Computational Experiments Using Randomized Algorithms for Robust Stability Analysis ⋆

Albert Yoon† and Pramod Khargonekar‡

Dept. of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122, USA

Abstract

In this paper, we take a "computational experiments" approach to robust stability analysis problems. Many robust control problems have been shown to be NP hard but in spite of this, it is important to develop effective techniques for solving them. A typical robust stability analysis problem is taken and formulated as an optimization problem to which several optimization algorithms are applied.

Key words: randomized algorithms, global optimization, robust stability analysis, real parametric uncertainty

1 Introduction

In the last few years, it has become increasingly clear that the problem of robust stability analysis (even for linear time-invariant finite dimensional control systems) is extremely difficult to tackle analytically. This appears to be even more so for the case of real parameter uncertainty. Results on computational complexity [5,14] of robust stability analysis provide strong support for such conclusions. The fundamental underlying reason appears to be the fact that these problems typically involve nonconvex optimization having lots of local minima. It is only in very special cases that one has neat analytical solutions.

⋆ This work was supported in part by the U.S. Army Research Office under Grant Nos. DAAH 04-93-G-0012 and DAAH 04-95-1-0263

† Tel. (313) 763-6724, Email: optimus@ eecs.umich.edu

‡ Tel. (313) 764-4328, Email: pramod@ eecs.umich.edu

These results notwithstanding, from an engineering perspective, it is important to have effective tools to deal with the problems of robust stability and performance.

While much of the effort in the previous research has been on analytical approaches and results, it seems to us that at this point in the evolution of the subject, it may be useful to take a "computational experiments" approach to the robust stability analysis problem. In other words, it may be useful to carry out empirical investigations to guide the development of alternative ways of attacking the problem. For such empirical computational investigations, it would be useful to have a collection of well designed benchmark problems which can pose challenges and serve as objective tests to evaluate competing computational approaches. However, there are no such widely accepted benchmark problems, at least as of this writing.

There are some results in the literature [19,10,3,24], which indicate that randomized algorithms may provide tractable approaches to the problems of robust stability analysis and performance. Genetic algorithms for robust stability analysis have been investigated in [12,28]. In [10], it was shown that even simple random search has potentially attractive computational complexity properties. Despite these results, it is quite unclear whether problems of moderate to large size (as measured by the dimensions of the state-space and the space of uncertain parameters) can be effectively solved using such computational techniques. In particular, the following issues need much better understanding:

- Are there certain types of optimization algorithms that are well suited for robust stability and performance analysis and synthesis problems? More generally, systematic comparative evaluation of different computational algorithms is necessary.
- Do randomized algorithms offer significant benefits over deterministic algorithms?

It seems unlikely that these questions can be attacked using analytical or theoretical techniques. Thus, we believe that the goal of developing efficient algorithms for solving this class of problems can be best furthered by an "experimental" approach.

This paper takes a modest step in this empirical computational direction. The main objective of the paper is to present results that compare several different direct, randomized optimization methods for robust stability analysis with real parametric uncertainty. We begin with two different formulations of the robust stability analysis problem as optimization problems. One of these formulations utilizes a result on maximal unidirectional perturbation bounds due to Fu and Barmish [9]. For global optimization, we restrict our attention to direct methods, where "direct" refers to methods that use only the value of the

objective function but no derivative or any other analytical information. Since robust control problems often lead to nondifferentiable optimization problems, direct methods will have broader applicability. The following methods are investigated here:

(1) Multi-start of Rosenbrock's method [4]
(2) Crude and adaptive random search [10]
(3) Adaptive partitioned random search [23]
(4) Controlled random search [1] and shuffled complex evolution [7]

These methods are described in some detail in Section 3 where we also discuss the reasons for choosing to focus on these algorithms. It is very difficult to establish theoretical properties of all but the simplest of these algorithms. Therefore, we evaluate the efficacy of the algorithms on two robust stability analysis problems with real-valued uncertainties for this investigation. One comes from [20] and represents a linearized model of the closed-loop system of a bank-to-turn, air-to-air missile with a linear $\mathcal{H}_\infty$ controller. The second is the 55 state, 20 uncertain real parameters multivariable robust stability analysis problem investigated in [10]. To measure the computational cost, we use the number of function evaluations and flops as metrics. Run time is dependent on the computing platform and software; however, typical run times are also mentioned to give an idea of the computational burden.

2 Robust Stability Analysis Problem

The problem that is considered in this paper is the robust stability analysis problem. Consider the feedback system (M, Δ) containing an LTI system $M = (A, B, C)$ and the perturbation Δ shown in Figure 1.

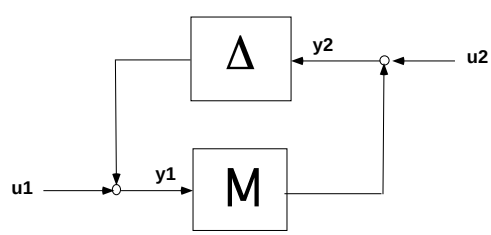

Fig. 1. Nominal system in feedback with structured uncertainty.

The perturbations are unknown, but are assumed to have a known structure. In this paper, the class of allowable perturbations is given by:

$$\mathbf{\Delta} = \{diag[\delta_1, \delta_2, \cdots, \delta_n] : \delta_i \in \mathbf{R}\}. \tag{1}$$

The size of the perturbation is its infinity norm:

$$\|\Delta\|_\infty = \overline{\sigma}(\Delta).$$

The measure of robustness, γ_{opt}, is defined to be the size of the smallest perturbation $\Delta \in \mathbf{\Delta}$ that destabilizes the feedback interconnection shown in Figure 1. If we define

$$\theta(M, \Delta) := max(Re(\lambda(A + B\Delta C))) \tag{2}$$

where $\lambda(X)$ is the set of eigenvalues of X, then

$$\begin{aligned} \gamma_{opt} &= \inf(\|\Delta\|_\infty : (M, \Delta) \text{ is unstable}) \\ &= \inf(\|\Delta\|_\infty : \theta(M, \Delta) \geq 0). \end{aligned} \tag{3}$$

Solving this problem would typically be handled by the real structured singular value (real μ) theory [6,26]. However, real μ analysis and synthesis problems are very difficult and one usually resorts to computing upper and lower bounds on the quantities of interest such as γ_{opt}. Experience has shown that these bounds can be arbitrarily conservative and recently, it has been proven that solving these problems is NP hard [5]. Therefore a reasonable course of action is to move our focus towards efficient means of coming to an approximate solution and reduce the conservatism of the bounds. We note that the definition of γ_{opt} naturally poses this problem as an optimization problem.

2.1 Optimization Problem

The optimization problem can be formulated in different ways. The performance of a particular optimization algorithm may depend greatly on the formulation that is chosen and so two possibilities are presented here. The first method is a very natural one but has the drawback of having a "rough" cost function. The second is also a fairly natural choice that may be smoother but requires greater computational effort to evaluate.

