Locating and Characterizing the Stationary Points of the Extended Rosenbrock Function

Schalk Kok schalk.kok@up.ac.za
Department of Mechanical and Aeronautical Engineering, University of Pretoria, Pretoria, 0002, South Africa

Carl Sandrock carl.sandrock@up.ac.za
Department of Chemical Engineering, University of Pretoria, Pretoria, 0002, South Africa

Abstract

Two variants of the extended Rosenbrock function are analyzed in order to find the stationary points. The first variant is shown to possess a single stationary point, the global minimum. The second variant has numerous stationary points for high dimensionality. A previously proposed method is shown to be numerically intractable, requiring arbitrary precision computation in many cases to enumerate candidate solutions. Instead, a standard Newtonian method with multi-start is applied to locate stationary points. The relative magnitude of the negative and positive eigenvalues of the Hessian is also computed, in order to characterize the saddle points. For dimensions up to 100, only two local minimizers are found, but many saddle points exist. Two saddle points with a single negative eigenvalue exist for high dimensionality, which may appear as "near" local minima. The remaining saddle points we found have a predictable form, and a method is proposed to estimate their number. Monte Carlo simulation indicates that it is unlikely to escape these saddle points using uniform random search. A standard particle swarm algorithm also struggles to improve upon a saddle point contained within the initial population.

Keywords

Numerical optimization, stationary points, saddle points, benchmark functions.

1 Introduction

Suitable test functions are indispensable in the development of optimization algorithms, since practical optimization problems are frequently computationally expensive. However, the conclusions that can be drawn about the abilities of an algorithm are limited by the knowledge of the challenges that a particular test function poses. Therefore, we analyze the extended Rosenbrock function in this paper, in order to highlight the challenges that this popular test function poses.

Similar to Shang and Qiu (2006), we analyze the Hessian of the test functions at a stationary point. A stationary point $\boldsymbol{x}$ of a function $f(\boldsymbol{x})$ is any point where the gradient vector vanishes, that is, $\nabla f(\boldsymbol{x}) = \mathbf{0}$. However, instead of only determining whether or not the Hessian is positive definite, we compute the eigenvalues of the Hessian. At a stationary point, the ith eigenvalue λ_i is the curvature of the function in the direction of the associated eigenvector $\hat{\boldsymbol{x}}_i$ (Himmelblau, 1972). If all eigenvalues are positive

(negative) at a stationary point, the stationary point is a local minimizer (maximizer). If some eigenvalues are positive, and some negative, the stationary point is a saddle point. In some cases, all but one of the eigenvalues are positive. If in addition the magnitude of this single negative eigenvalue is significantly less than the magnitude of the positive eigenvalues, a very limited range of descent directions exist, and optimization algorithms may find such saddle points difficult to escape. One such an example is found by Deb et al. (2002) for the 20D Rosenbrock problem, where the "near minimum" they report is in fact a saddle point with only one negative eigenvalue. One million small random perturbations about this point do not generate any superior solutions (Shang and Qiu, 2006). However, a small perturbation in the direction of the eigenvector associated with the negative eigenvalue generates a superior solution. This example motivates the characterization of a test function in terms of the presence of saddle points, particularly those saddle points with very few negative eigenvalues.

2 Rosenbrock Variants

The 2D Rosenbrock function (parabolic valley problem), given by

$$f(x_1, x_2) = 100\big(x_1^2 - x_2\big)^2 + (x_1 - 1)^2, \tag{1}$$

is a well-known test function in classical optimization that was originally analyzed by de Jong (1975). This test function has a single stationary point, at $x_1 = x_2 = 1$. The Hessian matrix at this stationary point is positive definite, signifying that the stationary point is a local minimizer.

This 2D function has been extended to higher dimensions, in order to assess the performance of optimization algorithms on problems of high dimensionality. As pointed out by Shang and Qiu (2006), many researchers assume that the extended versions also contain a single stationary point, the global minimum at $\boldsymbol{x} = [1\,1\ldots1]^T$. Several variants of the extended Rosenbrock function have been proposed. Although they have a completely different character, researchers often refer only to "the extended Rosenbrock function." We subsequently analyze two commonly encountered variants.

2.1 Variant A

Dixon and Mills (1994), among others, propose

$$f(x_1, x_2, \ldots, x_N) = \sum_{i=1}^{N/2} \big[100\big(x_{2i\text{-}1}^2 - x_{2i}\big)^2 + (x_{2i\text{-}1} - 1)^2\big]. \tag{2}$$

This variant is only defined for even N, and is the sum of $N/2$ uncoupled 2D Rosenbrock problems. The gradient vector of f is computed as

$$\begin{aligned} \frac{\partial f}{\partial x_{2i\text{-}1}} &= 400x_{2i\text{-}1}\big(x_{2i\text{-}1}^2 - x_{2i}\big) + 2(x_{2i\text{-}1} - 1) \\ \frac{\partial f}{\partial x_{2i}} &= -200\big(x_{2i\text{-}1}^2 - x_{2i}\big), \end{aligned} \qquad \text{for } i = 1, 2, \ldots, \frac{N}{2} \tag{3}$$

while the Hessian $[H]$ is 2×2 block diagonal, with every 2×2 block given by

$$200 \begin{bmatrix} 6x_{i\text{-}1}^2 - 2x_i + 0.01 & -2x_{i\text{-}1} \\ -2x_{i\text{-}1} & 1 \end{bmatrix} \quad \text{for } i = 2, 4, \ldots, N. \tag{4}$$

The only stationary point for this variant is $x_{2i\text{-}1} = x_{2i} = 1$. The Hessian is positive definite at this point for all N (all eigenvalues are positive), hence the only stationary point is the global minimum.

