The Numerical Analyses of Global Optimization and Simulated Annealing

Jacques Nel*
jmn23@my.yorku.ca
ID. 212588109

Celina Landolfi*
clandolfi17@gmail.com
ID. 215587892

Gian Alix*
gian.alix@gmail.com
ID. 214760870

KEYWORDS

global optimization, simulated annealing, continuous variables

1 INTRODUCTION

Global optimization, a branch of applied mathematics and numerical analysis, is concerned with finding points on a bounded subset S of $\mathbb{R}^n$ in which some function f assumes its optimal value [1]. The function may have numerous local optima values and instead of searching for local points that cannot be improved, global optimization aims to find the global optimum on the bounded subset [2]. Thus, global optimization is a stronger version of local optimization and is desirable, but not necessary, in many practical applications; however, there are several applications where the use of global optimization is crucial including problems in safety verification, chemistry and hard feasibility problems [2]. In these cases, using local optimization methods could potentially return useless information, could be unrealistic with respect to the real world or could potentially underestimate the problem at hand [2]. Despite the obvious importance of global optimization and the efforts that have been invested into developing algorithms, the results have been unsatisfactory thus far and more work is needed for the optimization of more complicated functions [1]. Currently, numerical methods are used for the optimization of complicated functions, but these often cannot produce optimal results and will return a value close to a global optimum instead. By 'close to', we mean the following:

Definition 1.1 ('Close To' Global Minimum). For $\epsilon > 0, B_f(\epsilon)$ is the set of points with a value close to the minimal point, i.e.

$$B_f(\epsilon) = \{x \in S | \exists_{x_{min}} : |f(x) - f(x_{min})| < \epsilon\} \tag{1.1}$$

Furthermore, we construct a formal definition for what it means for a set of points to be close to a minimal point:

Definition 1.2. Let ϵ be any positive real number. Then we define $B_x(\epsilon)$ to be the set such that:

$$B_x(\epsilon) = \{x \in S \mid \exists x_{\min} \, : \, ||x - x_{\min}|| < \epsilon\} \tag{1.2}$$

and this is a set containing all points close to the minimal.

There are two classes of numerical methods in regards to global optimization: deterministic, in which the minimization process depends on probabilistic events, and stochastic methods, which does not use probabilistic information [1]. Unfortunately, deterministic global optimization methods have several disadvantages including that the global optimum can only be found after an exhaustive search over S, there is no guarantee of success from the method and that many assumptions must be made about f [1]. On the other hand, stochastic methods generally have better computational results than deterministic methods and can almost all be proven to find a global optimum that is asymptotically convergent with probability 1 [1]. With this taken into consideration, the concentration of global optimization methods will be on stochastic methods, specifically simulated annealing.

2 PROBLEM DEFINITION

Let $S \subset \mathbb{R}^n$ be a bounded set on $\mathbb{R}^n$ and $f : S \to \mathbb{R}^n$ be an n-dimensional real-valued function.
The problem that simulated annealing wishes to solve is to find a point $x_{min} \in S$ such that $f(x_{min})$ is a global minimum on S.

Specifically,

$$\forall x \in S : f(x_{\min}) \leq f(x) \tag{2.1}$$

The problem deals with global minimization; however, simulated annealing can find global maximization in the same way by reversing the sign of f without loss of generality [1].

3 THE NUMERICAL METHOD

Since local optimality does not guarantee global optimality, a fundamental concern in global optimization is getting stuck at a local optimum point [1]. In order to avoid and overcome this, a specific class of stochastic optimization is used, which is simulated annealing. Simulated annealing originates from the physical annealing process that is well known in condensed matter physics [1]. Physical annealing is a thermal process where the heating and slow cooling of a metal in a heat bath brings it into a more uniformly crystalline state where the free energy takes its global minimum. The role of temperature in physical annealing is to allow the particles to reach higher energy states in order to overcome energy barriers that would force them into local minima [2].

In greater detail, a state with energy level E_1 is compared to a state with energy level E_2, which is obtained by moving one of the particles into another location by a small displacement. E_2 is only accepted if the movement of the particle brings the system in a state of lower energy (i.e. $E_2 - E_1 \leq 0$). If this is not the case, a movement to a state of higher energy E_2 (also called a deterioration) is only accepted with probability $e^{-\frac{(E_2-E_1)}{kT}}$, where k is the Boltzmann constant and T is the temperature [1]. However, the probability of accepting these deteriorations descends slowly towards zero as this process continues. Thus, the repetition of this process for a large enough number of particle movements using this stochastic acceptance criterion for deteriorations make it possible to overcome becoming stuck at local minima and leads to a (near) global minima [1].

In 1983, Kirkpatrick, Gellatt and Vecchi applied the physical annealing of metal with combinatorial minimization by replacing energy with a cost function and the movement of the particles in

*All authors contributed equally to this research.

the physical system became analogous to a trial in the combinatorial minimization problem [1]. This algorithm proves to have many benefits when applied to combinatorial minimization problems including guarantee of convergence to global minimum, generally applicable to the cost function and easy to implement with good performance [1].

The approach of the simulated annealing algorithm is to generate homogeneous Markov chains of finite length at a finite sequence of descending values of the control parameter [1]. The cooling schedule consists of a set of parameters that controls the convergence of the algorithm, as listed below:

- initial value of the control parameter c
- a decrement function for decreasing the value of the control parameter c
- a stop criterion
- a finite length, L, of each Markov chain [1]

The algorithm for Simulated Annealing is algorithm 2. It makes use of 3 sub-procedures **(1)** algorithm 3 - `gen_point_a`, **(2)** algorithm 4 - `gen_point_b`, and **(3)** algorithm 1 - `init_schedule`. The following explain each part and their associated parameters and in more detail.

3.1 Initial value of the control parameter (c_0)

The basic assumption regarding the initial value of the control parameter is that c_0 should be sufficiently large such that approximately all transitions are accepted at this value. If we let $\chi(c)$ denote the ratio between the number of accepted transitions and number of accepted transitions along the Markov chain at c. The problem of determining c_0 can be posed in terms of requiring the initial acceptance ratio $\chi_0 = \chi(c_0)$ to be close to 1.

The authors of [1] propose that c_0 be determined by the following scheme: The objective function $f(\boldsymbol{x})$ is sampled m times with $\boldsymbol{x} \sim$ Uniform $(\mathcal{S})$. Let $\Delta f = f(\boldsymbol{y}) - f(\boldsymbol{x})$, and $\Delta f^+ = \{\Delta f : \Delta f > 0\}$. Let m_1 denote the number of accepted transitions, with $m = m_1 + m_2$, and let $\overline{\Delta f^+}$ be the average values of Δf_{xy} for which $\Delta f_{xy} > 0$

$$c_0 = \overline{\Delta f^+} \left(\ln \frac{m_2}{m_2 \chi_0 + (1 - \chi_0) m_1} \right)^{-1} \tag{3.1}$$

In practice, eq. (3.1) was found to be problematic. The logarithm can result in a negative c_0 which does not make sense. Furthermore, if m is not sufficiently large, c_0 exhibits very high variance. This occurs because the mean in $\overline{\Delta f^+}$ is very sensitive to extreme values. Quick examination of the histograms of Δf^+ of the Rosenbrock test function suggested that Δf^+ is heavy-tailed.

