Multi-criteria Optimization Using the AMALGAM Software Package: Theory, Concepts, and MATLAB Implementation

Jasper A. Vrugt[a,b]

[a]*Department of Civil and Environmental Engineering, University of California Irvine, 4130 Engineering Gateway, Irvine, CA 92697-2175*
[b]*Department of Earth System Science, University of California Irvine, Irvine, CA*

Abstract

The evolutionary algorithm AMALGAM implements the novel concept of adaptive multimethod search to ensure a fast, reliable and computationally efficient solution to multiobjective optimization problems. The method finds a well-distributed set of Pareto solutions within a single optimization run, and achieves an excellent performance compared to commonly used methods such as SPEA2, NSGA-II and MOEA/D. In this paper, I review the basic elements of AMALGAM, provide a pseudo-code of the algorithm, and introduce a MATLAB toolbox which provides scientists and engineers with an arsenal of options and utilities to solve multiobjective optimization problems involving (among others) multimodality, high-dimensionality, bounded parameter spaces, dynamic simulation models, and distributed multi-core computation. The AMALGAM toolbox supports parallel computing to permit inference of CPU-intensive system models, and provides convergence diagnostics and graphical output. Four different case studies are used to illustrate the main capabilities and functionalities of the AMALGAM toolbox.

Keywords: Evolutionary search, Multiple objectives, Global optimization, Pareto front, Convergence analysis, Prior distribution, Residual analysis, Environmental modeling, Multi-processor distributed computation

AMALGAM MANUAL

Email address: jasper@uci.edu (Jasper A. Vrugt)
URL: http://faculty.sites.uci.edu/jasper (Jasper A. Vrugt), http://scholar.google.com/citations?user=zkNXecUAAAAJ\&hl=en (Jasper A. Vrugt)

1. Introduction and Scope

Evolutionary optimization is a subject of intense interest in many fields of study, including computational chemistry, biology, bioinformatics, economics, computational science, geophysics, and environmental science (*Holland*, 1975; *Bounds*, 1987; *Barhen et al.*, 1997; *Wales and Scheraga*, 1999; *Lemmon and Milinkovitch*, 2002; *Glick et al.*, 2002; *Nowak and Sigmund*, 2004; *Schoups et al.*, 2005). The goal is to determine values for model parameters or state variables that provide the best possible solution to a predefined cost or objective function, or a set of optimal tradeoff values in the case of two or more conflicting objectives. However, locating optimal solutions often turns out to be painstakingly tedious, or even completely beyond current or projected computational capacity (*Achlioptas et al.*, 2005).

Here, I consider multiobjective optimization problems, with d decision variables (parameters) and m ($m > 1$) objectives: $\mathbf{F} = \{f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})\}$, where $\mathbf{x} = \{x_1, \ldots, x_d\}$ denotes the decision vector, and $\mathbf{F}$ is the objective space. I restrict attention to optimization problems in which the parameter search space $\mathbf{X}$, although perhaps quite large, is bounded: $\mathbf{x} \in \mathcal{X} \in \mathbb{R}^d$. The presence of multiple objectives in an optimization problem gives rise to a set of Pareto-optimal solutions, instead of a single optimal solution. A Pareto-optimal solution is one in which one objective cannot be further improved without causing a simultaneous degradation in at least one other objective. As such, they represent globally optimal solutions to the tradeoff problem.

In mathematical terms, a multi-objective optimization problem can be formulated as

$$\arg\min_{\mathbf{x}\in\mathcal{X}} \mathbf{F}(\mathbf{x}) = \{f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})\} \tag{1}$$

where $f(\cdot)$ is a function (numerical model) that computes m ($m \leq 2$) different objective function values, $\mathbf{x}$ denotes a d-vector with decision variables (parameters), and $\mathcal{X}$ represents the feasible search space.[1] The solution to this problem will in general, no longer be a single 'best' parameter value but will consist of a Pareto set $\mathbf{P}(\mathbf{x})$ of solutions corresponding to various trade-offs among the m objectives. The Pareto set of solutions defines the minimum uncertainty in the parameters that can be achieved without stating a subjective relative preference for minimizing one specific component of $\mathbf{F}(\cdot)$ at the expense of another. Without additional subjective preference information, all Pareto optimal solutions are considered equally good (as vectors cannot be ordered completely).

To illustrate Pareto optimality, please consider Figure 1Illustration of the concept of Pareto optimality for the toy problem with two parameters $\{x_1, x_2\}$ and two criteria $\{f_1, f_2\}$ in the (A) parameter and (B) objective function space. The points A and B indicate the solutions that minimize each of the individual criteria f_1 and f_2. The dotted blue line joining A and B corresponds to the Pareto set of solutions. The point β is an element of the solution set, and is superior in the multi-criteria sense to any other point not on this line.figure.caption.1 which displays the sampled parameter (left) and objective function (right) space of a simple toy problem involving $m = 2$ conflicting objectives

$$\arg\min_{\mathbf{x}\in\mathcal{X}} \mathbf{F}(\mathbf{x}) = \begin{cases} f_1(\mathbf{x}) = x_1 + (x_2 - 1)^2 \\ f_2(\mathbf{x}) = x_2 + (x_1 - 1)^2, \end{cases} \tag{2}$$

[1] Notation in Equation 1Introduction and Scopeequation.1.1 assumes minimization problems. For maximization problems, please use $-f(\mathbf{x})$.

where $\mathbf{x} \in [0,1]^2$. The aim is to simultaneously minimize the two objectives $\{f_1, f_2\}$ with respect to the two parameters $\{x_1, x_2\}$. The individual points A and B minimize objectives f_1 and f_2, respectively, whereas the dotted blue line joining A and B represents the Pareto optimal front. Moving along the dotted line from A to B results in the improvement of f_2 while simultaneously causing deterioration in f_1. The points falling on the dotted AB line represent the trade-offs between the objectives and are called nondominated, noninferior, or efficient solutions. All the other points of the search domain are hence called inferior, dominated, or inefficient.

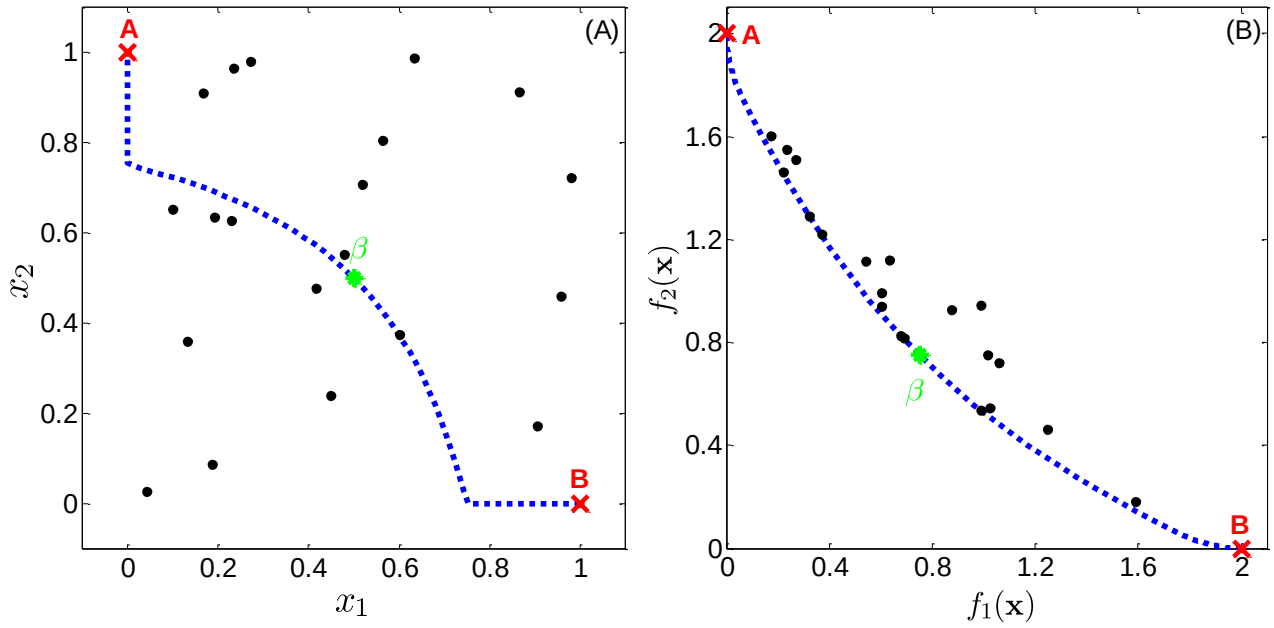


Figure 1: Illustration of the concept of Pareto optimality for the toy problem with two parameters $\{x_1, x_2\}$ and two criteria $\{f_1, f_2\}$ in the (A) parameter and (B) objective function space. The points A and B indicate the solutions that minimize each of the individual criteria f_1 and f_2. The dotted blue line joining A and B corresponds to the Pareto set of solutions. The point β is an element of the solution set, and is superior in the multi-criteria sense to any other point not on this line.

Numerous approaches have been proposed to efficiently find Pareto-optimal solutions for multiobjective optimization problems (*Zitzler and Thiele*, 1999; *Zitzler et al.*, 2000; *Deb et al.*, 2002; *Knowles and Corne*, 1999). In particular, evolutionary algorithms have emerged as the most powerful approach for solving search and optimization problems involving multiple conflicting objectives. Beyond their ability to search intractably large spaces for multiple Pareto-optimal solutions, these algorithms are able to maintain a diverse set of solutions and exploit similarities of solutions by recombination. These attributes lead to efficient convergence to the Pareto-optimal front in a single optimization run (*Zitzler et al.*, 2000). Of these, the SPEA/SPEA2 *Zitzler and Thiele* (1999); *Zitzler et al.* (2001), nondominated sorted genetic algorithm II (NSGA-II) (*Deb et al.*, 2002), and MOEA/D (*Zhang and Li*, 2007; *Li and Zhang*, 2009) algorithms have received most attention because of their demonstrated ability to solve difficult benchmark problems.

Although the multiobjective optimization problem has been studied quite extensively, current available evolutionary algorithms typically implement a single algorithm for population evolution. Reliance on a single biological model of natural selection and adaptation presumes that a single method exists that efficiently

evolves a population of potential solutions through the parameter space. However, existing theory and numerical experiments have demonstrated that it is impossible to develop a single algorithm for population evolution that is always efficient for a diverse set of optimization problems (*Wolpert and Macready*, 1997).

In the past decade, memetic algorithms (also called hybrid genetic algorithms) have been proposed to increase the search efficiency of population based optimization algorithms *Hart et al.* (2005). These methods are inspired by models of adaptation in natural systems, and use a genetic algorithm for global exploration of the search space, combined with a local search heuristic for exploitation. Memetic algorithms have shown to significantly speed up the evolution toward the global optimal solution for a variety of real-world optimization problems. However, our conjecture is that a search procedure that adaptively changes the way it generates offspring, based on the shape and local peculiarities of the fitness landscape, will further improve the efficiency of evolutionary search. This approach is likely to be productive because the nature of the fitness landscape (objective functions mapped out in the parameter space, also called the response surface) often varies considerably between different optimization problems, and dynamically changes en route to the global optimal solutions.

Drawing inspiration from the field of ensemble weather forecasting (*Gneiting and Raftery*, 2005), we have presented a novel multimethod evolutionary search approach (*Vrugt and Robinson*, 2007a; *Vrugt et al.*, 2009a). The method combines two concepts, simultaneous multimethod search, and self-adaptive offspring creation, to ensure a fast, reliable, and computationally efficient solution to multiobjective optimization problems. We called this approach a multi-algorithm, genetically adaptive multiobjective, or AMALGAM, method, to evoke the image of a procedure that blends the attributes of the best available individual optimization algorithms. Benchmark results using a set of well known multiobjective test problems show that AMALGAM approaches a factor of 10 improvement over current optimization algorithms for the more complex, higher dimensional problems (*Vrugt and Robinson*, 2007a). AMALGAM scales well with increasing number of dimensions, converges adequately for stochastic (noise induced) objective functions, and is designed to take full advantage of the power of distributed computer networks (*Vrugt et al.*, 2009a).

In this paper, I introduce a MATLAB toolbox of the AMALGAM multi-criteria optimization algorithm. This MATLAB toolbox provides scientists and engineers with a comprehensive set of utilities for application of the AMALGAM algorithm to multiobjective optimization. The AMALGAM toolbox supports parallel computing and includes tools for convergence analysis and post-processing of the results. Some of the built-in options and utilities are demonstrated using four different case studies involving (for instance) multimodality, numerous local optima, bounded parameter spaces, dynamic simulation models, Bayesian model averaging, and distributed multi-core computation. These example studies are easy to run and adapt and serve as templates for other inference problems. The present contribution follows closely the DREAM toolbox *Vrugt* (2016) developed for Bayesian inference of complex system models.

The remainder of this paper is organized as follows. Section 2 discusses the various building blocks of AMALGAM. This is followed in section 3 with a pseudo-code of the algorithm. Section 4 then presents a MATLAB toolbox of AMALGAM. In this section I am especially concerned with the input and output arguments of AMALGAM and the various utilities and options available to the user. Section 5 considers four different case studies that illustrate how to use the AMALGAM toolbox. The penultimate section of this paper (section 6) highlights recent research efforts aimed at further improving the efficiency of multimethod multiple objective optimization. Finally, section 7 concludes this manual with a short summary of the main

capabilities of the AMALGAM toolbox.

2. Inference of the Pareto distribution

A key task in multiple criteria optimization is to summarize the so called Pareto distribution. When this task cannot be carried out by analytical means nor by analytical approximation, evolutionary algorithms can be used to generate a sample from the Pareto distribution. The desired summary of the Pareto distribution is then obtained from this sample. The Pareto distribution, also referred to as the target or limiting distribution, is often high dimensional. A large number of iterative methods have been developed to generate samples from the Pareto distribution. All these methods rely in some way on evolutionary principles. The next section discusses the AMALGAM multimethod search algorithm.

2.1. AMALGAM: in words

Whereas classical evolutionary algorithms employ a single recombination method to create offspring, the AMALGAM method uses q different optimization algorithms concurrently to evolve a population of particles (also called parents) through a multidimensional search space in pursuit of the Pareto distribution.

The algorithm is initiated by an initial population $\mathbf{X}_0$ of size $N \times d$, drawn randomly from some prior ranges using, for instance, Latin hypercube sampling. Then, each parent is assigned a rank using the fast nondominated sorting (FNS) algorithm of *Deb et al.* (2002) (see Figure 2Two-dimensional scatter plots of the parent objective function values for a (A) min-min, (B) min-max, (C) max-min, and (D) max-max optimization problem. Color coding is used to denote the Pareto ranks of the solutions.figure.caption.2). A population of offspring $\mathbf{Y}_1$, of size $N \times d$, is subsequently created by using the multimethod search concept that lies at the heart of the AMALGAM method. Instead of implementing a single operator for reproduction, I simultaneously use q different recombination methods to generate the offspring population, $\mathbf{Y}_1 = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$. These reproductive methods implement different selection and crossover operators, and the number of children they each contribute individually to the offspring population, $\mathbf{Y}_1$, is determined by the selection probability, $\mathbf{p}_0 = \{p_0^1, \ldots, p_0^q\}$ of each recombination method. These selection probabilities are restricted to the unit simplex, Δ^q, in $\mathbb{R}_+^q$, or $\{\mathbf{p}.|p^j \geq 0 \text{ and } \sum_{j=1}^q p^k = 1\}$, and can vary per generation, based on each method's reproductive success during the previous generation. After the offspring population, $\mathbf{Y}_1$, has been created and evaluated, the parents and children are merged together into one "big" population, $\mathbf{R}_1 = \mathbf{X}_0 \bigcup \mathbf{Y}_1$ of size $2N \times d$, and the corresponding objective function values, stored in the $2N \times m$ matrix $\mathbf{Q}$, are subsequently ranked using FNS. Finally, the N members of the new population, $\mathbf{X}_1$, are selected from the nondominated fronts of $\mathbf{R}_1$ based on rank and crowding distance (*Deb et al.*, 2002). The new population, $\mathbf{X}_1$ is then used to create an offspring population and the aforementioned algorithmic steps are repeated until convergence of the population is achieved to a limiting distribution. This final population then consists of a sample set of nondominated solutions that make up the Pareto solution space. Note, that nondominated solutions of previous generations are preserved in the new population through the combined ranking of the parent and offspring population, $\mathbf{R}$. (*Zitzler and Thiele*, 1999; *Zitzler et al.*, 2000; *Deb et al.*, 2002). This elitism is particularly beneficial as it speeds up convergence to the Pareto set (*Laumanns et al.*, 2000).

The core of the FNS algorithm can be coded in just a few lines (see below) and requires as input argument the objective function values, `Q` (size $2N \times m$) of the combined parent and offspring population.

MATLAB function of FNS algorithm. The matrix Q of size $2N \times m$ with objective function values is ranked. Built-in functions are highlighted with a low dash - their names are often sufficient to understand their mathematical operation. `zeros()` defines a zeroth vector (matrix), `cell()` creates a cell-array with all the solutions each point of Q dominates, and `bsxfun(@fun,A,B)` applies an element-by-element binary operation to arrays A and B, with singleton expansion enabled; The operator @fun is either less than, `lt` or less than or equal, `le`. I refer to introductory textbooks and/or the MATLAB "help" utility for the remaining functions `size()`, `find()`, `sum()`, `numel()`, and `isempty()`.

```
function [ rank ] = FNS ( Q );
% Fast nondominated sorting for a N x m matrix with objective function values

[N,m] = size ( Q );      % Number of individuals and objective functions
Fr = {[]};               % Pareto-optimal fronts
rank = zeros(N,1);       % Initialize vector with ranks
Sp = cell(N,1);          % Set of individuals a particular individual dominates
np = zeros(N,1);         % Number of individuals by which a particular point is dominated

for j = 1 : N,           % Now loop over all elements of R
    idx = find( (sum( bsxfun(@le,Q(j,1:m),Q) , 2 ) == m ) & ...
        ( sum( bsxfun(@lt,Q(j,1:m),Q) , 2 ) > 0 ) );          % Find which of Q, j
           dominates
    if numel(idx), Sp{j} = idx; end                           % Store these points in Sp
    idx = find( (sum(bsxfun(@le,Q,Q(j,1:m)) , 2) == m ) & ...
        ( sum(bsxfun(@lt,Q,Q(j,1:m)) , 2 ) > 0 ) );           % Find which of Q dominate
           j
    np(j) = np(j) + numel(idx);                               % Store number of points in
         np
    if np(j) == 0,                                            % j is member of first
        front
        Fr{1}(end+1) = j;
    end
end

i = 1;
while ~isempty(Fr{i})                                         % Continue if not all
    ranked
    NextFr = [];                                              % Next front is empty
    for j = 1 : numel(Fr{i}),                                 % For each member j of Fr{i
        }
        q = Sp{Fr{i}(j)};                                     % Modify each member of set
             Sp
        np(q) = np(q) - 1;                                    % Decrement np(q) by one
        id = find ( np(q) == 0 ); NextFr = [NextFr ; q(id)]; % n(q) zero, q member next
           front
    end
    i = i + 1;                                                % Go to next front
    Fr{i} = NextFr;                                           % Current members of NextFr
end

for j = 1:i, rank(Fr{j}) = j; end                             % Extract N-vector with
    ranks
```

First, for each solution I calculate two entities: (i) `np`, the number of solutions which dominate the solution `p`, and (ii) `Sp`, a set of solutions which the solution `p` dominates. The calculation of these two entities requires $\mathcal{O}(mN^2)$ operations (comparisons). We identify all those points which have `np` $= 0$ and put them in a list `Fr{1}`. This is called the current front. Now, for each solution, `p` in the current front we visit each member, `q`, of its relevant set `Sp` and reduce its `np` count by one. In doing so, if for any member `q` the count becomes zero, we put it in a separate list `NextFr`. When all members of the current front have been checked, we declare the members in the list `Fr` as members of the first front. We then continue this process using the newly identified front `NextFr` as our current front.

To illustrate the FNS algorithm please consider Figure 2Two-dimensional scatter plots of the parent objective function values for a (A) min-min, (B) min-max, (C) max-min, and (D) max-max optimization problem. Color coding is used to denote the Pareto ranks of the solutions.figure.caption.2 that plots the ranks of the different solutions (points) for four different multi-criteria optimization problems.

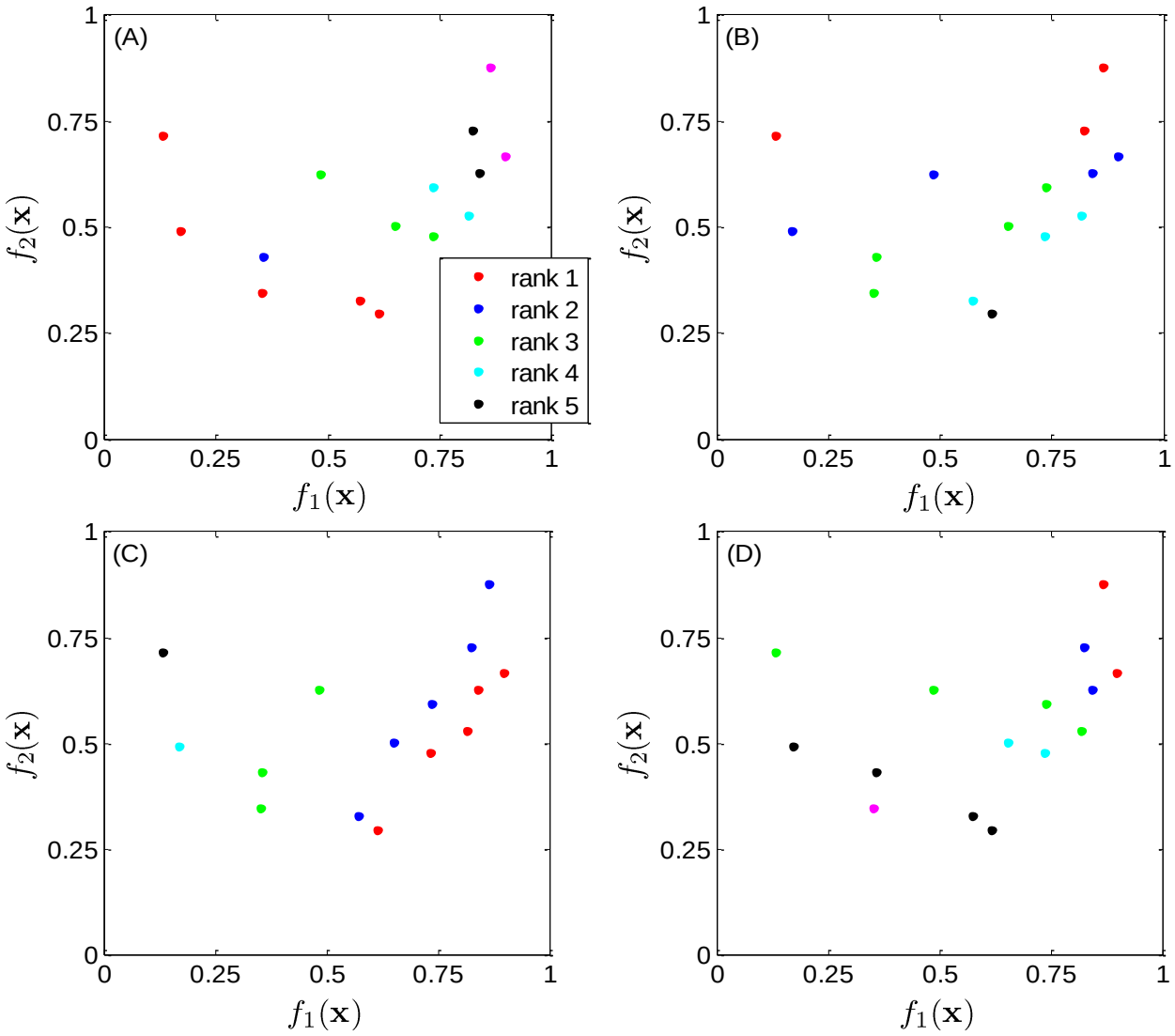


Figure 2: Two-dimensional scatter plots of the parent objective function values for a (A) min-min, (B) min-max, (C) max-min, and (D) max-max optimization problem. Color coding is used to denote the Pareto ranks of the solutions.

In the case of conventional Pareto ranking, points having an identical rank number are not distinguishable, even though the solutions at the extreme end are in some sense much more unique than other solutions having the same rank. Hence, another criteria is needed that penalizes individuals with many neighbors in their niche. The crowding distance is such metric and preserves the diversity of the population, thereby avoiding a collapse of the Pareto solutions to the most compromised region of the solution space.

The crowding distance provides an estimate of the density of solutions surrounding each point and is computed using the following MATLAB subroutine.

MATLAB function that calculates the crowding distance of the combined parent and offspring population. This function has as input, the matrix Q of size $2N \times m$ with objective function values, and the $2N \times 1$-vector `rank` with associated ranks derived from the function `FNS`. Built-in functions are highlighted with a low dash. `max()` computes the maximum value of a vector, `sort()` sorts the elements of a vector in ascending order, and `inf` stands for infinity.

```
function [ C ] = Crowding_distance ( Q , rank );
% Calculates the crowding distance of the solutions of Q

m = size(Q,2);                % How many objectives?

for r = 1:max(rank),

    idx = find(rank == r);        % First find points with rank j
    R_sel = Q(idx,1:m);           % Select those points
    N = numel(idx);               % How many points with rank j
    C_d = zeros(N,1);             % (Re)-initialize the crowding distance

    for k = 1:m,
        [R_sort sort_idx] = sort(R_sel(:,k));          % Sort objective k ascending
            order
        C_d(sort_idx(1),1) = C_d(sort_idx(1),1) + inf; % Set boundary solution to inf
        C_d(sort_idx(N),1) = C_d(sort_idx(N),1) + inf; % Set other boundary point to inf

        for j = 2:(N - 1),        % Now determine crowding distance of other individuals
            C_d(sort_idx(j),1) = C_d(sort_idx(j),1) + (R_sort(j+1) - R_sort(j-1));
        end

    end

    C(idx,1) = C_d;       % Store crowding distance of solutions idx of Q

end
```

Thus, to get an estimate of the density of solutions surrounding a particular point in the population we take the average distance of the two points on either side of this point along each of the objectives. This quantity, referred to as crowding distance serves as an estimate of the size of the largest cuboid enclosing the point i without including any other point in the population. In Figure 3Schematic example of computation of the crowding distance of the ith point of the population. The red dots are Pareto solutions (rank 1), whereas the blue solutions are dominated and hence belong to the next front (rank = 2).figure.3, the crowding distance of the ith solution in its front (marked with solid circles) is the average side-length of the cuboid (shown with a dashed box).