2.2 Variant B

This variant, from Goldberg (1989), and as quoted by Shang and Qiu (2006), given by

$$f(x_1, x_2, \ldots, x_N) = \sum_{i=1}^{N-1} \left[100 \left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2\right], \tag{5}$$

is much more challenging to analyze. The gradient vector of f is given by

$$\frac{\partial f}{\partial x_1} = 400x_1\left(x_1^2 - x_2\right) + 2(x_1 - 1) \tag{6}$$

$$\frac{\partial f}{\partial x_i} = -200\left(x_{i\text{-}1}^2 - x_i\right) + 400x_i\left(x_i^2 - x_{i+1}\right) + 2(x_i - 1) \text{ for } i = 2, \ldots, N-1 \tag{7}$$

$$\frac{\partial f}{\partial x_N} = -200\left(x_{N\text{-}1}^2 - x_N\right), \tag{8}$$

while the nonzero components of the tri-diagonal Hessian $[H]$ are given by

$$H_{i,i} = 200\left(6x_i^2 - 2x_{i+1} + 0.01\right) \quad \text{for} \quad i = 1, 2, \ldots, N-1. \tag{9}$$

$$H_{N,N} = 200. \tag{10}$$

$$H_{i,i+1} = H_{i+1,i} = -400x_i \quad \text{for} \quad i = 1, 2, \ldots, N-1. \tag{11}$$

Shang and Qiu (2006) detail a technique to determine the stationary points. Assuming a value for x_1, Equation (6) is used to solve for x_2:

$$x_2 = \frac{200x_1^3 + x_1 - 1}{200x_1} \tag{12}$$

Equation (7) is then used to solve for x_3 to x_N:

$$x_{i+1} = \frac{-100x_{i\text{-}1}^2 + 101x_i + 200x_i^3 - 1}{200x_i} \text{ for } i = 2, \ldots, N-1. \tag{13}$$

Finally, if Equation (8) equals zero, the generated sequence $x_1, x_2, \ldots, x_N$ is a stationary point. Shang and Qiu (2006) use points generated with the method outlined above as starting points for a steepest descent algorithm to find local minimizers. This strategy seems very attractive, since it transforms the N dimensional problem to a 1D problem:

search Equation (8) as a function of x_1, and enumerate the roots. However, this strategy has severe limitations, as discussed in Section 2.2.2.

2.2.1 Newton's Method

An alternative method to find the stationary points of f is to use Newton's method. Given an initial guess $\boldsymbol{x}^0$, a sequence of improved candidates to a stationary point is computed from

$$\boldsymbol{x}^{i+1} = \boldsymbol{x}^i + \Delta\boldsymbol{x} \quad \text{for } i = 0, 1, \ldots \tag{14}$$

where the update $\Delta\boldsymbol{x}$ is solved from Newton's method:

$$\boldsymbol{H}(\boldsymbol{x}^i)\Delta\boldsymbol{x} = -\nabla f(\boldsymbol{x}^i). \tag{15}$$

Here $\boldsymbol{H}$ is the Hessian, ∇f is the function gradient, and the superscripts indicate the iteration number. This process repeats until the norm of the gradient is less than some small tolerance ϵ. Since $\boldsymbol{H}$ is symmetric and tri-diagonal, the linear system in Equation (15) can be solved very efficiently. The quadratic convergence of Newton's method provides highly accurate stationary points in a small number of iterations, if the algorithm converges. The convergence of the method is improved by imposing a maximum update norm, chosen as $\|\Delta\boldsymbol{x}\| < 10N$ for this problem.

A large number of random starts are performed in the domain $[-1; 1]$ on all dimensions, and only those Newton runs that converge are recorded. Using this approach, 108 stationary points are found for the $N = 30$ case, with 100 million random starts. Two of the stationary points are the minimizers found by Shang and Qiu (2006), the global minimum at $x_1 = [1 \cdots 1]^T$ and the local minimum near $x_1 = [-1\ 1 \cdots 1]^T$. The remaining stationary points are saddle points.

Upon scrutiny of the located stationary points, it seems that our results are in disagreement with those of Shang and Qiu (2006). We identify a large number of *distinct* stationary points with a first component value of $x_1 = -0.55537607608450$ within the accuracy afforded by IEEE double precision floating-point numbers. This deserves further investigation since this value of x_1 should yield a single, unique stationary point according to the method of Shang and Qiu (2006).

2.2.2 Sensitivity of the Numerical Scheme

The explanation of the "disagreement" lies in the numerical sensitivity of the algorithm proposed by Shang and Qiu (2006), which they mention suffers from numerical issues related to accumulation of errors in the terms. To quantify this phenomenon, we compute the sensitivity of x_N with respect to x_1. This follows from the chain rule applied to Equation (13):

$$\frac{dx_{i+1}}{dx_1} = \frac{\partial x_{i+1}}{\partial x_i}\frac{dx_i}{dx_1} + \frac{\partial x_{i+1}}{\partial x_{i-1}}\frac{dx_{i-1}}{dx_1} \quad \text{for } i = 2, \ldots, N-1. \tag{16}$$

The partial derivatives $\frac{\partial x_{i+1}}{\partial x_i}$ and $\frac{\partial x_{i+1}}{\partial x_{i-1}}$ are given by

$$\frac{\partial x_{i+1}}{\partial x_i} = \frac{600x_i^2 + 101 - 200x_{i+1}}{200x_i} \tag{17}$$

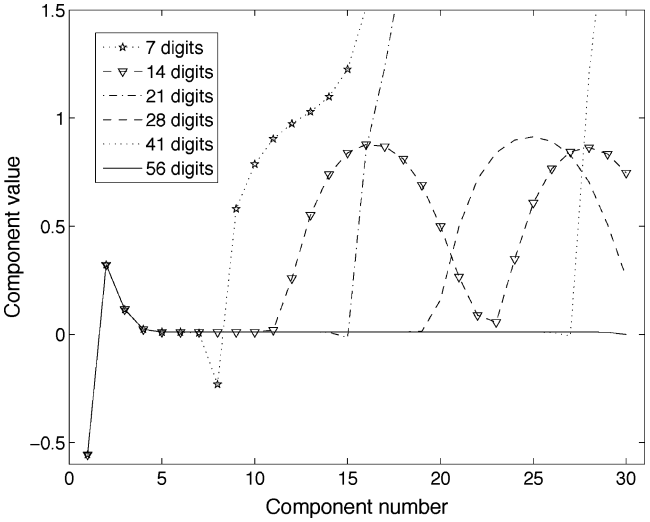

Figure 1: Numerical sensitivity of the sequences $x_1, x_2, \ldots, x_N$ generated with the algorithm of Shang and Qiu (2006) for $N = 30$, with varying precision representations of $x_1 = -0.55537607608450250523360402547909222298104783106761027618392 8$.