2.1.1 Norm-based Cost

If θ is the maximum real part of the eigenvalues of $A+B\Delta C$, the optimization problem can be posed as:

$$min\|\Delta\|_\infty \text{ subject to } \theta \geq 0. \tag{4}$$

To restrict the scope of the search, it is assumed that the size of the smallest destabilizing perturbations is at most $\overline{\gamma}$. This simply reflects the requirement that we have a prior (possibly loose) bound on the size of perturbations. The constraint, $\theta \geq 0$, and the size constraint are handled by assigning a high value for the cost function if a sample is taken from the infeasible region. The cost function, $J(\Delta)$, we used is:

$$J(\Delta) = \begin{cases} 2\overline{\gamma} - \|\Delta\|_\infty & ,\theta < 0 \text{ and } \|\Delta\|_\infty \leq \overline{\gamma} \\ \|\Delta\|_\infty & ,\theta \geq 0 \text{ or } \|\Delta\|_\infty \geq \overline{\gamma} \end{cases} \tag{5}$$

The only costs that will be less than $\overline{\gamma}$ will result from those perturbations that destabilize and also have size less than $\overline{\gamma}$. The optimization problem can thus be posed in an unconstrained setting as

$$\min_{\|\Delta\| \in \mathbf{\Delta}} J(\Delta). \tag{6}$$

2.1.2 Unidirectional Perturbation-Based Cost

This formulation of the problem utilizes an existing result from robust control theory. Consider all perturbations such that $\|\Delta\|_\infty$ is unity and for $M = (A, B, C)$ and $\xi \in \mathbf{R}$, define $\overline{A} = A + \xi B \Delta C$. Fu and Barmish [9] provide an analytic expression for the smallest $|\xi|$ such that $A_0 + \xi A_1$ is unstable, where $A_0 \in \mathbf{R}^{\mathbf{n}\times\mathbf{n}}$ is stable. The unidirectional perturbation-based cost function is

$$J(\Delta) = min(|\xi|) \text{ such that } max(Re(\lambda(A + \xi B \frac{\Delta}{\|\Delta\|_\infty} C))) \geq 0 \tag{7}$$

and the optimization problem is posed as in 6. For each, Δ, $J(\Delta)$ can be computed by applying the result of Fu and Barmish.

The drawback of this approach is that computing ξ is very expensive. The analytic expression requires the inversion and computation of eigenvalues of matrices with dimension of order $n^2 \times n^2$ where $n \times n$ is the dimension of A_0.

There is a natural connection of this formulation with the frequency sweep based μ analysis. In particular, the optimal ξ leads to a perturbed system with imaginary axis eigenvalues. Thus, by examining the frequency of the imaginary axis eigenvalues of the perturbed system we can find an upper bound for the value of μ at that frequency. Moreover, for the optimal Δ we get the frequency where the worst case μ is attained.

Figure 2 shows what each of the cost functions looks like for a randomly generated 3 state, 1 input/output system that is nominally stable. The value of $\overline{\gamma}$ for the norm-based cost function is 3. The minimum of both cost functions,

γ_{opt}, is 2.76 but the norm-based cost function is more jagged than the unidirectional perturbation-based cost function and can be expected to be more difficult to optimize. (The contrast is exaggerated in the scalar case since the unidirectional perturbation-based cost function only takes on two values but it nevertheless illustrates a potential benefit of that cost function.) When the perturbations are negative, the system is not destabilized and the unidirectional perturbation-based cost is infinity.

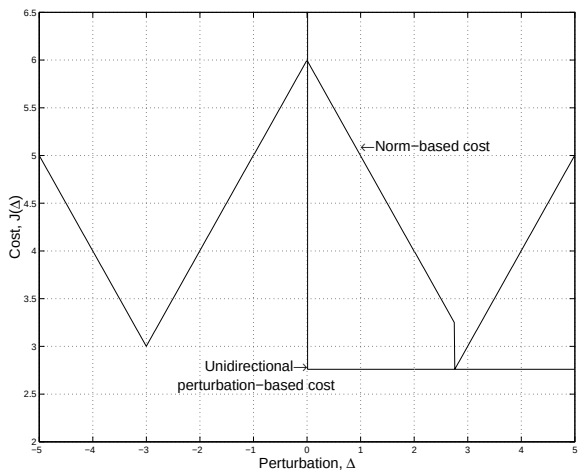

Fig. 2. Comparison of the cost functions for a scalar example.

3 Optimization Algorithms

There exist many algorithms which attempt to find the minimum of a function. Of course, testing out all of them is impractical for a single study, and only a few of them are selected for this one. All of the methods considered here are direct. They are called direct because they do not compute the gradient of the function to optimize the function or to decide which direction to perform the line search and where along that direction the next point should be chosen. Direct methods are chosen since gradient information is computationally expensive to approximate. Moreover, we wish to try to solve several problems in the future which are too complex to hope to have gradient information available. For example, in [25] the cost function is nondifferentiable and each evaluation involves the solution of a system of nonlinear differential equations with table look-ups, which is done using simulation tools.

A fairly wide range of random search algorithms is tested in this study. The first algorithm is a multistart version of Rosenbrock's line search method. There are several line search methods such as Zangwill's [27] and Hooke and Jeeve's [4] but they are not examined here. Apart from selecting the initial conditions randomly, the algorithm is purely deterministic. Another set of algorithms contains the pure random search and some variants of it. The pure

random search uniformly samples the space of all possible perturbations and records the best current estimated value of γ_{opt}. Variations on the pure random search such as partitioning or shrinking the input space are made to improve its efficiency. A third algorithm is the controlled random search (CRS) which has many variants. The version used in this study is from [1]. Finally, the Shuffled Complex Evolution (SCE) approach of [7] which uses elements of the CRS and partitioned random search techniques is implemented and tested. All of these algorithms place an upper bound on γ_{opt}.

3.1 Line Searches and Random Multistart

There are many line search methods for solving optimization problems. The algorithm of Rosenbrock [4] is presented here as an example. It is deterministic and also differs from the random searches in that it moves only a single point; the random algorithms attempt to move a group of points. While line search techniques do not compute gradients, they in essence attempt to do the same thing as a gradient descent technique. A gradient descent technique uses the gradient to find a descent direction along which a one dimensional minimization is performed in order to select the next best point. These direct, single point, line search techniques estimate that descent direction by cycling through n linearly independent directions, where n is the dimension of the input space, and trying to make a descending step in each of them. This algorithm is performed multiple times with varying, random initial points (hence random multistart). Rosenbrock's algorithm is below.

- <u>Rosenbrock</u>

  <u>Inputs:</u> x_0, ϵ, $\alpha > 1$, $\beta \in (-1, 0)$, $\overline{S}_i > 0$ for $i = 1, 2, \ldots, n$

  <u>Initialize:</u> Let k=1 and j=1. Choose $d_1, d_2, \ldots, d_n$ as the coordinate directions. Let $S_i > 0$ be the initial step size lengths along each direction d_i and let $S_i = \overline{S}_i$. Let $y_1 = x_0$.

  <u>Step 1:</u> If $J(y_j + S_j d_j) < J(y_j)$ then let $y_{j+1} = y_j + S_j d_j$ and $S_j = \alpha S_j$. Otherwise, let $y_{j+1} = y_j$ and $S_j = \beta S_j$. If $j < n$ let $j = j + 1$ and repeat step 1. Otherwise, go to step 2.

  <u>Step 2:</u> If $J(y_{n+1}) < J(y_1)$, let $y_1 = y_{n+1}$ and $j = 1$ and go to step 1. Otherwise, $J(y_{n+1}) = J(y_1)$. If $J(y_{n+1}) < J(x_k)$ go to step 3. Otherwise, check if $|S_j| \leq \epsilon$ for each j. If so, then end with x_k as the estimate of the optimal solution. Otherwise, let $y_1 = y_{n+1}$, let $j = 1$, and go to step 1.

  <u>Step 3:</u> Let $x_{k+1} = y_{n+1}$. If $\|x_{k+1} - x_k\| < \epsilon$ then end with x_{k+1} as the estimate of the optimal solution. Otherwise, compute λ_i from $x_{k+1} - x_k = \sum_{i=1}^{n} \lambda_j d_j$ and form a new set of orthogonal search directions $b_i, i = 1, 2, \ldots, n$ with the following expressions:

$$a_j = \begin{cases} d_j & \text{if } \lambda_j = 0 \\ \sum_{i=j}^{n} \lambda_i d_i & \text{if } \lambda_j \neq 0 \end{cases} \tag{8}$$

$$b_j = \begin{cases} a_j & j = 1 \\ a_j - \sum_{i=1}^{j-1}(a_j^T \overline{d}_i)\overline{d}_i & j \geq 2 \end{cases} \tag{9}$$

$$\overline{d}_j = \frac{b_j}{\|b_j\|} \tag{10}$$

Let $S_j = \overline{S}_j$ and $d_j = \overline{d}_j$ for each j, let $y_1 = x_{k+1}$, let $k = k+1$, let $j = 1$, and go to step 1.