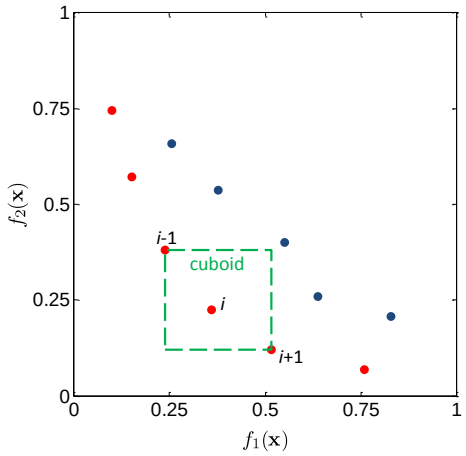


Figure 3: Schematic example of computation of the crowding distance of the ith point of the population. The red dots are Pareto solutions (rank 1), whereas the blue solutions are dominated and hence belong to the next front (rank = 2).

An alternative strategy to differentiate among solutions with equal rank is the strength Pareto approach of (*Zitzler and Thiele*, 1999) and used in the MOSCEM-UA algorithm (*Vrugt et al.*, 2003), a predecessor of AMALGAM. This option is implemented in the MATLAB toolbox of AMALGAM, of which more in the next section.

MATLAB function that calculates the strength Pareto of the combined parent and offspring population. This function has as input, the matrix Q of size $2N \times m$ with objective function values.

```
function [ s ] = Strength_pareto ( Q );
% Compute strength of each solution according to Zitzler et al. (1999)

[N,m] = size ( Q ) ;     % How many individuals and objective functions?
s = zeros(N,1);          % Initialize strength of each solution

for j = 1 : N,           % Loop over all N individuals and calculate their strength

    s( j , 1 ) = 1/N * sum ( sum ( bsxfun(@lt,Q(j,1:m),Q) , 2 ) == m );

end
```

The strength Pareto approach was introduced in SPEA/SPEA-2 (*Zitzler and Thiele*, 1999; *Zitzler et al.*, 2001) and details of how to compute the strength of each solution of the population can be found in the cited references. In words, the strength of a solution is equivalent to the reciprocal of the number of solutions it dominates. As solutions in the compromise region will have many more points in their niche than the solutions at the extreme ends, selection based on rank and strength will ensure that unique individuals continue to exist after each generation. This is a necessity for rapid evolution of the initial sample to the Pareto solution set. Benchmark experiments on known target distributions suggest that the crowding distance operator is preferred, but because this might not always be the case (e.g. real world applications) the user is free to experiment with the density operator.

Another strategy is to use differential weighting of the individual objective functions - a methodology introduced by *Zhang and Li* (2007) in MOEA/D and which has been implemented in AMALGAM$_{(D)}$ (*Vrugt*, 2016). This approach generates a perfectly uniform approximation of the Pareto front - and achieves

particularly excellent performance on problems involving more than $m = 2$ objectives.

2.2. Multimethod adaptation

Now I have discussed the computational heart of AMALGAM I am left with the question how to adapt the selection probability of each of the recombination methods. This adaptation method is designed specifically to favor recombination methods with higher reproductive success. I update the q-vector of selection probabilities, $\mathbf{p}_t$ after each generation, t using the following equation

$$p_{t+1}^j = (p_t^j)^{-1} \left(\frac{M_t^j}{\sum_{j=1}^q M_t^j} \right) \tag{3}$$

where $\mathbf{M}_t = \{M_t^1, \ldots, M_t^q\}$ is a q-vector with the number of offspring each recombination method, $j = \{1, \ldots, q\}$ contributes to $\mathbf{X}_t$. The first term on the right hand side takes into consideration the number of offspring a given operator has contributed to the offspring population, whereas the second term between brackets normalizes the reproductive success of each algorithm. In the absence of prior information about the expected performance of individual recombination method, I assume that they each create an equal number of offspring during the first generation.

The number of offspring each algorithm has contributed to the new parent population thus serves a guiding principle to determine the selection probability of each recombination method during the next generation. This ensures that the most productive genetic operators in AMALGAM are rewarded in the next generation by allowing them to generate more offspring. To avoid inactivation of a particular recombination method during the course of the optimization, the minimum selection probability is set larger than some nominal threshold, $p_{\min}$ (*Vrugt and Robinson*, 2007a). Prior information about the expected performance of individual recombination methods is readily incorporated in AMALGAM by using a q-vector of different $p_{\min}$ values.

2.3. Recombination methods: Selection and implementation

Now I have discussed the main algorithmic building blocks of AMALGAM, we have to determine which recombination methods to use to generate offspring. A host of different evolutionary algorithms could in principle be used, yet it would seem most productive to consider recombination methods that complement each other and, by definition, thus exhibit dissimilar searching behavior. Here, I combine differential evolution (*Storn and Price*, 1997), particle swarm optimization (*Kennedy et al.*, 2001), the adaptive Metropolis algorithm (*Haario et al.*, 2001), and the NSGA-II genetic algorithm (*Deb et al.*, 2002). Preliminary runs with a host of different recombination methods has shown that this group of recombination methods provides adequate performance across a range of different benchmark problems (*Vrugt and Robinson*, 2007a). I now shortly discuss the implementation of each of the four different recombination operators.

2.3.1. Differential Evolution

While traditional evolutionary algorithms are well suited to solve many difficult optimization problems, interactions among decision variables (parameters) introduces another level of difficulty in the evolution. Previous work has demonstrated the poor performance of a number of multiobjective evolutionary optimization algorithms, including the NSGA-II, in finding Pareto solutions for rotated problems exhibiting

strong interdependencies between parameters (*Deb et al.*, 2002). Rotated problems typically require correlated, self-adapting mutation step sizes to make timely progress in the optimization.

Differential evolution (DE) has been demonstrated to be able to cope with strong correlation among decision variables, and exhibits rotationally invariant behavior (*Storn and Price*, 1997). Unlike genetic algorithms and other evolutionary strategies, DE generates offspring using a fixed multiple of the difference between two or more randomly chosen parents of the population. I use variant DE/rand/1/bin of *Storn and Price* (1997) and create offspring as follows

$$\mathbf{y}_{i,t} = \mathbf{X}_{a,t-1} + \beta_1(\mathbf{X}_{b,t-1} - \mathbf{X}_{a,t-1}) + \beta_2(\mathbf{X}_{c,t-1} - \mathbf{X}_{d,t-1}), \tag{4}$$

where a, b, c, d are selected without replacement from the integers $\{1, \ldots, N\}$ and $\beta_1 \in (0, 1]$ and $\beta_2 \in (0, 1]$ are control parameter that determine the diversity of the offspring.

2.3.2. Adaptive Metropolis sampler

Evolutionary algorithms are prone to genetic drift in which the majority of the population is inclined to converge toward a single solution, thereby relinquishing occupations in other parts of the search space. The adaptive Metropolis sampler (AMS) is a Markov chain Monte Carlo (MCMC) simulation method that actively prevents the search of becoming mired in the relatively small region of a single best solution by adopting an evolutionary strategy that allows replacing parents with offspring of lower fitness (*Haario et al.*, 2001; *Vrugt et al.*, 2008a, 2009b; *Laloy and Vrugt*, 2012a). While this is a much appreciated strength, the AMS algorithm has another desirable property that is of more interest in the current study: the sampler is very efficient in sampling from high-dimensional target distributions. So, if our multimethod evolutionary optimization has progressed toward the Pareto-optimal front, then the AMS algorithm is able to rapidly explore the entire Pareto distribution, successively visiting and generating a large number of solutions.

The AMS recombination method creates offspring as follows

$$\mathbf{y}_{i,t} = \mathbf{X}_{i,t-1} + \mathcal{N}_d(0, \gamma\Sigma_{t-1}), \tag{5}$$

where $\mathcal{N}_d(a, b)$ is the d-variate normal distribution with mean a and covariance matrix, b, and γ is the jump rate which controls the spread (diversity) of the offspring. The covariance matrix is derived from the solutions of $\mathbf{X}_{t-1}$ with Pareto rank equivalent to one.

2.3.3. Particle Swarm Optimizer

Particle swarm optimization (PSO) is a population-based stochastic optimization method whose development was inspired from the flocking and swarm behavior of birds and insects. After its introduction by *Kennedy et al.* (2001), the method has gained rapid popularity in many fields of study. The method works with a group of potential solutions, called particles, and searches for optimal solutions by continuously modifying this population in subsequent generations. To start, the particles are assigned a random location and velocity in the d-dimensional search space. After initialization, each particle iteratively adjusts its position according to its own flying experience, and according to the flying experience of all other particles, making use of the best position encountered by itself, $\mathbf{x}_i^{\text{best}}$ and the entire population, $\mathbf{X}^{\text{best}}$. In contrast to standard genetic algorithms, PSO combines principles from local and global search to evolve a population of

points toward the Pareto-optimal front. The reproductive operator for creating offspring from an existing population is (*Kennedy et al.*, 2001)

$$\begin{aligned}\mathbf{v}_{i,t} &= \varphi\mathbf{v}_{i,t-1} + c_1 r_1(\mathbf{x}^{\text{best}}_{i,t-1} - \mathbf{X}_{i,t-1}) + c_2 r_2(\mathbf{X}^{\text{best}}_{t-1} - \mathbf{X}_{i,t-1}) \\ \mathbf{y}_{i,t} &= \mathbf{X}_{i,t-1} + \mathbf{v}_{i,t},\end{aligned} \tag{6}$$

where $\mathbf{v}_{i,t}$ represents the new velocity of the ith member of $\mathbf{X}_{t-1}$, φ is the inertia factor, c_1 and c_2 signify the cognitive and social factors of each particle, respectively, and r_1 and r_2 are drawn randomly from $\mathcal{U}[0,1]$, a standard uniform distribution.

To enhance search efficiency on multimodal problems I follow *Parsopoulos and Vrahatis* (2004) and perturb the offspring

$$\mathbf{y}_{i,t} = (1+\xi)\mathbf{y}_{i,t} \tag{7}$$

with a turbulence factor, $\xi \sim \mathcal{U}[-1,1]$.

2.3.4. Nondominated Sorted Genetic Algorithm, NSGA-II

The NSGA-II algorithm developed by *Deb et al.* (2002) has received the most attention of all evolutionary optimization algorithms because of its simplicity and demonstrated superiority over other existing methods. The algorithm uses well-known genetic operators of selection, crossover, and mutation to create a new population of points $\mathbf{Y}_t$ from an existing population, $\mathbf{X}_{t-1}$. I use simulated binary crossover (SBX) and polynomial mutation (*Deb and Agrawal*, 1995) to create offspring.

3. AMALGAM : Pseudo-code

I now provide a pseudo-code of AMALGAM. The variable $\Phi(\cdot|\mathbf{p})$ signifies the discrete multinomial distribution on $\cdot$ with selection probability $\mathbf{p} = (p_1, \ldots, p_q)$; $p_j \geq 0$ and $\sum_{j=1}^{q} p_j = 1$.

Algorithm: AMALGAM
Step 0: Problem formulation
Define $\mathbf{F}(\mathbf{x})$ which returns m values, $m \geq 2$
Define d, the dimensionality of $\mathbf{x}$, $d \geq 1$
$N \leftarrow$ population size, $N \geq 10$
$T \leftarrow$ maximum number of generations, $T \geq 2$
$m \leftarrow$ number of objectives, $m \geq 2$
Define $\mathcal{X}$, the domain ranges, $\mathbf{x} \in \mathcal{X} \in \mathbb{R}^d$
Step 1: Set main algorithmic variables
rec_methods $\leftarrow$ define constituent search methods
$p_{\text{cr}} \leftarrow$ NSGA crossover probability, $p_{\text{cr}} \in [0,1]$
$p_{\text{m}} \leftarrow$ NSGA mutation probability, $p_{\text{m}} \in [0,1]$
$\eta_{\text{C}} \leftarrow$ NSGA crossover distribution index, $\eta_{\text{C}} \in [1,250]$
$\eta_{\text{M}} \leftarrow$ NSGA mutation distribution index, $\eta_{\text{M}} \in [1,250]$
$\gamma \leftarrow$ AMS jumprate, $\gamma \geq 0$
$\beta_1 \leftarrow$ DE scaling factor, $\beta_1 \in [0,2]$

$\beta_2 \leftarrow$ DE scaling factor, $\beta_2 \in [0, 2]$
$c_1 \leftarrow$ PSO social factor, $c_1 \in [1, 2]$
$c_2 \leftarrow$ PSO cognitive factor, $c_2 \in [1, 2]$
$\varphi \leftarrow$ PSO inertia factor, $\varphi \in [0, 1]$
$K \leftarrow$ thinning rate, $K \geq 1$
$p_{\min} \leftarrow$ min. selection probability q offspring operators, $p_{\min} \in [0, 1/q]$
Step 2: Initialization
Define $\mathbf{p}_0 = \{p_0^1, \ldots, p_0^q\}$
Step 3: At generation $t = 0$
for $i = 1, \ldots, N$ **do**
    Sample $\mathbf{X}_0 \in \mathcal{X}$ using initial sampling distribution
    Evaluate model (function), $\mathbf{H}_i = \mathbf{F}(\mathbf{X}_i)$
end for
Step 4: At generation $1 \leq t \leq T$
for $i = 1, \ldots, N$ **do**
    Sample $j \sim \Phi(\{1, \ldots, q\}|\mathbf{p}_{t-1})$
    Create offspring, $\mathbf{Y}_{i,t}$ from $\mathbf{X}_{t-1}$ using the jth recombination method
    Repair infeasible offspring (or not)
    Evaluate model (function), $\mathbf{G}_{i,t} = \mathbf{F}(\mathbf{Y}_{i,t})$
end for
Step 5: Update population
Combine parents and offspring, $\mathbf{R}_t = \mathbf{X}_{t-1} \bigcup \mathbf{Y}_t$
Calculate rank and crowding distance of $\mathbf{R}_t$
Store in $\mathbf{X}_t$ the N members of $\mathbf{R}_t$ based on rank and crowding distance
Step 6: Search adaptation
Calculate $\mathbf{p}_t$ based on reproductive success (Equation 3Multimethod adaptationequation.2.3)
Step 7: Convergence check
If criteria satisfied, stop, otherwise go back to **Step 4**

The pseudo code of AMALGAM presented herein assumes serial evaluation of the N different children of the offspring population. This implementation is not particularly appealing, particularly if the forward model that is used to evaluate the fitness of each solution, $\mathbf{F}(\mathbf{x})$, is computationally demanding. Fortunately, many nature-inspired algorithms that rely on evolutionary principles such as survival of the fittest, are embarrassingly parallel. Indeed, it is not difficult to see that each of the N children of the offspring population can be evaluated in parallel using a distributed computing network. This approach, accelerates, sometimes dramatically, the computational efficiency of AMALGAM, thereby permitting multi-criteria inference of CPU-intensive system models. The software package of AMALGAM that I describe in the next section allows the user to activate multi-core evaluation of the offspring population using the distributed computing toolbox of MATLAB.

4. AMALGAM: MATLAB implementation

The basic code of AMALGAM was written in 2006 but many new functionalities and options have been added to the source code in recent years to support the needs of users. You can download the AMALGAM toolbox from my website at the following link `http://faculty.sites.uci.edu/jasper/software`. Appendix A explains how to setup the AMALGAM toolbox in MATLAB.

The AMALGAM code can be executed from the MATLAB prompt by the command

```
[X,F,output,Z] = AMALGAM(Func_name,AMALGAMPar,Par_info,
```

where `Func_name` (string), `AMALGAMPar` (structure array), and `Par_info` (structure array) are input arguments defined by the user, and `X` (matrix), `F` (matrix), `output` (structure array) and `Z` (matrix) are output variables computed by AMALGAM and returned to the user. An additional output argument SIM contains the simulations of the forward model, and is available if the user activates memory storage of the model output, details will be discussed later. To minimize the number of input and output arguments in the AMALGAM function call and related primary and secondary functions called by this program, I use MATLAB structure arrays and group related variables in one main element using data containers called fields, more of which later. Three optional input arguments that the user can pass to AMALGAM are the variables `options`, `func_in` and `Fpar` and their content and usage will be discussed below.

The AMALGAM function uses more than twenty other functions to help evolve the initial population to the Pareto distribution. All these functions are summarized briefly in Appendix B. In the subsequent sections I will discuss the MATLAB implementation of AMALGAM. This, along with prototype case studies presented herein and template input files should help users apply multi-criteria inference to their own models and data.

4.1. Input argument 1: Func_Name

The variable `Func_name` defines the name (enclosed in quotes) of the MATLAB function (.m file) used to calculate the objective functions of each parameter vector, `x`. The use of a m-file rather than anonymous function (e.g. `pdf`), permits AMALGAM to solve multi-criteria inference problems involving, for example, dynamic simulation models. If I conveniently assume `Func_name` to be equivalent to 'model' then the call to this function becomes

```
F = model(X,func_in),                                    (8)
```

where `x` (input argument) is a $d \times 1$ vector of parameter values, `func_in` is an optional input argument that allows the user to pass additional information to the model script, and `F` is a return argument with the values of the m different objective functions. The user is free to use a row or column vector for the elements of `F`, as long as it contains m values. The content of the variable `func_in` can be determined by the user. It can be declared a scalar, vector, matrix, structure, string or cell-array, whatever the user deems most appropriate for their respective model plugin.

The function `model` needs to be supplied by the user, yet its syntax is universal. Appendix C presents four different templates for the function `model` which are used in the case studies presented in section 5Numerical examplessection.5.

4.2. Input argument 2: `AMALGAMPar`

The structure `AMALGAMPar` defines the computational settings of AMALGAM. Table 1Main algorithmic variables of AMALGAM: Mathematical symbols, corresponding fields of `AMALGAMPar` and their default settings.table.caption.3 lists the different fields of `AMALGAMPar`, their default values, and the corresponding variable names used in the mathematical description of AMALGAM in section 3AMALGAM : Pseudo-codesection.3.

Table 1: Main algorithmic variables of AMALGAM: Mathematical symbols, corresponding fields of `AMALGAMPar` and their default settings.

Symbol	Description	Field `AMALGAMPar`	Default
	Problem dependent		
d	number of parameters	d	
N	population size	N	> 10
T	number of generations	T	
m	number of objective functions	m	> 1
	Problem independent		
	recombination methods	rec_methods	{'GA','PSO','AMS','DE'}
p_{cr}	NSGA crossover probability	p_cr	0.9
p_{m}	NSGA mutation probability	p_m	$1/d$
η_{C}	NSGA crossover distribution index	eta_C	10
η_{M}	NSGA mutation distribution index	eta_M	50
γ	AMS jump rate	gamma	$(2.38/\sqrt{d})^2$
β_1	DE scaling factor: variant-DE-I	beta_1	$\mathcal{U}_d(0.6, 1.0)$
β_2	DE scaling factor: variant-DE-II	beta_2	$\mathcal{U}_d(0.2, 0.6)$
c_1	PSO social factor	c_1	1.5
c_2	PSO cognitive factor	c_2	1.5
φ	PSO inertia factor	varphi	$\mathcal{U}(0.5, 1.0)$
K	thinning rate	K	1
p_{min}	minimum selection probability	p_min	0.05

The names of the different fields of `AMALGAMPar` are equivalent to the symbols (letters) used in the (mathematical) description of AMALGAM. The values of the fields d, N, T and m depend on the dimensionality of the Pareto distribution, and hence should be defined by the user. Default settings are assumed for the remaining fields of `AMALGAMPar`. These default settings are easily modified by the user, by simply specifying individual fields of `AMALGAMPar` explicitly and their respective values.

The field K of `AMALGAMPar` allows the user to specify the thinning rate of output argument `Z` which for $K = 1$ stores in a single matrix the final population at the end of each generation. This matrix stores a much larger number of Pareto solutions than present in the final population, and thus helps delineate sharply the nondominated solution set. Storage of each past population can be problematic however, for instance, for a $d = 100$ dimensional Pareto distribution with $m = 2$ objectives, $N = 100$ and $T = 1,000$, MATLAB would need a staggering 10.2-million bytes of memory to store all the T populations and their corresponding objective function values. In this case, thinning can be used to reduce memory requirements by storing only every Kth population. This option reduces memory storage with a factor of T/K, and also decreases the autocorrelation between parents of successively stored populations. A default value of $K = 1$ (no thinning) is assumed in AMALGAM. Note, large values for K ($K >> 10$) can be rather wasteful because many parents are not used in the computation of the Pareto moments and/or plotting of marginal/bivariate parameter distributions.

4.3. Input argument 3: `Par_info`

The structure `Par_info` stores all necessary information about the parameters of the target distribution, for instance their prior uncertainty ranges (for bounded search problems), initial values (how to draw initial population), prior distribution (if prior information is available) and boundary handling (what to do if out of feasible space), respectively. Table 2AMALGAM input argument `Par_info`: Different fields, description, options, default settings and variable types.table.caption.4 lists the different fields of `Par_info` and summarizes their content, default values and variable types.

Table 2: AMALGAM input argument `Par_info`: Different fields, description, options, default settings and variable types.

Field `Par_info`	Description	Options	Default	Type
initial	Initial sample	'uniform'/'latin'/'normal'/'prior'		string
min	Minimum values		-Inf^d	$1 \times d$-vector
max	Maximum values		Inf^d	$1 \times d$-vector
boundary	Boundary handling	'reflect'/'bound'/'fold'/'none'	'none'	string
mu	Mean 'normal'			$1 \times d$-vector
cov	Cov. 'normal'			$d \times d$-matrix
prior	Prior distribution			cell array †

† Data type with containers called cells. Each cell can contain any type of data.

The field initial of `Par_info` specifies with a string enclosed between quotes how to sample the initial population of N parents. The user can select from (1) 'uniform' random, (2) 'latin' hypercube, and (3) multivariate 'normal' sampling, and (4) sampling from a user-defined 'prior' distribution. Option (1) and (2) require specification of the fields min and max of `Par_info` that define with $1 \times d$ vectors the lower and upper bound values of each of the parameters, respectively. Option (3) 'normal' necessitates definition of fields mu ($1 \times d$ -vector) and cov ($d \times d$-matrix) of `Par_info` which store the mean and covariance matrix of the multivariate normal distribution, respectively. Finally, for 'prior' (option 4) the user needs to specify as cell array field prior of `Par_info`, for example

$$\texttt{Par\_info.prior} = \{\texttt{'normrnd(-2,0.1)'},\texttt{'trnd(10)'},\texttt{'unifrnd(-2,4)'}\} \tag{9}$$

uses a normal distribution with mean of -2 and standard deviation of -0.1 for the first parameter, a Student distribution with $\nu = 10$ degrees of freedom for the second dimension, and a uniform distribution between -2 and 4 for the last parameter of the target distribution, respectively. The first three options assume the prior distribution to be noninformative (uniform/flat), and consequently do not favor a-priori any particular values of the parameters. On the contrary, if an explicit 'prior' distribution is used then the samples of each dimension will follow the marginal distribution of prior.

The fields min and max of the structure `Par_info` serve two purposes. First, they define the feasible parameter space from which the initial population of parents is drawn if 'uniform' random or 'latin' hypercube sampling is used. Second, they can define a bounded search domain for problems involving one or more parameters with known physical/conceptual ranges. This does however require the bound to be actively enforced during chain evolution. Indeed, offspring generated with the recombination methods can fall outside the hypercube defined by min and max even if the initial state of each chain are well within the feasible search space. The field boundary of `Par_info` provides several options what to do if the parameters are outside their respective ranges. The four different options that are available are (1) 'bound', (2) 'reflect', (3) 'fold', and

(4) 'none' (default). These methods are illustrated graphically in Figure 4Different options for parameter boundary handling in the AMALGAM package, a) set to bound, b) reflection, and c) folding.figure.caption.5 and act on one parameter at a time.

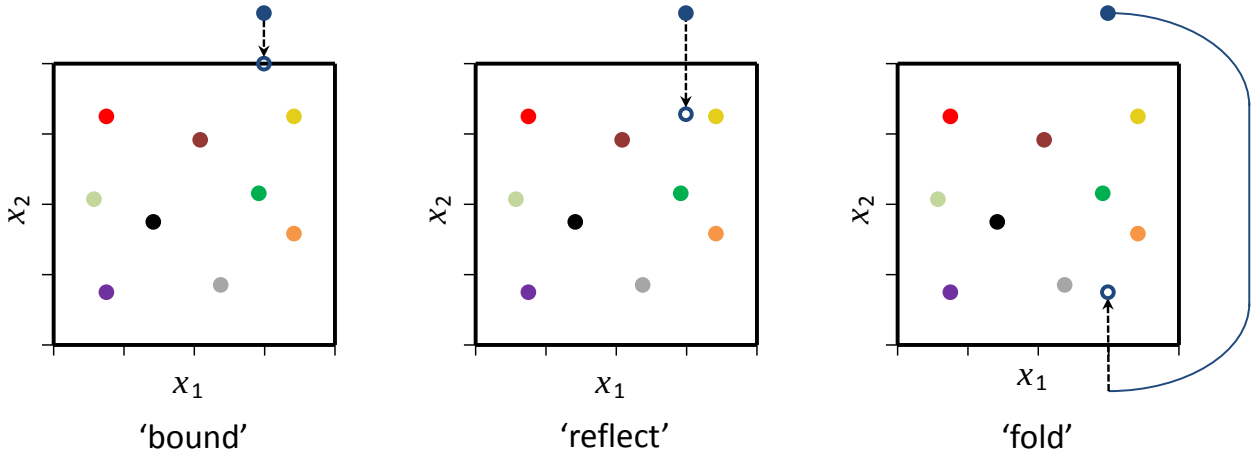


Figure 4: Different options for parameter boundary handling in the AMALGAM package, a) set to bound, b) reflection, and c) folding.

The option 'bound' is most simplistic and simply resets each dimension of the d-vector of parameters that is outside the bound equal to its respective bound. The option 'reflect' is somewhat more refined and views the boundary of the search space as a mirror through which individual dimensions are reflected back into the parameter space. The size of this reflection is set equal to the "amount" of boundary violation. The 'bound' and 'reflect' boundary treatment options are used often in the field of optimization concerned with finding the global optimum of a given cost or objective function. The third option 'fold' connects the upper and lower bound of the parameter space so as to create a continuum representation. Note, this approach can lead to "bad" children if at least one of the dimensions of the Pareto distribution is located at the edges of the search domain. For those dimensions the parameter values can transition directly from low to high values, and vice versa.

Practical experience suggests that a reflection approach works well in practice. The option 'bound' is least recommended as it can collapse the parameter values of a given dimension to a single point, thereby not only loosing diversity for sampling but also inflating the density of solutions at the bound. Most evolutionary algorithms apply the 'bound' option. For some test functions, with optimum for at least some of the dimensions on the bound this can artificially enhance the convergence rate to the nondominated solution set.