$$\frac{\partial x_{i+1}}{\partial x_{i-1}} = -\frac{x_{i-1}}{x_i}. \tag{18}$$

In addition, $\frac{dx_1}{dx_1} = 1$ and $\frac{dx_2}{dx_1}$ follows from direct differentiation of Equation (12):

$$\frac{dx_2}{dx_1} = \frac{600x_1^2 + 1 - 200x_2}{200x_1}. \tag{19}$$

The absolute value of the partial derivative $\frac{\partial x_N}{\partial x_1}$ for the 108 roots found by the random procedure varies between 10^3 and 10^{42}, with 38 of the 108 values greater than 10^{22}. Even the magnitude of the sensitivities for the two local minimizers are greater than 10^8. This has grave implications when attempting to use the algorithm proposed by Shang and Qiu (2006) using double precision arithmetic. To illustrate, the most frequently found saddle point was refined to 62 digits using Newton's method with the arbitrary precision computations implemented in the Python (2006) decimal module. Figure 1 shows the sequences generated when using less precision in Shang and Qiu's method. At least 56 digit precision is required to reproduce the stationary point accurately. The accompanying graph of $x_{N-1}^2 - x_N$ vs. x_1 is shown in Figure 2.

Transforming the original N dimensional problem to a 1D problem is therefore not a tenable method to solve the problem because the resolution required along the x_1 axis in order to locate all the stationary points is higher than conventional floating point precision provides. The required resolution also increases with N. Numerical experiments for $N = 100$ indicate than in some cases 170 digits are required to represent x_1, in order to generate a stationary point where x_{100} is accurate to only six digits. Locating candidate x_1 values for $x_1 \in [-1\ \ 0]$ using increments of $\Delta x_1 = 10^{-170}$ would require

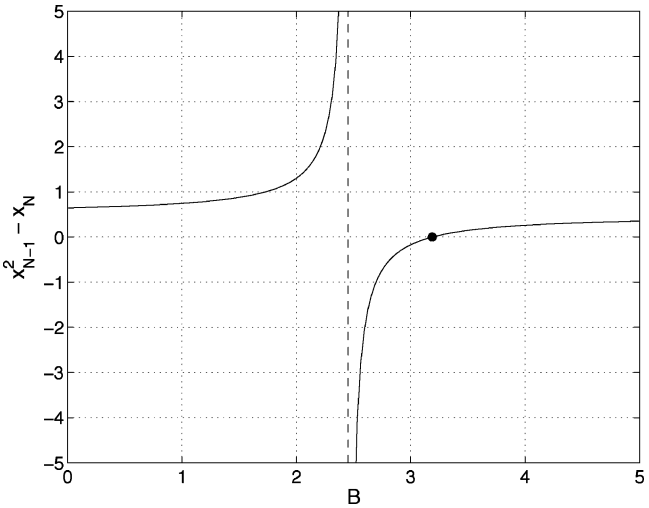

Figure 2: $x_{N\text{-}1}^2 - x_N$ vs. B, where $x_1 = A + 10^{\text{-}43}B$ and the constant $A = -0.55537607608450250523360402547909222298104815$. The curve is for $N = 30$, computed using 56 decimal point precision.

the generation of 10^{170} sequences, which is not feasible using current computational devices. Even for $N = 30$, the numerical experiments suggest the generation of 10^{25} sequences.

The large number of stationary points found for which $x_1 \approx -0.55537607608450$ is now resolved: these stationary points have each distinct x_1 values, but a large number of digits is required before these values can be distinguished. This is illustrated in Figure 3, which depicts Equation (8) as a function of $x_1 \in [A, A + 1.32 \times 10^{\text{-}23}]$, where $A = -0.55537607608450250523360398 5$. We also verified that we are able to reduce the gradient norm of those stationary points we located to arbitrarily small values using increasing precision with Newton's method using arbitrary precision arithmetic.

2.2.3 Analytical Solution

Due to the computational demands of the method by Shang and Qiu (2006), we attempted an analytical solution using their approach. x_2 is available as a function of x_1 from Equation (12). Repeated substitution of $x_i(x_1)$ and $x_{i\text{-}1}(x_1)$ into Equation (13) provides all $\boldsymbol{x}$ components as functions of x_1. Finally, we obtain

$$r_N(x_1) = x_{N\text{-}1}^2(x_1) - x_N(x_1). \tag{20}$$

Explicit forms for r_N have been obtained for $N = 2$ to 7. In all of these cases, $r_N(x_1)$ is of the form

$$r_N(x_1) = \frac{(x_1 - 1)p_N(x_1)}{q_N(x_1)} \tag{21}$$

where both $p_N(x_1)$ and $q_N(x_1)$ are polynomials in x_1 on the order of $3^{N\text{-}2} - 1$ and $3^{N\text{-}2}$, respectively. The asymptote visible in Figure 2 is now also explained: the asymptote coincides with a root of $q_N(x_1)$.