Rosenbrock's algorithm is attractive because, unlike say Hooke and Jeeve's algorithm [4], it changes its set of line search directions to suit the cost function. Numerical experiments have shown this algorithm to be more efficient than those that do not adapt its line search directions. Depending on previous step lengths, a new set of orthogonal directions is found with the first direction being chosen as the direction from the first to the last point of the previous iteration. The crude line search technique also provides some robustness. One drawback of this algorithm is that the operation of generating a new set of n orthogonal directions after each iteration is computationally expensive.

3.2 Random Search

The random search technique is the simplest and most robust for finding the global minimum of a function since the points that are chosen are almost if not completely independent of the function. Thus, it is not subject to being trapped by local minima. The price that is paid for this robustness is poor efficiency. Many more function evaluations are likely necessary to arrive at a solution of the same quality as one that is found by a more directed technique.

Two random search methods, taken from [10], and a method based on that used in [25] are grouped in this section. The first algorithm simply samples from the region of all perturbations with size less than or equal to $\overline{\gamma}$ a designated number of times and records the sample that provides the lowest cost. The second is different in that the size of the search space is determined by the size of the current best sample. In this way, samples are not wasted on regions that will not yield a smaller destabilizing perturbation. This algorithm applies only to the norm-based cost function. The third algorithm samples the space uniformly, finds the best points, and then in ensuing iterations performs crude random searches that are concentrated around the best points. The algorithms are presented below.

- <u>Crude Random Search</u>
  <u>Inputs:</u> *iterations*,$\overline{\gamma}$
  <u>Initialization:</u> Let $\gamma = \overline{\gamma}$ and let $i = 1$.
  <u>Step 1:</u> Generate Δ_i uniformly from $\Delta \in \mathbf{\Delta} : \overline{\sigma}(\Delta) \leq \overline{\gamma}$.

Step 2: If $A + B\Delta_i C$ is unstable then let $\gamma = min(\overline{\sigma}(\Delta_i), \gamma)$.
Step 3: If $i < iterations$ then let $i = i + 1$ and go to step 1.
Step 4: End and return γ.

- Adaptive Random Search I (ARS I)
  Inputs: $iterations$,$\overline{\gamma}$
  Initialization: Let $\gamma = \overline{\gamma}$ and let $i = 1$.
  Step 1: Generate Δ_i uniformly from $\Delta \in \mathbf{\Delta} : \overline{\sigma}(\Delta) \leq \gamma$.
  Step 2: If $A + B\Delta_i C$ is unstable then let $\gamma = \overline{\sigma}(\Delta_i)$.
  Step 3: If $i < iterations$ then let $i = i + 1$ and go to step 1.
  Step 4: End and return γ.
- Adaptive Random Search II (ARS II)
  Inputs: N,m,p,$iterations$,$\overline{\gamma}$
  Initialization: Let $\gamma = \overline{\gamma}$, let $i = 1$ and evaluate the cost function, J at N points which are generated uniformly from $\Delta \in \mathbf{\Delta} : \overline{\sigma}(\Delta) \leq \gamma$.
  Step 1: Select the best m points and about each of them, uniformly generate p points within a user-specified range given by $radius_i$.
  Step 2: Evaluate $J(\Delta_j)$ $j = 1, \ldots, mp$ and let $\gamma = \min_{j=1,\ldots,mp}(J(\Delta_j))$
  Step 3: If $i < iterations$ then let $i = i + 1$ and go to step 1.
  Step 4: End and return γ.

When seeking the minimum of the norm-based cost function, the efficiency of the ARS II algorithm can be improved by limiting the radius of each uniform sampling so that during the iterations the samples will not have norm greater than γ. This is similar to the modification of the crude random search to get ARS I. The radius is user-specified for each iteration.

3.3 Adaptive Partitioned Random Search (APRS)

An adaptive partitioned random search divides the input space into subregions and performs several smaller searches. The version of APRS presented here is from [23]. In this approach, the goal of finding the subregion with greatest expectation of containing the maximizer is abandoned and replaced with the goal of finding the partition that provides the greatest expected improvement from the current best estimate of the maximum. This expectation is called the promising index. Partitions with greater promising indexes are partitioned further while the remaining, inferior are lumped into one partition called the surrounding region. The algorithm of [23] for finding a global maximum in a rectangular input space is given below.

- Adaptive Partitioned Random Search (APRS)
  Inputs: p_0, p, p_{index}, $best_p$, $shrink_factor$, $tolerance$
  Initialize: Partition the input space into p_0 smaller regions.
  Step 1: Take p_{index} samples from each partition and compute their promising indices.

Step 2: Select the best $best_p$ partitions based on the values of p_{index}.

Step 3: In each of the $best_p$ regions, find the point which gives the largest function value and use that point as the center of a new rectangular partition that is proportional to the input space. After each iteration, the length of each of the sides of this partition is decreased by a factor of $shrink_factor$. In addition to the $best_p$ regions, another rectangular partition is made around the best point that wasn't in the $best_p$ partitions.

Step 4: Partition each of the $best_p$ partitions into p partitions so that the total number of partitions is $best_p \times p + 1$.

Step 5: If the largest length of any dimension of a partition is larger than $tolerance$ then go to step 1.

Step 6: End and return the maximum.

Note that the partitions are not necessarily disjoint in this algorithm. This eases some of the required data handling and also makes handling the non-rectangular surrounding region easier. The author suggests using four points per partition to estimate each promising index. More points will provide a better estimate but will also require more computation effort. The shrinking factor is recommended to be closer to one for problems of higher dimension or greater "roughness".

3.4 Controlled Random Search (CRS)

The controlled random search is another direct search technique that comes in many varieties. The first step is to generate a group of random points over the entire search space. In general, the strategy is to replace the worst points of the group with better points through a selected operation. The Nelder-Mead simplex method may be considered as one form of CRS. In [1], a numerical comparison of several CRS algorithms is presented and a new CRS algorithm is introduced. In light of the positive results of the algorithm presented in [1], it is used in this study.

- Controlled Random Search

  Inputs: N, δ , $tolerance$

  Initialization: Uniformly generate a group, A of N points and evaluate them. Designate the best and worst points and their function values by x_1, x_N, J_1, and J_N respectively.

  Step 1: Randomly choose two distinct points x_2 and x_3 from $A \backslash x_1$.

  Compute $x_p = (x_{p_1} x_{p_2}, \ldots, x_{p_n})$ where n is the dimension of the input space,

$$x_{p_i} = \frac{1}{2}\left[\frac{(x_{2_i}^2 - x_{3_i}^2)f(x_1) + (x_{3_i}^2 - x_{1_i}^2)f(x_2) + (x_{1_i}^2 - x_{2_i}^2)f(x_3)}{(x_{2_i} - x_{3_i})f(x_1) + (x_{3_i} - x_{1_i})f(x_2) + (x_{1_i} - x_{2_i})f(x_3)}\right] \quad (11)$$

  and $x = (x_1, x_2, \ldots, x_n)$

Step 2: If x_p is not in the feasible set then go to step 1. Otherwise, evaluate $f_p := f(x_p)$. If $f_p \geq f_N$ go to step 1.

Step 3: Replace worst point x_N with x_p. If $f_p < f_1$ then go to step 5.

Step 4: Repeat this step M times: Choose a new point $x_p = (x_{p_1} x_{p_2}, \ldots, x_{p_n})$ from a scaled β-distribution with the mean in each dimension being the ith co-ordinate of the best point x_1 and the standard deviation given by

$$s = \delta d \tag{12}$$

where

$$d_i = |x_{1_i} - x_{N_i}| \; i = 1, \ldots, n. \tag{13}$$

Evaluate f_p. If $f_p < f_N$ then replace x_N with x_p.