4.4. (Optional) input argument 4: options

The structure `options` is optional and passed as fourth input argument to AMALGAM. The fields of this structure can activate (among others) file writing, workspace saving, storage of model output, and distributed multi-core calculation. Table 3Content of (optional) input structure `options`. This fourth input argument of AMALGAM is required to activate built-in functionalities such as distributed multi-processor

calculation, workspace saving, and file writing.table.caption.6 summarizes the different fields of `options` and their default settings.

Table 3: Content of (optional) input structure `options`. This fourth input argument of AMALGAM is required to activate built-in functionalities such as distributed multi-processor calculation, workspace saving, and file writing.

Field of `options`	Description	Options	Default
parallel	Distributed multi-core calculation?	'no'/'yes'	'no'
IO	If parallel, IO writing of model?	'no'/'yes'	'no'
modout	Store output of model?	'no'/'yes'	'no'
save	Save AMALGAM workspace?	'no'/'yes'	'no'
restart	Restart run? ('save' required)	'no'/'yes'	'no'
density	Approach to compute density points?	'crowding'/'strength'	'crowding'
print	Print results to screen (figures/tables)	'no'/'yes'	'yes'

Multi-core calculation takes advantage of the MATLAB Parallel Computing Toolbox and evaluates the N different children created with AMALGAM on a different processor. This is especially useful if the forward model, `model` is CPU-demanding and requires at least a few seconds (often more) to run. The field IO of `options` determines the setup of the distributed computing environment. If file writing is used to communicate between AMALGAM and an external executable called from `model` using the built-in `dos` or `unix` command, then each processor is setup automatically to work in a different directory. This is a simple solution to file overwriting and corruption that occurs if multiple different children are evaluated in the same directory. If `model` consists of MATLAB code only with/without shared libraries linked through the built-in MEX-compiler then the parameter values, $\mathbf{x}$ can be passed directly to this .mex or .dll function and a single common directory for all workers suffices.

For CPU-intensive forward models it is convenient to not only store the parameter samples but also keep in memory the corresponding model simulations returned by `model` and used to calculate the objective function values of each parent/child. This avoids having to rerun the `model` script again after AMALGAM has terminated to assess Pareto simulation uncertainty. The field modout of `options` determines whether to store the output of the `model` script. If model output writing is activated then the N simulations of each population $\mathbf{X}$ are stored in memory after each generation. These simulations are then returned to the user as fifth output argument, `sim` of AMALGAM. If thinning is activated then this applies to the simulations stored in `sim` as well.

To help evaluate the progress of AMALGAM, it can be useful to periodically store the MATLAB workspace to a file. If the field save of `options` is set to 'yes' then all variables in the workspace of the AMALGAM function are saved to the file "AMALGAM.mat" at completion of each successive generation. This binary file is not only useful to analyze, on the fly, the results of AMALGAM, but is also a necessary requirement to restart AMALGAM (field restart of structure `options` is set to 'yes') and continue a previously aborted run or to extend an AMALGAM trial if convergence to the Pareto front has not been achieved with the assigned computational budget in `AMALGAMPar.T`. In this last case, the user can list in an ascii file called "T.txt" how many additional generations should be performed. This text file should be saved to the relevant case study folder, which also contains the file "AMALGAM.mat". For CPU-intensive models I recommend to save the workspace, that is, `options.save = 'yes'`), in case a restart run is required to enhance further the approximation of the Pareto front.

The field density of `options` determines which method is used to calculate the density of each solution of

the population. The default option is the crowding distance operator, explicated on Page 10lstlisting.-2 and implemented in the NSGA-II algorithm (*Deb et al.*, 2002). Alternatively, the user can select the strength Pareto approach of *Zitzler and Thiele* (1999) and *Zitzler et al.* (2001) which is used by SPEA and SPEA-2. The MATLAB script on Page 11lstlisting.-3 details how to compute the Pareto strength of each solution of the population. Practical experience suggests that the crowding distance operator is preferred and leads to the best approximation of the Pareto front - yet, a synergy of both methods might receive the best performance - an idea that deserves further investigation.

Finally, the field `print` of structure `options` allows the user to control output writing to the screen. This output consists of tables and figures that summarize the results of AMALGAM. Graphical output is suppressed if the field `print` of structure `options = 'no'`.

4.5. (Optional) input argument 5: func_in

The variable `func_in` constitutes the fifth (optional) input argument of AMALGAM. This variable allows the user to pass to `model` any additional input arguments (beyond the parameter values) that are required to calculate the objective function values. The content of `func_in` can be determined by the user. It can be declared a scalar, vector, matrix, structure, string or cell-array, whatever is deemed appropriate for the model plugin. The different case studies in section 5Numerical examplessection.5 illustrate how to use the variable `func_in`. These templates can be used for other models/functions (plugins) as well.

4.6. (Optional) input argument 6: Fpar

The sixth and last input argument `Fpar` of the AMALGAM function is optional and useful only for case studies with a known Pareto solution set. The matrix `Fpar` is of size $n \times m$ and stores n different Pareto solutions (in objective space). This input argument allows AMALGAM to compute convergence metrics such as the hypervolume (*While et al.*, 2006; *Beume et al.*, 2009) and widely used inverse generational distance. The inverted generation distance, or IGD-metric *Veldhuizen* (1998); *Li and Zhang* (2009) is used to measure the distance of the solutions, $\mathbf{A} = \{\mathbf{a}^1, \ldots, \mathbf{a}^N\}$ to a reference set, $\mathbf{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^n\}$ of n uniformly distributed Pareto solutions,

$$\mathrm{IGD}(\mathbf{A}, \mathbf{P}) = \frac{1}{n} \sum_{l=1}^{n} \min_{1 \leq i \leq N} ||\mathbf{p}^l - \mathbf{a}^i||, \tag{10}$$

where $||\cdot||$ denotes the Euclidean distance. If the reference set is sufficient large, say $n > 250$, the IGD-metric measures not only the distance of the sampled solutions to the true Pareto front, but also their spread (diversity). The IGD values are strictly positive. The closer its value to zero, the better the approximation of the true Pareto front.

4.7. Output arguments

I now briefly discuss the three output (return) arguments of AMALGAM including `X`, `F`, `output` and `Z`. These four variables summarize the results of the AMALGAM algorithm and are used for plotting of the Pareto front, analysis of the nondominated solution set, convergence assessment, and investigation of the selection probabilities of the individual recombination methods.

The variable `X` is a matrix of size $N \times d$ with the parameter vectors (as rows) of the final population. The corresponding objective function values are stored in the $N \times m$ matrix `F`.

The following MATLAB command

```
plot(F(1:AMALGAMPar.N,1),F(1:AMALGAMPar.N,2),'r+')
```
(11)

plots the objective function space of the N members of the final population. If the analysis involves three-different objective functions, then the statement

```
plot3(F(:,1),F(:,2),F(:,3),'r+')
```
(12)

can be used to create a three-dimensional plot of the sampled objective function values of the final population.

The structure `output` contains important (diagnostic) information about the progress of the AMALGAM algorithm. The field RunTime (scalar) stores the wall-time (seconds), p_alg (matrix) the selection probability of each of the recombination methods, and IGD (matrix) the value of the IGD-convergence diagnostic, after each successive generation.

The MATLAB command

```
plot(output.IGD(1:end,1),output.IGD(1:end,2),'b')
```
(13)

generates a trace plot of the IGD-convergence diagnostic.

The matrix `Z` ($NT \times d+m$) stores the population and their corresponding objective function values after each successive generation. If thinning is applied then the number of rows of `Z` is reduced and equivalent to $NT/K+1; K \geq 2$.

Finally, a fifth output argument, the matrix `Y_sim` is available if the user has set the field modout of structure `options`. This output argument can be used to return to the user the model simulations of the Pareto solutions to determine the simulation uncertainty of the rank 1 solutions. The matrix `Y_sim` is only available for dynamic models involving (spatio)temporal simulation of one or more variables of interest. This mandates the function `model` to return the vector/matrix of simulated variables as second output argument besides the vector of objective function values. Case study 3 in the next section will illustrate how to do this in practice.

The directory "\postprocessing" (in main folder) contains a number of different functions that can be used to tabulate and visualize the different output arguments of AMALGAM. The script postproc_AMALGAM is executed from the MATLAB prompt after the main AMALGAM function has terminated its calculations. Appendix D provides an overview of the graphical output of AMALGAM for the last (fourth) case study.

5. Numerical examples

I now demonstrate the application of the MATLAB AMALGAM package to four different multi-criteria optimization problems. These case studies cover a diverse set of problem features and involve (among others) multimodality, local optima, and high-dimensional Pareto distributions.

5.1. Case Study I: ZDT1

I revisit the two-objective scalable test function suite of *Zitzler et al.* (2000) and consider ZDT1 which has a convex Pareto-optimal front and is given by

$$\arg\min_{\mathbf{x}\in\mathcal{X}} \mathbf{F}(\mathbf{x}) = \begin{cases} f_1(\mathbf{x}) = x_1 \\ f_2(\mathbf{x}) = g(\mathbf{x})h(\mathbf{x}), \end{cases} \tag{14}$$

where

$$g(\mathbf{x}) = 1 + \frac{9}{d-1}\sum_{i=2}^{d} x_i \quad , \quad h(\mathbf{x}) = 1 - \sqrt{\frac{x_1}{g(\mathbf{x})}}, \tag{15}$$

and $x_i \in [0, 1]$. The Pareto optimal front is formed with $g(\mathbf{x}) = 1$. Equation 14Case Study I: ZDT1equation.5.14 is solved numerically in the script `AMALGAM_ZDT1` of Appendix C.

I derive the Pareto distribution of ZDT1 assuming $d = 30$ using the input file called "example_1.m" in the folder \example_1 in the main directory of the AMALGAM toolbox.

MATLAB input script "example_1.m", with the setup of case study I: Test function ZDT1. The last line of this script calls the main program of the AMALGAM toolbox.

```
%
    ------------------------------------------------------------------------------
     %
%       AAA       MM        MM       AAA       LL         GGGGGGGGGG       AAA       MM
    MM   %
%      AA AA      MMM      MMM      AA AA      LL         GGG      GGG      AA AA      MMM
    MMM   %
%      AA AA      MMMM    MMMM      AA AA      LL         GG        GG      AA AA      MMMM
    MMMM   %
%     AA   AA     MM MM MM MM     AA   AA     LL         GGG      GGG     AA   AA     MM MM MM
    MM   %
%    AAAAAAAAA    MM  MMM  MM    AAAAAAAAA    LL         GGGGGGGGGG    AAAAAAAAA    MM  MMM
    MM   %
%    AA     AA    MM        MM    AA     AA    LL                 GG    AA     AA    MM
    MM   %
%   AA       AA   MM        MM   AA       AA   LLLLLLLL           GG   AA       AA   MM
    MM   %
%   AA       AA   MM        MM   AA       AA   LLLLLLLL  GGGGGGGGGG   AA       AA   MM
    MM   %
%
    ------------------------------------------------------------------------------
     %

%% Define fields of AMALGAMPar
AMALGAMPar.N = 100;                         % Define population size
AMALGAMPar.T = 50;                          % How many generations?
AMALGAMPar.d = 30;                          % How many parameters?
AMALGAMPar.m = 2;                           % How many objective functions?

%% Define fields of Par_info
Par_info.initial = 'latin';                 % Latin hypercube sampling
Par_info.boundhandling = 'bound';           % Explicit boundary handling
Par_info.min = zeros(1,AMALGAMPar.d);       % If 'latin', min values
Par_info.max = ones(1,AMALGAMPar.d);        % If 'latin', max values

%% Define name of function (Zitzler et al., Evolutionary Computation, 8 (2), 183-195,
    2000)
Func_name = 'AMALGAM_ZDT1';

%% Define func_in (additional input to "model" function)
func_in = AMALGAMPar.d;

%% Now load Pareto front -- this is a benchmark problem
Fpar = load('ZDT1.txt');                    % Load Pareto solution set (known)

%% Define structure options
options.print = 'yes';                      % Print output to screen (tables and figures
    )

%% Run the AMALGAM code and obtain non-dominated solution set
[X,F,output,Z] = AMALGAM(AMALGAMPar,Func_name,Par_info,options,func_in,Fpar);
```

The initial sample is drawn using Latin hypercube sampling, and boundary handling is activated to enforce the parameters to stay within their prior ranges. The asc-ii file "ZDT1.txt" contains an equidistant sample of the known Pareto front, and is used by AMALGAM to compute the evolution of the IGD convergence diagnostic.

Figure 5(A) True (black line) and AMALGAM sampled (red dots) Pareto optimal front. (B) Trace plot of the IGD convergence metric - the closer the value of this diagnostic to zero the better the population approximates the true nondominated solution set.figure.caption.7 (left hand side) plots the objective function values of the final population (red dots) after 100 generations. The true Pareto optimal front is indicated with the solid black line. The plot at the right hand side displays the evolution of the IGD convergence metric.

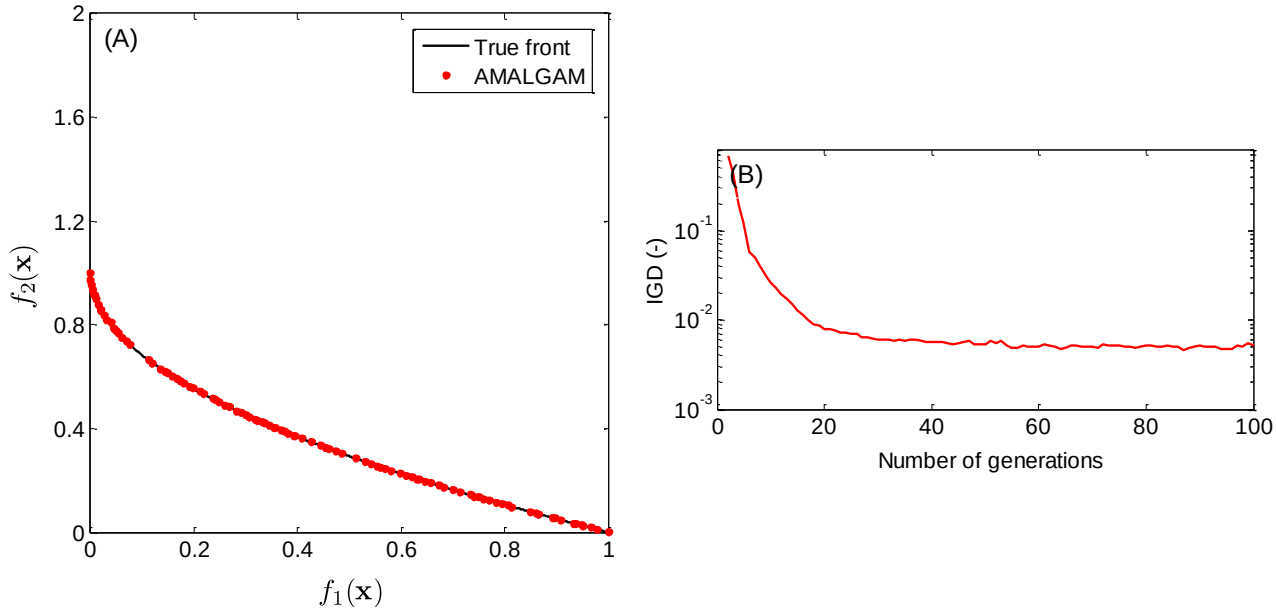


Figure 5: (A) True (black line) and AMALGAM sampled (red dots) Pareto optimal front. (B) Trace plot of the IGD convergence metric - the closer the value of this diagnostic to zero the better the population approximates the true nondominated solution set.

The samples generated with AMALGAM closely approximate the true Pareto optimal front. About $2,000$ ZDT1 function evaluations are required to converge within 0.01 of the known Pareto front.

5.2. Case Study II: ZDT4

Our second case study involves test function ZDT4 of *Zitzler et al.* (2000). This problem involves 21^9 local Pareto optimal fronts, and is therefore a strong test for AMALGAM's ability to deal with multimodality. The ZDT4 test function is given by

$$\arg\min_{\mathbf{x}\in\mathcal{X}} \mathbf{F}(\mathbf{x}) = \begin{cases} f_1(\mathbf{x}) = x_1 \\ f_2(\mathbf{x}) = g(\mathbf{x})h(\mathbf{x}), \end{cases} \tag{16}$$

where

$$g(\mathbf{x}) = 1 + 10(d-1) + \sum_{i=2}^{d}\left(x_i^2 - 10\cos(4\pi x_i)\right) \quad , \quad h(\mathbf{x}) = 1 - \sqrt{\frac{x_1}{g(\mathbf{x})}}, \tag{17}$$

$x_1 \in [0,1]$ and $x_2, \ldots, x_m \in [-5,5]$. The global Pareto-optimal front is formed with $g(\mathbf{x}) = 1$, the best local Pareto-optimal front with $g(\mathbf{x}) = 1.25$. Note that not all local Pareto-optimal sets are distinguishable in the objective space. Equation 16Case Study II: ZDT4equation.5.16 is solved numerically in script `AMALGAM_ZDT4` of Appendix C.

I derive the Pareto distribution of ZDT4 for $d = 10$ using the input file called "example_4.m" in the folder \example_4 in the main directory of the AMALGAM toolbox.

MATLAB input script "example_4.m", with the setup of case study II: Test function ZDT4. The last line of this script calls the main program of the AMALGAM toolbox.

```
%
    ------------------------------------------------------------------------------
     %
%       AAA       MM        MM       AAA       LL         GGGGGGGGGG       AAA       MM
    MM   %
%      AA AA      MMM      MMM      AA AA      LL         GGG      GGG      AA AA      MMM
    MMM   %
%      AA AA      MMMM    MMMM      AA AA      LL         GG        GG      AA AA      MMMM
    MMMM   %
%     AA   AA     MM MM MM MM     AA   AA     LL         GGG      GGG     AA   AA     MM MM MM
    MM   %
%    AAAAAAAAA    MM  MMM  MM    AAAAAAAAA    LL         GGGGGGGGGG    AAAAAAAAA    MM  MMM
    MM   %
%    AA     AA    MM        MM    AA     AA    LL                 GG    AA     AA    MM
    MM   %
%   AA       AA   MM        MM   AA       AA   LLLLLLLL           GG   AA       AA   MM
    MM   %
%   AA       AA   MM        MM   AA       AA   LLLLLLLL  GGGGGGGGGG   AA       AA   MM
    MM   %
%
    ------------------------------------------------------------------------------
     %

%% Define fields of AMALGAMPar
AMALGAMPar.N = 100;                             % Define population size
AMALGAMPar.T = 100;                             % How many generations?
AMALGAMPar.d = 10;                              % How many parameters?
AMALGAMPar.m = 2;                               % How many objective functions?

%% Define fields of Par_info
Par_info.initial = 'latin';                     % Latin hypercube sampling
Par_info.boundhandling = 'bound';               % Explicit boundary handling
Par_info.min = [0 -5*ones(1,AMALGAMPar.d-1)];   % If 'latin', min values
Par_info.max = [1  5*ones(1,AMALGAMPar.d-1)];   % If 'latin', max values

%% Define name of function
Func_name = 'AMALGAM_ZDT4';

%% Define func_in (additional input to "model" function)
func_in = AMALGAMPar.d;

%% Now load Pareto front -- this is a benchmark problem
Fpar = load('ZDT4.txt');                        % Load Pareto solution set (known)

%% Define structure options
options.ranking = 'C';                          % Use C script for ranking (faster)
options.print = 'yes';                          % Print output to screen (tables and
    figures)

%% Run the AMALGAM code and obtain non-dominated solution set
[X,F,output,Z] = AMALGAM(AMALGAMPar,Func_name,Par_info,options,func_in,Fpar);
```

The initial sample is drawn using Latin hypercube sampling, and boundary handling is used to enforce the parameters to stay within their respective ranges. The known Pareto optimal front, stored in the file "ZDT4.txt", is used by AMALGAM to monitor convergence of the rank 1 solutions via the IGD statistic.

To demonstrate the advantages of multimethod optimization, consider Figure 6Pareto-optimal fronts after 5, 25, and 50 generations with NSGA-II (blue diamonds), PSO (green circles), AMS (cyan plusses), DE (yellow crosses), and AMALGAM (red crosses) optimization algorithms for test problem ZDT4. This benchmark problem has 21^9 different local Pareto-optimal fronts in the search space, of which only one corresponds to the global Pareto-optimal front. The true Pareto optimal front is separately indicated with the solid black line. Combining the individual algorithms into a adaptive multimethod search algorithm ensures a faster and more reliable solution to multiobjective optimization problems.figure.caption.8, which plots the nondominated fronts generated with the individual NSGA-II (blue diamonds), PSO (green circles), AMS (cyan plusses) and DE (yellow crosses) recombination methods, and AMALGAM (red crosses) after $2,500$, $5,000$ and $7,500$ function evaluations. The true Pareto distribution is separately indicated in each graph with a black line. After only 7,500 function evaluations, AMALGAM has progressed toward the true Pareto-optimal front, and has generated solutions that are far more evenly distributed along the Pareto front than any of the individual algorithms.

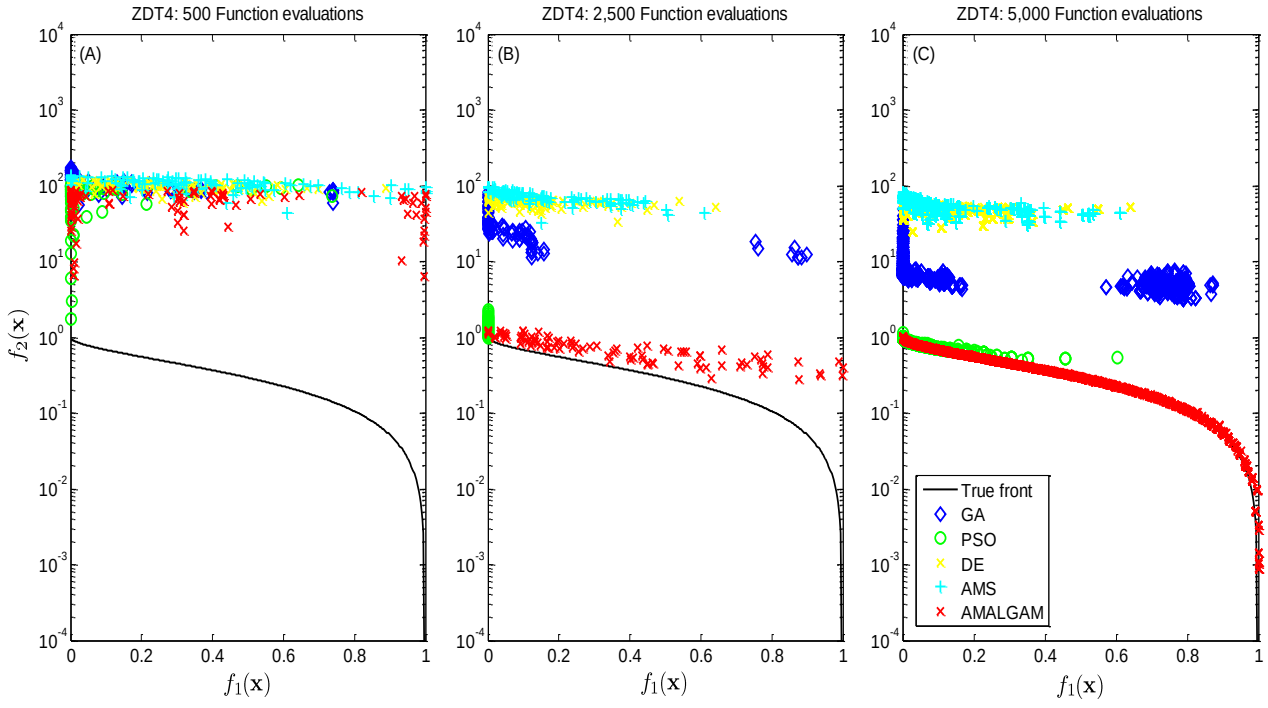


Figure 6: Pareto-optimal fronts after 5, 25, and 50 generations with NSGA-II (blue diamonds), PSO (green circles), AMS (cyan plusses), DE (yellow crosses), and AMALGAM (red crosses) optimization algorithms for test problem ZDT4. This benchmark problem has 21^9 different local Pareto-optimal fronts in the search space, of which only one corresponds to the global Pareto-optimal front. The true Pareto optimal front is separately indicated with the solid black line. Combining the individual algorithms into a adaptive multimethod search algorithm ensures a faster and more reliable solution to multiobjective optimization problems.

Figure 7Illustration of the concept of self-adaptation in multimethod evolutionary optimization. (A)

Evolution of the selection probability of each individual recombination method used in AMALGAM. (B) Trace plot of the IGD convergence diagnostic. These results illustrate the utility of individual search algorithms during different stages of the optimization, and provide numerical evidence for the 'No Free Lunch' theorem of *Wolpert and Macready* (1997).figure.caption.9 presents a trace plot of the selection probabilities of the recombination methods used by AMALGAM. Initially, the NSGA-II algorithm (blue) exhibits the highest reproductive success, yet after about 20 generations, the DE (yellow), AMS (cyan), and PSO (green) algorithms are suddenly more favored. This combination of methods proves to be extremely effective at increasing the diversity of solutions along the Pareto front once the NSGA-II method does its job. The performance of AMALGAM on the other benchmark problems provides further justification for this conclusion.