Instead let's suppose we want the χ_0-th quantile transition to have a high ($p = 0.9$) probability of being accepted. Section 3.1 shows that $\exp -\Delta f / c$ maps Δf onto a probability. If we let $k = \chi$-th quantile of the Δf^+ values, we can determine c_0 with

$$\exp -\frac{k}{c_0} \geq p \implies c_0 = -\frac{k}{\log p} \tag{3.2}$$

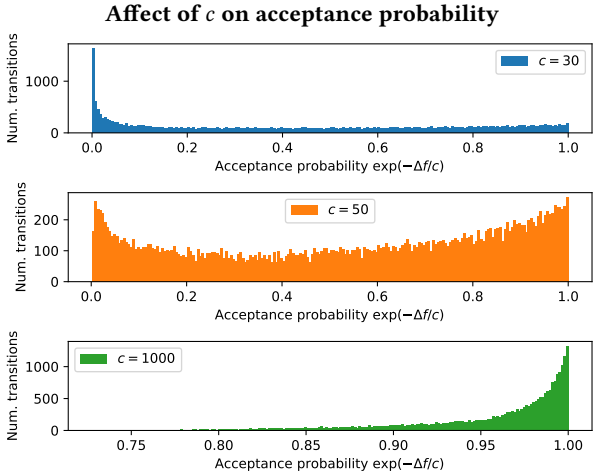


Figure 3.1: Using the Rosenbrock function, 50,000 transitions were computed. Among those, the positive transitions were selected and the exponential function $exp(-\Delta f/c)$ was applied and used as the acceptance criterion. Using three different c values of 30, 50 and 1000, it is shown that a greater value of c leads to more transitions given a higher probability of being accepted.

Algorithm 1: Init. schedule - `init_schedule`

Input: $f : \mathcal{S} \longrightarrow \mathbb{R}, \mathcal{S}, \chi, m$

$\boldsymbol{x}_0 \longleftarrow$ gen_point_a($\mathcal{S}$)

for $i \in \{1, \ldots m-1\}$ **do**

  $\boldsymbol{x}_i \longleftarrow$ gen_point_b($\boldsymbol{x}_{i-1}$)

$\Delta_i = f(\boldsymbol{x}_{i+1}) - f(\boldsymbol{x}_i)$ for $i = 0, \ldots m-1$

$\Delta^+ = \{\Delta_i : \Delta_i > 0\}$

cutoff $\longleftarrow$ χ-quantile(Δ^+)

$C_0 \longleftarrow -$ cutoff/log(χ)

return C_0

3.2 Decrement of the control parameter

The decrement function is used to decrease the value of c The new value of c, c' is calculated by:

$$c' = c \left(1 + \frac{c \ln(1+\delta)}{3\sigma(c)} \right)^{-1} \tag{3.3}$$

where $\sigma(c)$ is the standard deviation of the values of the cost function of the points of the Markov chain at c and the constant δ is the distance parameter that determines the speed of the decrement of the control parameter c [1].

Particular care must be given towards the $\sigma(c)$ term in the denominator of eq. (3.3). Especially when c is small, towards the later stages of the annealing schedule, a Markov chain at the current

c can fail to make any transitions. In such situations, $\sigma(c) = 0$, and we have both a divide-by-zero exception, and c can be set to 0 prematurely, which has the result of triggering the stop condition, before a good solution is reached.

One way to address this issue is to repeatedly sample a Markov chain until some minimum number of transitions are performed. Due to descent direction component of point generation alternative B, the Markov chain will usually make at least one transition, even if it is very small, thus avoiding the divide-by-zero problem.

Figure 3.2: Cooling schedule for Branin

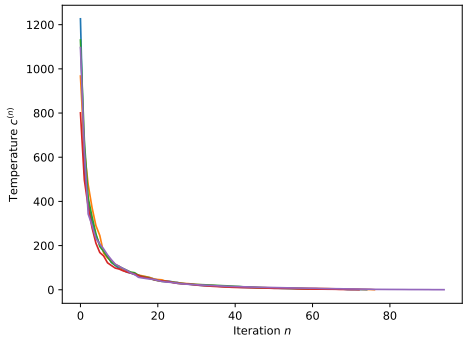


Figure 3.3: Cooling schedule for Goldstein-Price

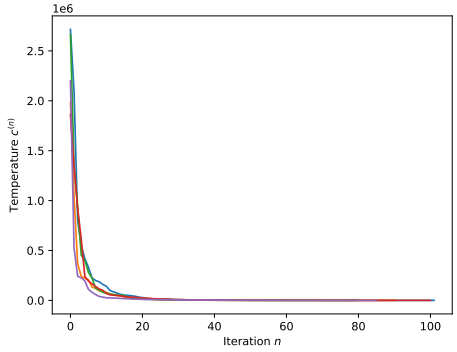


Figure 2 and Figure 3 represent an example Cooling Schedule for Branin and Golden-Price respectively, where each colour on the curve represents a different run, for 4 total runs.

3.3 Stop condition

Particular care needs to be given to devising an appropriate stopping condition. First, let $\overline{f}(c_0)$ denote the mean value of the points in the initial Markov chain, and $\overline{f}(c)$ denote the average values of the points in the chain at c. Then $\overline{f_s}(c)$ is the smoothed value of $\overline{f}$ over a number of chains in order to reduce fluctuations of $\overline{f}(c)$. In practice the smoothing is applied using a small smoothing parameter $0 < \gamma < 1$ and

$$\begin{aligned} \overline{f_s}^{(n+1)} &= \gamma \overline{f}^{(n+1)} + (1-\gamma)\overline{f_s}^{(n)} \\ \overline{f_s}^{(0)} &= \overline{f}^{(0)} \end{aligned} \tag{3.4}$$

The algorithm is terminated if:

$$\left| \frac{d\overline{f_s}(c)}{dc} \frac{c}{\overline{f}(c_0)} \right| < \epsilon_s \tag{3.5}$$

where ϵ_s is the stop parameter, a small positive real number. The derivative in eq. (3.5) is simply calculated using a finite difference method [1].

3.4 Length of the Markov chains

Let n be the dimension of S and let L_0 be a constant called the standard length. Then

$$L = L_0 \cdot n \tag{3.6}$$

is the length of the Markov chain, which must be sufficiently large in order to enable the algorithm to explore the neighbourhood of a given point in all directions [1].

3.5 Generation of Points

There are several ways to generate new points from a given point and two alternatives are used for the purposes of this project, Alternative A and Alternative B:

(A) A uniform distributions on S:

$$g_{xy} = \frac{1}{m(S)} \tag{3.7}$$

Refer to algorithm 3 for details. A disadvantage about Alternative A is that no structural information about function values is used [1].

(B) Either a point is drawn from a uniform distribution over S or a step is made into a descent direction from the current point:

$$g_{xy} = \begin{cases} LS(x) & \text{if } w > t \\ \dfrac{1}{m(S)} & \text{if } w \leq t \end{cases} \tag{3.8}$$

where t is a fixed number in the interval [0,1) and w is a random number from U[0,1). $\mathrm{LS}(x)$ is a Local Search procedure (i.e Steepest Descent Method) that generates a point $\boldsymbol{y}$ in a descent direction of $\boldsymbol{x}$. [1] Refer to algorithm 4 for details.

4 THEORETICAL RESULTS

The mathematical model of the simulated annealing algorithm is based on the theory of Markov chains and thus, the following definitions are necessary:

Definition 4.1 (Markov Chains). A *Markov chain* in the simulated annealing algorithm is a sequence of trials. $X(k)$ is a random variable denoting the kth trial by simulated annealing, and the outcome of a trial is a point $x \in S$ that is dependent on the previous trial.

Algorithm 2: Simulated Annealing

Input: $f : \mathcal{S} \longrightarrow \mathbb{R}, \nabla f, \mathcal{S}, l_0, \epsilon_s, \gamma$