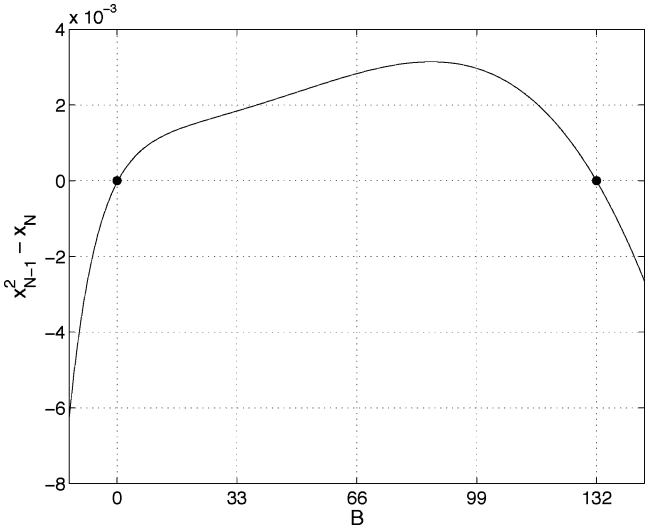

Figure 3: $x_{N\text{-}1}^2 - x_N$ vs. B, where $x_1 = A + 10^{\text{-}25}B$ and the constant $A = -0.55537607608450250523360 3985$. The curve is for $N = 30$, computed using 65 decimal point precision. Notice the two distinct roots, spaced approximately $132 \times 10^{\text{-}25}$ apart.

Since we aim to locate the roots of $r_N(x_1)$, we only have to locate the roots of $(x_1 - 1)p_N(x_1)$. According to Cauchy's bound, all the real roots of $p_N(x_1)$ are within the interval $x_1 \in [-M;\ M]$, with M given by

$$M = 1 + \frac{\max_{i=1}^{m\text{-}1} |a_i|}{|a_m|}, \tag{22}$$

where a_i is the ith coefficient of the polynomial $p_N(x_1) = a_m x_1^m + \cdots + a_1 x_1 + a_0$.

Furthermore, the number of real roots can be determined using the Sturm sequence P_i of p_N. This is done by defining Sturm functions (Weisstein, 2008)

$$P_0(x_1) = p_N(x_1) \tag{23}$$

$$P_1(x_1) = p'_N(x_1) \tag{24}$$

$$P_n(x_1) = -\text{rem}\,(P_{n\text{-}2},\, P_{n\text{-}1}), \quad n \geq 2. \tag{25}$$

Here rem$(P_{n\text{-}2},\, P_{n\text{-}1})$ denotes the remainder upon division of $P_{n\text{-}2}$ by $P_{n\text{-}1}$. The sequence terminates when a constant is obtained. Now, the difference in the number of sign changes between the Sturm functions when evaluated at $x_1 = a$ and $x_1 = b$ gives the number of nonrepeated real roots N_r in the interval (a, b).

The leading coefficient a_m, the maximum absolute remaining coefficient of p_N, the Cauchy bound M and the number of real roots N_r in $[-M, M]$ are given in Table 1 for $N = 3$ to 7. In all these cases, the minimum absolute coefficient is 1. Not shown is the rapid growth in the number of significant digits required to exactly represent the integer coefficients: 49 digits are required for $N = 6$, while at most 147 digits are required for $N = 7$.

Table 1: Leading Coefficient a_m, Maximum Absolute Remaining Coefficient, Cauchy Bound M, and Number of Real Roots N_r for $p_N(x_1) = a_m x_1^m + \cdots + a_1 x_1 + a_0$

N	m	a_m	$\max_{i=1}^{m-1} \lvert a_i \rvert$	M	N_r
3	2	200	200	2	0
4	8	8×10^6	8.12×10^6	2.015	2
5	26	5.12×10^{20}	5.3504×10^{20}	2.045	2
6	80	$1.34217728 \times 10^{62}$	$1.5233712128 \times 10^{62}$	2.135	2
7	242	$2.417851639229258 \times 10^{186}$	$3.397081553117108 \times 10^{186}$	2.405	2

The problem of locating the stationary points of the Rosenbrock function is now transformed to locating the real roots of the polynomial $p_N(x_1)$. An analytical solution is possible for $N = 3$ ($m = 2$, a quadratic) but thereafter numerical procedures are again required. Using arbitrary precision computation in Maxima (2008), the two real roots are computed for $N = 4$ to 7. Since we were unable to determine $p_N(x_1)$ for $N > 7$, we could not attempt to compute the associated roots. If double precision numbers are used to represent the coefficients approximately, we find the two correct real roots for $N = 4$ and 5, six real roots for $N = 6$ (the two correct roots and four spurious roots), and 10 real roots for $N = 7$ (only one of the two correct roots, and nine spurious roots). Conventional double precision representation is clearly inadequate due to inevitable rounding errors.

As a final attempt to find all the stationary points of the Rosenbrock problem, we attempt to solve the system of homogeneous polynomial equations (Equations [6–8]). Using a polyhedral homotopy continuation method (Gunji et al., 2004), we could solve the system up to $N = 10$. This method attempts to find all the solutions to the system, including complex solutions. For the $N = 10$ case, 6,377 solutions are found, of which only three are real. Based on the analytical results for $N < 8$, we expect $3^{(10-2)} = 6{,}561$ solutions. These computations, using a single CPU of an AMD Athlon 4400+ dual core with 4 GB memory, required more than 6 h to complete. As a comparison, the Newton scheme using 1,000 random starts frequently requires less than 0.1 s of CPU time to locate all three real solutions. Homotopy continuation is therefore not currently suitable since it is limited to small N, and it might not locate all the (real) solutions.

At this point it becomes evident that we have to abandon our attempt to locate all real stationary points. We have to resort to some efficient numerical procedure that allows us to locate "many" stationary points. At the very least this provides a lower bound on the number of stationary points.

2.2.4 Predictive Pattern

Close scrutiny of the stationary points found using Newton's method revealed a pattern that allows us to predict the stationary points for a given problem dimension N. For the $N = 100$ problem, approximately 71% on the random starts converge to a stationary point. The four stationary points listed in Table 2 account for 99.9% of these converged solutions. The first two stationary points are local minimizers, also found by Shang and Qiu (2006). The remaining two stationary points are saddle points, with only one negative eigenvalue. Notice that the magnitude of the negative eigenvalue is significantly less than the median eigenvalue.