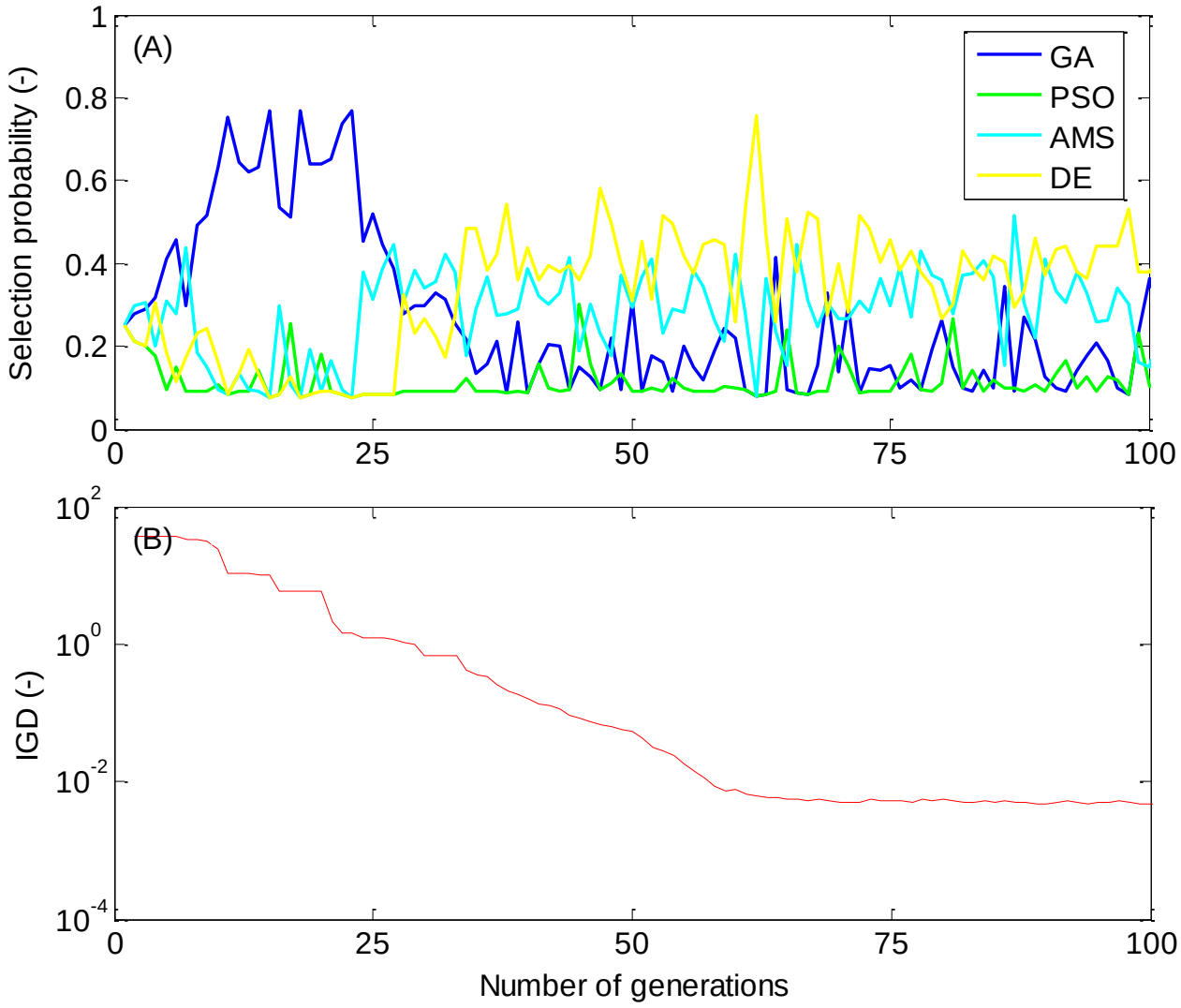


Figure 7: Illustration of the concept of self-adaptation in multimethod evolutionary optimization. (A) Evolution of the selection probability of each individual recombination method used in AMALGAM. (B) Trace plot of the IGD convergence diagnostic. These results illustrate the utility of individual search algorithms during different stages of the optimization, and provide numerical evidence for the 'No Free Lunch' theorem of *Wolpert and Macready* (1997).

5.3. Case Study III: hmodel conceptual watershed model

I now consider a real world problem involving a relatively simple conceptual watershed model (*Schoups and Vrugt*, 2010). The model transforms rainfall into runoff at the watershed outlet using explicit process descriptions of interception, throughfall, evaporation, runoff generation, percolation, and surface and subsurface routing (see Figure 8Schematic representation of the hmodel conceptual watershed model.figure.caption.10).

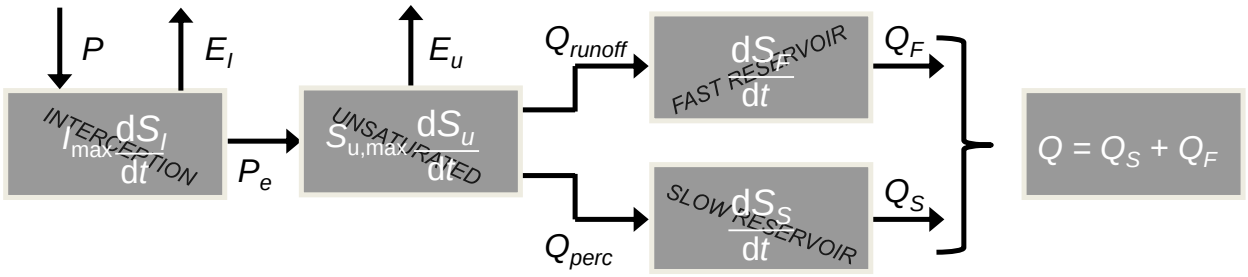


Figure 8: Schematic representation of the hmodel conceptual watershed model.

Runoff generation is assumed to be dominated by saturated overland flow and is simulated as a function of basin water storage without an explicit dependence on rainfall intensity. This assumption is typically valid for temperate climates but may be violated in semiarid watersheds. Snow accumulation and snowmelt are also not accounted for, yet this is not a problem if we focus on "warm" watersheds. Table 4Model parameters and their prior uncertainty ranges.table.caption.11 summarizes the seven different model parameters, and their prior uncertainty ranges.

Table 4: Model parameters and their prior uncertainty ranges.

Parameter	Symbol	Minimum	Maximum	Units
Maximum interception	$I_{\max}$	1	10	mm
Soil water storage capacity	$S_{\max}$	10	1000	mm
Maximum percolation rate	$Q_{\max}$	0.1	100	mm/d
Evaporation parameter	α_{E}	0.1	100	-
Runoff parameter	α_{F}	-10	10	-
Time constant, fast reservoir	K_{F}	0.1	10	days
Time constant, slow reservoir	K_{S}	0.1	150	days

I now like to estimate the parameters of the model. I use a seven-year record with daily data of discharge (mm/day), mean areal precipitation (mm/day), and mean areal potential evapotranspiration (mm/day) from a watershed in the USA (from MOPEX data set). Details of the basin, experimental data, and model can be found in various publications. I use a two-year spin-up period to reduce sensitivity of the model to state-value initialization. In other words, only the last five years are used for our analysis.

We would like the hmodel to fit both the driven and nondriven part of the hydrograph equally well. Unfortunately, practical experience with a suite of different hydrologic models suggests that this is practically impossible. Due to epistemic errors (model structural errors) and rainfall data errors, the model is unable to accurately fit both parts of the hydrograph. Single objective optimization (for instance minimization of sum of squared error) would give a compromise model calibration and hence simulation. We like to understand the trade-off in fitting both parts of the hydrograph as it might help us to understand which part of the model is erroneous and in need of improvement (epistemic errors due to lack of knowledge).

To better understand how these objective functions are computed please consider Figure 9Partitioning of the observed hydrograph into driven (blue) and nondriven (red) part. Classification is based on the rainfall data record. Days with precipitation define the driven part of the hydrograph, whereas dry days constitute

the nondriven part.figure.9 that schematically illustrates how the observed hydrograph (discharge data) is partitioned in a driven (solid circles) and nondriven (open circles) part. This partitioning follows ideas presented in (*Boyle et al.*, 2000).

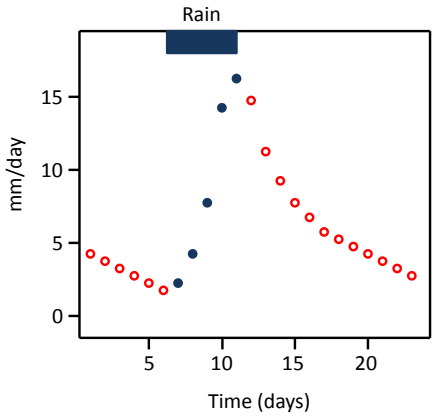


Figure 9: Partitioning of the observed hydrograph into driven (blue) and nondriven (red) part. Classification is based on the rainfall data record. Days with precipitation define the driven part of the hydrograph, whereas dry days constitute the nondriven part.

I derive the Pareto distribution using the input file called "example_6.m" in the folder \example_6 in the main directory of the AMALGAM toolbox.

MATLAB input script "example_6.m", with the setup of case study III: Modeling of the rainfall-runoff transformation. The file "03451500.dly" stores the daily discharge, precipitation and potential evapotranspiration data (in mm/day) of the French Broad River basin. The last line of this script calls the main program of the AMALGAM toolbox. This call to AMALGAM requests as fifth output argument the discharge simulations, `Y_sim`, of the final population.

```
%
    ------------------------------------------------------------------------------------
     %
%       AAA       MM        MM       AAA       LL         GGGGGGGGGG       AAA       MM
    MM   %
%      AA AA      MMM      MMM      AA AA      LL         GGG      GGG      AA AA      MMM
    MMM   %
%      AA AA      MMMM    MMMM      AA AA      LL         GG        GG      AA AA      MMMM
    MMMM   %
%     AA   AA     MM MM MM MM     AA   AA     LL         GGG      GGG     AA   AA     MM MM MM
    MM   %
%    AAAAAAAAA    MM  MMM  MM    AAAAAAAAA    LL         GGGGGGGGGG    AAAAAAAAA    MM  MMM
    MM   %
%    AA     AA    MM        MM    AA     AA    LL                 GG    AA     AA    MM
    MM   %
%   AA       AA   MM        MM   AA       AA   LLLLLLLL           GG   AA       AA   MM
    MM   %
%   AA       AA   MM        MM   AA       AA   LLLLLLLL  GGGGGGGGGG   AA       AA   MM
    MM   %
%
    ------------------------------------------------------------------------------------
     %

%% Define fields of AMALGAMPar
AMALGAMPar.N = 100;                              % Define population size
AMALGAMPar.T = 150;                              % How many generations?
AMALGAMPar.d = 7;                                % How many parameters?
AMALGAMPar.m = 2;                                % How many objective functions?

%% Define fields of Par_info
Par_info.initial = 'latin';                      % Latin hypercube sampling
Par_info.boundhandling = 'reflect';              % Explicit boundary handling
Par_info.min = [1   10  0.1 0.1 -10 0.1 0.1];    % If 'latin', min values
Par_info.max = [10 1000 100 100  10 10  150];    % If 'latin', max values

%% Define name of function
Func_name = 'AMALGAM_hmodel';

%% Define data and setup to be used for multi-criteria model calibration
mopex = load('03451500.dly');                          % Now load the mopex data
idx = find(mopex(:,1) > 1959 & mopex(:,1) < 1999);     % Select calibration set
n = size(mopex,1); tout = 0:n;                         % Number of observations and time
Y_obs = mopex(idx(1:n),6); Y_obs = Y_obs(731:n)';      % Measured discharge data (mm/day)
data.P      = mopex(idx(1:n),4)';                      % Daily rainfall (mm/d)
data.Ep     = mopex(idx(1:n),5)';                      % Daily evaporation (mm/d)
data.aS     = 1e-6;                                    % Percolation coefficient
hmodel_options.InitialStep = 1;                        % Initial time-step (d)
hmodel_options.MaxStep     = 1;                        % Maximum time-step (d)
hmodel_options.MinStep     = 1e-6;                     % Minimum time-step (d)
hmodel_options.RelTol      = 1e-3;                     % Relative tolerance
hmodel_options.AbsTol      = 1e-3*ones(5,1);           % Absolute tolerances (mm)
hmodel_options.Order       = 2;                        % 2nd order method (Heun)
y0 = 1e-6*ones(5,1);                                   % Initial conditions
id_d = find(data.P(731:n)>0); N_d = numel(id_d);       % Index and total driven part
id_nd = find(data.P(731:n)==0); N_nd = numel(id_nd); % Index and total nondriven part
```

The initial population is drawn using Latin hypercube sampling, and a reflection step is used if the parameter values are outside their respective prior ranges specified in the structure `Par_info`. Distributed multi-core computing is used to evaluate the N children and calculate their objective function values. The simulations of the hmodel of each population are stored after each generation and used in the post-processor script for plotting of the Pareto simulation uncertainty ranges (of which more later). The model is defined in the script `AMALGAM_hmodel` of Appendix C, and returns the values of the two objective functions for each of the parameter vectors of matrix $\mathbf{X}$ using the MOPEX data record as input. The actual model `crr_model` is written in the C-language and linked to MATLAB into a shared library called a MEX-file. The use of such MEX function significantly reduces the CPU-time required to approximate the Pareto distribution with AMALGAM.

Figure 10Normalized parameter plots for each of the hmodel parameters using a two-criteria $\{f_{\mathrm{D}}, f_{\mathrm{ND}}\}$ calibration. Each line across the graph denotes a single parameter set: gray is Pareto solution set; red and blue lines are single-criterion solutions of f_{D} and f_{ND}, respectively. The squared plot at the right-hand side are a two-dimensional projection of the objective space of the Pareto set of solutions. The red and blue cross signify the single criterion solutions derived separately with the SCE-UA optimization algorithm (*Duan et al.*, 1992). The Pareto solution set encompasses the SCE-UA solutions - this inspires thrust that AMALGAM has converged adequately as the nondominated solution set includes the extreme ends.figure.caption.12 presents the Pareto uncertainty for each parameter of the hmodel. The individual parameters are listed along the x-axis, while the y-axis defines their normalized ranges using the data tabulated in Table 1Main algorithmic variables of AMALGAM: Mathematical symbols, corresponding fields of `AMALGAMPar` and their default settings.table.caption.3. Each line across the graph represents one Pareto solution from the final population sampled with AMALGAM. The solid and dashed black lines going from left to right across the plots correspond to the single-objective solutions of f_{D} and f_{ND} obtained by separately fitting to each criterion using the SCE-UA global optimization algorithm (*Duan et al.*, 1992). The right-hand side in Figure 10Normalized parameter plots for each of the hmodel parameters using a two-criteria $\{f_{\mathrm{D}}, f_{\mathrm{ND}}\}$ calibration. Each line across the graph denotes a single parameter set: gray is Pareto solution set; red and blue lines are single-criterion solutions of f_{D} and f_{ND}, respectively. The squared plot at the right-hand side are a two-dimensional projection of the objective space of the Pareto set of solutions. The red and blue cross signify the single criterion solutions derived separately with the SCE-UA optimization algorithm (*Duan et al.*, 1992). The Pareto solution set encompasses the SCE-UA solutions - this inspires thrust that AMALGAM has converged adequately as the nondominated solution set includes the extreme ends.figure.caption.12 plots the objective function values of the final population. The red and blue cross denote the single criterion ends derived from the SCE-UA algorithm, and are used to benchmark the AMALGAM results.

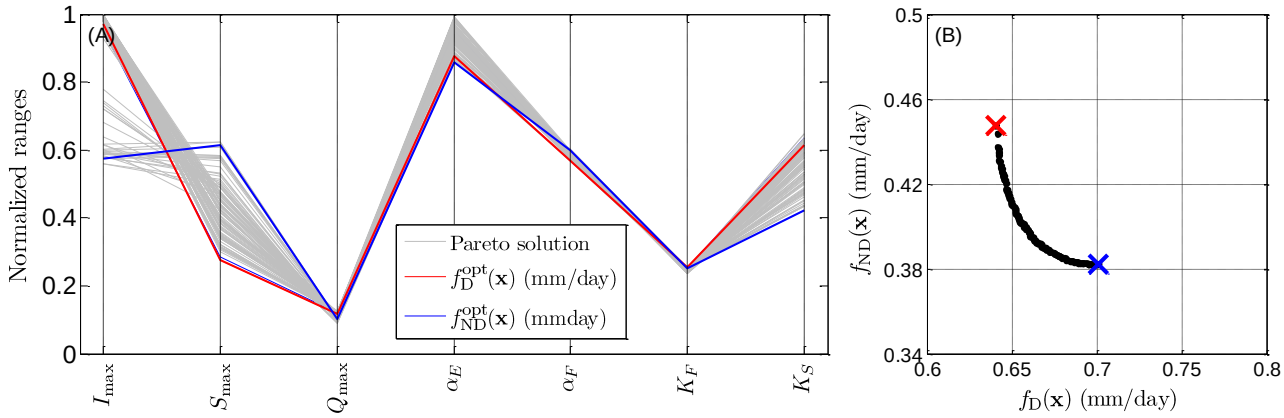


Figure 10: Normalized parameter plots for each of the hmodel parameters using a two-criteria $\{f_D, f_{ND}\}$ calibration. Each line across the graph denotes a single parameter set: gray is Pareto solution set; red and blue lines are single-criterion solutions of f_D and f_{ND}, respectively. The squared plot at the right-hand side are a two-dimensional projection of the objective space of the Pareto set of solutions. The red and blue cross signify the single criterion solutions derived separately with the SCE-UA optimization algorithm (*Duan et al.*, 1992). The Pareto solution set encompasses the SCE-UA solutions - this inspires thrust that AMALGAM has converged adequately as the nondominated solution set includes the extreme ends.

AMALGAM has generated a fairly uniform approximation of the Pareto front with solutions that occupy the single-criterion solutions at the extreme ends. For most of the hmodel parameters, the Pareto solution set tends to cluster closely in the parameter space for the two objectives. However, there is considerable uncertainty associated with the recession parameters K_F and K_S in the hmodel, which play a major role in determining the shape of the hydrograph during recession periods.

To better understand how the Pareto parameter distribution translates into hydrograph simulation uncertainty, please consider Figure 11Time series plot of hmodel Pareto simulation uncertainty (grey lines) and the observed data (magenta dots). The hmodel tracks the data quite nicely but systematic deviations are visible, for instance in the nondriven part of the hydrograph around days 980-1,000 and 1,020 - 1,050. This demonstrates that the model is in need of further refinement.figure.caption.13 that plots the Pareto uncertainty intervals (gray interval) of the hmodel for a selected portion of the calibration data set.

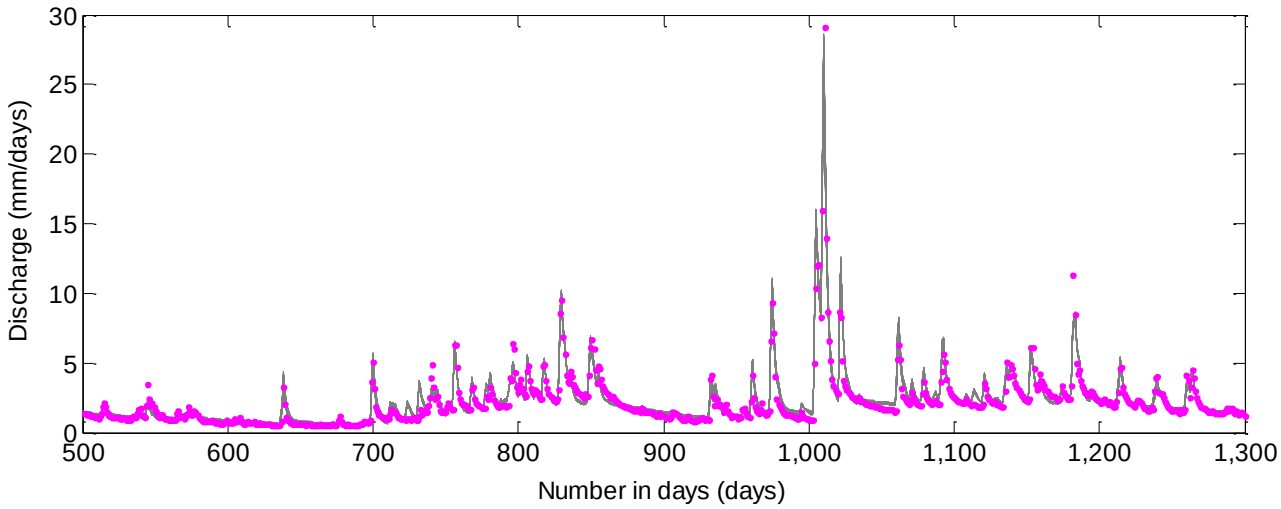


Figure 11: Time series plot of hmodel Pareto simulation uncertainty (grey lines) and the observed data (magenta dots). The hmodel tracks the data quite nicely but systematic deviations are visible, for instance in the nondriven part of the hydrograph around days 980-1,000 and 1,020 - 1,050. This demonstrates that the model is in need of further refinement.

The observed data are indicated with the magenta dots. The streamflow prediction uncertainty ranges match the medium- and high-flow events very well, but do not bracket the observations and display bias (systematic error) on the long recessions, suggesting that the model structure may be in need of further improvement. The relatively large uncertainty found during low flow and recession periods is consistent with the relatively large uncertainty in the $S_{\max}$ and K_S parameters. The issue of model structural errors is best addressed using diagnostic model evaluation, and interested readers are referred to (*Vrugt and Sadegh*, 2013).

5.4. Case Study IV: Bayesian Model Averaging

Ensemble Bayesian Model Averaging (BMA) proposed by *Raftery et al.* (2005) is a widely used method for statistical post-processing of forecasts from an ensemble of different models. The BMA predictive distribution of the quantity of interest is a weighted average of the probability density functions of the models' (bias-corrected) forecasts (see Figure 12Schematic illustration of Bayesian model averaging using a $K = 3$ member ensemble for the sea surface temperature in degrees Celsius. The BMA predictive pdf, g_j, is indicated with the solid black line and equivalent to a weighted average of the conditional forecast distributions, $f_k(\cdot)$, of the jth forecast of the members, $k = \{1, \ldots, K\}$, of the ensemble (displayed with solid red, blue and green lines). The forecast density of BMA can be used to compute prediction uncertainty ranges of the quantity of interest (sea surface temperature) at any desired confidence interval, $\alpha = 0.9$, 0.95 or 0.99. Also shown are the individual model forecasts ('×' symbol), the BMA deterministic point forecast ('o' symbol), and the verifying observation ('+' symbol). The deterministic point forecast of BMA can be compared to the ensemble mean and/or point predictors of other model averaging methods.figure.caption.14). The weights of the models are the estimated posterior model probabilities, representing each model's relative forecast skill in the training (calibration) period.

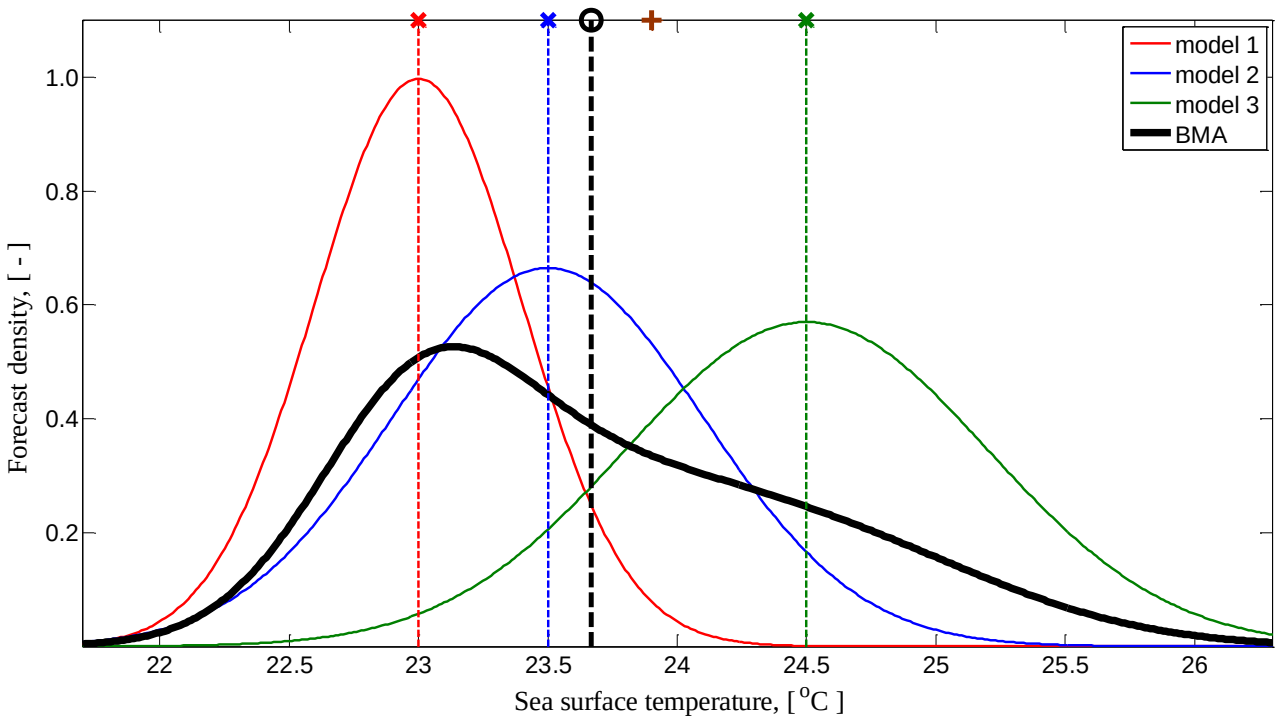


Figure 12: Schematic illustration of Bayesian model averaging using a $K = 3$ member ensemble for the sea surface temperature in degrees Celsius. The BMA predictive pdf, g_j, is indicated with the solid black line and equivalent to a weighted average of the conditional forecast distributions, $f_k(\cdot)$, of the jth forecast of the members, $k = \{1, \ldots, K\}$, of the ensemble (displayed with solid red, blue and green lines). The forecast density of BMA can be used to compute prediction uncertainty ranges of the quantity of interest (sea surface temperature) at any desired confidence interval, $\alpha = 0.9$, 0.95 or 0.99. Also shown are the individual model forecasts ('×' symbol), the BMA deterministic point forecast ('o' symbol), and the verifying observation ('+' symbol). The deterministic point forecast of BMA can be compared to the ensemble mean and/or point predictors of other model averaging methods.

To formalize the BMA method, let me denote by $\tilde{\mathbf{Y}} = \{\tilde{y}_1, \ldots, \tilde{y}_n\}$ a $n \times 1$ vector of measurements of a certain quantity of interest. These observations can be made at different times and locations in space, yet without loss of generality I conveniently ignore these two coordinates. Further assume that there is an ensemble of K different models that predict the observed data. The point forecasts of each model are stored in the $n \times K$ matrix $\mathbf{D}$. Thus, each column of $\mathbf{D}$ stores the n forecasts of a different model. In the remainder of this case study, the index j is used to mean "*for all* $i \in \{1, \ldots, n\}$".

The BMA point predictor, $g_j^\bullet$, is simply a weighted average of the individual models of the ensemble

$$g_j^\bullet = \sum_{k=1}^{K} w_k D_{jk} \tag{18}$$

which is a deterministic forecast in its own right, whose predictive performance can be compared with the individual models of the ensemble, or with the ensemble mean (median). The BMA weight, w_k, of each ensemble member, $k = \{1, \ldots, K\}$, can be viewed as each model's relative contribution to predictive skill over the training (calibration) period. The BMA weights can thus be used to assess the usefulness of ensemble members, and this can be used as a basis for selecting ensemble members given the CPU-cost of

running large ensembles (*Raftery et al.*, 2005).