$l \longleftarrow l_0 \cdot \texttt{dim}(\mathcal{S})$

$c_0 \longleftarrow \texttt{init\_schedule}(\mathcal{S}, \chi, m)$

$c \longleftarrow c_0$

$\boldsymbol{x} \longleftarrow \texttt{gen\_point\_a}(\mathcal{S})$

f_x $\longleftarrow f(\boldsymbol{x})$

while *true* **do**

  f_at_c $\longleftarrow \{f(\boldsymbol{x})\}$

  chain $\longleftarrow$ { }

  for $i \in \{1, \ldots, l\}$ **do**

    $\boldsymbol{y} \longleftarrow \texttt{gen\_point\_b}(\boldsymbol{x}, f, \nabla f, \mathcal{S}, t)$

    f_y $\longleftarrow f(\boldsymbol{y})$

    $\Delta \longleftarrow$ f_y - f_x

    ▷ Check acceptance criteria

    if $\Delta \leq 0$ *or* $\exp(-\Delta/c) > \texttt{random}[0, 1)$ **then**

      $\boldsymbol{x} \longleftarrow \boldsymbol{y}$ ▷ accept new point

      chain.push($\boldsymbol{y}$)

      f_at_c.push(f_y)

      f_x $\longleftarrow$ f_y

  $\overline{f} \longleftarrow \texttt{mean}(f\_at\_c)$

  $\sigma \longleftarrow \texttt{std}(f\_at\_c)$

  if $\sigma = 0$ **then**

    $\sigma \longleftarrow 10^{-14}$

  if $n = 0$ **then**

    $\overline{f}_s \longleftarrow \overline{f}$

    $\overline{f}_0 \longleftarrow \overline{f}$

  else

    $\overline{f}_s^{(0)} \longleftarrow \overline{f}$

    $\overline{f}_s = (1 - \gamma)\overline{f}_s + \gamma\overline{f}$

  $c_{n-1} \longleftarrow = c$

  $c \longleftarrow c\left(1 + \log\frac{1+\delta}{3\sigma}\right)^{-1}$ ▷ decrement c

  dc $\longleftarrow c - c_{n-1}$

  if $n > 0$ *and* $dc \neq 0$ **then**

    $d\overline{f}_s \longleftarrow \overline{f}_s - \overline{f}_s^{(0)}$

    if $\left|\frac{d\overline{f}_s}{dc}\frac{c}{\overline{f}_0}\right| < \epsilon_s$ **then**

      return $\boldsymbol{x}^\star = \boldsymbol{x}, f(\boldsymbol{x}^\star)$ ▷ check stop cond.

  $n \longleftarrow n + 1$

Algorithm 3: Generate point alternative A - gen_point_a

Input: $\mathcal{S}$

$\boldsymbol{x} \longleftarrow$ sample(Uniform($\mathcal{S}$))

return $\boldsymbol{x}$

Algorithm 4: Generate point alternative B - gen_point_b

Input: $f : \mathcal{S} \longrightarrow \mathbb{R}, \nabla f, \mathcal{S}, t$

$w \longleftarrow$ random($[0, 1)$)

if $w > t$ **then**

  $\boldsymbol{x} \longleftarrow$ sample(Uniform($\mathcal{S}$))

else

  $\boldsymbol{x} \longleftarrow \texttt{grad\_descent}(f, \nabla f, \boldsymbol{x_0}, N = 1)$

return $\boldsymbol{x}$

Definition 4.2 (The Generation Probability Distribution). g_{xy}, as seen in **Section 3.5**, is the *generation probability distribution*. That is, g_{xy} is the probability distribution function for generating a point y from point x at a fixed value of the control parameter c.

Definition 4.3 (Acceptance Probability). A_{xy} is the *acceptance probability*. That is, A_{xy} is the probability of accepting point y as a possible new point if x is the current point in a Markov chain.

Definition 4.4 (Transition Probability). A point $x \in S$ transformed to a point $y \in T \subset S$, with the probability of generating and accepting a point in T given that $x \notin T$ is called a *transition probability*. Hence, for any current point x in the Markov Chain, then the probability that an element in T is the next point of the chain is given as:

$$P(T \mid x;\, c) = \begin{cases} \displaystyle\int_{y \in T} p_{xy}(c)\, \mathrm{d}y & \text{for } x \notin T \\ \displaystyle\int_{y \in T} p_{xy}(c)\, \mathrm{d}y + \left(1 - \int_{y \in S} p_{xy}(c)\, \mathrm{d}y\right) & \text{for } x \in T \end{cases} \tag{4.1}$$

where

$$p_{xy}(c) = g_{xy} \cdot A_{xy}(c) \tag{4.2}$$

and

$$P(T \mid x;\, c) = \mathbb{P}\{X(l) \in T \mid X(l-1) = x\,;\, c\} \tag{4.3}$$

4.1 Asymptotic Convergence of the Algorithm

Before asymptotic convergence of the algorithm is proved, the following definitions are necessary:

Definition 4.5 (Stationary Probability Distribution Function). A probability distribution function $r(x, c)$ is said to be *stationary* if

the following two conditions are met:

$$(1)\ \forall_{x\in S} : r(x,c) = \int_{y\in S} r(y,c)p_{yx}(c)\mathrm{d}y + r(x,c)\left(1 - \int_{y\in S} p_{xy}(c)\,\mathrm{d}y\right) \tag{4.4}$$

$$(2)\ \int_{x\in S} r(x,c)\mathrm{d}x = 1 \tag{4.5}$$

From this definition, the following is a stationary probability distribution since it meets the two conditions, which will later be used to show that the simulated annealing algorithm converges to a near minimal solution:

$$q(x,c) = \exp\left(-\frac{f(x)-f_{\min}}{c}\right)\left[\int_{y\in S}\exp\left(-\frac{f(y)-f_{\min}}{c}\right)\mathrm{d}y\right]^{-1} \tag{4.6}$$

Definition 4.6 (Probability of Transformation). The probability that a point $x \in S$ is transformed into a point $y \in T \subset S$ in k trials is:

$$P^{(k)}(T \mid x;\, c) = \begin{cases} \displaystyle\int_{y\in T} p_{xy}^{(k)}(c)\,\mathrm{d}y & \text{for } x \notin T \\ \displaystyle\int_{y\in T} p_{xy}^{(k)}(c)\,\mathrm{d}y + \left(1 - \int_{y\in S} p_{xy}(c)\,\mathrm{d}y\right)^{k} & \text{for } x \in T \end{cases} \tag{4.7}$$

where

$$p_{xy}^{(k)}(c) = \int_{z\in S} p_{xz}^{(k-1)}(c)p_{zy}(c)\,\mathrm{d}z + p_{xz}^{(k-1)}(c) + \left(1 - \int_{z\in S} p_{yz}(c)\,\mathrm{d}z\right) + \left(1 - \int_{z\in S} p_{xz}(c)\,\mathrm{d}z\right)^{k-1} p_{xy}(c) \tag{4.8}$$

i.e. $p_{xy}^{(k)}(c)$ is the quasi probability distribution function of transforming x into y in k trials. $p_{xy}^{(k)}(c)$ is the summation of three terms, which are outlined below:
(i). Term 1: the quasi probability distribution function of transforming x into z in $k-1$ trials, and from z to y in the next trial integrated over all z.
(ii). Term 2: the quasi probability distribution function of transforming x into y in $k-1$ trials and then rejecting the kth trial.
(iii). Term 3: the quasi probability distribution function of transforming x into y in one trial after $k-1$ rejected trials from x.

In **Eq. (4.6)**, we present a stationary probability distribution function, which is the necessary requirement for the Simulated Annealing algorithm to converge to the minimum solution. We present the theorem that:

Theorem 4.7. If there is a finite number of local minima for a uniformly continuous function f, then:

$$\forall\epsilon > 0 \ :\ \lim_{c\to 0}\int_{y\in B_f(\epsilon)} q(y,c)\,\mathrm{d}y > 1-\epsilon \tag{4.9}$$

Proof. For a finite number of local minima, we have:

$$\exists\epsilon_1 > 0 \ :\ |f(x_{\text{loc}}) - f_{\min}| > \epsilon_1 \tag{4.10}$$

$$\exists\epsilon_2 > 0,\ \forall x_{\min} \ :\ ||x_{\text{loc}} - x_{\min}|| > \epsilon_2 \tag{4.11}$$

where $f_{\min} = f(x_{\min})$, $\forall x_{\min}$ as per **Eq. (2.1)** and x_{loc} is a local, non-global minimum point. Then pick a positive ϵ such that:

$$\epsilon < \frac{1}{4}\min\{\epsilon_1, \epsilon_2\} \tag{4.12}$$

It should be noted that if all minima are global, then select ϵ such that $\exists x \in S : f(x) - f_{\min} > \epsilon$.

As f is uniformly continuous, then:

$$\exists\delta_1 > 0,\ \forall x, y \in S : ||x-y|| \le \delta_1 \implies |f(x) - f(y)| < \frac{\epsilon}{2} \tag{4.13}$$

Choosing a δ such that $\delta = \min\{\delta_1/2, \epsilon\}$, we have:

$$\forall y \in B_x(\delta) : f(y) - f_{\min} < \frac{\epsilon}{2} \tag{4.14}$$

where $B_x(\delta)$ is given by **Definition 1.2**.