For large N, the function values of the first three stationary points are insensitive of N, while the function value associated with the last stationary point is approximately a

Table 2: The Four Stationary Points Found Most Often Using 100 Million Random Starts and Newton's Method for $N = 100$, Reported to Eight Decimal Points*

x_1	1	−0.99932861	−0.01094139	−0.55537608
x_2	1	0.99665107	0.46210002	0.32244549
x_3	1	0.99833032	0.70758673	0.11517821
x_4	1	0.99916774	0.84772207	0.02350620
x_5	1	0.99958520	0.92242609	0.01066138
x_6	1	0.99979328	0.96091538	0.01021797
x_7	1	0.99989698	0.98041576	0.01020862
x_8	1	0.99994866	0.99021374	0.01028428
x_9	1	0.99997441	0.99511648	0.01020842
⋮	⋮	⋮	⋮	⋮
x_{96}	1	1.00000000	1.00000000	0.01020842
x_{97}	1	1.00000000	1.00000000	0.01020834
x_{98}	1	1.00000000	1.00000000	0.01020421
x_{99}	1	1.00000000	1.00000000	0.01000409
x_{100}	1	1.00000000	1.00000000	0.00010008
$P(\%)$	81.52	17.16	0.52	0.70
f	0	3.98662385	65.02536346	98.69667141
λ_1	0.49875312	0.49875312	−182.76136633	−2.81157458
λ_2	202.39471252	202.39100938	0.49875312	166.57951322
λ_{50}	976.86321108	976.48330537	936.15044478	197.99795121
λ_{51}	1001.99201591	1001.60968201	961.45622931	198.25623977
λ_{100}	1801.60524391	1801.60154965	1801.54623931	516.47175711

*Also included are the percentage found P, function value f, and four eigenvalues from the sorted list $\lambda_1 < \lambda_2 < \cdots < \lambda_{50} < \cdots < \lambda_{100}$.

linear function of N, given by

$$f \approx 0.989896874N - 0.293015997. \tag{26}$$

The remaining 0.1% of the random starts converge to saddle points with a very specific form. The basic solution is the fourth saddle point listed in Table 2, with a number of roughly half-sinusoidal curves (humps) superimposed upon it. Examples of these types of saddle points are presented in Figure 4 for the $N = 50$ case. As can be seen, the width of the hump is roughly 12 units, and it is found to be independent of N. Given such a stationary point, new stationary points are found if the hump is translated approximately 0.5 units to either side. This provides a numerical procedure to locate a large number of the stationary points. All possible combinations of humps that will fit are superimposed on the basic solution and used as the initial point for Newton's method. Update norms are limited to $\|\Delta \boldsymbol{x}\| \leq 0.1$ to assist convergence to a stationary point close to the starting point, and a maximum of 20 Newton steps are allowed. The list of roots is pruned based on roots already found. A MATLAB implementation of this numerical strategy is included in the Appendix.

This strategy found the number of stationary points listed in Table 3. Notice that for $N = 30$ we now locate 128 stationary points, including the four dominant stationary points, compared to 108 stationary points found using 100 million random starts. The function value associated with these stationary points can also be estimated. The

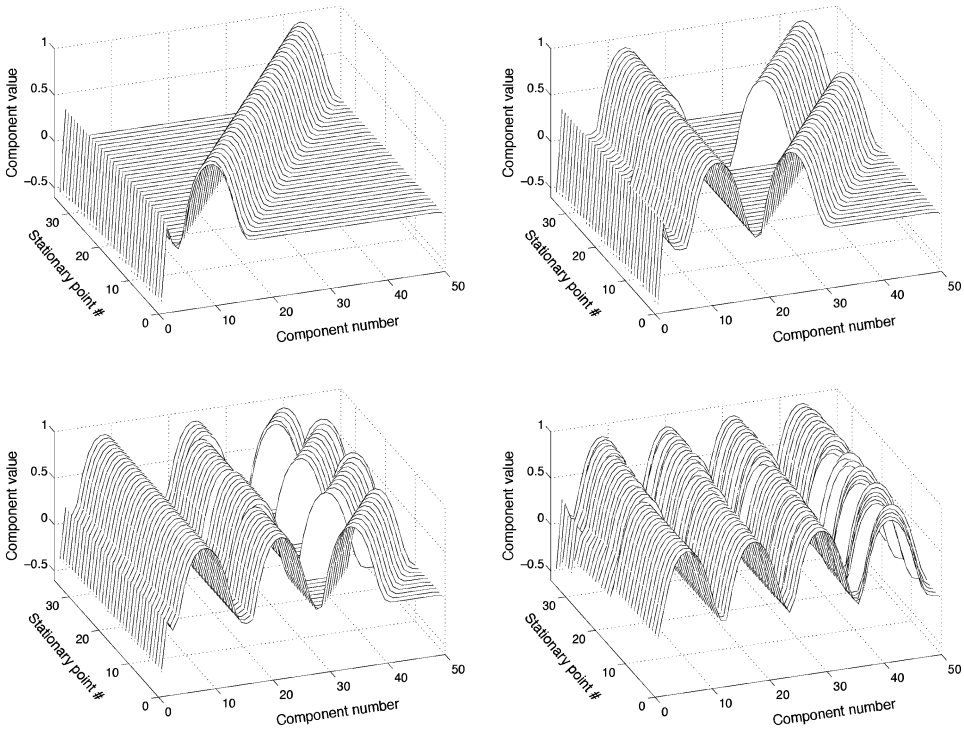

Figure 4: Examples of saddle points for $N = 50$, containing one to four humps.

function value of the base solution follows from Equation (26), while the presence of each hump contributes approximately 53.9 to the function value.

Also of interest is that the norm of the stationary points containing a certain number of humps are similar, and it increases as the number of humps increase. The distance between a stationary point and its nearest neighbor is distributed discretely, ranging between 0.01 and 0.45 for the $N = 50$ case, but with the most common distance being 0.29.