Lets now assume that the forecasts of each model are subject to uncertainty. We can describe this uncertainty with an (unknown) forecast distribution, $f_k(\cdot)$, whose parameters are subject to inference from a training data set. For now, I conveniently assume that the forecast distribution is centered at the forecasts of each individual model of the ensemble. We can then compute the forecast density of the BMA model, g_j, as follows

$$g_j = \sum_{k=1}^{K} w_k f_k(\tilde{y}_j) \tag{19}$$

A common choice for the conditional pdf, $f_k(\cdot)$, is a normal distribution

$$f_k(\tilde{y}_j|D_{jk},\sigma_k^2) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\!\big(-\frac{1}{2}\sigma_k^{-2}(\tilde{y}_j - D_{jk})^2\big), \tag{20}$$

with mean D_{jk} and (unknown) standard deviation σ_k. The BMA forecast variance, var$(\cdot)$, of Equation (19Case Study IV: Bayesian Model Averagingequation.5.19) can be computed directly using (*Raftery et al.*, 2005)

$$\mathrm{var}\big(g_j^{\bullet}|D_{j1},\ldots,D_{jK}\big) = \sum_{k=1}^{K} w_k \big(D_{jk} - \sum_{l=1}^{K} w_l D_{jl}\big)^2 + \sum_{k=1}^{K} w_k \sigma_k^2 \tag{21}$$

This variance consists of two terms, the first representing the ensemble spread, and the second representing the within-ensemble forecast variance.

The MATLAB toolbox of AMALGAM presented herein allows the user to implement two different distributions for the conditional pdf, $f_k(\cdot)$, of the ensemble members. This includes the normal and gamma distribution, and allows a proper characterization of variables with/without a skew (see Figure 13Histograms of daily measurement records of five different variables, including (A) outside temperature [K], (B) atmospheric pressure [mbar], (C) wind speed [m/sec], (D) precipitation [mm/day], and (E) river discharge [mm/day]. Whereas the first two variables (temperature and pressure) follow a normal distribution, the last three variables are truncated at zero and much better described with a gamma (skewed) distribution.figure.caption.15). The gamma distribution is given by

$$f_k(\tilde{y}_j|a,b) \sim \frac{1}{b^a\Gamma(a)} \tilde{y}_j^{(a-1)} \exp(-\tilde{y}_j/b), \tag{22}$$

where $a > 0$ and $b > 0$ are a shape and scale parameter, respectively and $f_k(\tilde{y}_j) = 0$ if $\tilde{y}_j \leq 0$. The mean and variance of the gamma distribution are determined by the values of a and b, as follows, $\mu = ab$ and $\sigma^2 = ab^2$. Hence, the values of a and b cannot be chosen freely as their product should equate to D_{jk} and the mean of the gamma distribution centers around the actual forecast of the kth member of the ensemble. I therefore calculate the values of a and b as follows

$$a_{jk} = \frac{|D_{jk}|^2}{\sigma_k^2} \quad ; \quad b_{jk} = \frac{\sigma_k^2}{|D_{jk}|}, \tag{23}$$

and estimate the weight, w_k, and standard deviation, σ_k, of the gamma distribution, where $k = \{1,\ldots,K\}$, using AMALGAM. This involves the inference of $d = 2K$ parameters.

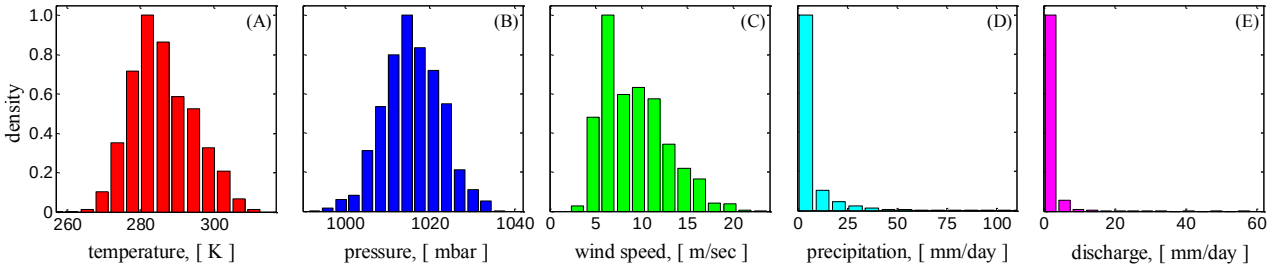


Figure 13: Histograms of daily measurement records of five different variables, including (A) outside temperature [K], (B) atmospheric pressure [mbar], (C) wind speed [m/sec], (D) precipitation [mm/day], and (E) river discharge [mm/day]. Whereas the first two variables (temperature and pressure) follow a normal distribution, the last three variables are truncated at zero and much better described with a gamma (skewed) distribution.

The MATLAB toolbox of AMALGAM therefore implements four different parameterizations of the standard deviation of the forecast distribution.

(1) common constant variance: all members of the ensemble have the same standard deviation, that is $\sigma_1 = \sigma_2 = \ldots = \sigma_K$. This simplifies somewhat the inference and involves $d = K+1$ fitting parameters.

(2) individual constant variance: all members of the ensemble have their own standard deviation, and thus $\boldsymbol{\sigma} = \{\sigma_1, \ldots, \sigma_K\}$. This assumption was made in our notation thus far and results in a BMA mixture distribution with $d = 2K$ unknowns.

(3) common non-constant variance: the standard deviation of the forecast distribution is dependent on the magnitude of the forecast. This approach is implemented using $\sigma_{jk} = cD_{jk}$, where the multiplier c applies to all models and forecasts of the ensemble. This approach involves $d = K + 1$ fitting parameters.

(4) individual non-constant variance: the standard deviation of the forecast distribution is member and forecast dependent. This approach is implemented using $\sigma_{jk} = c_k D_{jk}$, where the K multipliers are subject to inference. This involves $d = 2K$ unknowns.

The first two approaches assume a homoscedastic (constant) variance of the conditional distribution of each ensemble member. This approach might be appropriate for variables such as the temperature and pressure of the atmosphere that are known to have a constant measurement error (*Vrugt et al.*, 2006). The last two approaches assume a heteroscedastic (non-constant) variance of the conditional forecast distribution. The variance of this distribution increases with value of the forecast. This assumption is appropriate for variables such as rainfall (*Sloughter et al.*, 2007), discharge (*Vrugt and Robinson*, 2007b) and wind speed (*Sloughter et al.*, 2010) whose measurement errors are known to increase with magnitude of the observation. Note, one can augment the heteroscedastic variance with a constant value, for instance, $\sigma_{jk} = c_k D_{jk} + c_2$ so that the standard deviation does not become zero if $D_{jk} = 0$. This adds one additional parameter, c_2 to the BMA forecast density of Equation (19Case Study IV: Bayesian Model Averagingequation.5.19) and now involves $d = 2K + 1$ unknowns.

Successful application of BMA requires estimates of the weights and variances of the individual competing models in the ensemble. In their seminal paper, *Raftery et al.* (2005) recommends using the Expectation Maximization (EM) algorithm (*Dempster et al.*, 1997). This method is relatively easy to implement, computationally efficient, but does not provide uncertainty estimates of the weights and standard deviations of

the forecast distributions. *Vrugt et al.* (2008b) therefore introduced Bayesian inference of the BMA weights and standard deviations using Markov chain Monte Carlo (MCMC) simulation with the DREAM algorithm (*Vrugt et al.*, 2008a, 2009b, 2011; *Laloy and Vrugt*, 2012a; *Vrugt*, 2016).

Here, I follow *Vrugt et al.* (2006) and use a multi-criteria approach to BMA model training. This multi-criteria framework is fundamentally different than the maximum likelihood BMA training approach used in the literature as it uses simultaneously three different but complementary metrics of forecast skill. I use AMALGAM to solve for the Pareto set of BMA model parameters that have consistently good performance across multiple performance metrics. I use herein the root mean square error (RMSE), the ignorance score (IS), and the continuous rank probability score (CRPS). The RMSE has units mm/day and the other two metrics are unitless. The mathematical definition of each objective function is given below

$$\arg\min_{\mathbf{x}\in\mathcal{X}} \mathbf{F}(\mathbf{x}) = \begin{cases} \text{RMSE}: & f_1(\mathbf{x}) = \sqrt{\dfrac{1}{n}\displaystyle\sum_{t=1}^{n}(g_t^{\bullet} - \tilde{y}_t)^2} \\ \text{IS}: & f_2(\mathbf{x}) = -\dfrac{1}{n}\displaystyle\sum_{t=1}^{n}\log\left\{\sum_{k=1}^{K} w_k g_k(\tilde{y}_t)\right\} \\ \text{CRPS}: & f_3(\mathbf{x}) = \dfrac{1}{n}\displaystyle\sum_{t=1}^{n}\left\{\int_{-\infty}^{\infty}\left(G_t(y) - 1\{y \geq \tilde{y}_t\}\right)^2 \mathrm{d}y\right\} \end{cases} \tag{24}$$

where $\{f_1, f_2, f_3\}$ denote the value of the RMSE, IS and CRPS metrics, respectively, $\mathbf{w} = \{w_1, \ldots, w_K\}$ signifies the weights of the K models, $g_j^{\bullet}$ is the mean forecast of the BMA model (see Equation (18Case Study IV: Bayesian Model Averagingequation.5.18)), and $g(y)$ and $G(y)$ represent the probability density function (pdf) and cumulative distribution function (cdf) of the BMA model, respectively. The Heaviside step function, $1\{y \geq \tilde{y}_t\}$ attains the value of one if the statement between accolades is true, and zero otherwise. The three metrics measure simultaneously the qualify of fit, and sharpness and spread of the BMA model. Smaller values are preferred for each of the performance metrics.

The CRPS can be decomposed into a reliability and a resolution part (*Hersbach*, 2000; *Gneiting and Raftery*, 2007)

$$\text{CRPS}: \quad f_3(\mathbf{x}) = \mathbb{E}|\{\mathbf{G}_1\} - \tilde{\mathbf{Y}}| - \frac{1}{2}\mathbb{E}|\{\mathbf{G}_1\} - \{\mathbf{G}_2\}| \tag{25}$$

where $\{\mathbf{G}_1\}$ and $\{\mathbf{G}_2\}$ are $n \times 1$-vectors with samples from the evolving mixture distribution of the BMA model. The rows of $\{\mathbf{G}_1\}$ and $\{\mathbf{G}_2\}$ correspond exactly with $\tilde{\mathbf{Y}}$. Thus, $\{\mathbf{G}_1\}$ and $\{\mathbf{G}_2\}$ represent two different forecast time series drawn randomly from the BMA model for the length of the training data period. Without a closed-form expression of the CRPS, Equation (25Case Study IV: Bayesian Model Averagingequation.5.25) is much easier to evaluate in practice than its counterpart of Equation (24Case Study IV: Bayesian Model Averagingequation.5.24).

The MATLAB function `BMA_rnd` returns to the user N different trajectories of the BMA mixture distribution for the calibration data period.

MATLAB code of `BMA_rnd`: This function returns to the user the cell array, `g` with `P` different trajectories from the evolving BMA mixture distribution derived from the ensemble forecasts, `D` and weights, `w` of the training data set. `P = 2` suffices to calculate the CRPS metric. These two BMA model trajectories are stored in `G{1}` and `G{2}`, respectively. Built-in functions are highlighted with a low dash. The function `randsample(1:K,n,'true',c)` returns the n-vector `id` with values $\{1, \ldots, K\}$ sampled at random, with replacement, using selection probabilities, w, of the K integers. The built-in function `random(PDF,A,B)` draws from the distribution `PDF` using the parameters `A` and `B` respectively. `sum(a)` computes the sum of the vector `a`, and `find` returns the index of all entries of `id` equal to k.

```
function [ G ] = BMA_rnd ( PDF , w , A , B , P )
% This function creates sampled trajectories of the BMA forecast distribution
% Reference: J.A. Vrugt, M.P. Clark, C.G.H. Diks, Q. Duan, and B. A. Robinson (2006),
%   Multi-objective calibration of forecast ensembles using Bayesian model averaging,
%   Geophysical Research Letters, 33, L19817, 2006

if nargin < 5,
    error('AMALGAM:BMA_rnd:TooFewInputs','Requires at least five input arguments.');
end

[n,K] = size(A);    % How many forecasts and ensemble members

for r = 1:P,        % Draw N trajectories from BMA forecast distribution
    id = randsample(1:K,n,'true',w);            % Select n times from 1:K using weights
        w
    for k = 1:K,
        T = find(id == k);                      % kth component used for indices of T
        G{r}(T,1) = random(PDF,A(T,k),B(T,k));  % Draw sample from kth mixture component
    end
end
```

The MATLAB function `AMALGAM_BMA` appears in Appendix C on Page 61lstlisting.-12 and calculates the root mean square error (RMSE, mm/day), ignorance score (IS, -) and continuous rank probability score (CRPS, -) of the BMA model and its mean forecast (second output argument) for a given vector, $\mathbf{x}$ (input argument) of weights and standard deviations (or proxies thereof) of the ensemble members. The second input argument, `func_in` of this function is defined by the user in the input file "example_8.m" (see below) and passes to `AMALGAM_BMA` the $n \times K$ matrix `D` with ensemble forecasts, the $n \times 1$ vector `Y` with verifying observations, and the fields `PDF` and `VAR` that define with a string the members' forecast distribution and variance option, respectively (see Page 39Case Study IV: Bayesian Model Averagingfigure.caption.15).

Theory, concepts and applications of multi-criteria BMA model calibration are presented in *Vrugt et al.* (2006) and interested readers are referred to this publication for further details. Here, I apply the multi-criteria BMA model training approach to discharge forecasting. I use a 3000-day record of daily streamflows of an eight-member ensemble of calibrated watershed models for the Leaf River, near Collins, Mississippi. This discharge ensemble is stored in the $n \times K$ matrix $\mathbf{D}$ and was created by *Vrugt and Robinson* (2007b) to evaluate the sharpness and coverage of the BMA forecast distribution. Figure 14Streamflow predictions of the eight individual models of the ensemble for a representative portion of the calibration period. The red dots represent the verifying observations.figure.caption.16 provides a snapshot of the model ensemble for a portion of the training data set. The verifying observations, $\tilde{\mathbf{Y}} = \{\tilde{y}_1, \ldots, \tilde{y}_n\}$ are also shown.

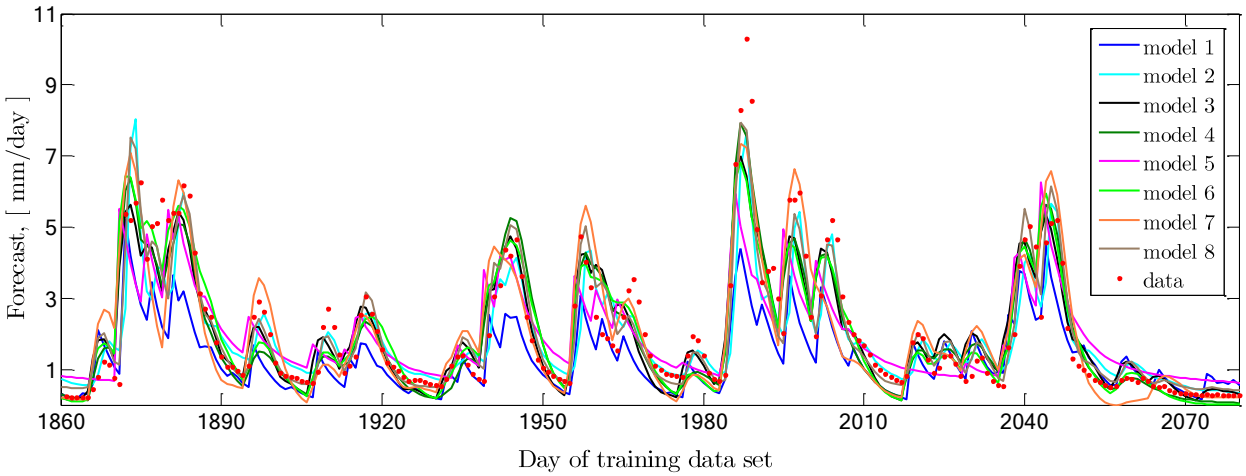


Figure 14: Streamflow predictions of the eight individual models of the ensemble for a representative portion of the calibration period. The red dots represent the verifying observations.

The spread of the ensemble is sufficient and generally brackets the observations (red dots). The calibrated models appear to provide somewhat different forecasts. This is a necessary requirement for an ensemble prediction system, otherwise model averaging cannot improve the forecast skill and distribution. Note, the ensemble members have been bias-corrected using simple linear regression of the forecasts of each model on the verifying data.

The file "example_8.m" in the folder \example_8 in the main directory of the AMALGAM toolbox defines the setup of this case study.

MATLAB input script "example_8.m", with the setup of case study IV: Multi-criteria Bayesian model averaging. The file "discharge.txt" stores the ensemble forecasts of the eight watershed models and the verifying discharge observations. The script `setup_BMA` uses linear bias correction of the ensemble members' forecasts. The last line calls the main program of the AMALGAM toolbox. This call to AMALGAM requests as fifth output argument the simulations, `Y_sim`, of the final population.

```
%
    ------------------------------------------------------------------------------------
     %
%       AAA       MM        MM       AAA       LL         GGGGGGGGGG       AAA       MM
    MM   %
%      AA AA      MMM      MMM      AA AA      LL         GGG      GGG      AA AA      MMM
    MMM   %
%      AA AA      MMMM    MMMM      AA AA      LL         GG        GG      AA AA      MMMM
    MMMM   %
%     AA   AA     MM MM MM MM     AA   AA     LL         GGG      GGG     AA   AA     MM MM MM
    MM   %
%    AAAAAAAAA    MM  MMM  MM    AAAAAAAAA    LL         GGGGGGGGGG    AAAAAAAAA    MM  MMM
    MM   %
%    AA     AA    MM        MM    AA     AA    LL                 GG    AA     AA    MM
    MM   %
%   AA       AA   MM        MM   AA       AA   LLLLLLLL           GG   AA       AA   MM
    MM   %
%   AA       AA   MM        MM   AA       AA   LLLLLLLL  GGGGGGGGGG   AA       AA   MM
    MM   %
%
    ------------------------------------------------------------------------------------
     %

%% Define fields of AMALGAMPar
AMALGAMPar.N = 100;                     % Define population size
AMALGAMPar.T = 100;                     % How many generations?
AMALGAMPar.m = 3;                       % How many objective functions?

%% Define fields of Par_info
Par_info.initial = 'latin';             % Latin hypercube sampling
Par_info.boundhandling = 'reflect';     % Explicit boundary handling

%% Define name of function (.m file) for posterior exploration
Func_name = 'AMALGAM_BMA';

%% Load data from Vrugt and Robinson, WRR, 43, W01411, doi:10.1029/2005WR004838, 2007
data = load('discharge.txt');           % Daily discharge forecasts (mm/day) and
    verifying data
T_idx = 1:1:3000;                       % Start/end day of training period

%% Define the second input argument of BMA_multcalc
func_in.D = data(T_idx,1:8);            % Store ensemble forecasts
func_in.Y = data(T_idx,9);              % Store verifying observations
func_in.PDF = 'gamma';                  % Store name of forecast distribution ('normal
    '/'gamma')
func_in.VAR = '3';                      % Store variance option ('1'/'2'/'3'/'4')
```



```
%% Setup the BMA model (apply linear bias correction)
[AMALGAMPar,Par_info] = setup_BMA(AMALGAMPar,Par_info,func_in);

%% Define structure options
options.ranking = 'C';                  % Use Pareto ranking in C
options.print = 'yes';                  % Print output to screen
options.save = 'yes';           % Save memory of DREAM for restart run
options.modout = 'yes';                 % Return simulation of BMA model
```

The predictive distribution of each constituent member of the ensemble is assumed to follow a gamma distribution with unknown heteroscedastic standard deviation, $\sigma_k = cD_k$, where c applies to all models of the ensemble. The workspace is saved after each iteration, and the FNS algorithm is ran in C through a mex-compiled function to speed up the calculation of the rank and crowding distance. The MATLAB function`AMALGAM_BMA` (see Page 61lstlisting.-12) uses as input the $d \times 1$ vector of BMA parameters (weights and standard deviations of each member's forecast distribution) and returns a $1 \times m$ vector, `F` = { RMSE , IS , CRPS }, of objective function values and the $n \times 1$ vector `G_dot` with mean forecast of the BMA model.

Table 5Results of AMALGAM$_{(\text{BMA})}$ by application to eight different watershed models using daily discharge data from the Leaf River in Mississippi, USA. I list the individual RMSE (mm/day) of each of model of the ensemble, followed by the mean Pareto values of the weights and their standard deviation using a Gamma (second column) and normal (third column) forecast distribution with heteroscedastic standard deviation, $\sigma_k = c_k D_k$. The bottom rows of the Table list the Pareto mean values of the RMSE (mm/day), IS (-) and CRPS (-) performance statistics.table.caption.17 summarizes the results of AMALGAM and presents (in column "Gamma") the mean Pareto (MP) values of the BMA weights for the different models of the ensemble. Values listed in parentheses denote the standard deviation derived from the nondominated solutions in the final population of AMALGAM. I also summarize the MP values of the weights for a normal forecast distribution with constant (second column) and non-constant (third column) standard deviation, and report the lowest values of the RMSE (mm/day), IS (-), and CRPS (-) performance statistics in the Pareto solution set for the BMA model and $n = 3000$-day calibration data period. For completeness, the RMSE (mm/day) values of each model is listed as well.

Table 5: Results of AMALGAM$_{(\text{BMA})}$ by application to eight different watershed models using daily discharge data from the Leaf River in Mississippi, USA. I list the individual RMSE (mm/day) of each of model of the ensemble, followed by the mean Pareto values of the weights and their standard deviation using a Gamma (second column) and normal (third column) forecast distribution with heteroscedastic standard deviation, $\sigma_k = c_k D_k$. The bottom rows of the Table list the Pareto mean values of the RMSE (mm/day), IS (-) and CRPS (-) performance statistics.

Model	RMSE	Gamma		Normal	
		Mean	Std	Mean	Std
ABC	1.3957	0.0001	0.0003	0.0007	0.0007
GR4J	0.8453	0.1613	0.0348	0.1750	0.0454
HYMOD	0.8370	0.0010	0.0025	0.0078	0.0065
TOPMO	0.7777	0.2059	0.0752	0.2374	0.0589
AWBM	1.1569	0.0002	0.0005	0.0001	0.0001
NAM	0.8895	0.0001	0.0004	0.0001	0.0000
HBV	0.8546	0.0390	0.0236	0.0004	0.0004
SAC-SMA	0.7233	0.5923	0.0203	0.5784	0.0078
BMA: RMSE		0.6993	0.0027	0.6990	0.0026
BMA: IS		-0.1223	0.0199	-0.0679	0.0110
BMA: CRPS		0.2099	0.0202	0.2108	0.0179

The Pareto values of the weights hardly depend on the assumed forecast distribution of the ensemble members. The GR4J, TOPMO and SAC-SMA models consistently receive the highest weights and are thus most important for the BMA model for this discharge data set. Note also that the HBV model receives a very low BMA weight, despite it having quite a low RMSE. Correlation between the individual forecasts of the watershed models affects strongly the Pareto values of the BMA weights.

To better understand the multi-criteria calibration results, please consider Figure 15Normalized parameter plots for the BMA weights of the Gamma conditional distribution using a three-criterion {RMSE, IS,

CRPS} calibration with AMALGAM for the 3000-day training data set of daily discharge values simulated with eight watershed models. Each line across the graph denotes a single Pareto solution. The plots at the right-hand side are three-dimensional projections of the objective space of the Pareto set of solutions. The solutions of the single criterion ends are indicated with a red (RMSE), blue (IS) and green (CRPS) line (plot A) and cross (plot B), respectively.figure.caption.18 that presents (left-hand side) normalized parameter plots of the Pareto BMA weights and values of $\{c_1, \ldots, c_K\}$ (multipliers standard deviation) for the individual models of the ensemble, and a three-dimensional plot of the objective function values of the final population sampled with AMALGAM (right-hand side). These results pertain to the gamma distribution as forecast distribution for each model of the ensemble.

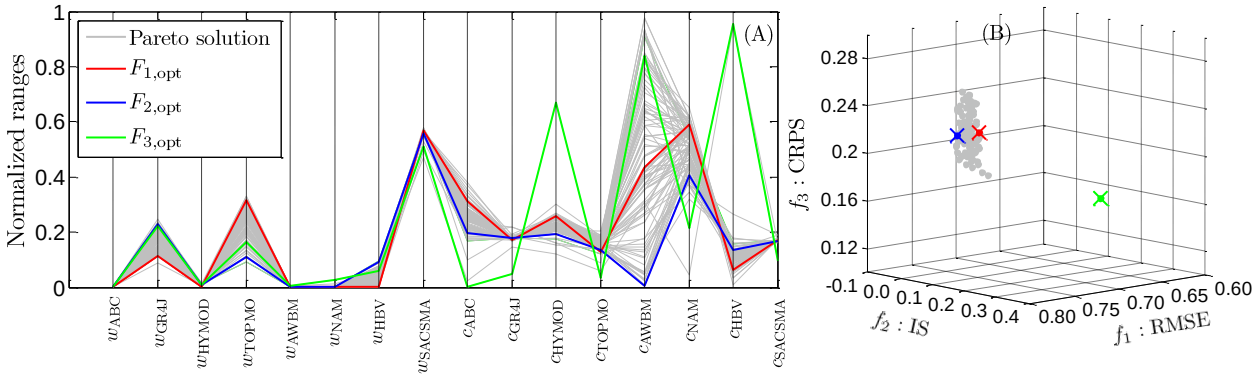


Figure 15: Normalized parameter plots for the BMA weights of the Gamma conditional distribution using a three-criterion {RMSE, IS, CRPS} calibration with AMALGAM for the 3000-day training data set of daily discharge values simulated with eight watershed models. Each line across the graph denotes a single Pareto solution. The plots at the right-hand side are three-dimensional projections of the objective space of the Pareto set of solutions. The solutions of the single criterion ends are indicated with a red (RMSE), blue (IS) and green (CRPS) line (plot A) and cross (plot B), respectively.