Consider a point $x_0 \in S \setminus B_x(\delta)$ where $f(x_0) - f_{\min} = \epsilon$. Note that this is possible due to the continuity of f. Then therefore:

$$\begin{aligned}
\lim_{c\to 0} q(x_0,c) &= \lim_{c\to 0}\frac{\exp(-(f(x_0)-f_{\min}/c))}{\int_{y\in S}\exp(-(f(y)-f_{\min})/c)}\,\mathrm{d}y \\
&= \lim_{c\to 0}\frac{\exp(-\epsilon/c)}{\int_{y\in S}\exp(-(f(y)-f_{\min})/c)}\,\mathrm{d}y \\
&= \lim_{c\to 0}\left[\int_{y\in S}\exp((\epsilon-(f(y)-f_{\min}))/c)\,\mathrm{d}y\right]^{-1} \\
&= \lim_{c\to 0}\left[\int_{y\in S\setminus B_x(\delta)}\exp((\epsilon-(f(y)-f_{\min}))/c)\,\mathrm{d}y \right.\\
&\qquad \left. + \int_{y\in B_x(\delta)}\exp((\epsilon-(f(y)-f_{\min}))/c)\,\mathrm{d}y\right]^{-1} \\
&\le \lim_{c\to 0}\left[\int_{y\in B_x(\delta)}\exp((\epsilon-(f(y)-f_{\min}))/c)\,\mathrm{d}y\right]^{-1} \\
&\le \left[\lim_{c\to 0}\int_{y\in B_x(\delta)}\exp((\epsilon-\epsilon/2)/c)\,\mathrm{d}y\right]^{-1} \\
&= \left[\lim_{c\to 0}\exp(\epsilon/2c)m(B_x(\delta))\right]^{-1}
\end{aligned} \tag{4.15}$$

and can clearly be seen that this approaches to 0 as $c \to 0$. And so with the Lebesgue measure of S, $m(S)$, then we have:

$$\exists c_0 > 0,\ \forall c < c_0 \ :\ q(x_0,c) < \frac{\epsilon}{m(S)} \tag{4.16}$$

And hence:

$$\forall c < c_0,\ \forall x \in S^{+}(x) \ :\ q(x,c) \le q(x_0,c) < \frac{\epsilon}{m(S)} \tag{4.17}$$

and that:

$$\forall c < c_0,\ \forall x \in S^{-}(x) \ :\ f(x) - f_{\min} < \epsilon \tag{4.18}$$

with $S^+(x)$ and $S^-(x)$ defined as follows:

$$S^+(x) = \{y \in S \,|\, f(y) \leq f(x)\} \tag{4.19}$$
$$S^-(x) = \{y \in S \,|\, f(y) > f(x)\} \tag{4.20}$$

$\forall c < c_0$, then we have:

$$\begin{aligned} 1 &= \int_{y\in S} q(y,c)\,\mathrm{d}y \\ &= \int_{y\in S^-(x_0)} q(y,c)\,\mathrm{d}y + \int_{y\in S^+(x_0)} q(y,c)\,\mathrm{d}y \\ &< \int_{y\in B_f(\epsilon)} q(y,c)\,\mathrm{d}y + \int_{y\in S^+(x_0)} \frac{\epsilon}{m(S)}\,\mathrm{d}y \\ &\leq \int_{y\in B_f(\epsilon)} q(y,c)\,\mathrm{d}y + \epsilon \end{aligned} \tag{4.21}$$

Know that $B_f(\epsilon) = S^-(x_0)$. Because of **Eq. (4.11)** and **Eq. (4.12)**, then there is no local minimum in $B_f(\epsilon)$. Thus showing that

$$\lim_{c\to 0} \int_{y\in B_f(\epsilon)} q(y,c)\,\mathrm{d}y > 1 - \epsilon \tag{4.22}$$

proving the **Theorem 4.7** as desired. □

5 NUMERICAL RESULTS

5.1 Examples of running Simulated Annealing

Table 5.1: Simulated Annealing with BR test to $\tau = 10^{-7}$

	$x^\star$	$f(x^\star)$	#f evals	#∇f evals
1	(9.42478, 2.47500)	0.39789	448	166
2	(9.42478, 2.47500)	0.39789	607	224
3	(9.42478, 2.47500)	0.39789	688	232
4	(-3.14159, 12.27500)	0.39789	567	167
5	(-3.14159, 12.27500)	0.39789	648	195
6	(-3.14159, 12.27500)	0.39789	568	199
7	(3.14159, 2.27500)	0.39789	529	183
8	(3.14159, 2.27500)	0.39789	486	172
9	(3.14159, 2.27500)	0.39789	528	204
10	(-3.14159, 12.27500)	0.39789	527	190
11	(3.14159, 2.27500)	0.39789	607	195
12	(-3.14159, 12.27500)	0.39789	568	201
13	(-3.14159, 12.27500)	0.39789	489	179
14	(-3.14159, 12.27500)	0.39789	608	203
15	(9.42478, 2.47500)	0.39789	728	228
16	(3.14159, 2.27500)	0.39789	528	179
17	(9.42478, 2.47500)	0.39789	728	235
18	(3.14159, 2.27500)	0.39789	567	201
19	(3.14159, 2.27500)	0.39789	649	225
20	(3.14159, 2.27500)	0.39789	767	244

Table 5.1: note that all 3 global minima are found. The following parameters were used and serve as good default paramters:
$L_0 = 20, \delta = 1.1, \epsilon_s = 10^{-6}, \chi = 0.9, \gamma = 10^{-2}$ and $T = 0.5$.

Figure 5.1: Average SA trajectories for BR

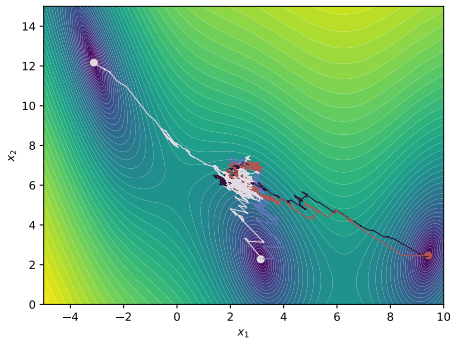


Figure 5.2: Average SA trajectories for GP

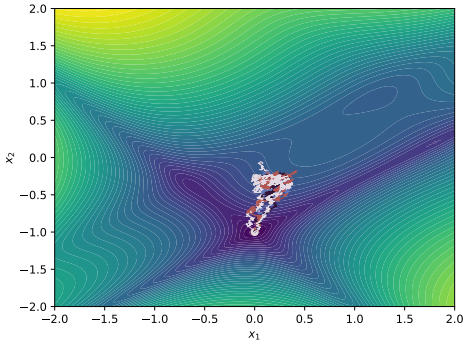


Table 5.2: Average number of runs required to solve problems

	Problem	Avg. num runs	Solutions found	#f evals	#∇f evals
1	RB	1.0	1	614.58	210.68
1	GP	1.6	1	1072.38	385.96
3	BR	5.18	3	3012.68	1041.06
1	H3	1.12	1	575.08	211.06
1	H6	1.58	1	737.7	284.1

Table 5.2 shows average required number of runs for the benchmark problems RB, GP, BR, H3, and H6, Simulated Annealing is run repeatedly until all known global minima are found for the problem. A solution $\hat{x}_i$ is considered the same if $|\hat{x}_i - x_j^\star|_2 \leq \tau = 10^{-4}$. This process is repeated $M = 50$ times and average number of runs, average function and Jacobian values are reported. In all cases, every global minimum was found. Test problems BR and H6 show an average higher number of required runs per global minima.

The following Simulated Annealing parameters were used: $l_0 = 20, \delta = 1.1, \epsilon_s = 1e-4, \chi = 0.9, \gamma = 10^{-2}$, and $t = 0.5$.