Step 5: If $f_N - f_1 < tolerance$ then end. Otherwise go to step 1.

In step 1, points that are potentially better are generated by doing a weighted quadratic fit between three points in A, one of which is the best point x_1. This step constitutes the global part of the search. It becomes more local as the points in A come together. Step 4 is a dedicated local search about the current best point. The number of points , N, in A is user-defined and a value of $10(n+1)$ is suggested.

3.5 Shuffled Complex Evolution (SCE)

The so-called Shuffled Complex Evolution (SCE) approach of [7] combines ideas of APRS and CRS. The APRS part of the algorithm separates the group of points into several partitions and for a certain period of time the partitions are treated independently (until the points are shuffled). The CRS part of the algorithm uses the Nelder-Mead [13] simplex algorithm to replace the worst points in each of the partitions. A simplex is a set of points in the input space which don't all lie in a subspace of lower dimension than the input space. The points in the simplex are moved by operations called reflection, contraction, and mutation which are described in the outline of the evolution algorithm. While some approaches attempt to avoid the trap of local minima by starting at multiple initial conditions, this approach starts several simplex searches ("complex evolutions") simultaneously. In addition, after each simplex has gone through a specified number of evolutions, all of the simplexes are "shuffled" by combining all of them and then repartitioning them into new simplexes based on the function values of each of the points. A simplex step is interpreted as an evolutionary step of a population while shuffling is interpreted as communication and co-operation between populations. The procedure can be found in [7] and is outlined below.

- Shuffling

Inputs: s (such that $s \geq n+1$ where n is the dimension of the input space), p, ϵ

Initialization: Let i=1 and evaluate the cost function at sp points where s is the number of samples and p is the number of complexes (simplexes).

Step 1: Rank the points in the order of increasing cost.

Step 2: Partition the points into p complexes by letting $complex_i$ contain the points $p(k-1)+i$ for $k = 1, 2, \ldots, s$.

Step 3: Evolve $complex_i$ for $i = 1, 2, \ldots, p$ (algorithm shown below).

Step 4: Combine all of the points of the p complexes.

Step 5: If all points do not lie in a hypercube which has sides of length ϵ then go to step 2.

Step 6: End and return the point with the lowest cost.

- Evolution

Inputs: α, β, $c_{reflect}$, $c_{contract}$, $complex_i$

Initialize: Let i=1 and j=1.

Step 1: Form a simplex by randomly selecting $n+1$ points from the s-point complex with a triangular probability distribution (described later).

Step 2: Find the worst point x_w of the simplex and the centroid $\overline{x}$ of the remaining points.

Step 3: (Reflect) Compute the reflection

$$x_r = (1 + c_{reflect})\overline{x} - c_{reflect} x_w \tag{14}$$

where $c_{reflect} \in (0, 1]$. If $J(x_r) < J(x_w)$ then replace x_w with x_r and go to step 6.

Step 4: (Contract) Compute the contraction

$$x_c = (1 - c_{contract})\overline{x} + c_{contract} x_w \tag{15}$$

where $c_{contract} \in (0, 1]$. If $J(x_c) < J(x_w)$ then replace x_w with x_c and go to step 6.

Step 5: (Mutate) Replace x_w with a randomly generated point x_m.

Step 6: If $i < \alpha$ then let $i = i + 1$ and go to step 2. Otherwise, let $i = 1$ and go to step 7

Step 7: If $j < \beta$ then let $j = j + 1$ and go to step 1. Otherwise, end and return the modified complex.

Randomness is introduced into this algorithm in several ways. One is in the way a failed contraction is handled (i.e. $J(x_c) > J(x_w)$). In this algorithm, mutation occurs and the worst point is replaced by a randomly generated one. By contrast, in the original Nelder-Mead algorithm a deterministic step called shrinkage takes place. In the shrinkage step, all of the points, besides the best one, are moved closer to the best point. Randomness is also introduced in step 1 of the evolution algorithm. A triangular density $\rho_i = \frac{2(s+1-i)}{s(s+1)}$, $i = 1, 2, \ldots, s$ assigns a probability, ρ_i, of the ith point of the complex being chosen. The points are numbered in order of increasing function evaluation and so the point with the lowest cost has the highest chance of being selected for evolution. In

this way, all points in the complex have a chance to participate in evolution, but it is the better points that will participate more often. Finally, randomness can enter when an infeasible point is picked due to reflection or contraction. In [7] it is suggested that a point randomly chosen from the smallest hypercube that contains the complex replace the infeasible one.

The parameters of this approach are s, p, $c_{reflect}$, $c_{contract}$, α, and β. Nelder and Mead suggested 1 and 0.5 for $c_{reflect}$ and $c_{contract}$ respectively. More complexes, p, or more points per complex, s, should generally improve robustness of the algorithm but will also decrease efficiency. A higher β may also increase robustness at the expense of decreased efficiency while a higher α may decrease robustness since more worst-point replacements occur before a new simplex is chosen from the complex. The relationships between these parameters and robustness and efficiency are examined in [8] and are based on numerical experiments.

3.6 Convergence Results

Almost all of the algorithms presented lack convergence results. Only the crude random search and ARS I are guaranteed to converge in probability to the global optimum as the number of function evaluations increases. On the other hand, ARS II is not even guaranteed to converge to a local optimum. In [1], it is stated that the CRS algorithm is totally heuristic and lacks theoretical convergence properties. According to Subrahmanyam in [22], convergence theorems for the Nelder-Mead algorithm alone are practically nonexistent even though the algorithm has was presented in 1965. Some very recent work in [11] has proven the convergence of the algorithm for strictly convex functions in one dimension and shown the difficulty of finding proofs in higher dimensions. Given this, it is unlikely that any theoretical convergence results for the SCE algorithm exist.

Tests for the efficiency of the algorithms have mainly been done through the use of numerical experiments. In [1], the proposed CRS algorithm is compared with other CRS algorithms on a set of benchmark problems using numerical experiments. In [16], they are used to compare the Nelder-Mead algorithm with other direct methods and in [17] they are used to propose modifications to the algorithm to improve its efficiency. Interestingly enough, several studies have provided different conclusions. Some claim that the simplex method is inefficient and suffer when the dimension of the input space is high while others claim that it is competitive with other methods such as that of Powell [18] especially in higher dimensions. Schwefel in [21] has performed a comprehensive numerical study involving many algorithms on a large number of different problems. His results tend to show that the simplex algorithm is very inefficient, especially as it comes close to a solution.

Due to the lack of theoretical results in the literature as well as the (perhaps grudging) use of numerical experiments to make statements about the quality of an algorithm, the focus of the work in this study is to compare the algorithms by their performance on a couple of robust analysis problems rather than investigate their theoretical properties.

4 Numerical Results and Discussion

Two real μ examples are presented here. In each case, the system $M = (A, B, C)$ is LTI and stable and we seek the smallest $\Delta \in \mathbf{\Delta}$ which destabilizes $A + B\Delta C$. The solution to this problem cannot be computed analytically, but there are methods which compute conservative bounds on γ_{opt}. The μ-Tools toolbox in MATLAB contains such algorithms.

4.1 Method of Comparison

Comparing the performance of the different algorithms is no easy task. Along with the solutions, the number of function calls and flops are recorded. The number of flops per trial is an especially useful measure when the two cost functions are compared. Because calling the unidirectional perturbation-based cost function requires more computational effort than calling the norm-based cost function, fewer calls of the former is not necessarily enough to deem it superior. The average time each algorithm takes per trial can be a volatile measure since it is influenced by other loads on the computer. However, under controlled conditions, time measurements can be useful since flops do not capture the overhead (ex. sampling and checking of conditional statements) associated with some of the algorithms. Some general statements about execution time are made in the course of the following discussions. All trials were run on a DELL 200 MHz Pentium Pro machine.