Finally, I now do a similar analysis but using 48-h forecasts of surface temperature in the North American Pacific Northwest in January-June 2000 from the University of Washington (UW) mesoscale short-range ensemble system (*Grimit and Mass*, 2002). This is a five member multianalysis ensemble (hereafter referred to as the UW ensemble) consisting of different runs (hereafter referred to as AVN, GEM, ETA, NGM and NOGAPS, respectively) of the fifth-generation Pennsylvania State University - National Center for Atmospheric Research Mesoscale Model (MM5), in which initial conditions are taken from different operational centers. I follow *Raftery et al.* (2005) and use a 25-day training period between April 16 and 9 June 2000 for BMA training. For some days the data were missing, so that the number of calendar days spanned by the training data set is larger than the number of days of training used. The individuals members of the ensemble were bias corrected using simple linear regression of $\mathbf{D}_k$ on $\tilde{\mathbf{Y}}$ for the training data set. I assume a Gaussian conditional distribution, $f_k(\cdot)$, with a homoscedastic (constant) standard deviation for each member of the ensemble (`func_in`.VAR = '2').

Figure 16Normalized parameter plots for the BMA weights and variance of the Gaussian conditional distribution using a three-criterion {RMSE, IS, CRPS} calibration with AMALGAM for the UW 48-hour ensemble data sets of surface temperature. Each line across the graph denotes a single parameter set: shaded is Pareto solution set. The plots at the right-hand side are three-dimensional projections of the objective

space of the Pareto set of solutions.figure.caption.19 presents normalized parameter plots of the Pareto solution set derived from AMALGAM for the surface temperature data set and three forecast skill objectives of Equation (24Case Study IV: Bayesian Model Averagingequation.5.24). Each line going from left to right across the plot corresponds to a different parameter combination. The gray lines represent members of the Pareto set (appears as a band). The weights of each model, and their corresponding standard deviations are listed along the x-axis, and the y-axis corresponds to the Pareto parameter values, normalized by their prior uncertainty ranges. The plot at the right-hand side in Figures 16Normalized parameter plots for the BMA weights and variance of the Gaussian conditional distribution using a three-criterion {RMSE, IS, CRPS} calibration with AMALGAM for the UW 48-hour ensemble data sets of surface temperature. Each line across the graph denotes a single parameter set: shaded is Pareto solution set. The plots at the right-hand side are three-dimensional projections of the objective space of the Pareto set of solutions.figure.caption.19 depicts a three-dimensional projection of the RMSE (Kelvin), IS (-) and CRPS (-) forecast skill metrics for the Pareto solution set. The Pareto rank 1 solutions in these plots are indicated with the gray dots. The single criterion solutions are separately indicated and color coded using red (RMSE), blue (IS) and green (CRPS).

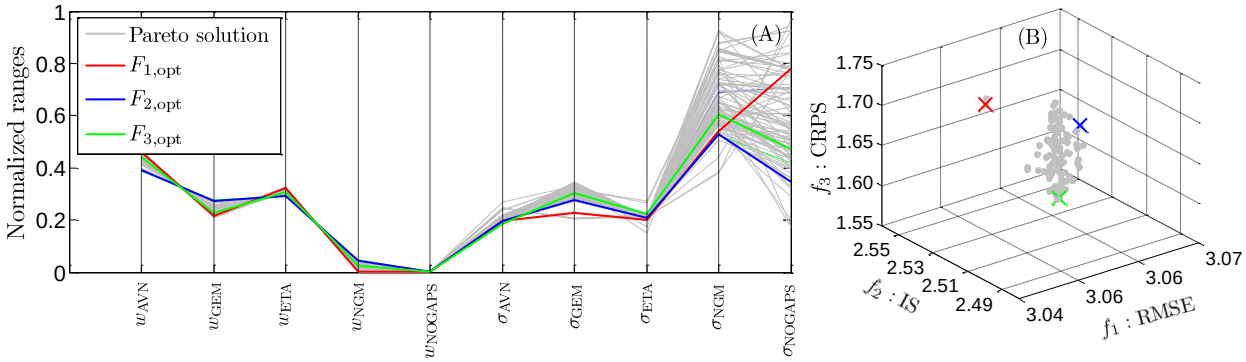


Figure 16: Normalized parameter plots for the BMA weights and variance of the Gaussian conditional distribution using a three-criterion {RMSE, IS, CRPS} calibration with AMALGAM for the UW 48-hour ensemble data sets of surface temperature. Each line across the graph denotes a single parameter set: shaded is Pareto solution set. The plots at the right-hand side are three-dimensional projections of the objective space of the Pareto set of solutions.

The Pareto solution space spans only a very small region interior to the prior parameter space. This is particularly true for the weights of the models, as the standard deviation of the forecast distribution of the NGM and NOGAPS models appears relatively large. This latter finding is explained by the low weight of these models in the BMA mixture density. Altogether, these findings illustrate that the BMA model is well defined by calibration against the three individual performance criteria. This is further confirmed by a check of the three-dimensional projection of the {RMSE, IS, CRPS} Pareto objective function space at the right hand side of Figure 16Normalized parameter plots for the BMA weights and variance of the Gaussian conditional distribution using a three-criterion {RMSE, IS, CRPS} calibration with AMALGAM for the UW 48-hour ensemble data sets of surface temperature. Each line across the graph denotes a single parameter set: shaded is Pareto solution set. The plots at the right-hand side are three-dimensional projections of the objective space of the Pareto set of solutions.figure.caption.19. This plot shows a negligible trade-off in the

fitting of the different objectives. This demonstrates that it is possible to identify a single 'optimal' BMA model that has good and consistent performance across each of the three forecast measures. The user is, of course, free to select any other Pareto BMA model - based on subjective preferences and the intended goal of application.

In general I conclude that the multi-criteria optimization helps to guide the search for an appropriate BMA model, and provides useful information about the trade-offs between the various performance metrics. The current practice of optimizing the BMA model using maximum likelihood theory seems to result in a calibrated forecast ensemble that receives a good performance in terms of the quadratic forecast error, and sharpness of the prediction intervals. However, such consistent performance of the ML method cannot be guaranteed for all forecasting problems. I refer interested readers to *Vrugt et al.* (2006), *Vrugt and Robinson* (2007b) and *Rings et al.* (2012) for a more comprehensive analysis of the BMA approach, including a comparison with filtering (data assimilation) methods.

6. Final remarks

The four case studies presented herein illustrate only some of the main capabilities of the AMALGAM software package. Yet, not all the options, discussed initially in the presentation of the toolbox have been demonstrated explicitly. The input files, "example_X.m" in the case study folders \example_X, where X = $\{1, \ldots, 10\}$, in the main directory of the AMALGAM toolbox present an exhaustive overview of the different options and capabilities of the AMALGAM software suite. Users can draw from these test problems and use them as templates for their own modeling problems.

Recent work includes implementation of a decomposition-based search strategy in AMALGAM. This methodology, called AMALGAM$_{(\mathrm{D})}$ uses the Tchebycheff approach (*Zhang and Li*, 2007; *Li and Zhang*, 2009) to calculate the fitness, $q(\cdot)$ of each individual, $\mathbf{X}_i$ of the population,

$$\begin{aligned} q(\mathbf{X}_i|\boldsymbol{w}_i, \mathbf{z}) &= \max_{1\leq j\leq m} \{w_{i,j}|f_{i,j}(\mathbf{X}_i) - z_j|\} \\ \text{subject to } & \mathbf{x}_i \in \mathcal{X} \in \mathbb{R}^d \end{aligned} \tag{26}$$

where d signifies the dimensionality of the variable (parameter) space, $\boldsymbol{w}_i = \{w_{i,1}, \ldots, w_{i,m}\}$ denotes the weight vector, $\mathbf{z} = \{z_1, \ldots, z_m\}$ is an anchor point. The Pareto front can be approximated by minimizing (26Final remarksequation.6.26) simultaneously for each of the N individuals using evenly distributed weight vectors, $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N\}; w_{i,j} \in [0, 1]; \sum_{j=1}^{m} w_{i,j} = 1$. Yet, this approach is not particularly efficient if each individual scalar subproblem is solved independently, without information exchange between the successive solutions evolved in parallel. I therefore follow *Zhang and Li* (2007) and *Li and Zhang* (2009) and exploit similarities from neighboring individuals with closest weight factors (in Euclidean space) to guide the selection and evolution process and speed-up convergence towards the Pareto front.

7. Summary

In this manuscript, I have introduced a MATLAB package of the AMALGAM multi-criteria optimization algorithm. This toolbox provides scientists and engineers with an arsenal of options and utilities to solve optimization problems involving (amongst others) two or more objectives. The AMALGAM toolbox supports

parallel computing and includes tools for convergence analysis of the sampled solutions and post-processing of the results. Four different case studies were used to illustrate the main capabilities and functionalities of the MATLAB toolbox. These example studies are easy to run and adapt and serve as templates for other inference problems.

A graphical user interface (GUI) of AMALGAM is currently under development and will become available in due course.

8. Acknowledgements

The MATLAB toolbox of AMALGAM is available upon request from the author, `jasper@uci.du`. I would like to thank Guilherme Gomes for careful proofreading of the final document.

Appendix A. Download and installation

The AMALGAM code can be downloaded from my website at the following link `http://faculty.sites.uci.edu/software`. Please, scroll down to find the appropriate toolbox and link. Please save the file "MATLAB-pCode-AMALGAM-V1.0" to your hard disk, for instance, in the directory "D:\Downloads\Toolboxes\MATLAB\AMALGAM". Now open Windows explorer in this directory (see Figure A.1figure.caption.20).

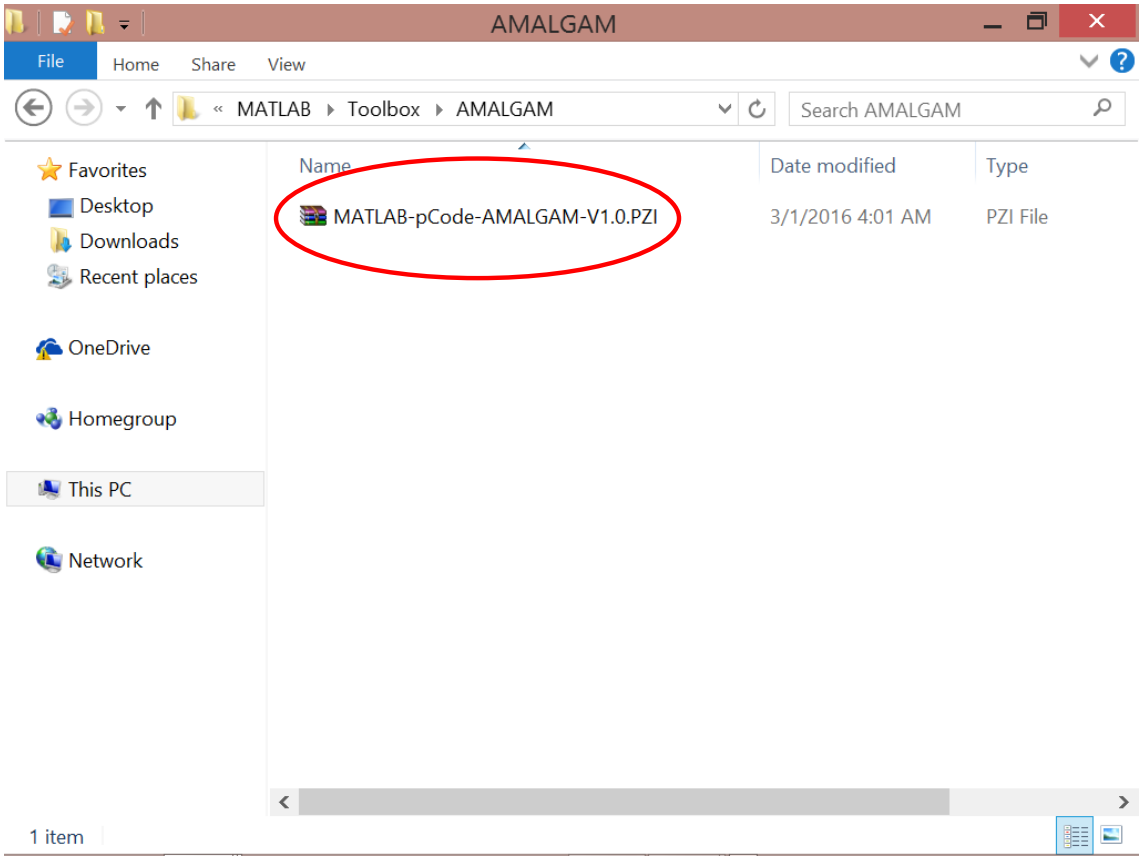


Figure A.1

You will notice that the file does not have an extension - it is just called **MATLAB-pCode-AMALGAM-V1.0**. That is because Windows typically hides extension names.

If you can already see file extensions on your computer, then please skip the next step. If you cannot see the file extension, please click the **View** tab. Then check the box titled "File name extensions" (see Figure A.2figure.caption.21).

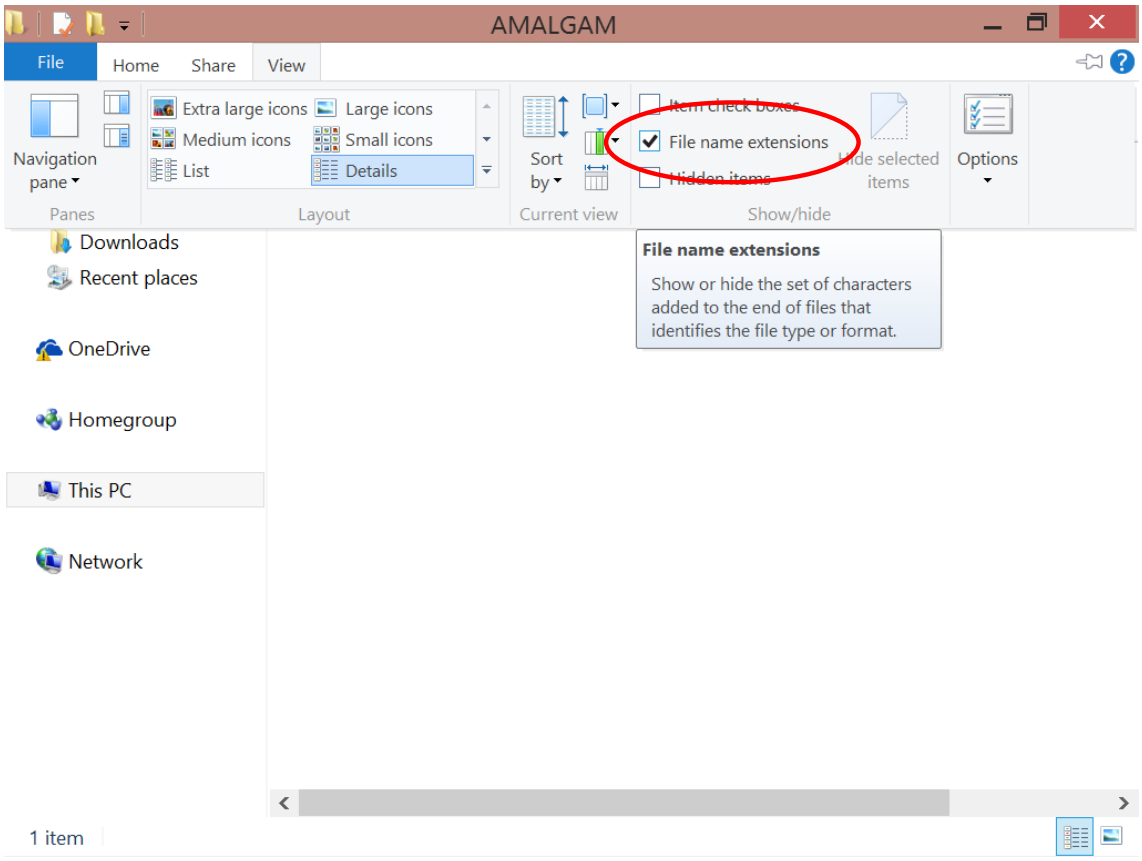


Figure A.2

Now you should be able to see the file extension. Right-click the file name and select **Rename** (see Figure A.3figure.caption.22).

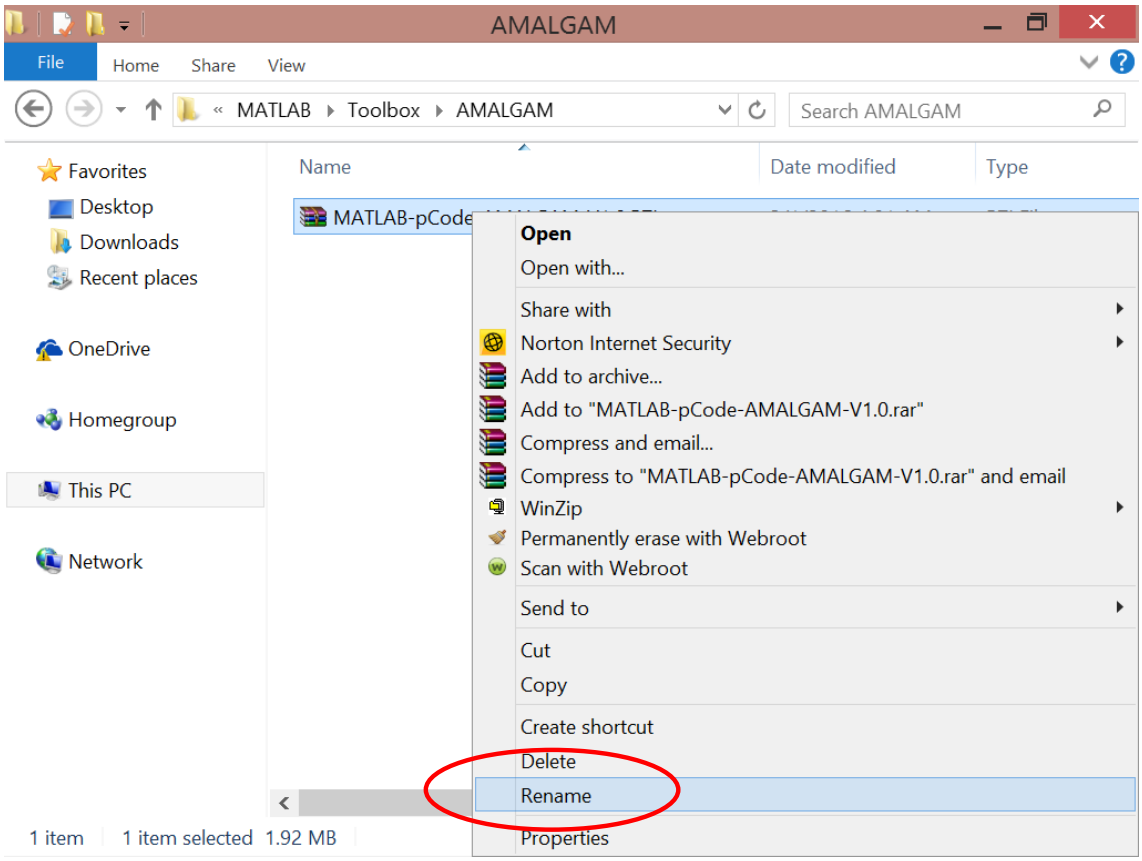


Figure A.3

Now change the extension of "MATLAB-pCode-AMALGAM-V1.0" from ".pdf" to ".rar" (see Figure A.4figure.caption.23).

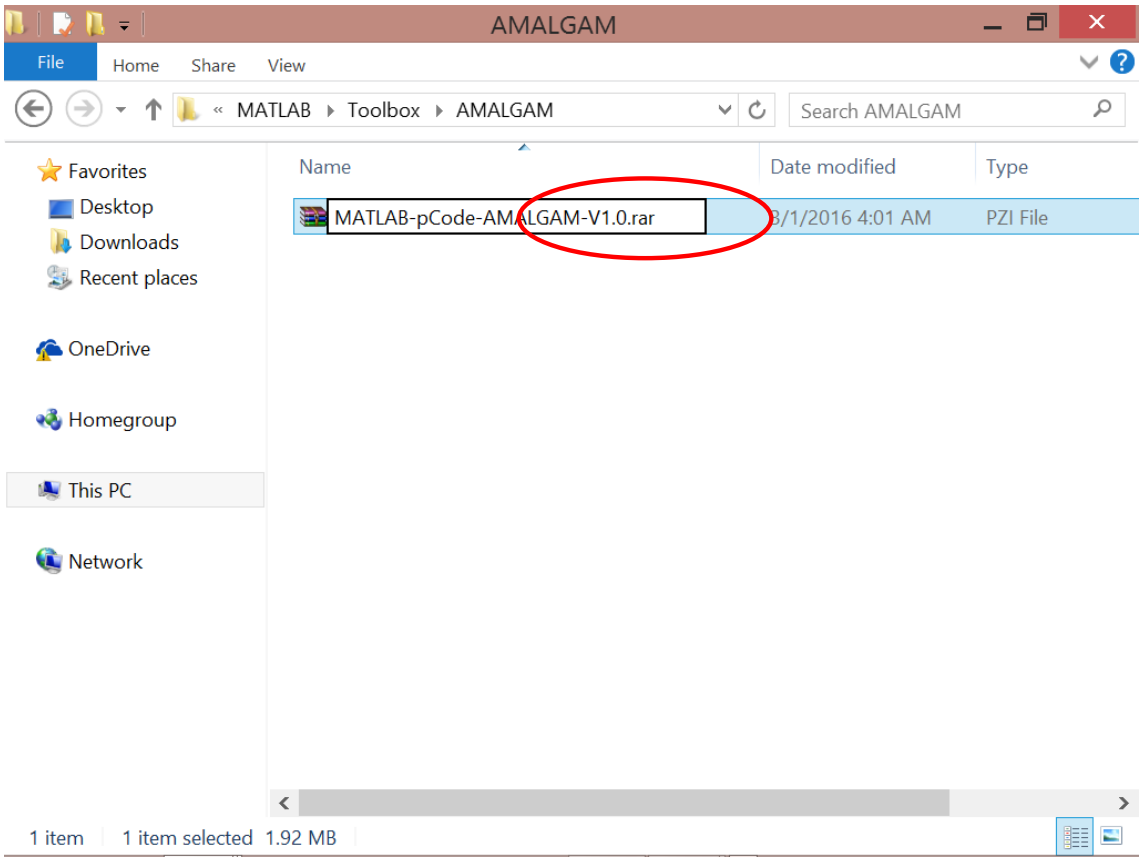


Figure A.4

After entering the new extension, hit the **Enter** (return) key. Windows will give you a warning that the file may not work properly (see Figure A.5figure.caption.24). This is quite safe - remember that you can restore the original extension if anything goes wrong.

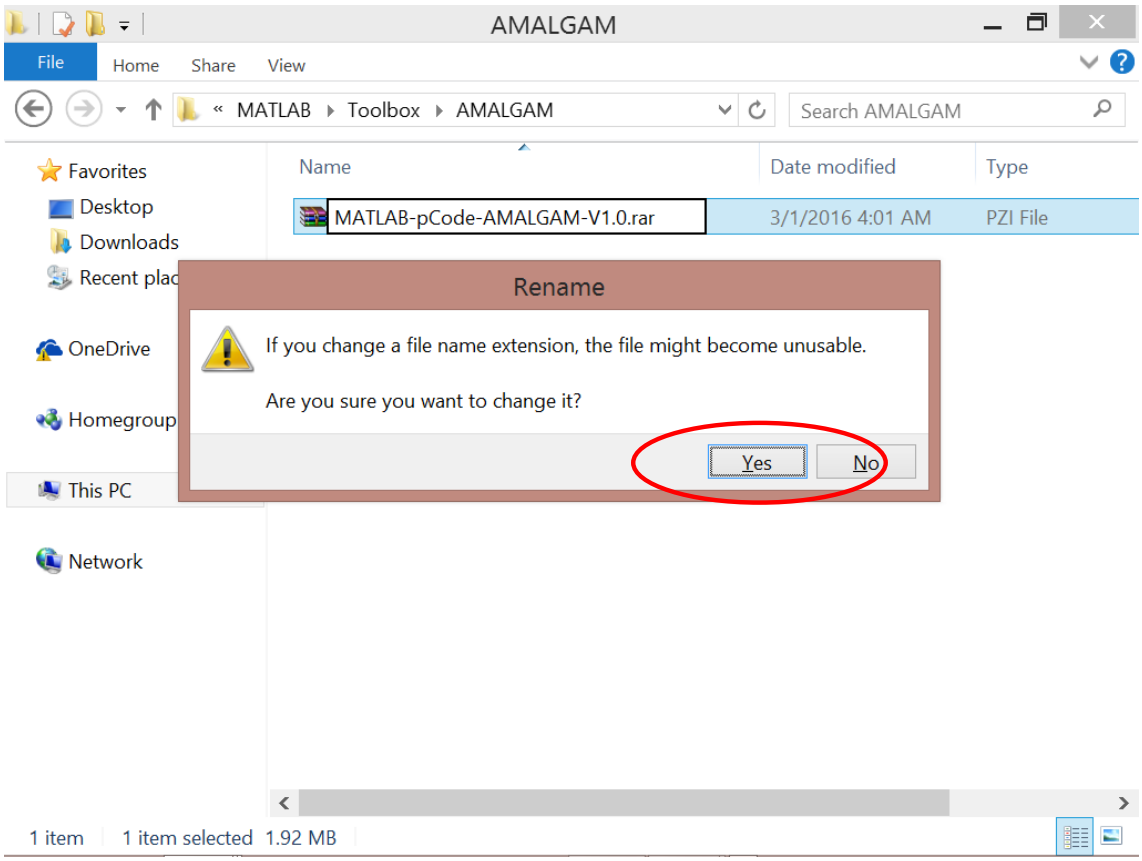


Figure A.5

It is also possible that you might get another message telling you that the file is "read-only". In this case either say yes to turning off read-only, or right-click the file, select **Properties** and uncheck the **Read-only** box.

If you do not have permission to change the file extension, you may have to login as Administrator. Another option is to make a copy of the file, rename the copy and then delete the original.

Now you have changed the extension of the file to ".rar" you can use the program WinRAR to extract the files to whatever folder your desire, for instance "D:\Downloads\Toolboxes\MATLAB\AMALGAM". Right-click the file name and select **Extract Here** (see Figure A6).

