel has become a more common choice for Bayesian *global* optimization because it is only twice-differentiable [5], whereas the SE and RQ kernels are infinitely differentiable – a 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 $P \subseteq \{1, \ldots, D\}$, 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 [35] 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, $\text{M}_{5/2}$), we define the composite ARD periodic kernel as

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

for $1 \leq d \leq D$, where L_d is the period in the d-th coordinate dimension, and the length scale ℓ_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 $L_d = \texttt{UB}_d - \texttt{LB}_d$ (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* [46] local GP approximation. Points are added to the training set sorted by their $\boldsymbol{\ell}$-scaled distance r^2 from the incumbent $\boldsymbol{x}_k$. 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 $k(\boldsymbol{x}, \boldsymbol{x}') = k(r^2(\boldsymbol{x}, \boldsymbol{x}'))$, we define ρ as the distance such that $k(2\rho^2) \equiv 1/(\sigma_f^2 e)$. We have then

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

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

A.3 Treatment of hyperparameters

We fit the GP hyperparameters by maximizing their posterior probability (MAP), $p(\boldsymbol{\theta}|\mathbf{X}, \boldsymbol{y}) \propto p(\boldsymbol{\theta}, \mathbf{X}, \boldsymbol{y})$, which, thanks to the Gaussian likelihood, is available in closed form as [12]

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

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 $p_{\text{hyp}} = p_{\text{hyp}}(\boldsymbol{\theta}; \mathbf{X}, \boldsymbol{y})$. 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 [13]. In particular, we assume a prior such that σ decreases as a function of the poll size Δ_k^{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 $\boldsymbol{y}$, 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 hyparameter 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 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 [47], $z^{(i)} = \left(y^{(i)} - \mu(\boldsymbol{x}^{(i)})\right) / \sqrt{s^2(\boldsymbol{x}^{(i)}) + \sigma^2}$ (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 [14] 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.

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

Table S1: **GP hyperparameter priors.** Empirical Bayes priors and bounds for GP hyperparameters. $\mathcal{N}\left(\mu, \sigma^2\right)$ denotes the normal pdf with mean μ and variance σ^2, and $\mathcal{N}_{\mathrm{T}}\left(\cdot, \cdot\right)$ 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; Δ^p_{min} is the minimum poll size (default 10^{-6}); L_d is the parameter range ($\mathtt{UB}_d - \mathtt{LB}_d$), for $1 \leq d \leq D$; $\mathrm{SD}(\cdot)$ denotes the standard deviation of a set of elements; Δ_k^{poll} is the poll size parameter at the current iteration k; $\mathrm{Q}_q(\cdot)$ denotes the q-th quantile of a set of elements ($\mathrm{Q}_{0.5}$ is the median).

Probability of improvement (PI) This strategy maximizes the probability of improving over the current best minimum y_{best} [16]. For consistency with the main text, we define here the *negative* PI,

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

where $\xi \geq 0$ is an optional trade-off parameter to promote exploration, and $\Phi\left(\cdot\right)$ is the cumulative distribution function of the standard normal. a_{PI} is known to excessively favor exploitation over exploration, and it is difficult to find a correct setting for ξ to offset this tendency [48].

Expected improvement (EI) We then consider the popular predicted improvement criterion [5, 15, 2]. The expected improvement over the current best minimum y_{best} (with an offset $\xi \geq 0$) is defined as $\mathbb{E}\left[\max\left\{y_{\text{best}} - y, 0\right\}\right]$. For consistency with the main text we consider the *negative* EI, which can be computed in closed form as

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

where $\mathcal{N}\left(\cdot\right)$ 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.

B.1 Problem definition and initialization

BADS solves the optimization problem

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

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

Feature	Description (defaults)
Surrogate model	GP
Hyperparameter treatment	optimization
GP training set size $n_{\max}$	70 ($D = 2$), 250 ($D = 20$) (+50 for noisy problems)
POLL directions generation	LTMADS with GP rescaling
SEARCH set generation	Two-step ES algorithm with search matrix $\boldsymbol{\Sigma}$
SEARCH evals. (n_{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.**