As can be seen from Table 3, the number of stationary points grows rapidly with N. Instead of using the random multi-start strategy to find each stationary point, we now exploit the repeating structure of the humps to deduce the number of stationary points.

Let us define n_a as the number of humps and a as their width. In addition, let us assume that there are n_b gaps of width b that represent the smallest incremental shift of a hump that can generate a new stationary point. If there are N dimensions and we want to fill the available space, it follows that

$$N = n_a \cdot a + n_b \cdot b \tag{27}$$

Thus,

$$n_b = \left\lfloor \frac{N - n_a \cdot a}{b} \right\rfloor \tag{28}$$

Table 3: Number of Stationary Points Containing Humps Found Using Newton's Method for Problem Dimension N, Exploiting the Predictable Location of the Humps

	Number of humps					
N	1	2	3	4	5	Total
12	2					2
20	17					17
25	27	5				32
30	37	87				124
35	47	277				324
40	57	561	184			808
50	77	1,447	5,066	144		6,734
60	97	2,726	22,696	27,619	27	53,165

Next we consider $n_a + n_b$ locations which could be either a hump or a space. Because the order in which these humps and spaces occur is not important, we can calculate the number of combinations as

$$\frac{(n_a + n_b)!}{n_a! n_b!} \tag{29}$$

Finally, we can exploit the knowledge that at most $\lfloor \frac{N}{a} \rfloor$ humps can fit into N spaces and write the total number of stationary points containing humps as

$$NSP = \sum_{n_a=1}^{\lfloor \frac{N}{a} \rfloor} \frac{(n_a + n_b)!}{n_a! n_b!} \tag{30}$$

where n_b is calculated from Equation (28). Since n_b is typically larger than n_a, it is beneficial to calculate Equation (30) by noticing that

$$\frac{(n_a + n_b)!}{n_b!} = (n_a + n_b) \cdot (n_a + n_b - 1) \cdot (n_a + n_b - 2) \cdots (n_b + 1)$$

to avoid calculating very large numbers in the numerator.

The predicted number of stationary points is depicted in Figure 5, for $n_a = 11.8$ and $n_b = 0.5$. The data from Table 3 are included. Notice that the trend rapidly approaches an exponential in N, with more than 145 million stationary points predicted for $N = 100$.

It is also found that the number of humps in a stationary point is related to the number of negative eigenvalues. Given that a stationary point contains n_a humps, the number of negative eigenvalues can range from $n_a + 1$ to $2n_a + 1$. Also, the relative magnitude of the negative eigenvalues is small compared to the positive eigenvalues. To illustrate, the eigenvalues at each of the 812 located stationary points (808 containing humps and four others) for $N = 40$ are plotted in Figure 6. Only 89 of the 812 stationary points have eigenvalues less than –50, while the median eigenvalue of all the points are greater than 198.

The effect of this eigenvalue spectrum is that evolutionary algorithms could find it very challenging to escape these saddle points. A Monte Carlo simulation is performed to investigate this further. M uniform random points are generated on the surface of an

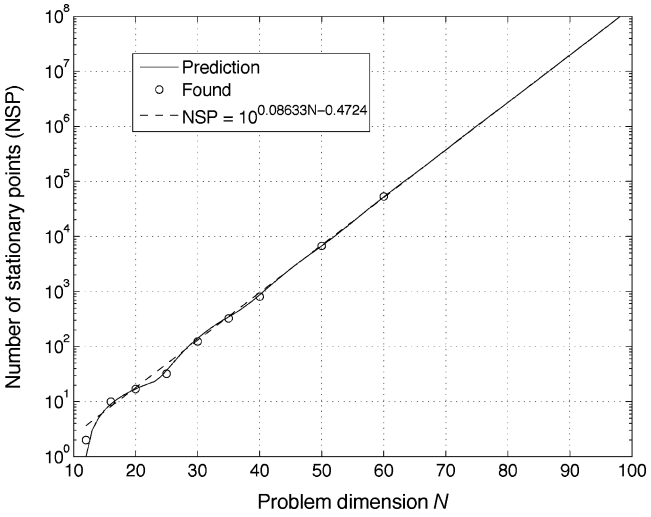

Figure 5: Predicted number of stationary points containing humps, compared to the actual number found.

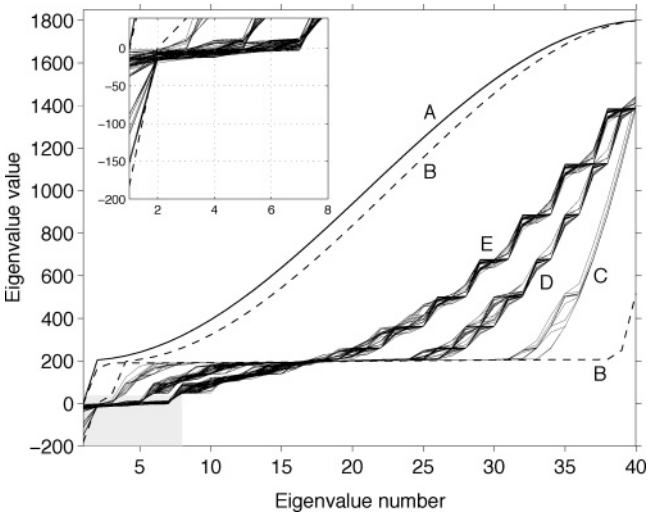

Figure 6: Eigenvalues at each of the 812 stationary points found for the 40D Rosenbrock problem. The two curves (A) are associated with local minima while the remaining curves are for saddle points that have (B) a single negative eigenvalue (C) one hump (D) two (97%) or three humps (3%) and (E) three (91%) or two (9%) humps.

N dimensional hypersphere of radius R, which is centered at some point of interest $\boldsymbol{x}^*$. After counting the number of points K with function value less than $f(\boldsymbol{x}^*)$, the integral of the fraction K/M from 0 to R (corrected for the probability of generating such a point) provides an estimate of the probability of generating a function value lower than $f(\boldsymbol{x}^*)$ using uniform random search.