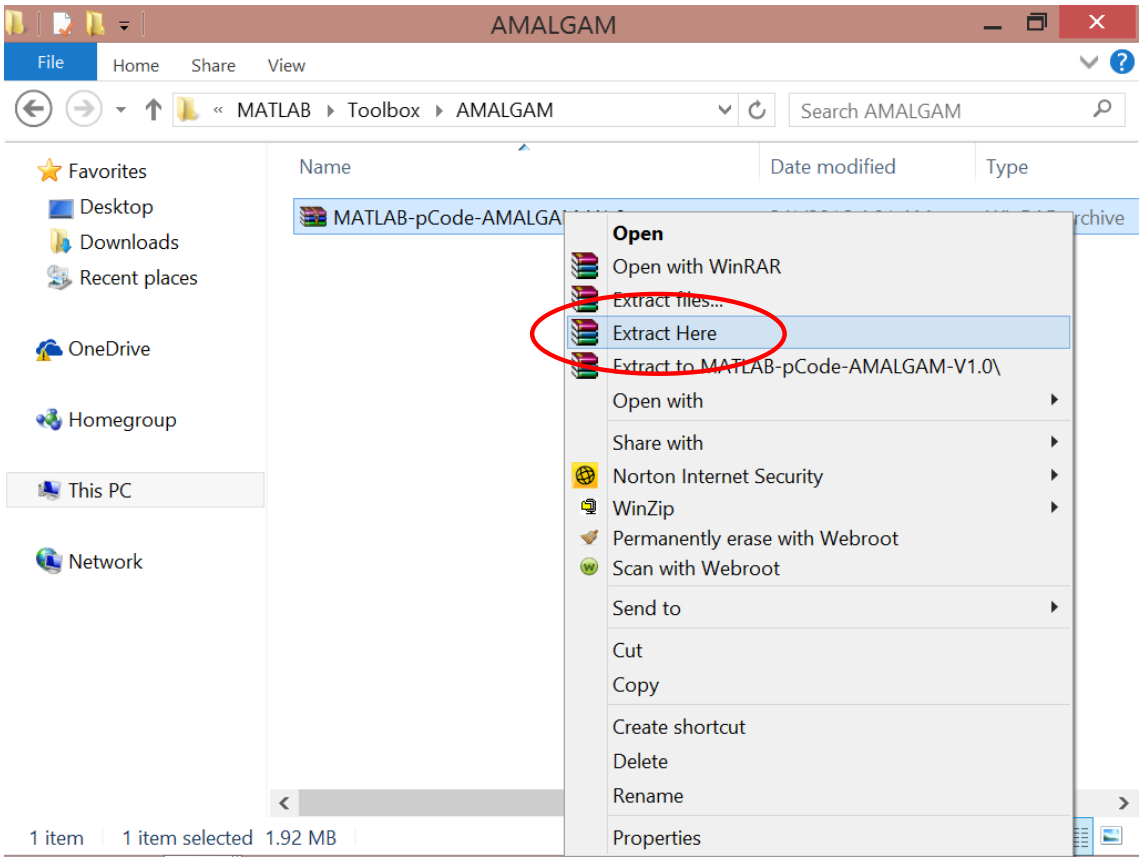


Figure A.6

Now WinRAR should extract the files to your folder. The end result should look as in Figure A.7figure.caption.26.

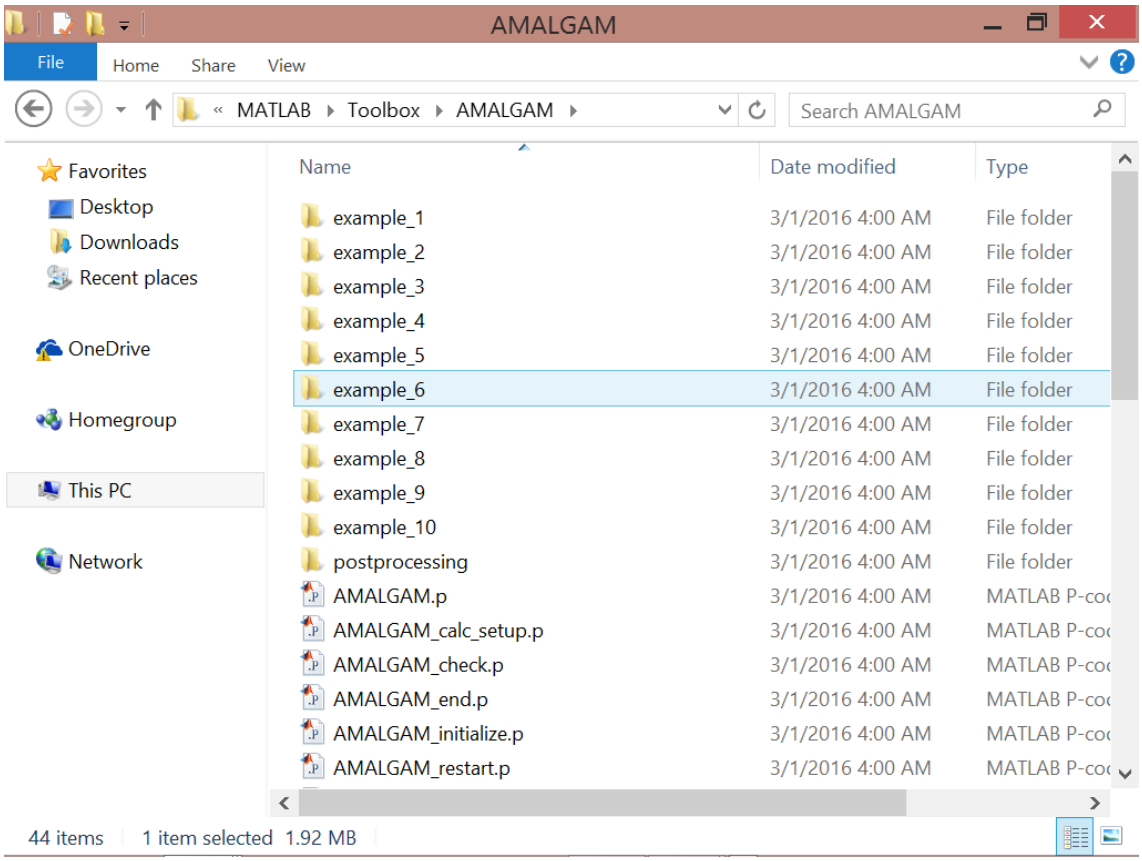


Figure A.7

As last step, please open MATLAB and go to the appropriate directory with the AMALGAM files, for instance "D:\Downloads\Toolboxes\MATLAB\AMALGAM". Now execute the following statement in the MATLAB prompt: `addpath(pwd,[pwd '/postprocessing']).` By adding the main and postprocessing directories of AMALGAM to the search path, the user can execute the toolbox from within any other directory. Now the AMALGAM toolbox is ready for use. If you want to execute the first built-in case study, then please change the current working directory of MATLAB to "...\example_1" via **cd example_1** or **cd('D:\Downloads\Toolboxes\MATLAB\AMALGAM\example_1')**, and then type in the MATLAB prompt: **example_1**.

Appendix B. Main functions of the AMALGAM toolbox

Table B1Description of the MATLAB functions and scripts (.m files) used by AMALGAM, version 1.0.table.caption.27 summarizes, in alphabetic order, the most important functions (scripts) of the AMALGAM package in MATLAB. The toolbox also includes two functions that are written in C/C++ including IGD.CPP and PARETO_FRONT.C. These functions are useful as they speed up calculation of the IGD statistic and the Pareto ranks.

The file AMALGAM.M is the main program of the toolbox. This function calls the different subroutines listed above and returns to the user the set of nondominated solutions, `X`, their associated objective function values, `F`, a structure with fields that summarize the algorithmic performance of AMALGAM, `output`, an historical archive of all past parent populations, `Z`, and if appropriate, a matrix `Y_sim` with the simulations of the final population, `X`. This program is called on the last line of the file "example_X.m" stored in the \example_X folders, where X = $\{1, \ldots, 10\}$. These directories store the the `model` script written by the user and all other data file(s), MATLAB functions, and external executable(s) necessary to compute the objective functions of each parameter vector. The example studies cover a large range of problem features, most of which have been published in the literature. The "example_X.m" files serve as templates for users to setup and solve their own specific multi-criteria calibration studies.

The directory "\postprocessing" in the root directory of the AMALGAM toolbox contains a number of different functions designed to visualize the results (output arguments) of AMALGAM. Graphical and tabulated output is produced by the function POSTPROC_AMALGAM which is called by AMALGAM. This function creates the text file "AMALGAM_output.txt" which is printed automatically to the MATLAB editor and contains two tables which list the Pareto mean and standard deviation of the rank 1 solutions of the final population, and their correlation coefficients. The postprocessor also generates automatically, a large number of figures that are printed to the screen. This includes one or more scatter plots (two- or three-dimensional) of the objective function values of the nondominated (Pareto) solution set, histograms and bivariate scatter plots of the Pareto samples, a two-dimensional plot with $d \times d$ cells that depict graphically the linear correlation coefficients between the different dimensions (parameters) of the Pareto solution set, trace plots of the IGD convergence diagnostic (if input argument `Fpar` has been defined by user) and the selection probability of each recombination method. Appendix D displays the output generated by the postprocessor for the last case study. Graphical output is suppressed if the field `print` of structure `options` is set to 'no'. Appendix D presents a screen copy of the graphical output of the forth case study discussed herein.

Table B1: Description of the MATLAB functions and scripts (.m files) used by AMALGAM, version 1.0.

Name of function	Description
AMALGAM	Main AMALGAM function that call the different functions listed here and returns Pareto solution set
AMALGAM_CALC_SETUP	Setup of computational heart of AMALGAM and (if activated) the distributed computing environment
AMALGAM_CHECK	Checks the AMALGAM setup for potential errors and/or inconsistencies in the settings defined by the user
AMALGAM_END	Terminates computing environment and creates return arguments
AMALGAM_INITIALIZE	Pre-allocates the main matrices and vectors used by AMALGAM and draws initial population
AMALGAM_RANK	Pareto ranking and crowding distance computation of parents and children
AMALGAM_RESTART	Initializes the setup of AMALGAM for a restart run (continue aborted trial)
AMALGAM_SETUP	Setup of the main algorithmic variables and options used by AMALGAM
CREATE_CHILDREN	Function that uses different recombination methods to create the child population
CROWDING_DISTANCE	Function that calculates the crowding distance of the combined parent and offspring population
EVALUATE_MODEL	Evaluates the offspring population and computes objective function values (executes script `Func_name`)
FNS	Fast nondominated sorting of matrix (population) of objective function values
GEN_NEW_POPULATION	Creates new population by comparing rank and crowding distance of parents and their offspring
HANDLE_CONSTRAINTS	Handle parameter constraints (if necessary)
LATIN	Latin hypercube sampling of initial parent population
LOAD_CALCULATION	Updates the selection probability of each recombination method
LOAD_DISTRIBUTION	Determines which parents to be used with each recombination method
README.TXT	Text file (ascii format) with details how to setup and use the AMALGAM toolbox
REC_METHOD_AMS	Produces offspring from selected parents using adaptive Metropolis algorithm
REC_METHOD_DE	Creates offspring from selected parents using differential evolution
REC_METHOD_NSGA	Produces offspring from selected parents using NSGA-II algorithm
REC_METHOD_PSO	Creates offspring from selected parents using particle swarm optimizer
STRENGTH_PARETO	Calculates the Pareto strength of each individual of population (e.g. (*Zitzler and Thiele*, 1999))
UPDATE_PSO	Update best known position of each particle and overall best position

Appendix C. MATLAB function scripts used in case studies

This Appendix presents the different `model` functions used in the four case studies presented in section 5Numerical examplessection.5 of this paper. These functions (.m file) serve as a template for users to help define their own forward model in AMALGAM. All `model` functions have as input argument, `x` a vector of size $d \times 1$ with d parameter values, and as output a vector of size $m \times 1$ with corresponding m values of the objective functions. A low-dash is used in the print out of each `model` scripts to denote the use of a built-in MATLAB function.

Appendix C.1. Case Study I: ZDT1

The MATLAB function `AMALGAM_ZDT1` listed below uses common built-in operators to calculate the pair of objective functions of test problem 1 described in (*Zitzler et al.*, 2000) for a given vector, `x` of size $d \times 1$ of parameter values, and value of the dimensionality, `d`.

MATLAB function of the ZDT1 test problem.

```
function F = AMALGAM_ZDT1 ( x , d )
% ZDT1 - test function
% Reference: E. Zitzler, K. Deb, and L. Thiele, Comparison of multiobjective
%   evolutionary algorithms: Empirical results, Evolutionary Computation, 8 (2),
%   183-195, 2000

f = x(1);                          % f
g = 1 + 9/(d-1)*sum(x(2:d));       % g
h = 1 - sqrt(f./g);                % h
F(1) = f; F(2) = g.*h;             % Return objective functions
```

Appendix C.2. Case Study II: ZDT4

The MATLAB function `AMALGAM_ZDT4` listed below uses common built-in operators to calculate the pair of objective functions of test problem 4 described in (*Zitzler et al.*, 2000) for a given vector, `x` of size $d \times 1$ of parameter values, and value of the dimensionality, `d`.

MATLAB function of the ZDT4 test problem.

```
function F = AMALGAM_ZDT4 ( x , d )
% ZDT4 - test function
% Reference: E. Zitzler, K. Deb, and L. Thiele, Comparison of multiobjective
%   evolutionary algorithms: Empirical results, Evolutionary Computation, 8 (2),
%   183-195, 2000

f = x(1);                                                    % f
g = 1 + 10*(d-1) + sum(x(2:d).^2 - 10*cos(4*pi*x(2:d)));     % g
h = 1 - sqrt(f./g);                                          % h
F(1) = f; F(2) = g.*h;                                       % Return objective functions
```

Appendix C.3. Case Study III: hmodel conceptual watershed model

The MATLAB function `AMALGAM_hmodel` listed below simulates the rainfall-runoff transformation for a vector, `x` of size $d \times 1$ of parameter values and returns two output variables; a pair of RMSE based objective function values for the driven and nondriven part of the hydrograph, and the corresponding simulation of the hmodel. This second output argument is required as the field `modout` of `option` has been set to 'yes' in the input script (see Figure 15Normalized parameter plots for the BMA weights of the Gamma conditional distribution using a three-criterion {RMSE, IS, CRPS} calibration with AMALGAM for the 3000-day training data set of daily discharge values simulated with eight watershed models. Each line across the graph denotes a single Pareto solution. The plots at the right-hand side are three-dimensional projections of the objective space of the Pareto set of solutions. The solutions of the single criterion ends are indicated with a red (RMSE), blue (IS) and green (CRPS) line (plot A) and cross (plot B), respectively.figure.caption.18). The two objective function values are computed from the simulated discharge time series using

$$f_1(\mathbf{x}) = \sqrt{\frac{1}{n_{\mathrm{D}}} \sum_{j \in \mathrm{D}} (y_j(\mathbf{x}) - \tilde{y}_j)^2} \quad , \quad f_2(\mathbf{x}) = \sqrt{\frac{1}{n_{\mathrm{ND}}} \sum_{j \in \mathrm{ND}} (y_j(\mathbf{x}) - \tilde{y}_j)^2}, \tag{C.1}$$

where $\tilde{\mathbf{Y}} = \{\tilde{y}_1, \ldots, \tilde{y}_n\}$ and $\mathbf{Y} = \{y_1(\mathbf{x}), \ldots, y_n(\mathbf{x})\}$ denote the observed and simulated discharge time series, respectively, and n_{D} (n_{ND}) signifies the number of driven (nondriven) discharge observations.

MATLAB function that executes hmodel and returns a pair of RMSE objective functions of the driven and nondriven part of the hydrograph.

```
function [ F , Y_sim ] = AMALGAM_hmodel ( x , func_in )
% hmodel - conceptual hydrologic model
% Reference: G. Schoups, and J.A. Vrugt, A formal likelihood function for parameter
%   and predictive inference of hydrologic models with correlated, heteroscedastic,
%   and non-Gaussian errors, Water Resources Research, 46, W10531, doi:10.1029/2009
    WR008933,
%   2010.

% Extract various input data
[tout,data,hmodel_options,y0,Y_obs,n,id_d,N_d,id_nd,N_nd] = v2struct(func_in,
    func_in.fields);

y = hmodel(x,tout,data,hmodel_options,y0);                  % Simulate discharge with x
Y = y(5,2:n+1) - y(5,1:n);                                  % Compute discharge from state
Y_sim = Y(731:n);                                           % Remove first two years (burn-
    in)
F(1) = sqrt(sum((Y_sim(id_d)-Y_obs(id_d)).^2)/N_d);         % Calculate RMSE driven part
F(2) = sqrt(sum((Y_sim(id_nd)-Y_obs(id_nd)).^2)/N_nd);      % Calculate RMSE nondriven part
```

The source code of the hmodel is written in C and linked into a shared library using the MEX-compiler of MATLAB. This avoids file writing, and allows for direct passing of the parameter values, forcing conditions, and settings of the numerical solver to `crr_model`. A second-order time-variable integration method is used to solve the differential equations of the hmodel.

Appendix C.4. Case Study IV: Bayesian Model Averaging

The MATLAB function `AMALGAM_BMA` computes for a given $d \times 1$ vector, `x`, of BMA weights and standard deviations (or proxies thereof) and structure array `func_in` the $1 \times m$ vector `F` with $m = 3$ metrics of forecast skill, and the $n \times 1$ vector`G_dot` with mean forecast of the BMA model. This second output argument can be used to quantify the Pareto simulation uncertainty of the BMA model. The second input argument, `func_in`, contains the different variables that are required for calculation of the RMSE, IS, and CRPS skill metrics.

MATLAB function which computes three diagnostics metrics of forecast skill of the BMA model.

```
function [ F , G_dot ] = AMALGAM_BMA ( x , func_in )
% This function calculates the RMSE, IS, and CRPS of the BMA mixture distribution
% Reference: J.A. Vrugt, M.P. Clark, C.G.H. Diks, Q. Duan, and B. A. Robinson (2006),
%   Multi-objective calibration of forecast ensembles using Bayesian model averaging,
%   Geophysical Research Letters, 33, L19817, 2006

if nargin < 2
    error('AMALGAM_BMA:TooFewInputs','Requires at least two input arguments.');
end

[D,Y,PDF,VAR] = v2struct(func_in,{'Fieldnames','D','Y','PDF','VAR'}); % Unpack fields
    func_in

[n,K] = size(D);            % Ensemble size (number forecasts, number models)
w = x(1:K);                 % Unpack weights from (d x 1) vector, x with parameter
    values

switch VAR % VARIANCE OPTION -> (n x K)-matrix "sigma" with standard deviations
    forecasts
    case {'1'} % 1: common constant variance
        sigma = x(K+1) * ones(n,K);
    case {'2'} % 2: individual constant variance
        sigma = bsxfun(@times,x(K+1:2*K)',ones(n,K));
    case {'3'} % 3: common non-constant variance
        c = x(K+1); sigma = c * D;
    case {'4'} % 4: individual non-constant variance
        c = x(K+1:2*K)'; sigma = bsxfun(@times,c,D);
end
sigma = max(sigma,eps);     % each element (n x K)-matrix sigma at least equal to 2.22e
    -16

switch PDF % CONDITIONAL DISTRIBUTION -> (n x K)-matrices "A" and "B"
    case {'normal'} % Gaussian with mean "D" and standard deviation "sigma"
        A = D; B = sigma;
    case {'gamma'}  % Gamma with shape "A" and scale "B"
        mu = abs(D); var = sigma.^2; A = mu.^2./var; B = var./mu;
end
L = pdf(PDF,repmat(Y,1,K),A,B);     % (n x K)-matrix of likelihoods forecasts at Y
lik = L*w + realmin;                % (n x 1)-vector of likelihoods BMA model at Y
G_dot = D*w; res = G_dot - Y;       % (n x 1)-vectors of BMA mean forecast and residuals
G = BMA_rnd(PDF,w,A,B,2);           % Draw randomly two [n x 1]-vectors of BMA forecasts
     at Y

F(1) = sqrt(1/n*(res'*res));                        % F_{1}: RMSE mean forecast BMA model
F(2) = -mean(log(lik));                             % F_{2}: Ignorance score BMA model
F(3) = mean(abs(G{1}-Y))-.5*mean(abs(G{1}-G{2}));   % F_{3}: CRPS score BMA model
```

This function `AMALGAM_BMA` uses as input arguments the BMA weights and standard deviations (or proxies thereof) of matrix `x` and the structure array `func_in` and returns to the user the vector `F` with the root

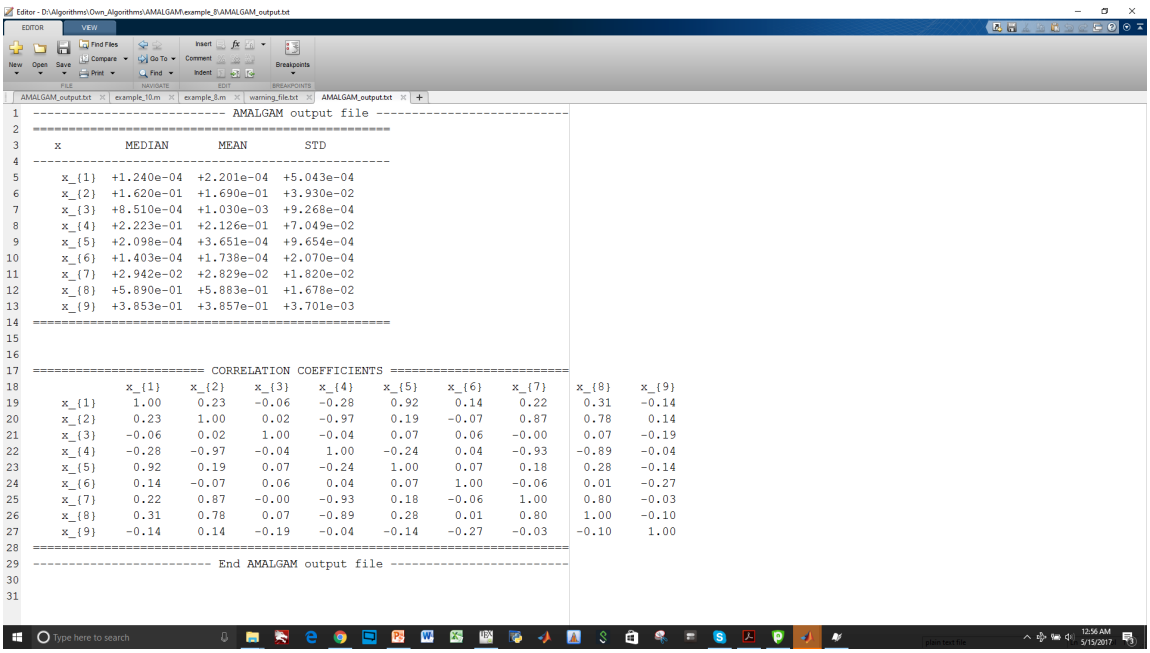
mean square error, RMSE (mm/day), ignorance score, IS (-), and continuous rank probability score, CRPS (-), and the vector `G_dot` with mean forecast of the BMA model. The second input argument, `func_in` is defined by the user in the input file "example_8.m" (see section 5.4) and passes to the BMA plugin the $n \times K$ matrix `D` with ensemble forecasts, the $n \times 1$ vector `Y` with verifying observations, and the fields PDF and VAR that define with a string the members' forecast distribution and variance option, respectively (see Page 39Case Study IV: Bayesian Model Averagingfigure.caption.15). Notation in the function `AMALGAM_BMA` is consistent with main text.

The function `v2struct` is part of the AMALGAM toolbox and extracts the fields of the structure `func_in`. Built-in functions are highlighted with a low dash. The `switch` function switches among the several cases listed in the code. The built-in function `pdf(PDF,repmat(Y,1,K),A,B)` calculates the $n \times K$ matrix `L` that stores the density of each member's forecast distribution at the verifying observations of Y. The density of the BMA mixture distribution (also called likelihood) is then simply equivalent to the sum of the individual forecast densities multiplied with their respective weights stored in `w`. `realmin` ($= 2.225 \cdot 10^{-308}$) avoids a zero likelihood. `log(L)` computes the natural logarithm of the `n` likelihood values of the BMA model, `mean(a)` computes the average of the vector `a`, and `BMA_rnd` is a function that returns two `n` vectors with samples from the BMA density (see Page 41lstlisting.-7).

Appendix D. Screen output

The AMALGAM toolbox presented herein returns to the user tables and figures which jointly summarize the results of the toolbox. This appendix displays all this output for the last case study involving application of AMALGAM to multi-criteria BMA model training using daily discharge data and corresponding simulated values of an eight member model ensemble for the Leaf River watershed near Collins, Mississippi, USA.

Figure D1Screen print of ascii file "AMALGAM_output.txt". This file is automatically generated by the postprocessor of the AMALGAM toolbox and printed to the screen in the MATLAB editor.figure.caption.28 displays the ascii file "AMALGAM_output.txt" which is created by the function AMALGAM_POSTPROC and printed to the screen in the MATLAB editor.



```
------------------------- AMALGAM output file -------------------------
=================================================
   x          MEDIAN        MEAN          STD
-------------------------------------------------
    x_{1}  +1.240e-04  +2.201e-04  +5.043e-04
    x_{2}  +1.620e-01  +1.690e-01  +3.930e-02
    x_{3}  +8.510e-04  +1.030e-03  +9.268e-04
    x_{4}  +2.223e-01  +2.126e-01  +7.049e-02
    x_{5}  +2.098e-04  +3.651e-04  +9.654e-04
    x_{6}  +1.403e-04  +1.738e-04  +2.070e-04
    x_{7}  +2.942e-02  +2.829e-02  +1.820e-02
    x_{8}  +5.890e-01  +5.883e-01  +1.678e-02
    x_{9}  +3.853e-01  +3.857e-01  +3.701e-03
=================================================


======================== CORRELATION COEFFICIENTS ========================
             x_{1}    x_{2}    x_{3}    x_{4}    x_{5}    x_{6}    x_{7}    x_{8}    x_{9}
    x_{1}     1.00     0.23    -0.06    -0.28     0.92     0.14     0.22     0.31    -0.14
    x_{2}     0.23     1.00     0.02    -0.97     0.19    -0.07     0.87     0.78     0.14
    x_{3}    -0.06     0.02     1.00    -0.04     0.07     0.06    -0.00     0.07    -0.19
    x_{4}    -0.28    -0.97    -0.04     1.00    -0.24     0.04    -0.93    -0.89    -0.04
    x_{5}     0.92     0.19     0.07    -0.24     1.00     0.07     0.18     0.28    -0.14
    x_{6}     0.14    -0.07     0.06     0.04     0.07     1.00    -0.06     0.01    -0.27
    x_{7}     0.22     0.87    -0.00    -0.93     0.18    -0.06     1.00     0.80    -0.03
    x_{8}     0.31     0.78     0.07    -0.89     0.28     0.01     0.80     1.00    -0.10
    x_{9}    -0.14     0.14    -0.19    -0.04    -0.14    -0.27    -0.03    -0.10     1.00
==========================================================================
------------------------ End AMALGAM output file ------------------------
```

Figure D1: Screen print of ascii file "AMALGAM_output.txt". This file is automatically generated by the postprocessor of the AMALGAM toolbox and printed to the screen in the MATLAB editor.

The toolbox also presents to the user a large number of figures that visualize the results. I now display the figures that were generated for the last case study.