Algorithm input The algorithm takes as input a starting point $\boldsymbol{x}_0 \in \mathcal{X}$; vectors of *hard* lower/upper bounds `LB`, `UB`; optional vectors of *plausible* lower/upper bounds `PLB`, `PUB`; and an optional barrier function c. We require that, if specified, $c(\boldsymbol{x}_0) \leq 0$; and for each dimension $1 \leq d \leq D$, $\texttt{LB}_d \leq (\boldsymbol{x}_0)_d \leq \texttt{UB}_d$ and $\texttt{LB}_d \leq \texttt{PLB}_d < \texttt{PUB}_d \leq \texttt{UB}_d$. 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. 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 $\texttt{UB}_d \geq 10 \cdot \texttt{LB}_d$ 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 $[\texttt{PLB}, \texttt{PUB}]$ 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 ($c(\boldsymbol{x}) > 0$). 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 ~ 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 ~ 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 $\boldsymbol{x}_0$ and $n_{\text{init}} = D$ additional points chosen via a low-discrepancy Sobol quasirandom sequence [17] 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 $\boldsymbol{x}_0$. For all practical purposes, a function is deemed noisy if the two evaluations at $\boldsymbol{x}_0$ differ more than $1.5 \cdot 10^{-11}$.[5]

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_{search} consecutive steps to find a *sufficient* improvement in function value,

[5] Since this simple test might fail, users are encouraged to actively specify whether the function is noisy.

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 Δ_k^{poll} is the poll size.

In each SEARCH step we choose the final candidate point to evaluate, $\boldsymbol{x}_{\text{search}}$, by performing a fast, approximate optimization of the chosen acquisition function in the neighborhood of the incumbent $\boldsymbol{x}_k$, using a two-step evolutionary heuristic inspired by CMA-ES [18]. This local search is governed by a *search covariance matrix* $\boldsymbol{\Sigma}$, and works as follows.

Local search via two-step evolutionary strategy We draw a first generation of candidates $\boldsymbol{s}_{\text{I}}^{(i)} \sim \mathcal{N}(\boldsymbol{x}_k, (\Delta_k^{\text{poll}})^2\boldsymbol{\Sigma})$ 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); $\boldsymbol{\Sigma}$ is a search covariance matrix with unit trace,[6] and $n_{\text{search}} = 2^{11}$ by default. For each candidate point, we assign a number of offsprings proportionally to the square root of its ranking according to $a(\boldsymbol{s}_{\text{I}}^{(i)})$, for a total of n_{search} offsprings [18]. We then draw a second generation $\boldsymbol{s}_{\text{II}}^{(i)} \sim \mathcal{N}(\boldsymbol{s}_{\text{I}}^{(\pi_i)}, c^2(\Delta_k^{\text{poll}})^2\boldsymbol{\Sigma})$ and project it onto the mesh grid, where π_i is the index of the parent of the i-th candidate in the 2nd generation, and $0 < c \leq 1$ is a zooming factor (we choose $c = 1/4$). Finally, we pick $\boldsymbol{x}_{\text{search}} = \arg\min_i a(\boldsymbol{s}_{\text{II}}^{(i)})$. At each step, we remove candidate points that violate non-bound constraints ($c(\boldsymbol{x}) > 0$), 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 $\boldsymbol{\Sigma}_{\boldsymbol{\ell}}$ with diagonal $\left(\ell_1^2/|\boldsymbol{\ell}|^2, \dots, \ell_D^2/|\boldsymbol{\ell}|^2\right)$, and a matrix $\boldsymbol{\Sigma}_{\text{WCM}}$ 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 [18]). At each step, we compute the probability of choosing $\boldsymbol{\Sigma}_s$, with $s \in \{\boldsymbol{\ell}, \text{WCM}\}$, according to a *hedging* strategy taken from the *Exp3* HEDGE algorithm [19],

$$p_s = \frac{e^{\beta_{\text{H}} g_s}}{\sum_{s'} e^{\beta_{\text{H}} g_{s'}}}(1 - \gamma_{\text{H}} n_{\boldsymbol{\Sigma}}) + \gamma_{\text{H}} \tag{S10}$$

where $\beta_{\text{H}} = 1$, $\gamma_{\text{H}} = 0.125$, $n_{\boldsymbol{\Sigma}} = 2$ 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

$$g_s^{\text{new}} = \alpha_{\text{H}} g_s^{\text{old}} + \frac{\Delta f_s}{p_s \Delta_k^{\text{poll}}} \tag{S11}$$

where $\alpha_{\text{H}} = 0.1^{1/(2D)}$ is a decay factor, and Δf_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 ($\boldsymbol{\Sigma}_{\boldsymbol{\ell}}$), and following an approximation of the local curvature around the incumbent ($\boldsymbol{\Sigma}_{\text{WCM}}$).