The APRS algorithm contains both C and MATLAB code and so flops could not be counted. This also renders execution time potentially unsuitable for comparison purposes. For interest, however, execution times are mentioned along with the number of function evaluations.

4.2 Real μ Missile Controller Example

The LTI system, M, comes from [20] and represents the closed-loop linearized model of a bank-to-turn, air-to-air missile and a $\mathcal{H}_\infty$ controller. It is stable, has 7 states and 13 inputs and outputs. Because the number of states

is relatively low, the unidirectional perturbation-based cost function can be evaluated as well as the norm-based one.

The upper and lower bounds found by the μ-Tools toolbox in MATLAB show that γ_{opt} must lie in the interval $[2.3129, \infty)$. Computation time for these bounds was about nine minutes.

4.2.1 Random Multistart

Although the initial points for this algorithm are selected randomly, the algorithm is essentially a deterministic one as all further points are selected by solely deterministic rules. Each result in the trials below is generated by selecting N points and using each of them as initial conditions for Rosenbrock's algorithm. The best of the N runs is recorded.

Table 1
Results of the multistart algorithm using the norm-based cost

	N=25		N=40	
Trial	γ_{opt}	Function Calls	γ_{opt}	Function Calls
1	2.7592	39103	2.7592	59671
2	3.7097	34059	2.5576	62687
3	2.5576	39623	2.5433	59684
4	2.6592	39883	2.7348	59749
5	2.5433	36308	2.8956	67783
Average	2.8458	37795	2.6981	61915
Average Flops	4.07×10^8		6.65×10^8	

Table 2
Results of the multistart algorithm using the unidirectional perturbation-based cost

	N=2		N=5	
Trial	γ_{opt}	Function Calls	γ_{opt}	Function Calls
1	2.3591	1614	2.3591	7012
2	2.3591	1653	2.3591	8078
3	2.3591	2017	2.3591	8936
4	2.3673	3707	2.3591	5062
5	2.4329	3590	2.3673	7519
Average	2.3755	2516	2.3607	7321
Average Flops	5.38×10^9		1.19×10^{10}	

The most striking observation here is that the algorithm performs much better on the unidirectional perturbation-based cost. The number of flops required to find a solution with this cost function is an order of magnitude higher than required by the norm-based cost function. However, the time increase was only about a factor of two. Since each trial with the unidirectional perturbation-based cost function takes only about five minutes when $N = 5$ and the quality of the solutions is much higher, the extra computational cost is worthwhile.

4.2.2 Pure and Adaptive Random Searches I & II

The results of the crude and adaptive random searches ARS I and ARS II are provided below. Two sets of trials are performed for ARS II. In the first case, $N = 65$ and $P = 5$ while in the second, $N = 130$ and $P = 10$. In both cases, $M = 13$ and 300 iterations are performed.

Table 3
Results of the crude and ARS algorithms using the norm-based cost

	Crude	ARS I	ARS II (P=5)	ARS II (P=10)
Trial	γ_{opt}	γ_{opt}	γ_{opt}	γ_{opt}
1	3.2969	2.6844	2.4502	2.3977
2	2.9838	2.6518	2.3680	2.3663
3	3.1466	2.6369	2.3693	2.3824
4	3.1403	2.6803	2.4418	2.3766
5	3.0302	2.6135	2.4048	2.3809
Average	3.1196	2.6534	2.4068	2.3808
Function Calls	35000	35000	19500	39000
Average Flops	3.79×10^8	3.99×10^8	2.21×10^8	4.32×10^8

Table 4
Results of the crude and ARS algorithms using the unidirectional perturbation-based cost

	Crude	ARS II (P=5)	ARS II (P=8)
Trial	γ_{opt}	γ_{opt}	γ_{opt}
1	2.5306	2.4367	2.4390
2	2.5428	2.3650	2.3761
3	2.6026	2.3953	2.3666
4	2.5637	2.3710	2.3668
5	2.5654	2.4354	2.3658
Average	2.5610	2.4007	2.3829
Function Calls	35000	19500	31200
Average Flops	5.24×10^{10}	2.89×10^{10}	4.57×10^{10}

With both cost functions, ARS II performs the best. However, in terms of quality of solutions, the crude random search performs much better on the unidirectional perturbation-based cost while the ARS II algorithm performs similarly on both considering the quality of solutions for the number of function evaluations that are used.

Figure 3 shows the estimated value of γ_{opt} as each of the algorithms progresses.

4.2.3 Adaptive Partitioned Random Search

The APRS algorithm is applied with 32 and 64 partitions. For each case, 4 and 8 points are taken per partition in order to estimate the promising indices. A shrinking factor of 0.9 is used.

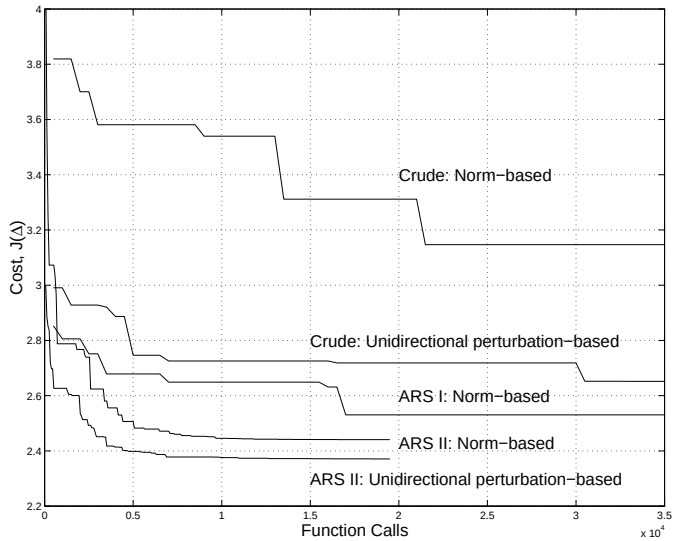

Fig. 3. Progression of best estimate of γ_{opt}.

Table 5
Results of the APRS algorithm using the norm-based cost

	32 Partitions (4)	32 Partitions (8)	64 Partitions (4)	64 Partitions (8)
Trial	γ_{opt}	γ_{opt}	γ_{opt}	γ_{opt}
1	2.4333	2.3801	2.4153	2.3926
2	2.4496	2.4373	2.4006	2.4046
3	2.3709	2.4051	2.4199	2.3994
4	2.5615	2.4944	2.4089	2.3758
5	2.4758	2.3800	2.4108	2.3861
Average	2.4582	2.4194	2.4110	2.3917
Function Calls	11744	23488	23136	46272
Average Time	7 min	15 min	14 min	26 min

Increasing the number of partitions improves solutions as does increasing the number of points taken per partition. Doubling the number of partitions seems to have a slightly greater effect than doubling the number of samples taken per partition to estimate the promising index. As will be seen in the following sections, this method performs comparably with the SCE method on the norm-based cost function. Also, using the unidirectional perturbation-based cost function improves the quality of the solutions but not as much as it does with the CRS and SCE algorithms.

4.2.4 Controlled Random Search

Five trials of the controlled random search are applied to the problem with different sizes of set A. The CRS algorithm performs roughly as well on both cost functions though more function evaluations are used for the norm-based

Table 6
Results of the APRS algorithm using the norm-based cost

	32 Partitions (4)	32 Partitions (8)	64 Partitions (4)	64 Partitions (8)
Trial	γ_{opt}	γ_{opt}	γ_{opt}	γ_{opt}
1	2.4510	2.3882	2.3942	2.3712
2	2.3707	2.3855	2.3819	2.3714
3	2.3834	2.3859	2.3674	2.3794
4	2.4058	2.3735	2.3885	2.3989
5	2.3759	2.3919	2.3915	2.3896
Average	2.3974	2.3850	2.3847	2.3821
Function Calls	9764	19528	19236	38472
Average Time	16 min	33 min	30 min	60 min

cost function. However, the unidirectional perturbation-based cost function performs two orders of magnitude more flops when $N = 10(n+1)$. Trials of $N = 2(n+1)$ were performed for the latter cost function in an attempt to bring the flop count to a level comparable to that when the former cost function is used. The quality of the answers is significantly poorer in this case.