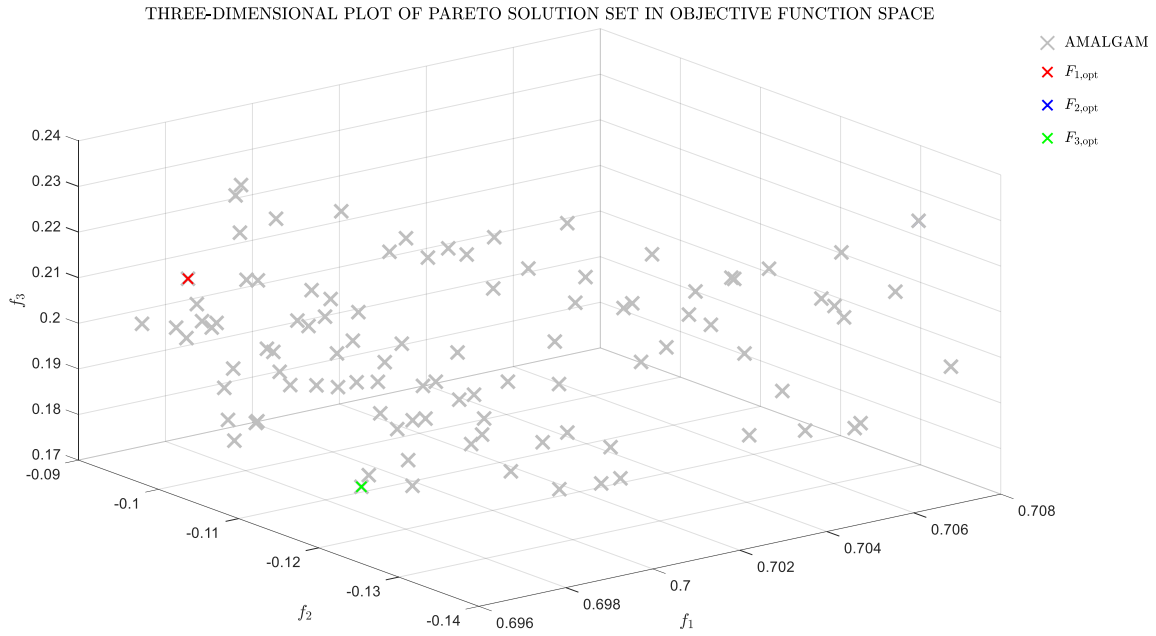


Figure D2: Three dimensional projection of the Pareto solution set (rank 1 solutions) derived from AMALGAM. The single criterion solutions of the RMSE, IS and CRPS are separately indicated with the blue, red, and green cross symbols, respectively.

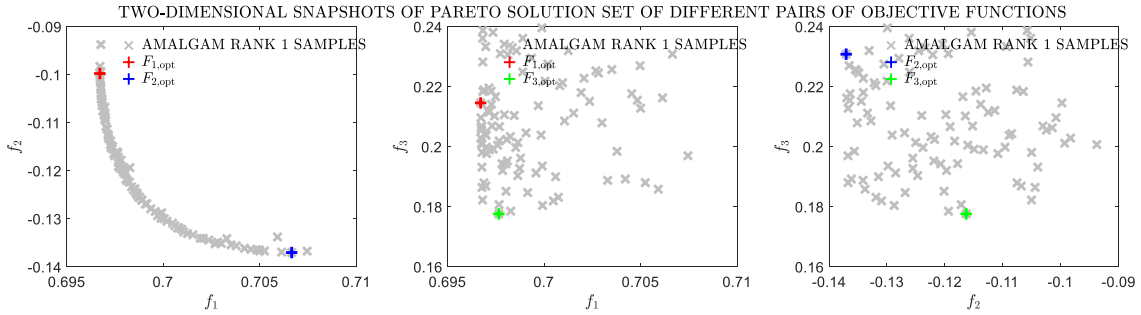


Figure D3: Two dimensional projections of the Pareto solution set (rank 1 members) for different pairs of the RMSE, IS and CRPS forecast skill metrics. The single criterion solutions of the RMSE, IS and CRPS are separately indicated in each panel with the blue, red, and green cross symbols, respectively.

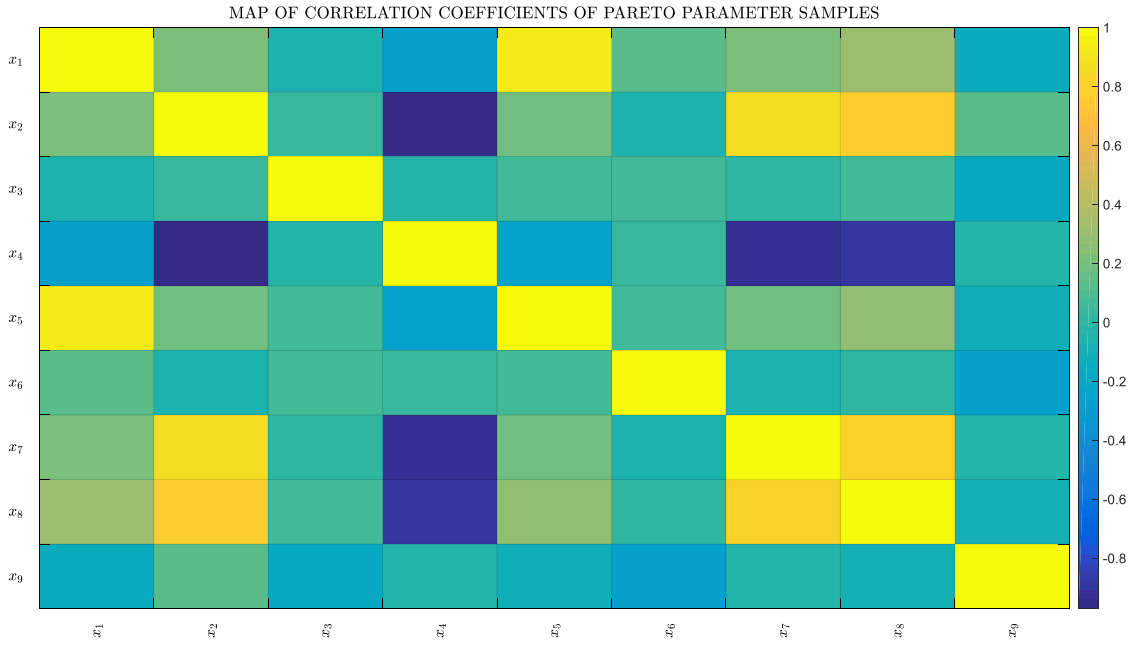


Figure D4: Two dimensional plot of the correlation coefficients of the parameters of the Pareto solution set.

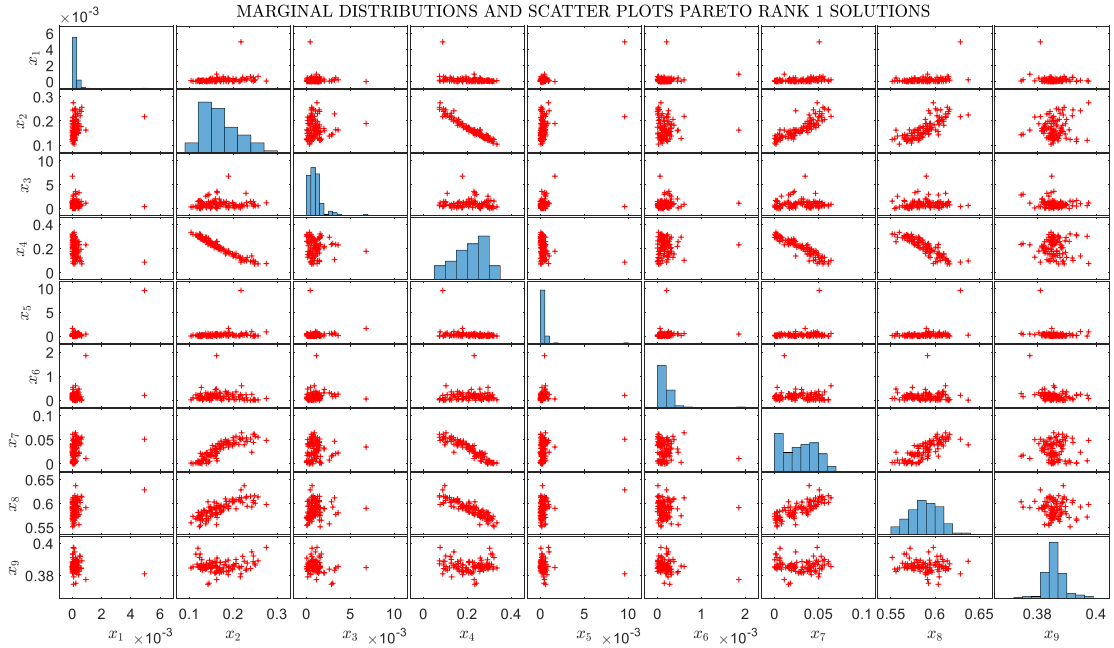


Figure D5: Scatter plot matrix of the AMALGAM derived Pareto parameter solutions of the BMA model. The main diagonal displays histograms of the marginal Pareto distribution of each individual BMA parameter, whereas the off-diagonal graphs display bivariate scatter plots of the Pareto samples.

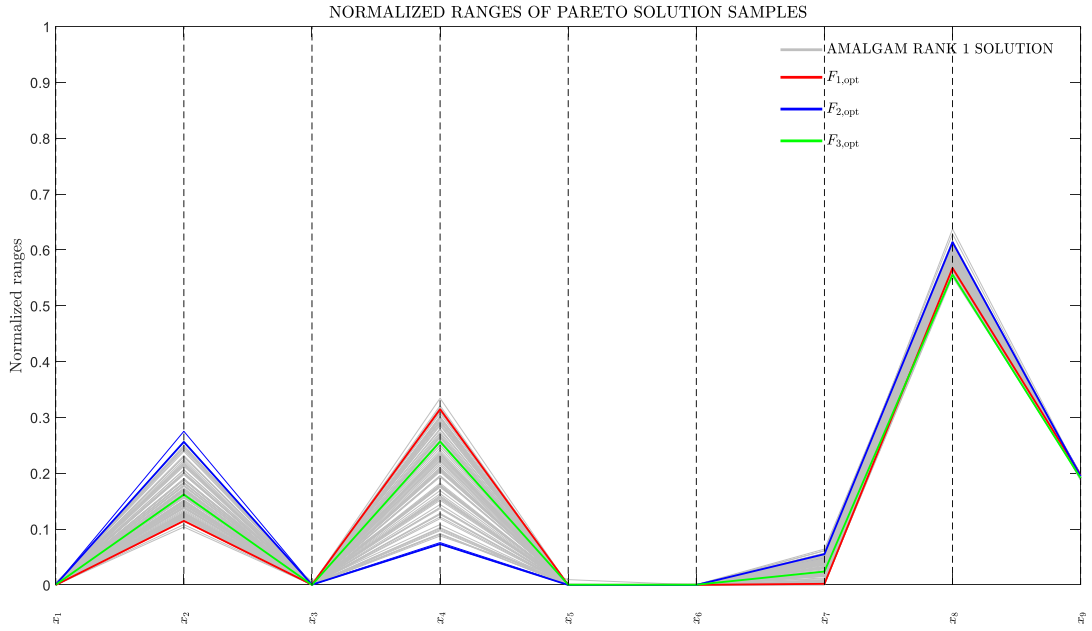


Figure D6: Normalized parameter plots of the $d = 9$ BMA parameters using three-criteria, {RMSE, IS, CRPS}, BMA model training with AMALGAM. Each line across the graph denotes a single Pareto parameter vector: grey lines are regular Pareto solutions, and colored lines signify the single-criterion solutions of RMSE, IS and CRPS, respectively.

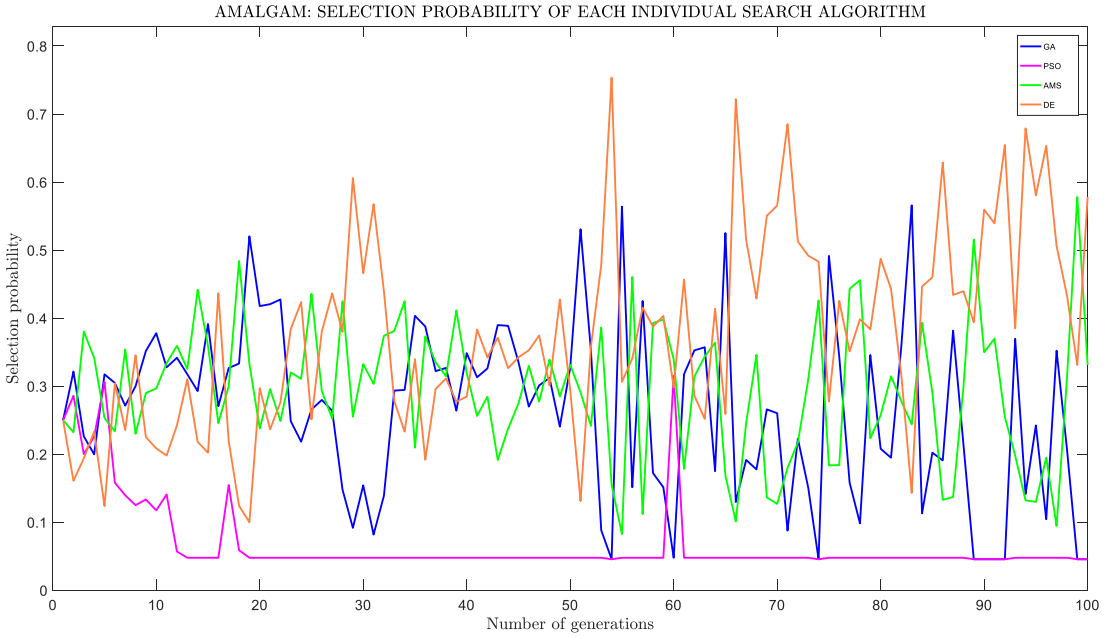


Figure D7: Trace plot of the selection probabilities of the different recombination methods of AMALGAM.

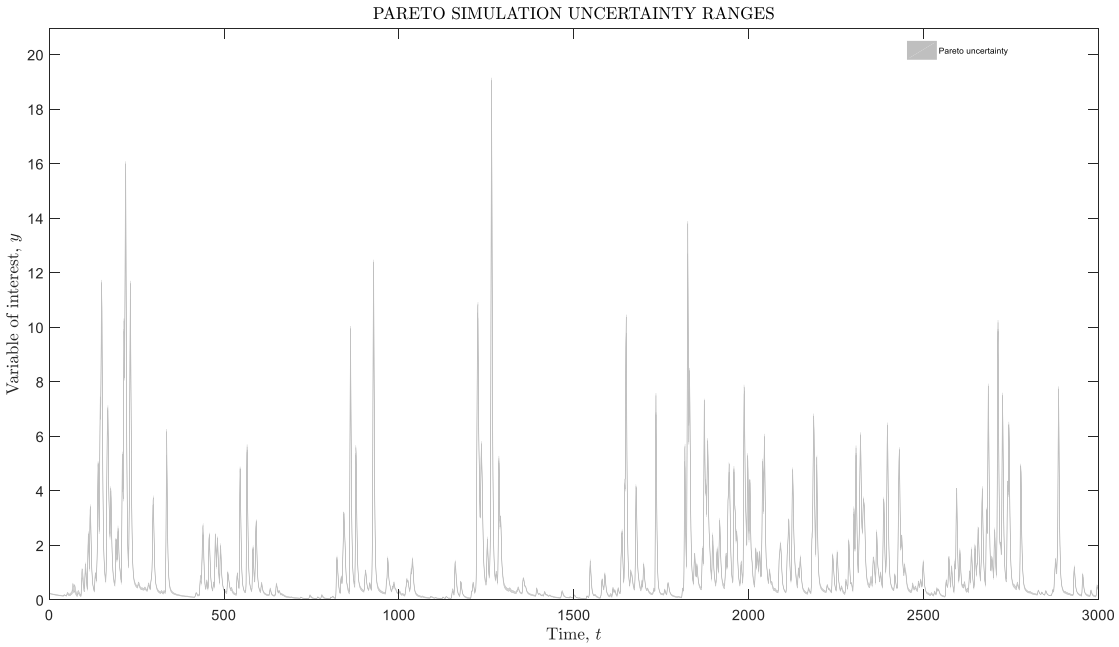


Figure D8: Pareto simulation uncertainty ranges of the BMA model.

Appendix E. References

References

D. Achlioptas, A. Naor, and Y. Peres, "Rigorous location of phase transitions in hard optimization problems," *Nature*, vol. 435, pp. 759-763, 2005.

J. Barhen, V. Protopopescu, and D. Reister, "TRUST: A deterministic algorithm for global optimization," *Science*, 276, pp. 1094-1097, 1997.

N. Beume, C.M. Fonseca, M. López-Ibáñez, L. Paquete, and J. Vahrenhold, "On the complexity of computing the hypervolume indicator," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1075-1082, doi:10.1109/TEVC.2009.2015575, 2009.

D.G. Bounds, "New optimization methods from physics and biology," *Nature*, vol. 329, pp. 215-219, 1987.

D.P. Boyle, H.V. Gupta, and S. Sorooshian, "Toward improved calibration of hydrologic models: Combining the strengths of manual and automatic methods," *Water Resources Research*, vol. 36, no. 12, 3663-3674, 2000.

K. Deb, and R.B. Agrawal, "Simulated binary crossover for continuous search space," *Complex Systems*, vol. 9, pp. 115-148, 1995.

K. Deb, "Multi-objective optimization using evolutionary algorithms,", Wiley, New York, 2001.

K. Deb, L. Thiele, M. Laumanns, and E. Zitzler "Scalable test problems for evolutionary multiobjective optimization," *Evolutionary multiobjective optimization*, pp. 105-145, 2005.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182-197.

A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society*, vol. 39(B), pp. 1-39, 1977.

Q. Duan, V.K. Gupta, and S. Sorooshian, "Effective and efficient global optimization for conceptual rainfall-runoff models," *Water Resources Resources*, vol. 28, no. 4, pp. 1015-1031, 1992.

M. Glick, A. Rayan, and A. Goldblum, "A stochastic algorithm for global optimization and for best populations: a test case of side chains in proteins," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, pp. 703-708.

T. Gneiting, and A.E Raftery, "Weather forecasting with ensemble methods," *Science*, vol. 310, pp. 248-249, 2005.

T. Gneiting, and A.E. Raftery, "Strictly Proper Scoring Rules, Prediction, and Estimation," *Journal of the American Statistical Association*, vol. 102, pp. 359-378, 2007.

D.E. Goldberg, "Genetic algorithms in search, optimization and machine learning," Addison-Wesley, Reading, M, 1989.

E.P. Grimit, and C.F. Mass, "Initial results of a mesoscale shortrange ensemble forecasting system over the Pacific Northwest", *Weather Forecasting*, vol. 17, pp. 192-205, 2002.

H. Haario, E. Saksman, and J. Tamminen, "An adaptive Metropolis algorithm," *Bernoulli*, vol. 7, pp. 223-242, 2001.

W.E. Hart, N. Krasnogor, and J.E. Smith, "Recent advances in memetic algorithms," Springer, Berlin, 2005.

H. Hersbach, "Decomposition of the continuous ranked probability score for ensemble prediction systems," *Weather Forecasting*, vol. 15, pp. 559-570, 2000.

J. Holland, "Adaptation in Natural and Artificial Systems," MIT Press, Cambridge, MA, 1975.

J. Kennedy, R.C. Eberhart, and Y. Shi, "Swarm Intelligence, " Morgan Kaufmann, San Francisco, 2001.

J. Knowles, and D. Corne, "The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation," *Proceedings of the 1999 Congress on Evolutionary Computation*, IEEE Press, New York, 1999.

E. Laloy, and J.A. Vrugt, "High-dimensional posterior exploration of hydrologic models using multiple-try DREAM$_{(ZS)}$ and high-performance computing," *Water Resources Research*, vol. 48, W01526, doi:10.1029/2011WR010608, 2012a.

M. Laumanns, E. Zitzler, and L. Thiele, "A unified model for multi-objective evolutionary algorithms with elitism," *Proceedings of the 2000 Congress on Evolutionary Computation*, vol. 1, pp. 46-53, doi:10.1109/CEC.2000.870274, 2000.

A.R. Lemmon, and M.C. Milinkovitch, "The metapopulation genetic algorithm: an efficient solution for the problem of large phylogeny estimation," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, pp. 10516-10521, 2002.

H. Li and Q. Zhang, "Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 284-302, 2009.

M.A. Nowak, and K. Sigmund, "Evolutionary dynamics of biological games," *Science*, vol. 303, pp. 793-799, 2004.

K.E. Parsopoulos, and M.N. Vrahatis, "On the computation of all global minimizers through particle swarm pptimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 211-224, 2004.

A.E. Raftery, T. Gneiting, F. Balabdaoui, and M. Polakowski, "Using Bayesian model averaging to calibrate forecast ensembles," *Monthly Weather Review*, vol. 133, pp. 1155-1174, 2005.

J. Rings, J.A. Vrugt, G. Schoups, J.A. Huisman, and H. Vereecken, "Bayesian model averaging using particle filtering and Gaussian mixture modeling: Theory, concepts, and simulation experiments," *Water Resources Research*, 48, W05520, doi:10.1029/2011WR011607, 2012.

G. Schoups, J.W. Hopmans, C.A. Young, J.A. Vrugt, W.W. Wallender, K.K. Tanji, and S. Panday "Sustainability of irrigated agriculture in the San Joaquin Valley, California,"*Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, pp. 15352-15356, 2005.

G. Schoups, and J.A. Vrugt, "A formal likelihood function for parameter and predictive inference of hydrologic models with correlated, heteroscedastic and non-gaussian errors," *Water Resources Research*, vol. 46, W10531, doi:10.1029/2009WR008933, 2010.

J.M. Sloughter, A.E. Raftery, T. Gneiting, and C. Fraley, "Probabilistic quantitative precipitation forecasting using Bayesian model averaging," *Monthly Weather Review*, vol. 135, pp. 3209-3220, 2007.

J.M. Sloughter, T. Gneiting, and A.E. Raftery, "Probabilistic wind speed forecasting using ensembles and Bayesian model averaging," *Monthly Weather Review*, vol. 105, no. 489, pp. 25-35, doi:10.1198/jasa.2009.ap08615, 2010.

R. Storn, and K. Price, "A simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341-359, 1997.

D.A. van Veldhuizen, and G.B. Lamont, "Multiobjective evolutionary algorithm research: A history and analysis," Dept. Elec. Comput. Eng., Graduate School of Eng., Air Force Inst. Technol., Wright-Patterson, AFB, OH, Tech. Rep. TR-98-03, 1998.

J.A. Vrugt, H.V. Gupta, L.A. Bastidas, W. Bouten, and S. Sorooshian, "Effective and efficient algorithm for multi-objective optimization of hydrologic models," *Water Resources Research*, vol. 39 (8), 1214, doi:10.1029/2002WR001746, 2003.

J.A. Vrugt, M.P. Clark, C.G.H. Diks, Q. Duan, and B.A. Robinson, "Multi-objective calibration of forecast ensembles using Bayesian model averaging," *Geophysical Research Letters*, vol. 33, L19817, doi:10.1029/2006GL027126, 2006.

J.A. Vrugt, and B.A. Robinson, "Improved evolutionary optimization from genetically adaptive multimethod search," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 104, pp. 708-711, doi:10.1073/pnas.0610471104, 2007a.

J.A. Vrugt, and B.A. Robinson, "Treatment of uncertainty using ensemble methods: Comparison of sequential data assimilation and Bayesian model averaging," *Water Resources Research*, vol. 43, W01411, doi:10.1029/2005WR004838, 2007b.

J.A. Vrugt, C.J.F. ter Braak, M.P. Clark, J.M. Hyman, and B.A. Robinson, "Treatment of input uncertainty in hydrologic modeling: Doing hydrology backward with Markov chain Monte Carlo simulation," *Water Resources Research*, vol. 44, W00B09, doi:10.1029/2007WR006720, 2008a.

J.A. Vrugt, C.G.H. Diks, and M.P. Clark, "Ensemble Bayesian model averaging using Markov chain Monte Carlo sampling," *Environmental Fluid Mechanics*, vol. 8 (5-6), pp. 579-595, doi:10.1007/s10652-008-9106-3, 2008b.

J.A. Vrugt, B.A. Robinson, and J.M. Hyman, "Self-adaptive multimethod search for global optimization in real-parameter spaces," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 243-259, doi:10.1109/TEVC.2008.924428, 2009.

J.A. Vrugt, C.J.F. ter Braak, C.G.H. Diks, D. Higdon, B.A. Robinson, and J.M. Hyman, "Accelerating Markov chain Monte Carlo simulation by differential evolution with self-adaptive randomized subspace sampling," *International Journal of Nonlinear Sciences and Numerical Simulation*, vol. 10, no. 3, pp. 273-290, 2009b.

J.A. Vrugt, and C.J.F. ter Braak, "DREAM$_{(D)}$: an adaptive Markov Chain Monte Carlo simulation algorithm to solve discrete, noncontinuous, and combinatorial posterior parameter estimation problems," *Hydrology and Earth System Sciences*, vol. 15, pp. 3701-3713, doi:10.5194/hess-15-3701-2011, 2011.

J.A. Vrugt, and M. Sadegh, "Toward diagnostic model calibration and evaluation: Approximate Bayesian computation," *Water Resources Research*, vol. 49, doi:10.1002/wrcr.20354, 2013.

J.A. Vrugt, "Markov chain Monte Carlo simulation using the DREAM software package: Theory, concepts, and MATLAB Implementation," *Environmental Modeling & Software*, vol. 75, pp. 273-316, doi:10.1016/j.envsoft.2015.08.013, 2016.

D.J. Wales, and H.A. Scheraga, "Global optimization of clusters, crystals, and biomolecules," *Science*, vol. 285 (5432), pp. 1368-1372, doi:10.1126/science.285.5432.1368.

L. While, P. Hingston, L. Barone, and S. Huband, "A faster algorithm for calculating hypervolume", *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 1, pp. 29-38, 2006.

D.H. Wolpert, and W.G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 67-82, 1997.

Q. Zhang and H. Li, "MOEA/D: A multi-objective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712-731, 2007.

E. Zitzler, K. Deb, L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary Computation*, vol. 8, pp. 173-195, 2000.

E. Zitzler, and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257-271, 1999.

E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," Comput. Eng. Net. Lab. (TIK), Dept. Elec. Eng, ETH, Zurich, TIK-Rep. 103, pp. 1-21, 2001.