Comparison of Algorithms for Including Equality Constraints in Signomial Programming

ACDL Technical Report TR-2017-1

Max M. J. Opgenoord[*] Brian S. Cohen[†] Warren W. Hoburg[‡]

August 31, 2017

Including equality constraints in signomial programming proves troublesome, because a straightforward difference of convex formulation for such a constraint results in a singular point. Signomial programs are usually solved by solving successive geometric programs, and the focus of this paper is on methods for approximating equality constraints in signomial programs as geometric programs. A general comparison between convergence characteristics of such methods is presented here, focusing on local approaches to signomial programming. The introduced methods locally approximate equality constraints as inequalities and Taylor approximations, and are compared to a recently developed algorithm from the literature, which claims to achieve global optimality for these types of problems. Trust region methods are proposed which perform comparably well to an algorithm from the literature yet are numerically much simpler to implement. Finally, one of these methods is used to solve a design problem for a solar unmanned aerial vehicle.

1 Introduction

Geometric programs (GPs) are optimization problems whose objective and constraint functions are convex in log-space. This feature makes them suitable for a range of efficient numerical solvers which also guarantee finding a global optimum, if one exists.[4] Geometric programming was introduced by Duffin and Zener in 1967,[9] and the theory behind it as well as its practical applications have received much attention since. The computational efficiency of convex optimization solvers – and thereby GP solvers – was improved dramatically by the introduction of interior point methods by Nesterov and Nemirovsky in 1994.[20] Geometric programming has been applied successfully to many engineering problems, such as chemical engineering,[23] statistics,[18] circuit design,[3,22] power control[6] and conceptual aircraft design.[11]

Signomial programs (SPs) are an extension of GPs that relaxes some of the restrictions of the GP restrictions to allow for a wider variety of applicable problems. However, because signomial programs are not necessarily convex in log-space, global optimality can no longer be guaranteed. Nevertheless active research in the field of signomial programming is ongoing, and is popular for

[*]Graduate student, Department of Aeronautical and Astronautical Engineering, Massachusetts Institute of Technology, Cambridge, MA. Corresponding author, mopg@mit.edu, http://web.mit.edu/mopg/www/.

[†]Graduate student, Department of Aeronautical and Astronautical Engineering, Massachusetts Institute of Technology, Cambridge, MA.

[‡]Assistant Professor, Department of Aeronautical and Astronautical Engineering, Massachusetts Institute of Technology, Cambridge, MA.

several applications because it extends the applicability of these types of optimization formulations to practical engineering problems. Signomial programming has been used on problems ranging from structural applications,[1] optimal control,[13] chemical engineering,[8] communication systems,[24] aircraft design,[14] and more.

Signomial programs can be solved iteratively as a sequence of GP subproblems, similar to difference of convex programming (DC). Forming these GP subproblems requires a GP approximation to the SP problem using some initial guess, where the initial guess for the next problem is the optimal point of the current problem. Equality constraints can be problematic for SPs since such GP approximations reduce the feasible space of an approximation to small regions or even singular points. Several researchers have proposed rapidly converging techniques for handling equality constraints.[15,25] However, oftentimes heuristic parameters are needed which require tuning for a specific problem.[25] Moreover certain algorithms are more suited for specific types of problems and are unable to solve others altogether. This fact has motivated a need to understand how the different algorithms compare and if a more universal approach can be applied. Such an algorithm can then be implemented in GPkit[5] – a Python package that is aimed at making GP and SP solution techniques available to engineering designers. Note that this paper considers only local approaches to Signomial Programming, although global approaches do exist.[17]

The goal of this paper is to (1) provide a comparison of several algorithms for handling equality constraints in a variety of signomial programs and (2) test these new methods on a range of illustrative test problems. A set of purely theoretic and one applied problem in aeronautical engineering for the sizing of a UAV are considered in this comparison. Comparison metrics include convergence properties, accuracy and computational efficiency of each of the algorithms. Parametric analysis is provided for those algorithms which employ heurstics to provide a sensible comparison. Initial starting points are also varied for the different test problems.