The time to execute one trial with the norm-based cost function is several times longer for the CRS than for the other algorithms. This is possibly due to the overhead required to find new points that are an improvement over the worst point. To save on computation time, when the sampling procedure finds a new point that is outside of the hypercube of radius 5, the point is rejected and a new sample is generated. This overhead consumes time but does not register in the number of flops.

Table 7
Results of the CRS algorithm using the norm-based cost

	$5(N+1)$		$10(N+1)$	
Trial	γ_{opt}	Function Calls	γ_{opt}	Function Calls
1	2.3609	36757	2.3603	52058
2	2.3676	29595	2.3627	30793
3	2.3752	30178	2.3621	35358
4	2.3614	33859	2.3606	46912
5	2.3722	31064	2.3604	44228
Average	2.3675	32291	2.3612	41870
Average Flops	3.66×10^8		4.92×10^8	

Table 8
Results of the CRS algorithm using the unidirectional perturbation-based cost

	$2(N+1)$		$10(N+1)$	
Trial	γ_{opt}	Function Calls	γ_{opt}	Function Calls
1	2.3944	3972	2.3594	14308
2	2.4453	3349	2.4329	12369
3	2.3642	5754	2.3594	15786
4	2.4811	4619	2.3594	15047
5	2.3688	4049	2.3594	14592
Average	2.4108	4349	2.3741	14420
Average Flops	6.41×10^{9}		2.11×10^{10}	

4.2.5 Shuffled Complex Evolution

Two sets of five trials of the SCE algorithm are applied with five and ten complexes each. The rest of the parameter values chosen were $\alpha = 1$, $\beta = 2n+1$ and $s = 2n+1$ in accordance with the values recommended in [8].

Table 9
Results of the SCE algorithm using the norm-based cost

	5 Complexes		10 Complexes	
Trial	γ_{opt}	Function Calls	γ_{opt}	Function Calls
1	2.4511	14030	2.3687	28433
2	2.4061	14039	2.4081	27446
3	2.5001	15208	2.4415	27777
4	2.4414	14190	2.4408	27872
5	2.4493	12991	2.4169	27060
Average	2.4496	14092	2.4152	27718
Average Flops	1.5×10^{8}		3.1×10^{8}	

Table 10
Results of the SCE algorithm using the unidirectional perturbation-based cost

	5 Complexes		10 Complexes	
Trial	γ_{opt}	Function Calls	γ_{opt}	Function Calls
1	2.3598	13153	2.3608	27241
2	2.3610	13089	2.3594	27363
3	2.3611	13011	2.3593	27538
4	2.3605	13211	2.3595	27622
5	2.3752	13235	2.3638	27419
Average	2.3635	13140	2.3606	27437
Average Flops	2.11×10^{10}		4.35×10^{10}	

Again, for roughly the same number of function evaluations the results are significantly better when using the unidirectional perturbation-based function but the number of flops is two orders of magnitude higher.

4.2.6 Summary

Of all the algorithms, the multistart operating on the unidirectional perturbation-based cost function performed the best. The CRS algorithm performed well and did better on the unidirectional perturbation based cost than on the norm-based cost. It should be noted that on the norm-based cost function the time required for one trial of CRS was considerably longer than for one trial of any other algorithm. The ARS II algorithm was next and seemed insensitive to the choice of cost function. For such a simple algorithm with little extra overhead than to compute function evaluations, it performs competitively with the other algorithms. The SCE and APRS algorithms performed comparably well in terms of number of function evaluations and quality of solution. Not surprisingly, the crude, multistart (on the norm-based cost function), and ARS I algorithms are among the poorest. The norm-based cost function is shown to have a more complex surface when the results of the multistart algorithm on both cost functions are contrasted.

Using the unidirectional perturbation-based cost function definitely helps most of the algorithms to find a better solution. However, this improvement comes at the cost of more computation time and greater memory requirements. The greater cost is much more pronounced in the next example where the cost of evaluating the function analytically is prohibitive.

Although the increase in the number of flops for the unidirectional perturbation-based cost function is about two orders of magnitude, the increase in time is only about one order higher.

4.3 Medium-sized Real μ Example

The LTI system, M, obtained from [10] is stable and contains 55 states and 20 inputs and outputs, i.e., the number of real parameters is 20. Unlike the previous example, the unidirectional perturbation-based cost cannot be computed analytically using the expression in [9] due to the size of this problem. Even one attempt to evaluate the cost function results in an out of memory error. For a given Δ, this cost function can be approximated numerically by evaluating $\theta(M, \xi\Delta)$, where ξ ranges over $(0, \overline{\xi}]$ and $\|\Delta\|_\infty = 1$, and finding which ξ gives the smallest positive value of θ. Any Δ for which θ over the given range of ξ does not go positive is assigned a cost $\xi_{high} > \overline{\xi}$. As long as there exist destabilizing perturbations with size less than $\overline{\xi}$, this is fine. This procedure approximately finds the smallest destabilizing perturbation along a direction, which is what the unidirectional perturbation-based cost computes. It is assumed that θ monotonically increases and a bisection search is used to find the desired value of ξ.

According to the upper and lower bounds found by the μ-Tools toolbox in MATLAB, γ_{opt} must lie in the interval $[0.31591, \infty)$. These bounds were computed in about half an hour.

4.3.1 Random Multistart

Results of five trials of random multistart are given below with 10 and 20 starts. Because the multistart method was so successful in the previous example with the unidirectional-perturbation based cost function, it is approximated for this example by solving for the smallest destabilizing perturbation numerically via a bisection method. Each evaluation of the unidirectional perturbation-based cost function is really about 20 evaluations of the norm-based cost function.

Table 11
Results of the multistart algorithm using the norm-based cost

	N=10		N=20	
Trial	γ_{opt}	Function Calls	γ_{opt}	Function Calls
1	0.7423	23270	0.6562	53600
2	0.6562	30330	0.6864	41460
3	0.7294	19990	0.7061	42320
4	0.6864	21470	0.6371	46020
5	0.9497	20110	0.5617	45260
Average	0.7528	23034	0.6495	45732
Average Flops	4.16×10^{10}		8.25×10^{10}	

Table 12
Results of the multistart algorithm using the unidirectional perturbation-based cost

	N=3		N=5	
Trial	γ_{opt}	Function Calls	γ_{opt}	Function Calls
1	0.3303	4103	0.3289	7425
2	0.3960	4703	0.3280	6505
3	0.4064	4503	0.3280	6325
4	0.3280	4483	0.3385	5785
5	0.3280	3283	0.3303	7425
Average	0.3578	4215	0.3307	6693
Average Flops	1.44×10^{11}		2.39×10^{11}	

The results are much worse for the norm-based cost function which suggests that it contains many local minima. The results found by this approach applied to the unidirectional perturbation-based cost are the best from all the algorithms tested in this study. Of course, it is also the most computationally expensive to use. The solutions have to be verified since the cost function is only an approximation. Each trial with $N = 5$ took about 80 minutes on average.

4.3.2 Crude and Adaptive Random Searches I & II

The results for five trials of the crude random search, ARS I, and ARS II are given below. For ARS II, $P = 5$, $M = 20$, $N = 100$, and 300 iterations are performed.