Table 4: Local Minima and Saddle Points Used for Monte Carlo Simulation, Reported to 14 Decimal Points

	$N = 4$		$N = 6$	
	Local minimum	Saddle point	Local minimum	Saddle point
x_1^*	−0.775659226565	−0.656124635719	−0.986574979571	−0.555419727179
x_2^*	0.613093365485	0.443120040972	0.983398228836	0.322493274297
x_3^*	0.382062846338	0.204312248228	0.972106670053	0.115207004099
x_4^*	0.145972018552	0.041743494776	0.947437436826	0.023502770405
x_5^*			0.898651184852	0.010447901205
x_6^*			0.807573952035	0.000109158640
$f(\boldsymbol{x}^*)$	3.701428610430	3.708241996647	3.973940500930	5.646383966588

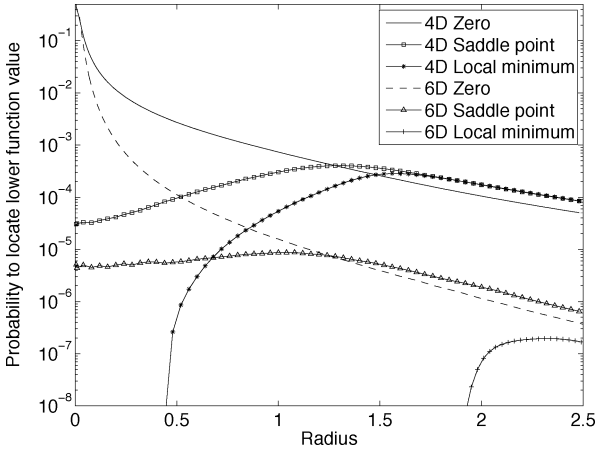

Figure 7: Probabilities to locate points with function values less that the specified points of interest, using random search.

The Monte Carlo simulation was performed for $N = 4$ and $N = 6$, choosing the points of interest as the saddle points and local minima in Table 4, as well as the zero vector. The zero vector provides a reference in that it is a point with nonzero gradient but with a relatively small function value. M was chosen as 10 million for all the simulations except the 6D local minimum, where we used 1 billion. For problem dimension $N > 6$, prohibitively large M is required to obtain reliable probabilities.

The results of the Monte Carlo simulations are presented in Figure 7, which confirms that the saddle points are indeed difficult to escape using random search. As the problem dimension increases, the probability to escape *using random search* decreases rapidly. This is true for all the saddle points we located, including those with multiple negative eigenvalues. Shang and Qiu (2006) did however report that their steepest descent method always located the global minimum if started at the 20D saddle point reported by Deb et al. (2002). We do not regard this as evidence that the saddle points are "easy" to escape *using evolutionary algorithms*, since few evolutionary algorithms use gradient information. Furthermore, since Deb et al. (2002) only reported the 20D saddle point to six digits, a nonzero gradient was computed at this point and hence the steepest descent algorithm moved away from the point.

3 Practical Implications

We conclude this study by illustrating the effect of the stationary points on a practical algorithm, the widely used particle swarm optimization (PSO) algorithm proposed by Kennedy and Eberhart (1995), even though we are aware that it might not be the best choice to solve this problem.

Consider a swarm of p particles in an N dimensional search space. The position vector $\boldsymbol{x}_k^i$ of each particle i is updated by

$$\boldsymbol{x}_{k+1}^i = \boldsymbol{x}_k^i + \boldsymbol{v}_{k+1}^i, \tag{31}$$

where k is a unit pseudo time increment (iteration number). $\boldsymbol{v}_{k+1}^i$ is the velocity vector that is obtained from

$$\boldsymbol{v}_{k+1}^i = w\boldsymbol{v}_k^i + c_1 r_1\big(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i\big) + c_2 r_2\big(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i\big), \tag{32}$$

where w is the inertia factor, c_1 and c_2 are cognitive and social scaling factors respectively, $\boldsymbol{p}_k^i$ is the best position vector of particle i, and $\boldsymbol{p}_k^g$ is the global best position vector of the complete swarm (i.e., a fully connected swarm), up to time step k. r_1 and r_2 are uniform random numbers between 0 and 1, and are generated independently for every dimension of every particle.

For each particle i, the initial position (iteration $k = 0$) is generated randomly within the allowable domain. We set the initial velocity of all particles to zero, and limited each velocity component to half the domain size to prevent instability of the swarm.

In our numerical experiments, we use problem domain ± 2.048 on all dimensions and parameter settings $w = 0.72$, $c_1 = c_2 = 1.49$, $p = 20$, and $N = 50$. In order to establish a reference point, we run the algorithm 100 times to a maximum of 100,000 iterations. The global minimum is found 18 times, the only known local minimum is found four times, and the mean function value is 17.8.

To illustrate the potential effect of a saddle point, we set the initial position of one of the particles equal to a known saddle point containing one hump, with function value approximately 103. After 100,000 iterations, only two out of 100 runs succeeded to improve upon this initial saddle point solution.

Now that we have established that the immediate neighborhoods of the Rosenbrock saddle points are difficult to escape by PSO once they have been entered, the one outstanding issue is whether it is likely that an algorithm searching for a minimum will locate one of these saddle points. To investigate this, we performed 10,000 PSO runs to 100,000 iterations. We found 544 cases where the global best solutions at 1,000 iterations are closer than 0.4 from a known saddle point. The vast majority is the base solution (the 50D equivalent to column four in Table 2), but stationary points containing one and two humps also occurred occasionally. The number of global best solutions that are close to known stationary points reduces to only three after 10,000 iterations, and none exists after 100,000 iterations. The point here is not that these numbers become negligible as the algorithm proceeds, but that saddle points do in fact feature in the searches of the PSO algorithm.