2 Background

A geometric program is a type of optimization problem where the objective and constraint functions have a specific form. These problems can be stated as

$$\begin{aligned} &\min_x f(x) \\ \text{subject to}\quad & g_j(x) \leq 1, \quad j = 1, ..., m \\ & h_j(x) = 1, \quad j = m+1, ..., p, \end{aligned} \tag{1}$$

where $f(x)$ and $g_j(x)$ are posynomials and $h_j(x)$ are monomials. In the context of geometric programming, a monomial is an expression of the form

$$m(x) = c x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}, \tag{2}$$

where $c > 0$, $(x_1, \ldots, x_n) > (0, \ldots, 0)$ is a vector of strictly positive variables, and $a_i \in \mathbb{R}$. A posynomial is a sum of one or more monomials and can be written as

$$p(x) = \sum_{k=1}^{K} c_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_n^{a_{nk}}, \tag{3}$$

where $c_k > 0$ and K is the number of terms in the posynomial. Note that a posynomial is strictly positive for $x_i > 0$, due to all coefficients being positive. A logarithmic change of variables is used to define a convex optimization problem, which can be solved efficiently using existing algorithms such as interior point methods. Consider the monomial equality constraint defined in Eq. (2). The logarithm of this function can be expanded into the sum

$$\log m(x) = \log c + a_1 \log x_1 + \cdots + a_n \log x_n. \tag{4}$$

Defining a new variable $y = (y_1, \ldots, y_n)$ such that $x_i = e^{y_i}, i = 1 \ldots n$, the above logarithm in Eq. (4) can be written in terms of the new variable y as

$$\log c + a_1 y_1 + \cdots a_n y_n = 0. \tag{5}$$

Eq. (5) is now an affine function of y. Similar techniques transform the objective functions and inequality constraints into log-sum-exp (LSE) functions.

Signomial programming is an extension of geometric programming where the objective and constraint functions are signomials. A signomial is defined exactly as a posynomial in Eq. (3) except that $c_k \in \mathbb{R}$ and hence the coefficients can be negative. A logarithmic change of variables is no longer guaranteed to result in a convex function. Following this approach, signomial programming algorithms can guarantee only locally and not globally optimal solutions.

Signomial programs can be solved iteratively as a sequence of GPs, similar to difference of convex programming. A formal statement for a signomial program can be written as,

$$\begin{aligned} &\min_x f(x) \\ \text{subject to} \quad & g_j(x) \leq 0, \quad j = 1, ..., m \\ & h_j(x) = 0, \quad j = m+1, ..., p \end{aligned} \tag{6}$$

where now $g_j(x)$ and $h_j(x)$ are signomials, and $f(x)$ is a posynomial.

The signomial inequality constraints $g_j(x) \leq 0$ can be rewritten in the form $p_1 \leq p_2$ where p_1 and p_2 are posynomials. The inequality constraint $g_j(x) \leq 0$ is split into a posynomial, $g_j^+(x)$, and a negynomial, $g_j^-(x)$, where $g_j^-(x)$ has strictly negative coefficients. The resulting constraint, $g_j^+(x) \leq -g_j^-(x)$ then has posynomials on both sides.

A common approach is to *monomialize* the right-hand side of the constraint to transform this inequality into a GP-compatible constraint. The monomialization of p_2 is made using a first order Taylor appoximation at the current point x^k and is denoted by $\hat{p}_2(x^k)$. Once the approximation has been made, dividing through by the monomial $\hat{p}_2(x^k)$ yields the inequality constraint

$$\frac{p_1}{\hat{p}_2(x^k)} \leq 1, \tag{7}$$

where the left hand term is now also a posynomial. When the signomial program only has inequality constraints, this method renders the subproblem a geometric program. The signomial program is then solved by optimizing a series of GP subproblems whose constraints have been monomialized. The solution to the current GP subproblem at x^k then yields the next iterate x^{k+1}. This is summarized in Algorithm 1.