Table 13
Results of the crude and ARS algorithms using the norm-based cost

	Crude	ARS I	ARS II (P=5)
Trial	γ_{opt}	γ_{opt}	γ_{opt}
1	0.6847	0.4980	0.3335
2	0.7196	0.5286	0.3385
3	0.6538	0.4962	0.3454
4	0.6707	0.5105	0.3560
5	0.7323	0.5006	0.3299
Average	0.6922	0.5068	0.3407
Function Calls	50000	50000	30000
Average Flops	9.02×10^{10}	9.12×10^{10}	5.54×10^{10}

Figure 4 shows the estimated value of γ_{opt} as each of the algorithms progresses. Because the crude search and ARS I use little information from previous function evaluations, they are very inefficient. However, the simple difference between the pure and adaptive random searches greatly improves the efficiency of the latter. Despite the simplicity of ARS II, it performs very well and is the second best of all the algorithms in this study in terms of the quality of the solutions for a given number of function evaluations.

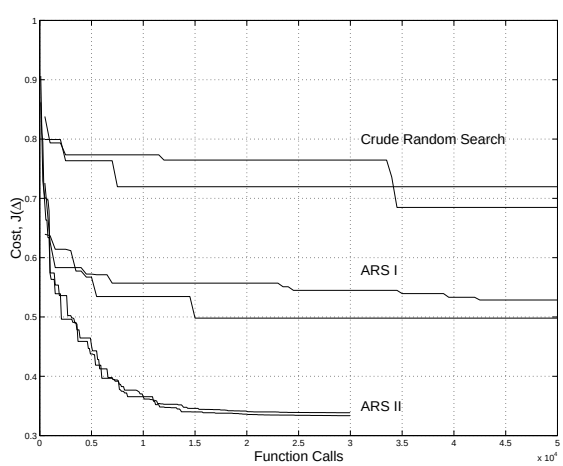

Fig. 4. Progression of the crude and adaptive random searches.

4.3.3 Adaptive Partitioned Random Search

As before, the APRS algorithm is applied with 32 and 64 partitions. Four or eight points are taken per partition in order to estimate the promising indices and a shrinking factor of 0.9 is used.

Table 14
Results of the APRS algorithm using the norm-based cost

	32 Partitions (4)	32 Partitions (8)	64 Partitions (4)	64 Partitions (8)
Trial	γ_{opt}	γ_{opt}	γ_{opt}	γ_{opt}
1	0.4477	0.3807	0.4053	0.3860
2	0.3724	0.3920	0.4102	0.3685
3	0.4588	0.3633	0.3512	0.3397
4	0.4634	0.3929	0.4030	0.3726
5	0.4721	0.3573	0.3866	0.3804
Average	0.4429	0.3772	0.3913	0.3694
Function Calls	12536	25072	24696	49392
Average Time	19 min	35 min	30 min	70 min

Unlike the previous example, doubling the number of samples taken to estimate the promising index rather than doubling the number of partitions appears to provide greater improvement. However, it is still the case that doubling either of them generally improves the quality of the solution.

4.3.4 Controlled Random Search

The results of the CRS algorithm are given below.

Table 15
Results of the CRS algorithm using the norm-based cost

	$10(n+1)$		$20(n+1)$	
Trial	γ_{opt}	Function Calls	γ_{opt}	Function Calls
1	0.4026	21343	0.4037	62818
2	0.3577	18911	0.3305	61399
3	0.4442	24854	0.4096	66143
4	0.3861	20667	0.3703	65649
5	0.4524	23081	0.3300	63458
Average	0.4086	21771	0.3688	63893
Average Flops	3.95×10^{10}		1.18×10^{11}	

Some very good solutions are found when$N = 20(n+1)$, but some poor solutions are found too. The number of function evaluations is also double that used by ARS II and SCE to get comparable results.

4.3.5 Shuffled Complex Evolution

Two sets of five trials of the SCE algorithm are applied with five or eight complexes. As before, the other parameter values chosen for the algorithm were $\alpha = 1$, $\beta = 2n + 1$ and $s = 2n + 1$.

Table 16
Results of the SCE algorithm using the norm-based cost

	5 Complexes		8 Complexes	
Trial	γ_{opt}	Function Calls	γ_{opt}	Function Calls
1	0.3525	22517	0.3486	37195
2	0.3521	22257	0.3388	35896
3	0.3701	22827	0.3535	35957
4	0.3445	22298	0.3485	35742
5	0.3519	22517	0.3558	36426
Average	0.3542	22441	0.3490	36243
Average Flops	4.14×10^{10}		6.68×10^{10}	

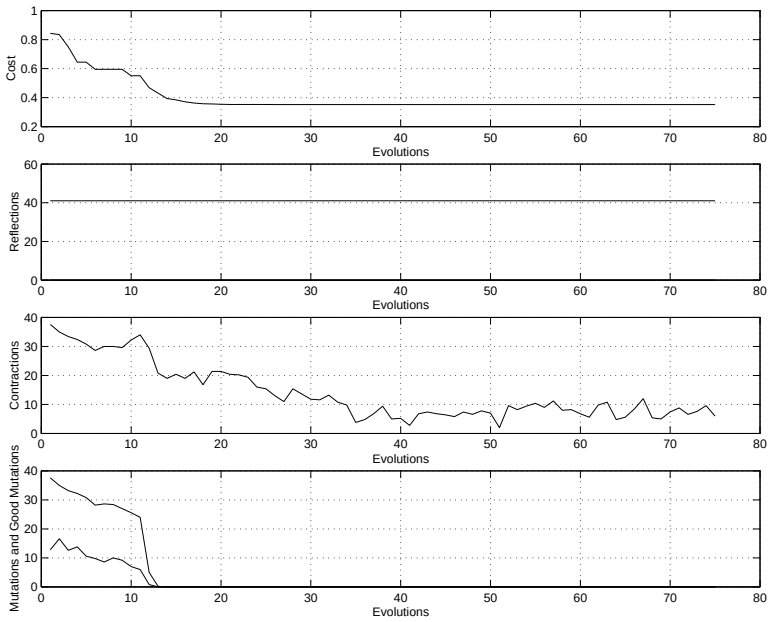

Fig. 5. Progression of best estimate of γ_{opt} for trial 1.

Figure 5 shows the estimate of γ_{opt} and the number of reflections, contractions, and mutations per evolution. The plot of mutations also includes a plot of how many of those mutations find a point that is better than the current worst point. The figure shows that in the early evolutions, many contractions and mutations are needed. However, just before the estimate of γ_{opt} settles down, no more mutations take place. The next few evolutions are spent in improving the estimate of γ_{opt} and afterwards, the rest are spent on collapsing all of the simplex points together so that the worst points move towards the best.

For comparison purposes, the mutation step is replaced by the shrinkage step

to make the SCE algorithm purely deterministic. The results are shown in Table 17.

Table 17
Results for the SCE algorithm with 5 complexes evolved a maximum of 100 times and a shrinkage step replacing the mutation.

Trial	γ_{opt}	Function Calls
1	0.473355	25388
2	0.433984	25473
3	0.416189	25560
4	0.426025	25591
5	0.478014	25373
6	0.558912	25543

With about the same number of function evaluations, the results are poorer than when mutation is allowed. The randomness may improve the robustness of the algorithm so that it can arrive at better points. Another reason for the poorer results may be that shrinkage drastically reduces the size of the simplex and so the simplex search becomes too localized prematurely. Mutations may or may not reduce the size of the simplex significantly. In fact, mutations may even increase the size of a simplex.

4.3.6 Summary

Again, the multistart algorithm on the unidirectional perturbation-based cost function found, in a couple trials, the best solutions of any of the algorithms but did so at the cost of many more flops. The ARS II algorithm generally performed very well on this problem both in terms of efficiency and quality of solutions. The SCE and CRS algorithms also generally performed well but for a similar number of flops, generally had slightly poorer solutions and took longer than ARS II due to the extra overhead required to find new points to sample. The APRS algorithm performed similarly with CRS and SCE with respect to the number of function calls and time required to complete a trial. Crude search and ARS I did poorly, as did the multistart algorithm on the norm-based cost function. It appears that the norm-based cost function has a large number of local minima.

The lower and upper bound computations of γ_{opt} in MATLAB take roughly the same amount of time as most of the randomized algorithms.