B.3 POLL stage

We perform the POLL stage only after a SEARCH stage that did not produce a *sufficient* improvement after n_{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.

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 [6]. We then transform it to a *rescaled* set $\mathbf{D}_k$ based on the current GP kernel length scales: for $\boldsymbol{v}' \in \mathbf{D}'_k$, we define a rescaled vector $\boldsymbol{v}$ with $v_d \equiv v'_d \cdot \omega_d$, for $1 \leq d \leq D$, and $\omega_d \equiv \min\{\max\{10^{-6}, \Delta_k^{\text{mesh}}, \ell_d/\text{GM}(\boldsymbol{\ell})\}, \texttt{UB}_d - \texttt{LB}_d\}$, where $\text{GM}(\cdot)$ denotes the geometric mean, and we use $\texttt{PLB}_d$ (resp. $\texttt{PUB}_d$) whenever $\texttt{UB}_d$ (resp. $\texttt{LB}_d$) 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 [9].

[6] Unit trace (sum of diagonal entries) for $\boldsymbol{\Sigma}$ implies that a draw $\sim \mathcal{N}(0, \boldsymbol{\Sigma})$ has unit expected squared length.

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 [6]). 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 Δ_k^{poll} goes below a threshold Δ_{min}^p (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 $\boldsymbol{x}_{\text{end}}$ (transformed back to original coordinates) that has the lowest objective value y_{end}. For a noisy objective, we return instead the stored point with the lowest quantile q_β across iterations, with $\beta = 0.999$; see Section 3.4 in the main text. We also return the function value at the optimum, y_{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, $n_{\text{min}} = 100$;
- increase the total number of points (within radius ρ) 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 [21] and replace y_i with the GP latent quantile function q_β [20] (see Eq. 3 in the main text). Moreover, we modify the MADS procedure by keeping an *incumbent set* $\{\boldsymbol{x}_i\}_{i=1}^k$, where $\boldsymbol{x}_i$ is the incumbent at the end of the i-th iteration. At the end of each POLL stage, we re-evaluate q_β 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 $q_\beta(\boldsymbol{x}_i)$. 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 $\boldsymbol{x}_{\text{end}}$ returned by the algorithm in a robust manner. For a noisy objective, instead of the noisy measurement y_{end}, we return either our best GP prediction $\mu(\boldsymbol{x}_{\text{end}})$ and its uncertainty $s(\boldsymbol{x}_{\text{end}})$, or, more conservatively, an estimate of $\mathbb{E}[f(\boldsymbol{x}_{\text{end}})]$ and its standard error, obtained by averaging N_{final} function evaluations at $\boldsymbol{x}_{\text{end}}$ (default $N_{\text{final}} = 10$). The latter approach is a safer option to obtain an unbiased value of $\mathbb{E}[f(\boldsymbol{x}_{\text{end}})]$, since the GP approximation may occasionally fail or have substantial bias.

Noise estimate The user can optionally provide a noise estimate σ_{est} which is used to set the mean of the hyperprior over the observation noise σ (see Table S1). We recommend to set σ_{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). We include here details that expand on Section 4.1 of the main text.

C.1 Algorithms

The list of tested algorithms is reported in Table S3. For all methods, we used their default options unless stated otherwise. For BADS and CMA-ES, we activated their *uncertainty handling* option when dealing with noisy problems (for CMA-ES, see [31]). 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 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 [43]. We note that we developed algorithmic details and internal settings of BADS by testing it on the CEC14 test set for expensive optimization [49] and on other model-fitting problems which differ from the test problems presented in this benchmark.

Package	Algorithm	Source	Ref.	Noise	Global
`bads`	Bayesian Adaptive Direct Search	GitHub page [7]	This	✓	≈
`fminsearchbnd`	Nelder-Mead (`fminsearch`) w/ bounded domain	File Exchange[8]	[22]	✗	✗
`cmaes`	Covariance Matrix Adaptation Evolution Strategy	Author's website[9]	[18]	✗	≈
— (`active`)	CMA-ES with active covariance adaptation	—	[29]	✗	≈
— (`noise`)	CMA-ES with uncertainty handling	—	[31]	✓	≈
`mcs`	Multilevel Coordinate Search	Author's website[10]	[28]	✗	✓
`snobfit`	Stable Noisy Optimization by Branch and FIT	Author's website[11]	[50]	✓	✓
`global`	GLOBAL	Author's website[12]	[30]	✗	✓
`randsearch`	Random search	GitHub page[13]	[27]	✗	✓
`fmincon`	Interior point (`interior-point`, default)	Opt. Toolbox	[23]	✗	✗
— (`sqp`)	Sequential quadratic programming	—	[24]	✗	✗
— (`active-set`)	Active-set	—	[25]	✗	✗
`patternsearch`	Pattern search	Global Opt. Toolbox	[51]	✗	✗
`ga`	Genetic algorithms	Global Opt. Toolbox	[26]	✗	≈
`particleswarm`	Particle swarm	Global Opt. Toolbox	[52]	✗	≈
`simulannealbnd`	Simulated annealing w/ bounded domain	Global Opt. Toolbox	[53]	✗	≈