Note that x^0 may not be feasible for the GP subproblem or even the original SP. In that case, a so-called feasibility solve is performed to find either a feasible point or to determine that the problem is in fact infeasible. In this feasibility problem all constraints are relaxed to find a feasible point for the original geometric problem. Thus, in the feasibility problem one tries to find out whether the constraints are mutually consistent, and step towards a feasible point, if it exists. The feasibility solve is therefore formulated as[4]

$$\begin{aligned} &\min_{x, s_i} \quad s_1 \cdots s_m \\ \text{subject to} \quad & g_i(x) \leq s_i, \quad i = 1, \ldots, m \\ & h_i(x) = 1, \quad i = 1, \ldots, p \\ & s_i \geq 1, \quad i = 1, \ldots, m, \end{aligned} \tag{8}$$

Algorithm 1 Algorithm for Solving SPs with a signomial inequality constraint $p_1(x) \leq p_2(x)$.

```
Input: Initial guess $x^0$
Output: Approximate solution $x^*$ to the signomial program within specified tolerance $\epsilon$
1: Initialization $k = 0$
2: while $\frac{|f(x^k)-f(x^{k-1})|}{f(x^k)+f(x^{k-1})} > \epsilon$ do
3:     $m_2(x, x^k) = \hat{p}_2(x, x^k)$
4:     $\hat{g}(x, x^k) = \frac{p_1(x)}{m_2(x)}$
5:     $x^* = \arg\min_x f(x^k)$
           subject to $\hat{g}(x, x^k) \leq 1$
6:     $x^{k+1} = x^*$
7:     $k = k + 1$
8: end while
```

where the design variables are x (same as the original problem), and the slack variables s_i are added. We solve for the optimum $\bar{x}$ and $\bar{s}_i$. If every $\bar{s}_i = 1$ then the original GP problem is feasible, and if any $\bar{s}_i > 1$, the original GP problem is infeasible. Therefore, if the original problem is feasible, but the initial point is not, a feasible point $\bar{x}$ is found quickly and the iteration continues from there.

For a problem with only inequality constraints, this technique is quite popular, it works well in practice, and again only suffers from the fact that only a locally optimal solution can be guaranteed.[4] It should also be noted that the monomialization results in a conservative approximation of the feasible region of the original SP, i.e. the feasible region of each GP subproblem is contained in the feasible region of the original SP. The solution x^k at each iteration – if it exists – is therefore guaranteed to be feasible for the original problem. The local optimum found by the signomial program is therefore also guaranteed to be a *feasible* local optimum, but it may not be globally optimal.

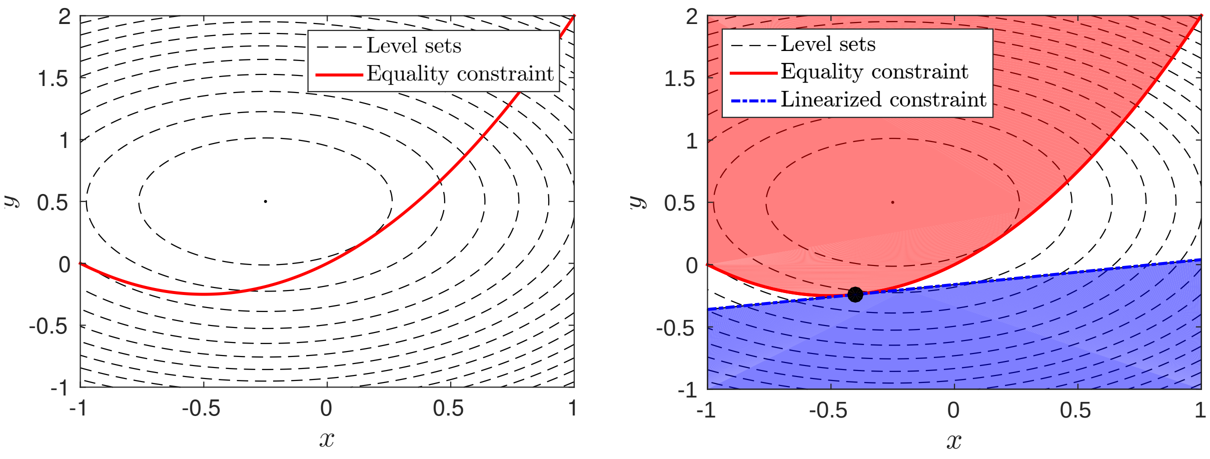
Using this approach for equality constraints, however, results in several problems. In the previous case, each GP subproblem is convex in log-space. An equality constraint of the form $p_1 = p_2$ is usually rewritten as the intersection of two constraints: $p_1 \leq p_2$ and $p_1 \geq p_2$. Using the same method as before, p_2 is monomialized while p_1 is kept as a posynomial form to yield $p_1 \leq \hat{p}_2$. Similarly, the latter inequality can be rewritten as $p_2 \leq \hat{p}_1$. Unfortunately this method results in a single feasible point. To illustrate this issue, we will use an example from sequential convex programming.[2] This is similar to signomial programming except for the conversion to log-space.