Table 5.3: Global and local minima found by Simulated Annealing

Prob	Type	$x_i^\star$	$f(x_i^\star)$
RB	g	(1.0, 1.0)	0.0
GP	g	(-0.0, -1.0)	3.0
	l	(-0.6, -0.4)	30.0
		(1.8, 0.2)	84.0
BR	g	(-3.14159, 12.275)	0.39789
		(3.14159, 2.275)	0.39789
		(9.42478, 2.475)	0.39789
H3	g	(0.11461, 0.55565, 0.85255)	-3.86278
	l	(0.10934, 0.86052, 0.56412)	-3.08976
H6	g	(0.20169, 0.15001, 0.47687, 0.27533, 0.31165, 0.6573)	-3.32237
	l	(0.40465, 0.88244, 0.8461, 0.57399, 0.13893, 0.0385)	-3.20316

6 NUMERICAL COMPARISONS

6.1 MS (Multi-start)

Multi-start techniques form a family of heuristic global optimization algorithms. These techniques work in a two-phase process. In the first global phase, starting points are sampled in the feasible region or domain $\mathcal{S}$ of the objective function. The most naive implementation of the global phase is to simply perform a uniform sampling of $\mathcal{S}$. Some members of this family of algorithms utilize sophisticated heuristics to generate starting points such as the Scatter Search technique.

The local phase of multi-start algorithms use the starting points from the global phase to perform a local optimization using one of the many algorithms for local optimization. Multi-start algorithms alternate between the global and local phases until a solution is found [5].

The hardest part of divising an efficient Multi-start method is the selection of an appropriate stopping rule. Variations in the literature range from several addhoc holes to Bayesian stoppic rules. For the sake of comparison we utilize a double-box stopping rule [cite box-cstr here].

Let $m(\mathcal{S})$ be the Lebesque measure of the search region $\mathcal{S}$. Since the local search method is deterministic, in the limit of $n \longrightarrow \infty$ runs of the local search method, the algorithm will converge to w unique local minima. Each local minimum $x_i^\star \in \mathcal{S}$ has an associated region of attraction A_i defined as

$$A_i := \left\{ x : x \in \mathcal{S}, \mathtt{LS}(x) = x_i^\star \right\}.$$

Since $\mathcal{S}$ contains w local minima, and the $A_i \cap A_j = \emptyset$, A_i partitions $\mathcal{S}$ i.e.,

$$\bigcup_{i=1}^{w} A_i = \mathcal{S}.$$

Furthermore, if $m(A_i)$ denotes the Lebesque measure of A_i, then

$$m(\mathcal{S}) = \sum_{i=1}^{w} m(A_i).$$

If some initial point $x_0 \sim \text{Uniform}(\mathcal{S})$, then the probability of x_0 begin contained in A_i is simply

$$P(x_0 \in A_i) = \frac{m(A_i)}{m(\mathcal{S})}.$$

The double-box stopping rule is based on the idea that we want to ensure that all A_i are sampled in $\mathcal{S}$, but additional sampling results in uncessary computation. Define C has a relative measure of the coverage after the discovery of w local minima, as

$$C = \sum_{i=1}^{w} \frac{m(\mathcal{A}_i)}{m(\mathcal{S})}. \tag{6.1}$$

A sensible heuristic is to stop when $C \longrightarrow 1$. The quantity inside the summation of eq. (6.1) is not calculatable in practice, but as $w \to \infty$, we approximate it with

$$C \approx \sum_{i=1}^{w} \frac{L_i}{L}, \tag{6.2}$$

where L_i is the number of starting points of the LS which converged to the local minimizer x_i, and L is the total of initial points so far. The quantity in eq. (6.2) is equal to 1 by definition, so another means must be determined. Construct a region $\mathcal{S}_2$ such that $\mathcal{S} \subset \mathcal{S}_2$ and $m(\mathcal{S}_2) = 2 \cdot m(\mathcal{S})$. For each iteration, we sample from $\mathcal{S}_2$ until we have a point in $\mathcal{S}$, in other words points in $A_0 = \mathcal{S}_2 \setminus \mathcal{S}$ are discared. Also let L_0 denote the number of sampled points in A_0. The total count of sampled points is now given by

$$L = L_0 + \sum_{i=1}^{w} L_i,$$

and the relative coverage C is now

$$C = \frac{1}{m(\mathcal{S})} \sum_{i=1}^{w} m(A_i) = 2 \sum_{i=1}^{w} \frac{m(A_i)}{m(\mathcal{S}_2)}$$

and finally we can approximate relative coverage with

$$C \approx \frac{2}{L} \sum_{i=1}^{w} L_i$$

After n iterations, let M_n denote the number of points in $\mathcal{S}_2$, and n are in $\mathcal{S}$. Then define $\delta_n := n/M_k$ which has an expectation which in the limit of large n

$$\langle \delta \rangle_n = \frac{1}{n} \sum_{i=1}^{n} \delta_i \longrightarrow \frac{m(\mathcal{S})}{m(\mathcal{S}_2)} = \frac{1}{2}$$

The variance is $\sigma_n^2(\delta) = \left\langle \delta^2 \right\rangle_n - \langle \delta \rangle_n^2$ which $\longrightarrow 0$ as $n \longrightarrow \infty$. Finally, the double-box rule is

Double-box stopping rule:

(1) Continue iterating if new minima are found. **(2)** If no new minima are found, let $\sigma_{\text{last}}(\delta)$ be the s.t.d. at the last iteration at which a minimum was found. Contnue iterating while

$$\sigma^2(\delta) < \rho \sigma_{\text{last}}^2(\delta) \tag{6.3}$$

where $\rho \in (0,1)$ is a paramter that performs exhaustive search when ρ is close to 0 and emphasises less iterations when ρ is close to 1.

Constructing $\mathcal{S}_2$ from $\mathcal{S}$:

In order to apply the double-box stopping rule, we need to construct the box region $\mathcal{S}_2$ such that $m(\mathcal{S}_2) = 2 \cdot m(\mathcal{S})$. Suppose $\mathcal{S} \in \mathbb{R}^n$, then we can scale the bounds of $\mathcal{S} = [l_1, u_1] \times \cdots [l_n, u_n]$ by $2^{1/n}$, i.e.,

$$l_i^{(2)} = l_i - \frac{1}{2}\left(2^{1/n} - 1\right)(u_i - l_i) \text{ for } i = 1, \dots n$$

$$u_i^{(2)} = u_i + \frac{1}{2}\left(2^{1/n} - 1\right)(u_i - l_i) \text{ for } i = 1, \dots n$$

6.2 DE (Differential Evolution)

Differential Evolution is global, gradient-free stochastic optimization algorithm. This population-based method mutates each solution candidate by mixing the solution with other members of the population [4]. We use `scipy.optim.differential_evolution` for comparison.

6.3 BH (Basin Hopping)

Basin-hopping is another two-phase global optimization algorithm inspired by energy minimization of clusters of atoms [6]. We use `scipy.optimize.basinhopping` for comparison.

6.4 Evaluation metrics

In section 6.4.1 and section 6.4.2, we consider a global minimum to have been found once $\left|\hat{\boldsymbol{x}}_j - \boldsymbol{x}_i^\star\right|_2 \leq \tau = 10^{-4}$. Over a total of M trials, let k_i denote the number of runs to find all w global minima on the i-th trial.

6.4.1 Metric 1: Number of runs required to find all global minima. One method to compare various global optimization methods is to calculate the number of runs required to find all global minima for a given benchmark, averaged over M runs, divided by the number of minima w. We can define metric ξ_1 with eq. (6.4).

$$\xi_1 = \frac{1}{wM}\sum_{i=1}^{M} k_i \tag{6.4}$$

6.4.2 Metric 2: Number of function evaluations to find all global minima. Different optimization methods use evaluations of f and ∇f in very different ways. Methods that utilize a local search will incur large numbers of calls to these functions inside of their inner loops. Examining the average number of these evaluations to find all global minima of a given benchmark function is a good way to compare the computational cost of various methods. Similiar to section 6.4.1, we average the number of required calls over M trials. Furtheremore, to make comparisons accross different benchmakrs, divide by the number of minima w. Suppose it takes k_i runs to find all w global minima, on the i-th trial, where $i \in \{1, \dots, M\}$. We define metric ξ_2 with eq. (6.5).