Table S3: **Tested algorithms.** *Top*: Freely available algorithms. *Bottom*: Algorithms in MATLAB's Optimization and Global Optimization 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.

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.

[7] `https://github.com/lacerbi/bads`

[8] `https://www.mathworks.com/matlabcentral/fileexchange/8277-fminsearchbnd--fminsearchcon`.

[9] `https://www.lri.fr/~hansen/cmaes_inmatlab.html`

[10] `https://www.mat.univie.ac.at/~neum/software/mcs/`

[11] `http://www.mat.univie.ac.at/~neum/software/snobfit/`

[12] `http://www.inf.u-szeged.hu/~csendes/index_en.html`

[13] `https://github.com/lacerbi/neurobench/tree/master/matlab/algorithms`

C.2 Procedure

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^t_{best} found so far after t function evaluations. We define the *immediate regret* (or error) at time t as $y^t_{\text{best}} - y_{\text{min}}$, where y_{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 ε). For problems in the BBOB09 set (both noiseless and noisy variants), we know the ground truth y_{min}. For problems in the CCN17 set, we do not know y_{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).

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 [43]. For noisy problems in the CCN17 set, we estimated the log likelihood at a each provided candidate solution via 200 function evaluations, and took the final estimate with lowest average.

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 ε 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 $\epsilon \in [0.1, 1]$ instead of the wider range $\epsilon \in [0.01, 10]$ 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.

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.

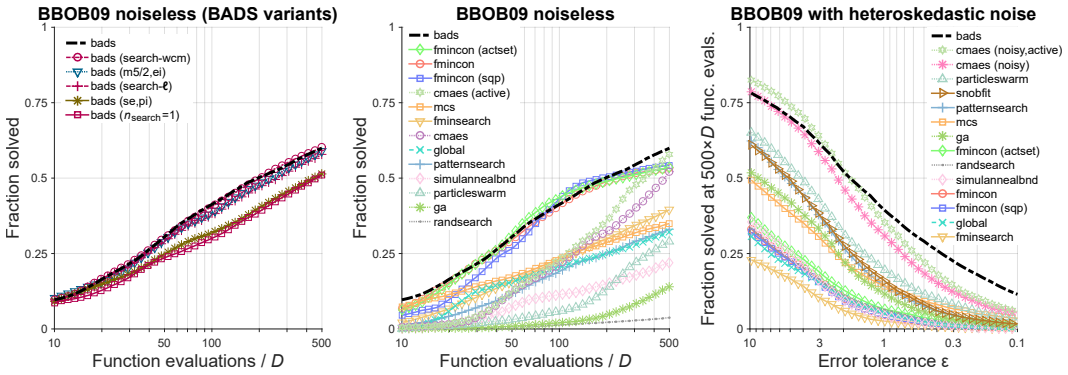

Figure S1: **Artificial test functions (BBOB09).** Same as Fig 1 in the main text, but with with alternative benchmark parameters (in bold). *Left & middle*: Noiseless functions. Fraction of successful runs ($\varepsilon \in [\mathbf{0.1}, \mathbf{1}]$) vs. # function evaluations per # dimensions, for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

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 [54] (v3.6), modified for increased efficiency of some algorithmic steps, such as computation of gradients,[14], 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.

Acknowledgments

We thank Will Adler, Robbe Goris, Andra Mihali, Bas van Opheusden, and Aspen Yoo for sharing data and model evaluation code that we used in the CCN17 benchmark set; Maija Honig, Andra Mihali, Bas van Opheusden, and Aspen Yoo for providing user feedback on earlier versions of the `bads` package for MATLAB; Will Adler, Andra Mihali, Bas van Opheusden, and Aspen Yoo for helpful feedback on a previous version of this manuscript; and John Wixted and colleagues for allowing us to reuse their data for the CCN17 'word recognition memory' problem set.

[14] 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.