Consider the problem

$$\begin{aligned} \min_{x,y} \quad & \left(x + \frac{1}{4}\right)^2 + \left(y - \frac{1}{2}\right)^2 \\ \text{subject to} \quad & x^2 + x - y = 0, \end{aligned} \tag{9}$$

where the feasible set is shown in Fig. 1(a) as a parabolic line representing the equality constraint.

One approach to solve this problem is to split the equality constraint $x^2+x = y$ into $x^2+x \geq y$ and $x^2 + x \leq y$. The latter is convex but the former is not. In order to convert this into a convex problem, the nonconvex constraint can be linearized at the current iterate to yield a convex affine approximation. However, the intersection of the two sets defined by the convex and linearized affine constraints is now a single feasible point as shown in Fig. 1(b): the first convex constraint is satisfied above the curve $y = x^2+x$ and the linearized constraint is satisfied below the blue line, which is the same problem we have described for signomial programming.



(a) Original problem (b) One constraint linearized around $x = -0.4$

Figure 1: Example of single feasible point when one side of the constraint is linearized.

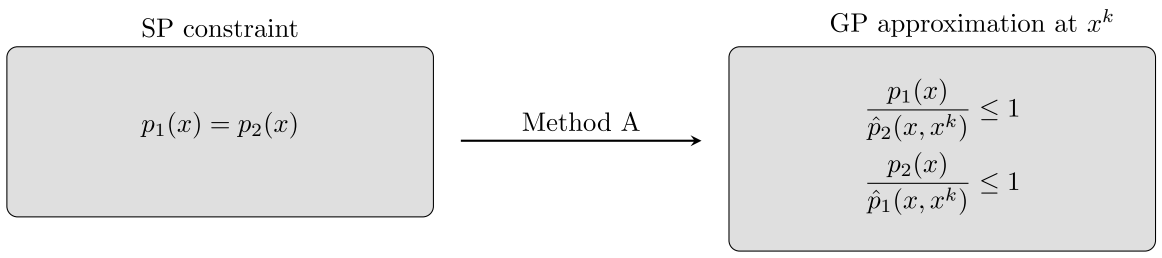
In practice, SPs are sometimes solved by replacing the equality constraint with a single inequality constraint. Having knowledge about which side of the equality constraint is pressured during the optimization can allow a designer to implement a tight inequality constraint. This knowledge, however, is not always known *a priori*. For problems where this knowledge is not available, we need a robust method to implement equality constraints in signomial programs.

3 Methodology

In this section several different methods of enforcing equality constraints for SPs are discussed. We compare these against a recent method from the literature developed by Xu in 2014.[25]

3.1 Linearizing only one of the two constraints at a time (Method A)

The first method we investigate is the original – and simplest – way of enforcing a signomial equality constraint. Fig. 2 deals with the case $p_1(x) = p_2(x)$, where $p_1(x)$ and $p_2(x)$ are posynomials. First, the posynomial equality constraint is split into two inequality constraints: $p_1(x) \leq p_2(x)$ and $p_1(x) \geq p_2(x)$. $p_2(x)$ is then monomialized in the former constraint, while $p_1(x)$ is monomialized in the latter constraint.