Algorithm 5: Multi-start

Input: $f : \mathcal{S} \longleftarrow \mathbb{R}, \nabla f, \mathcal{S}, \tau, \rho, \epsilon_s, N$

$\mathcal{S}_2 \longleftarrow$ `scale`$(\mathcal{S}, 2)$

$M_n \longleftarrow 0$

x_minima $\longleftarrow$ { }

deltas $\longleftarrow$ { }

f_best $\longleftarrow \infty$

for $n \in \{1, \dots, N\}$ **do**

  while *true* **do**

    $\boldsymbol{x} \longleftarrow$ `sample`$(\mathcal{S}_2)$

    $M_n \longleftarrow M_n + 1$

    if $\boldsymbol{x} \in \mathcal{S}$ **then**

      break

  $\boldsymbol{x}_{ls} \longleftarrow$ `LS`$(f, \boldsymbol{x}, \nabla f)$

  if $\boldsymbol{x}_{ls} \notin \mathcal{S}$ **then**

    continue

  $\Delta = n/M_n$

  deltas.`push`(δ)

  f_ls $\longleftarrow f(\boldsymbol{x}_{ls})$

  $\sigma^2 \longleftarrow$ `var`(deltas)

  if *f_ls < f_best* **then**

    f_best = f_ls

    $\boldsymbol{x}^\star \longleftarrow \boldsymbol{x}_{ls}$

  if $\boldsymbol{x}^\star \notin$ *x_minima* **then**

    $\sigma^2_{prev} \longleftarrow \sigma^2$

    x_minima.`push`$(\boldsymbol{x}_{ls})$

  else

    ▷ `Check double-box stop cond.`

    if $\sigma^2 < \rho \cdot \sigma^2_{prev}$ **then**

      return $\boldsymbol{x}^\star$

Output: 'failed to reach tolerance in N iterations'

return $\boldsymbol{x}^\star$

$$\xi_2 = \frac{1}{wM}\sum_{i=1}^{M}\sum_{j=1}^{k_i}\left(n(f_{ij}) + n(\nabla_{ij})\right), \tag{6.5}$$

where $n(f_{ij})$ and $n(\nabla_{ij})$ denotes the number of evaluations of the function f and its Jacobian respectively, incurred during the j-th run of the i-th trial.

Table 6.1: Finding all minima

Benchmark f	Method	$\overline{n(f_{ij})}$	$\overline{n(\nabla_{ij})}$	Δt (ms)	ξ_1	ξ_2
RB	BH	6590.6	2045.5	458	1.0	4318.0
	DE	3895.5	0.0	232	1.0	1948.0
	MS	1031.4	981.5	90	1.0	1006.0
	SA	631.8	220.0	15	1.0	426.0
GP	BH	19532.9	6181.6	1279	1.0	12857.0
	DE	966.4	0.0	63	1.0	483.0
	MS	608.2	575.2	73	1.0	592.0
	SA	1056.3	378.4	48	1.6	717.0
BR	BH	20369.5	6665.0	1422	1.8	4506.0
	DE	9163.4	0.0	569	3.1	1527.0
	MS	1437.3	1293.0	139	2.02	455.0
	SA	2761.4	946.5	77	1.57	618.0
H3	BH	8817.8	2177.6	580	1.0	5498.0
	DE	22736.6	0.0	1625	16.15	11368.0
	MS	444.9	418.3	61	1.0	432.0
	SA	570.2	207.8	34	1.1	389.0
H6	BH	22008.6	3132.0	1267	1.0	12570.0
	DE	22513.6	0.0	1807	4.5	11257.0
	MS	876.7	845.6	215	1.0	861.0
	SA	629.1	252.2	72	1.45	441.0

Figure 6.1: Finding all minima

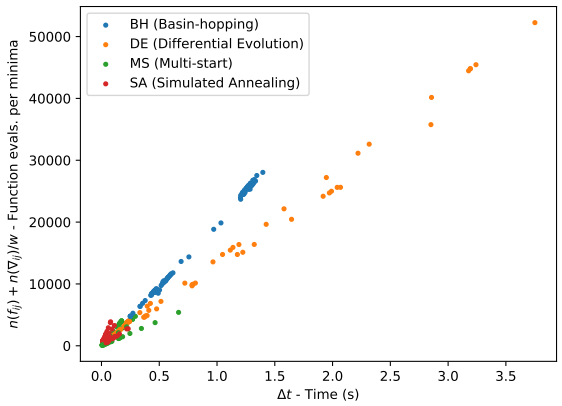
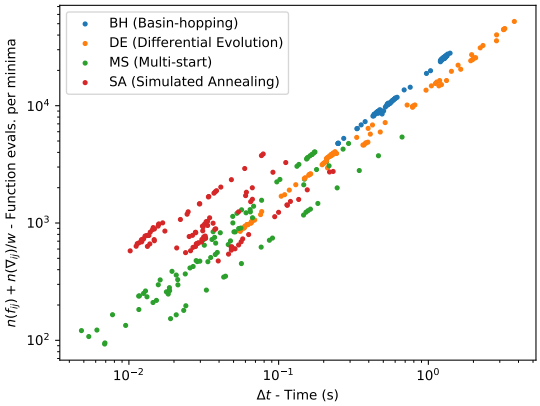


Table 6.1 shows that, in terms of average running time $\overline{\Delta t}$, our implementation of Simulated Annealing outperformed all other methods on every benchmark. Multi-start is arguably the second best performing method. Figure 6.1 shows that SA has generally faster running time. Mutli-start might use less function evaluations.

One aspect of the robustness of a numerical method, is that there will not be large unexpected running times. In this aspect, Simulated Annealing is significantly better than the other methods that were evaluated. Function evaluation/ running time points are tightly clustered towards the origin in fig. 6.1 in comparison with the other methods. In comparison, Differential Evolution, on some

Figure 6.2: Finding all minima (log scale)



instances, produced extreme outliers. Even the Multi-start method does not show such a tight clustering.

7 CONCLUSIONS

The work of Dekkers and Aarts have placed Simulated Annealing on a firm rigorous framework that enables further work and understanding of stochastic global optimization methods.

7.1 A robust and efficient method

Our emperical results, achieved with a non-optimized, and rather crude implementation, show that Simlulated Annealing is both a robust and efficient method. Simulated Annealing has outperformed the choosen global optimization methods provided by `scipy.optimize`, in respect to our multiple global minima test.

7.2 Further work

Simulated Annealing, Mutli-start, and the various other methods provided by `scipy.optimize` were used with default parameters. Further exploration and analysis should be done to determine optimal parameters for all methods to get a fairer comparison between the different methods.

Regretably our small selection of test problems had a only a small number of global and local minima. It would be insightful to explore test problems that have large numbers of minima. It is uncertain if our results would hold in such a regime.

Lastly, another interesting avenue of exploration would be to develop different methods of point generation for the Markov chains. Does there exist a way to sample points directly from a probability distribution using function information? This would avoid the costly and wasteful discarding or a large number of points.

REFERENCES

[1] Anton Dekkers and Emile Aarts. 1991. Global Optimization and Simulated Annealing. *Mathematical Programming* 50 (1991), 367–393.
[2] Arnold Neumaier. 2003. Complete Search in Continuous Global Optimization and Constraint Satisfaction.
[3] H.H. Rosenbrock. 1960. An automatic method for finding the greatest or least value of a function. *Comput. J.* 3 (1960), 175–184.

[4] R Storn and K Price. 1997. Differential Evolution-a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization* 11 (1997), 341–359.
[5] Zsolt Ugray, L. Lasdon, J. Plummer, Fred W. Glover, J. P. Kelly, and R. Martí. 2007. Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization. *INFORMS J. Comput.* 19 (2007), 328–340.
[6] D.J. Wales and J.P.K Doye. 1997. Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *Journal of Physical Chemistry* 101 (1997), 5111–5116.