5 Conclusions

The main objective of this paper was to conduct computational experiments using randomized algorithms for robust stability analysis. We make some concluding remarks and address some work that remains to be done:

- Seven algorithms were used in this study. Of all of them, the multistart algorithm found the best answers when applied to the unidirectional perturbation-based cost function. However, when applied to the norm-based cost function, it was one of the worst algorithms. Algorithms ARS II, APRS, CRS, and SCE also performed very well and were not as sensitive to the choice of cost function giving only slightly worse results when the norm-based cost was used. Of those four algorithms, none was the best for both problems. For a fixed number of flops, the ARS II algorithm takes the shortest amount of time while the CRS algorithm takes the longest, sometimes significantly longer due to the extra overhead. When the cost function is simple, a line search algorithm is likely the best approach to take. However, when the cost function is more complex, a randomized algorithm may be more suitable. The random algorithm should be discerning about where to sample in order to improve its efficiency over a crude random search. The relative insensitivity of the randomized algorithms to the choice of cost functions along with the ease with which they were implemented shows the potential of these types of algorithms.
- Cost function formulation is an important problem. It seems that the best combination of algorithm and cost function is the multistart algorithm using the unidirectional perturbation-based cost. In general, the unidirectional perturbation-based cost function is demonstrated to be easier to optimize in terms of the number of function evaluations needed. However, the cost function requires much more effort to evaluate. This trade-off must be considered in light of the size of the example. For the missile problem, which is relatively small, the more expensive evaluations can be tolerated. In the second example, the price of evaluating the function is very high. In fact, the function must be approximated.
- It should be noted that one drawback of the optimization methods used in this study is that they provide only upper bounds on the solution. In contrast, a branch and bound technique has been investigated in [15] using upper and lower bounds to compute μ. Another branch and bound approach to compute minimum stability degree has been developed in [2]. The advantage of our formulation over the μ computation is that frequency gridding is not necessary. Also, the approach in [2] can be modified to compute the smallest destabilizing perturbation through a one parameter iteration of the branch and bound optimization program that computes the minimum stability degree. Thus, a direct comparison of our algorithms and the branch and bound based algorithms is not a straightforward matter. A very important next step is to devise good branch and bound algorithms for the

problem considered in this paper and comparing their performance with the randomized algorithms considered in this paper. This line of attack may lead to new algorithms for robust stability analysis that combine the best features of both types of algorithms.

References

[1] M. Ali, A. Torn, and S. Viitanen. A numerical comparison of some modified controlled random search algorithms. Technical Report 98, Turku Centre for Computer Science, March 1997.

[2] V. Balakrishnan, S. Balemi, and S. Boyd. Computation of the Minimum Stability Degree of Parameter-dependent Linear Systems by a Branch and Bound Algorithm. In *Proceedings of 1st IFAC Symposium on Design Methods of Control Systems*, pages 133–138, Zürich, Switzerland, Sept. 1991.

[3] B. R. Barmish and C. M. Lagoa. The uniform distribution: a rigorous justification for its use in robustness analysis. In *Proceedings of the 35th IEEE International Conference on Decision and Control*, pages 3418–3423, Kobe, Japan, 1996.

[4] M. Bazaraa, H. Sherali, and C. M. Shetty. *Nonlinear Programming - Theory and Algorithms*. Wiley & Sons, Inc., New York, second edition, 1993.

[5] R. Braatz, P. Young, J. Doyle, and M. Morari. Computational complexity of μ calculation. In *Proceedings of the American Control Conference*, pages 1682–1683, San Francisco, California, 1993.

[6] R. R. E. de Gaston and M. G. Safonov. Exact calculation of the multiloop stability margin. *IEEE Transactions on Automatic Control*, 33:156–171, 1988.

[7] Q. Duan, S. Sorooshian, and V. Gupta. Effective and efficient global optimization for conceptual rainfall-runoff models. *Water Resources Research*, 28(4):1015–1031, 1992.

[8] Q. Duan, S. Sorooshian, and V. Gupta. Optimal use of the SCE-UA global optimization method for calibrating watershed models. *Journal of Hydrology*, 158:265–284, 1994.

[9] M. Fu and R. Barmish. Maximal unidirectional perturbation bounds for stability of polynomials and matrices. *Systems & Control Letters*, 11:173–179, 1988.

[10] P. Khargonekar and A. Tikku. Randomized algorithms for robust control analysis and synthesis have polynomial complexity. In *Proceedings of the 35th IEEE International Conference on Decision and Control*, pages 3470–3475, Kobe, Japan, 1996.

[11] J. Lagarias, J. Reeds, M. Wright, and P. Wright. Convergence properties of the nelder-mead simplex algorithm in low dimensions. Technical Report 96-4-07, Lucent Technologies, May 1997.

[12] C. Marrison and R. Stengel. The use of random search and genetic algorithms to optimize stochastic robustness functions. In *Proceedings of American Control Conference*, pages 1484–1489, Baltimore, Maryland, 1994.

[13] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[14] A. Nemirovskii. Several NP-hard problems arising in robust stability analysis. *Math. of Control, Signals, and Systems*, 6:99–105, 1993.

[15] M. P. Newlin and P. M. Young. Mixed μ problems and branch and bound techniques. In *Proceedings of the 31st IEEE International Conference on Decision and Control*, pages 3175–3180, Tucson, Arizona, 1992.

[16] J. M. Parkinson and D. Hutchinson. A consideration of nongradient algorithms for the unconstrained optimization of function of high dimensionality. In F. A. Lootsma, editor, *Numerical Methods for Nonlinear Optimization*, pages 99–113. Academic Press, 1972.

[17] J. M. Parkinson and D. Hutchinson. An investigation into the efficiency of variants of the simplex method. In F. A. Lootsma, editor, *Numerical Methods for Nonlinear Optimization*, pages 115–136. Academic Press, 1972.

[18] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*, 7:155–162, 1964.

[19] L. Ray and R. Stengel. Stochastic robustness of linear time-invariant control systems. *IEEE Transactions on Automatic Control*, 36:82–87, 1991.

[20] C. Schumacher and P. Khargonekar. A comparison of missle autopilot designs using $\mathcal{H}_\infty$ control with gain scheduling and nonlinear dynamic inversion. In *Proceedings of the American Control Conference*, pages 2759–2763, Albuquerque, New Mexico, 1997.

[21] H. Schwefel. *Evolution and Optimum Seeking*. Wiley & Sons, Inc., New York, 1995.

[22] M. B. Subrahmanyam. An extension of the simplex method to constrained nonlinear optimization. *Journal of Optimization Theory and Applications*, 62(2):311–319, 1989.

[23] Z. B. Tang. Adaptive partitioned random search to global optimization. *IEEE Transactions on Automatic Control*, 39(11):2235–2244, November 1994.

[24] R. Tempo, E. W. Bai, and F. Dabbene. Probabilistic robustness analysis: explicit bounds for the minimum number of samples. In *Proceedings of the 35th IEEE International Conference on Decision and Control*, pages 3424–3428, Kobe, Japan, 1996.

[25] A. Yoon, P. Khargonekar, and K. Hebbale. Design of computer experiments for open-loop control and robustness analysis of clutch-to-clutch shifts in automatic transmissions. In *Proceedings of the American Control Conference*, pages 3359–3364, Albuquerque, New Mexico, 1997.

[26] P. M. Young, M. P. Newlin, and J. C. Doyle. Let's get real. *ASME, Dynamic Systems and Control Division (Publication) DSC*, 43:5–12, 1992.

[27] W. Zangwill. Minimizing a function without calculating derivatives. *Computer Journal*, 10:293–296, 1967.

[28] X. Zhu, Y. Huang, and J. Doyle. Genetic algorithms and simulated annealing for robustness analysis. In *Proceedings of the American Control Conference*, pages 3756–3760, Albuquerque, New Mexico, 1997.