SP constraint
$p_1(x) = p_2(x)$
Method A
GP approximation at x^k
$\frac{p_1(x)}{\hat{p}_2(x, x^k)} \leq 1$
$\frac{p_2(x)}{\hat{p}_1(x, x^k)} \leq 1$

Figure 2: Illustration of changes to the original SP constraints to form the GP subproblem using Method A for one posynomial equality constraint.

As mentioned in Section 2, such an approach yields a GP with a single feasible point. While this approach therefore does not seem robust, some cases exist where it can be effective, especially when combined with a feasibility solve.[4] When starting from an infeasible initial point, a feasibility problem should be solved first. That feasibility problem has more than one single

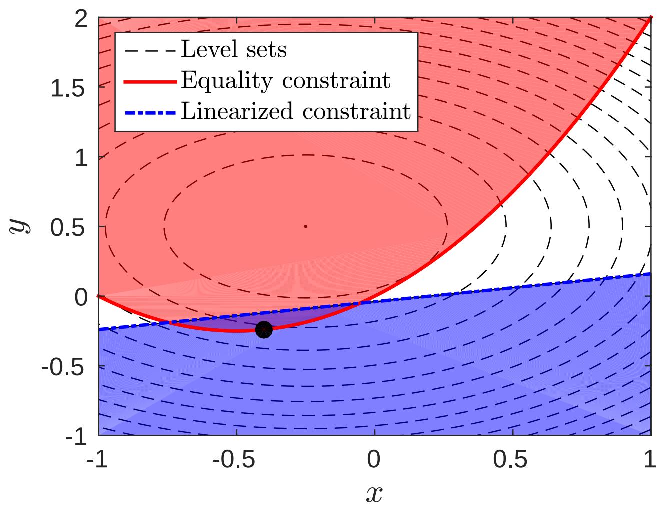


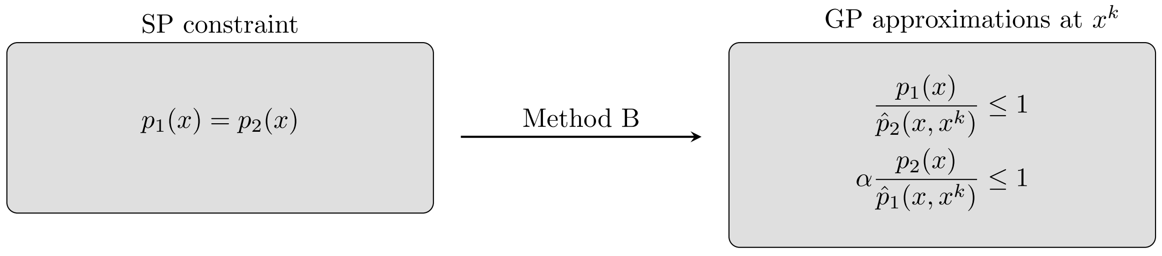
Figure 3: Design space for fully linearized constraint. Feasible design space is now between the parabolic line and the line above the linearization point (the dark blue region).

feasible point, because the constraints are relaxed with the slack variables s_i. This allows us to take a large step towards feasibility during the first iteration.

3.2 Linearizing and relaxing one constraint (Method B)

Another method is to expand the feasible region of the inequality constraints derived from the equality constraint. Thus, the equality constraint $p_1(x) = p_2(x)$ is transformed into $p_1(x) \leq p_2(x)$ as before, and $p_1(x) \geq \alpha p_2(x)$, where $\alpha < 1$. The scaling factor α dictates the size of the trust region. As is shown in Figure 3, the trust region now includes feasible points located on the parabolic line segment as well as slighty above the parabolic line segment (which are not feasible in the original SP). By keeping the trust region small, the number of such infeasible points is limited.

The approach for this method is illustrated in Fig. 4. This approach is the most similar of all methods presented in this paper to Xu's algorithm,[25] which is discussed in Section 3.5.



SP constraint
$p_1(x) = p_2(x)$
Method B
GP approximations at x^k
$\frac{p_1(x)}{\hat{p}_2(x, x^k)} \leq 1$
$\alpha \frac{p_2(x)}{\hat{p}_1(x, x^k)} \leq 1$

Figure 4: Illustration of changes to the original SP constraints to form the GP subproblem using Method B for one posynomial equality constraint.