A BENCHMARK FUNCTIONS USED

A.1 RB (Rosenbrock)

The Rosenbrock function is a widely used non-convex test function with a one global minimum inside of a long, narrow valley. It is defined as

$$f(x_1, x_2) := (a - x)^2 + b\left(x_2 - x_1^2\right)^2$$

The global minimum is at $\boldsymbol{x}^\star = \begin{pmatrix} a & a^2 \end{pmatrix}^T$. The typical values are $a = 1$ and $b = 100$ [3].

A.2 GP (Goldstein-Price)

The Goldstein-Price[1] test function in $\mathbb{R}^2$ is given by

$$f(x_1, x_2) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right]$$

$$\times \left[30 + (2x_1 - 3x_2)^2 \left(18 - 32x_1 + 12x_2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$$

$\mathcal{S} = \{\boldsymbol{x} : -2 \le x_i \le 2, i = 1, 2\}$. It has a global minimum at $\boldsymbol{x}^\star = \begin{pmatrix} 0 & -1 \end{pmatrix}^T$.

A.3 H3 and H6 (Hartmann's family)

$$f(\boldsymbol{x}) = -\sum_{i=1}^{m} c_i \exp\left(-\sum_{j=1}^{n} a_{ij}\left(x_i - p_{ij}\right)^2\right)$$

where for H3 in $\boldsymbol{x} \in \mathbb{R}^4$, with $m = 4$ and $n = 3$ is given by

$$A = [a_{ij}] = \begin{bmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{bmatrix},$$

$$P = [p_{ij}] = \begin{bmatrix} 0.3689 & 0.1170 & 0.2673 \\ 0.4699 & 0.4387 & 0.7470 \\ 0.1091 & 0.8732 & 0.5547 \\ 0.038150 & 0.5743 & 0.8828 \end{bmatrix}$$

where for H6 in $\boldsymbol{x} \in \mathbb{R}^4$, with $m = 4$ and $n = 6$ is given by

$$A = [a_{ij}] = \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} 1 \\ 1.2 \\ 3 \\ 3.2 \end{bmatrix},$$

$$P = [p_{ij}] = \begin{bmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{bmatrix}$$

A.4 S5, S7 and S10 (Shekel's Family)

The Shekel's family of functions, S5, S7, and S10 [1], is given as:

$$f(x) = -\sum_{i=1}^{m} \left((x - a_i)^T (x - a_i) + c_i\right)^{-1}$$

which has a dimension of $n = 4$, and $m = 5$, $m = 7$, and $m = 10$ for the S5, S7, and S10 respectively. Let $x = (x_1, x_2, ..., x_n)^T$, and $a_i = (a_{i1}, a_{i2}, ..., a_{in})^T$.

Thus for S5:

$$A = [a_{ij}] = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \end{bmatrix},$$

Then for S7:

$$A = [a_{ij}] = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \end{bmatrix},$$

Then for S10:

$$A = [a_{ij}] = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{bmatrix},$$

B DETAILED COMPARISON OF BH, DE, MS, AND SA METHODS

B.1 RB (Rosenbrock)

```
Benchmark 1 of 5: RB (Rosenbrock)
  Method BH (Basin-hopping)
  Run 1 of 5: runs=1, nfev=6281, njev=1970, time=0.4385
  Run 2 of 5: runs=1, nfev=6677, njev=2079, time=0.4605
  Run 3 of 5: runs=1, nfev=6315, njev=1971, time=0.4441
  Run 4 of 5: runs=1, nfev=6625, njev=2050, time=0.4591
  Run 5 of 5: runs=1, nfev=6689, njev=2067, time=0.4668
  ----------------------------------------
  avg_runs=1.0, avg_nfev=6517.4, avg_njev=2027.4, avg_time=0.4538
  metric_1=1.0, metric_2=4272.4

  Method DE (Differential Evolution)
  Run 1 of 5: runs=1, nfev=3873, njev=0, time=0.2291
  Run 2 of 5: runs=1, nfev=3723, njev=0, time=0.2175
  Run 3 of 5: runs=1, nfev=3753, njev=0, time=0.2215
  Run 4 of 5: runs=1, nfev=3723, njev=0, time=0.2204
  Run 5 of 5: runs=1, nfev=3693, njev=0, time=0.2182
  ----------------------------------------
  avg_runs=1.0, avg_nfev=3753.0, avg_njev=0.0, avg_time=0.2213
  metric_1=1.0, metric_2=1876.5

  Method MS (Multi-start)
  Run 1 of 5: runs=1, nfev=2045, njev=1946, time=0.2199
  Run 2 of 5: runs=1, nfev=2085, njev=1986, time=0.1833
  Run 3 of 5: runs=1, nfev=2088, njev=1989, time=0.1811
  Run 4 of 5: runs=1, nfev=2060, njev=1961, time=0.1789
  Run 5 of 5: runs=1, nfev=77, njev=74, time=0.0065
  ----------------------------------------
  avg_runs=1.0, avg_nfev=1671.0, avg_njev=1591.2, avg_time=0.1539
  metric_1=1.0, metric_2=1631.1

  Method SA (Simulated Annealing)
  Run 1 of 5: runs=1, nfev=771, njev=249, time=0.0168
  Run 2 of 5: runs=1, nfev=657, njev=227, time=0.015
  Run 3 of 5: runs=1, nfev=651, njev=218, time=0.0145
  Run 4 of 5: runs=1, nfev=540, njev=205, time=0.0131
  Run 5 of 5: runs=1, nfev=694, njev=221, time=0.0151
  ----------------------------------------
  avg_runs=1.0, avg_nfev=662.6, avg_njev=224.0, avg_time=0.0149
  metric_1=1.0, metric_2=443.3
```

B.2 GP (Goldstein-Price)

```
Benchmark 2 of 5: GP (Goldstein-Price)
  Method BH (Basin-hopping)
  Run 1 of 5: runs=1, nfev=18292, njev=5774, time=1.2044
  Run 2 of 5: runs=1, nfev=19211, njev=6091, time=1.2639
  Run 3 of 5: runs=1, nfev=18272, njev=5770, time=1.2058
  Run 4 of 5: runs=1, nfev=19670, njev=6222, time=1.2926
  Run 5 of 5: runs=1, nfev=20413, njev=6489, time=1.3348
  ----------------------------------------
  avg_runs=1.0, avg_nfev=19171.6, avg_njev=6069.2, avg_time=1.2603
  metric_1=1.0, metric_2=12620.4

  Method DE (Differential Evolution)
  Run 1 of 5: runs=1, nfev=939, njev=0, time=0.061
  Run 2 of 5: runs=1, nfev=1089, njev=0, time=0.0708
  Run 3 of 5: runs=1, nfev=909, njev=0, time=0.0588
  Run 4 of 5: runs=1, nfev=1089, njev=0, time=0.0711
  Run 5 of 5: runs=1, nfev=1059, njev=0, time=0.069
  ----------------------------------------
  avg_runs=1.0, avg_nfev=1017.0, avg_njev=0.0, avg_time=0.0661
  metric_1=1.0, metric_2=508.5

  Method MS (Multi-start)
  Run 1 of 5: runs=1, nfev=101, njev=95, time=0.013
  Run 2 of 5: runs=1, nfev=69, njev=66, time=0.0088
  Run 3 of 5: runs=1, nfev=313, njev=289, time=0.0359
  Run 4 of 5: runs=1, nfev=75, njev=72, time=0.0096
  Run 5 of 5: runs=1, nfev=685, njev=637, time=0.0816
  ----------------------------------------
  avg_runs=1.0, avg_nfev=248.6, avg_njev=231.8, avg_time=0.0298
  metric_1=1.0, metric_2=240.2

  Method SA (Simulated Annealing)
  Run 1 of 5: runs=1, nfev=816, njev=255, time=0.0345
  Run 2 of 5: runs=4, nfev=2690, njev=925, time=0.1195
  Run 3 of 5: runs=2, nfev=1402, njev=519, time=0.0654
  Run 4 of 5: runs=1, nfev=776, njev=231, time=0.0312
  Run 5 of 5: runs=1, nfev=532, njev=200, time=0.0247
  ----------------------------------------
  avg_runs=1.8, avg_nfev=1243.2, avg_njev=426.0, avg_time=0.0551
  metric_1=1.8, metric_2=834.6
```