4 Conclusion

We investigated the stationary points of two variants of the extended Rosenbrock problem. Variant A is shown to have a single stationary point, the well known global

minimum. Variant B, however, contains numerous stationary points, including two local minima. The numerical scheme proposed by Shang and Qiu (2006) is shown to be a numerically intractable method to find all stationary points. Instead, Newton's algorithm is employed with multiple random starts. This uncovered sufficient stationary points in order to deduce a pattern in which the stationary points appear. The stationary points are characterized in terms of the number of negative eigenvalues, and the relative magnitude. Few negative eigenvalues of small magnitude as compared to the positive eigenvalues indicates "near" local minima, in that very limited search directions exist that can improve the solution. Variant B contains two such "near" local minimum saddle points, each with a single negative eigenvalue. It also contains numerous others that have a small fraction of negative eigenvalues, again with a small magnitude as compared to the positive eigenvalues. Using Monte Carlo simulation we illustrated that uniform random search is unlikely to escape these saddle points of Variant B. An algorithm like the particle swarm optimization (PSO) for instance struggles to escape a saddle point contained in the initial population. Finally, we found that the PSO algorithm does in fact locate saddle points while searching for the minimum, but manages to escape most of the time.

Acknowledgments

The authors gratefully acknowledge the contribution of Daniel N. Wilke for his insightful comments and suggestions.

References

Deb, K., Anand, A., and Joshi, D. (2002). A computationally efficient evolutionary algorithm for real-paremeter optimization. *Evolutionary Computation*, 10(4):371–395.

de Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. PhD Thesis, University of Michigan, Ann Arbor, Michigan.

Dixon, L. C. W., and Mills, D. J. (1994). Effect of rounding errors on the variable metric method. *Journal of Optimization Theory and Applications*, 80(1):175–179.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, Massachusetts: Addison-Wesley.

Gunji, T., Kim, S., Kojima, M., Takeda, A., Fujisawa, K., and Mitzutani, T. (2004). PHoM—A polyhedral homotopy continuation method for polynomial systems. *Computing*, 73:57–77.

Himmelblau, D. M. (1972). *Applied nonlinear programming*. New York: McGraw-Hill.

Kennedy, J., and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, Vol. 4 (pp. 1942–1948). Perth, Australia.

Maxima. (2008). Maxima 15.5.0, http://maxima.sourceforge.net/.

Python. (2006). Python Software Foundation, Python® 2.4.3, http://python.org/.

Shang, Y.-W., and Qiu, Y.-H. (2006). A note on the extended Rosenbrock function. *Evolutionary Computation*, 14(1):119–126.

Weisstein, E. W. (2008). Sturm function. From *MathWorld*—A Wolfram resource. http://mathworld.wolfram.com/SturmFunction.html.

Appendix Code Listing

The following MATLAB functions can be used to generate roots using the predictive pattern described in Section 2.2.4. Call `humper` from the commandline as `>> humper(N)` to generate a list of roots for the $N\dagger$ dimensional problem.

```
function allroots = humper(N)
spacing = 11; % How close the humps are allowed to come to one another
shift = 0.1; % How much the hump shifts to the next 'position'
rootnorm = 1e-9; % norm of difference between distinct roots
gradnorm = 1e-8; % gradient norm considered a valid root
allroots = zeros(0,N);
    function addstarts(Nstarts, starts)
        if Nstarts == 0
            [x,normdf] = RosenNewton(guessroot(N, starts));
            x = x';
            if normdf < gradnorm && all(sqrt(sum(bsxfun(@minus,
                allroots, x).^2, 2))>rootnorm)
                allroots = [allroots; x];
                Nfound = Nfound + 1;
            end
        else
            if isempty(starts)
                a = 1;
            else
                a = starts(end)+spacing;
            end
            b = N-spacing;
            for i = a:shift:b
                addstarts(Nstarts-1, [starts i]);
            end
        end
    end
for Nstarts = 1:floor((N-1)/spacing)
    Nfound = 0;
    fprintf('Searching for %i hump(s): ', Nstarts);
    addstarts(Nstarts, [])
    fprintf(' found %i roots\n', Nfound)
end
end
```

```
function [df] = RosenGrad(x)
n = length(x);
df = zeros(n,1);
df(1) = -400.0*x(1)*(x(2)-x(1)^2) + 2.0*(x(1)-1.0);
for i=2:n-1
    df(i) = -400.0*x(i)*(x(i+1)-x(i)^2) + 2.0*(x(i)-1.0) + ...
        200.0*(x(i) - x(i-1)^2);
end
df(n) = 200.d0*(x(n) - x(n-1)^2);
function [H] = RosenHess(x)
n = length(x);
H = zeros(n,n);
H(1,1) = 1200*x(1)^2 - 400*x(2) + 2.0;
H(1,2) = -400*x(1);
for i=2:n-1
    H(i,i-1) = -400*x(i-1);
    H(i,i)   = -400.0*(x(i+1)-x(i)^2) + 800*x(i)^2 + 202;
    H(i,i+1) = -400*x(i);
end
H(n,n-1) = -400.0*x(n-1);
H(n,n)   =  200.0;
end
```

```
function [x,NrmDF] = RosenNewton(x)
iter = 1;
maxiter = 20;
tolx = 1e-8;
while iter==1 || (iter<maxiter && NrmDF<1e5 && Nrmdx>tolx)
    iter = iter + 1;
    df = RosenGrad(x);
    H  = RosenHess(x);
    Delta_x = -H\df;
    scale = 1.0;
    Nrmdx = norm(Delta_x);
    if Nrmdx > 0.1
        scale = 0.1/Nrmdx;
    end
    x = x + scale*Delta_x;
    NrmDF = norm(df);
end
```

```
function r = guessroot(N, allstarts)
baseline = 0.01;
peak = 0.9;
first = [-0.55; 0.35; 0.1; 0.02];
width = 12;
r = ones(N, 1)*baseline;
r(1:length(first)) = first;
humpi = (0:width-1)';
for start = allstarts
    x = (humpi+(start-floor(start))+0.5);
    y = sin(x/width*pi)*peak;
    y(1) = 0.05;
    index = min(ceil(start)+humpi, N);
    r(index) = max(r(index), y);
end
```