B.3 BR (Branin)

```
Benchmark 3 of 5: BR (Branin)
  Method BH (Basin-hopping)
  Run 1 of 5: runs=37, nfev=136289, njev=44679, time=9.6403
  Run 2 of 5: runs=5, nfev=17928, njev=5892, time=1.2717
  Run 3 of 5: runs=7, nfev=29709, njev=9663, time=2.0686
  Run 4 of 5: runs=10, nfev=33036, njev=10908, time=2.3967
  Run 5 of 5: runs=11, nfev=40615, njev=13299, time=2.8692
  ----------------------------------------
  avg_runs=14.0, avg_nfev=51515.4, avg_njev=16888.2, avg_time=3.6493
  metric_1=4.67, metric_2=11400.6

  Method DE (Differential Evolution)
  Run 1 of 5: runs=23, nfev=23217, njev=0, time=1.4408
  Run 2 of 5: runs=5, nfev=5259, njev=0, time=0.3265
  Run 3 of 5: runs=4, nfev=3963, njev=0, time=0.2458
  Run 4 of 5: runs=12, nfev=12318, njev=0, time=0.7611
  Run 5 of 5: runs=7, nfev=6720, njev=0, time=0.4159
  ----------------------------------------
  avg_runs=10.2, avg_nfev=10295.4, avg_njev=0.0, avg_time=0.638
  metric_1=3.4, metric_2=1715.9

  Method MS (Multi-start)
  Run 1 of 5: runs=6, nfev=2253, njev=2025, time=0.2174
  Run 2 of 5: runs=4, nfev=190, njev=172, time=0.0185
  Run 3 of 5: runs=5, nfev=375, njev=341, time=0.0352
  Run 4 of 5: runs=3, nfev=293, njev=264, time=0.0284
  Run 5 of 5: runs=4, nfev=672, njev=601, time=0.0642
  ----------------------------------------
  avg_runs=4.4, avg_nfev=756.6, avg_njev=680.6, avg_time=0.0727
  metric_1=1.47, metric_2=239.53

  Method SA (Simulated Annealing)
  Run 1 of 5: runs=7, nfev=3972, njev=1404, time=0.1125
  Run 2 of 5: runs=5, nfev=2800, njev=990, time=0.0791
  Run 3 of 5: runs=3, nfev=1509, njev=539, time=0.0434
  Run 4 of 5: runs=3, nfev=1783, njev=596, time=0.049
  Run 5 of 5: runs=9, nfev=5074, njev=1754, time=0.1423
  ----------------------------------------
  avg_runs=5.4, avg_nfev=3027.6, avg_njev=1056.6, avg_time=0.0853
  metric_1=1.8, metric_2=680.7
```

B.4 H3 (Hartmann's family n=3)

```
Benchmark 4 of 5: H3 (Hartmann's family n=3)
  Method BH (Basin-hopping)
  Run 1 of 5: runs=1, nfev=8594, njev=2122, time=0.5755
  Run 2 of 5: runs=1, nfev=8571, njev=2128, time=0.574
  Run 3 of 5: runs=1, nfev=9231, njev=2270, time=0.6155
  Run 4 of 5: runs=1, nfev=8658, njev=2141, time=0.5785
  Run 5 of 5: runs=1, nfev=8504, njev=2100, time=0.5694
  ----------------------------------------
  avg_runs=1.0, avg_nfev=8711.6, avg_njev=2152.2, avg_time=0.5826
  metric_1=1.0, metric_2=5431.9

  Method DE (Differential Evolution)
  Run 1 of 5: runs=6, nfev=8462, njev=0, time=0.6085
  Run 2 of 5: runs=18, nfev=24777, njev=0, time=1.7846
  Run 3 of 5: runs=49, nfev=68519, njev=0, time=4.9169
  Run 4 of 5: runs=4, nfev=5108, njev=0, time=0.3628
  Run 5 of 5: runs=3, nfev=4773, njev=0, time=0.3436
  ----------------------------------------
  avg_runs=16.0, avg_nfev=22327.8, avg_njev=0.0, avg_time=1.6033
```

```
  metric_1=16.0, metric_2=11163.9

  Method MS (Multi-start)
  Run 1 of 5: runs=1, nfev=120, njev=112, time=0.0176
  Run 2 of 5: runs=1, nfev=239, njev=227, time=0.0322
  Run 3 of 5: runs=1, nfev=47, njev=44, time=0.0065
  Run 4 of 5: runs=1, nfev=53, njev=50, time=0.0078
  Run 5 of 5: runs=1, nfev=131, njev=123, time=0.0183
  ----------------------------------------
  avg_runs=1.0, avg_nfev=118.0, avg_njev=111.2, avg_time=0.0165
  metric_1=1.0, metric_2=114.6

  Method SA (Simulated Annealing)
  Run 1 of 5: runs=1, nfev=451, njev=166, time=0.0272
  Run 2 of 5: runs=1, nfev=455, njev=170, time=0.0274
  Run 3 of 5: runs=1, nfev=571, njev=208, time=0.0341
  Run 4 of 5: runs=2, nfev=1065, njev=388, time=0.0628
  Run 5 of 5: runs=1, nfev=453, njev=187, time=0.0291
  ----------------------------------------
  avg_runs=1.2, avg_nfev=599.0, avg_njev=223.8, avg_time=0.0361
  metric_1=1.2, metric_2=411.4
```

B.5 H6 (Hartmann's family n=6)

```
Benchmark 5 of 5: H6 (Hartmann's family n=6)
  Method BH (Basin-hopping)
  Run 1 of 5: runs=1, nfev=22161, njev=3159, time=1.2873
  Run 2 of 5: runs=1, nfev=22604, njev=3217, time=1.3167
  Run 3 of 5: runs=1, nfev=21752, njev=3094, time=1.2685
  Run 4 of 5: runs=1, nfev=22480, njev=3191, time=1.3095
  Run 5 of 5: runs=1, nfev=22361, njev=3181, time=1.3032
  ----------------------------------------
  avg_runs=1.0, avg_nfev=22271.6, avg_njev=3168.4, avg_time=1.297
  metric_1=1.0, metric_2=12720.0

  Method DE (Differential Evolution)
  Run 1 of 5: runs=5, nfev=25340, njev=0, time=2.0397
  Run 2 of 5: runs=2, nfev=9873, njev=0, time=0.7906
  Run 3 of 5: runs=3, nfev=15024, njev=0, time=1.2112
  Run 4 of 5: runs=1, nfev=5075, njev=0, time=0.4041
  Run 5 of 5: runs=2, nfev=9333, njev=0, time=0.7445
  ----------------------------------------
  avg_runs=2.6, avg_nfev=12929.0, avg_njev=0.0, avg_time=1.038
  metric_1=2.6, metric_2=6464.5

  Method MS (Multi-start)
  Run 1 of 5: runs=1, nfev=233, njev=225, time=0.0567
  Run 2 of 5: runs=1, nfev=2643, njev=2544, time=0.6528
  Run 3 of 5: runs=1, nfev=1820, njev=1752, time=0.4478
  Run 4 of 5: runs=1, nfev=101, njev=97, time=0.0252
  Run 5 of 5: runs=1, nfev=542, njev=523, time=0.1325
  ----------------------------------------
  avg_runs=1.0, avg_nfev=1067.8, avg_njev=1028.2, avg_time=0.263
  metric_1=1.0, metric_2=1048.0

  Method SA (Simulated Annealing)
  Run 1 of 5: runs=1, nfev=345, njev=166, time=0.0463
  Run 2 of 5: runs=1, nfev=460, njev=180, time=0.0524
  Run 3 of 5: runs=2, nfev=813, njev=337, time=0.0966
  Run 4 of 5: runs=2, nfev=1001, njev=363, time=0.1076
  Run 5 of 5: runs=2, nfev=879, njev=349, time=0.1006
  ----------------------------------------
  avg_runs=1.6, avg_nfev=699.6, avg_njev=279.0, avg_time=0.0807
  metric_1=1.6, metric_2=489.3
```