3.3 Linearizing both constraints (Method C)

In this method, both sides of the derived inequality constraints are monomialized to yield a monomial equality constraint, which is illustrated in Fig. 5.

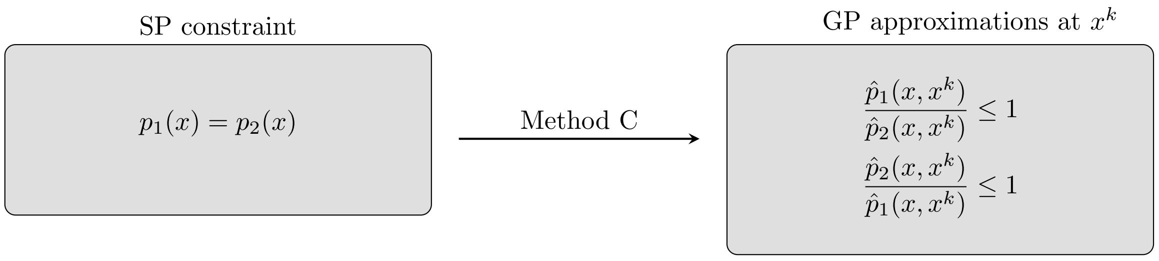

SP constraint
$p_1(x) = p_2(x)$
Method C
GP approximations at x^k
$\frac{\hat{p}_1(x,x^k)}{\hat{p}_2(x,x^k)} \leq 1$
$\frac{\hat{p}_2(x,x^k)}{\hat{p}_1(x,x^k)} \leq 1$

Figure 5: Illustration of changes to the original SP constraints to form the GP subproblem using Method C for one posynomial equality constraint.

The feasible region for the GP subproblem is now a line (in log-space) through the current iterate x^k. Note that this therefore allows for points on the line which are not feasible in our original problem, as illustrated in Fig. 6. However, even when the current iterate is an infeasible point, the monomialization for the next iterate will linearize the problem around a feasible point, in most cases that linearization point will lay on the original SP constraint.

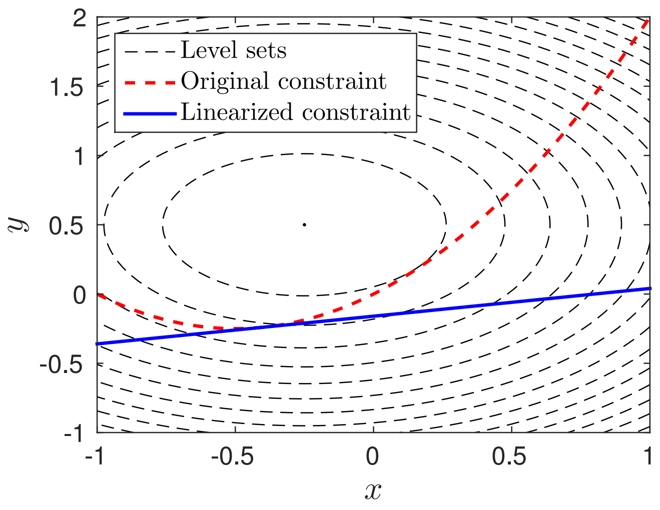


Figure 6: Design space for fully linearized constraint. Feasible design space is now on the blue line.

3.4 Linearizing both constraints with added trust region (Method D)

As illustrated in Fig. 6, Method C linearizes the derived inequality constraints such that the tangent line to the original equality constraint is considered feasible in the suproblem. This allows for a large infeasible region for the original SP, which may be problematic. Therefore, Method D utilizes a trust region to bound the points on the new feasible region (illustrated as a line segment in Fig. 7). For every design variable that is included in the equality constraint we add a trust region as follows

$$\alpha x_{i,0} \leq x_i \leq \frac{1}{\alpha} x_{i,0}, \tag{10}$$

where $\alpha < 1$ is a tuning parameter. The new optimization formulation is shown in Fig. 8.

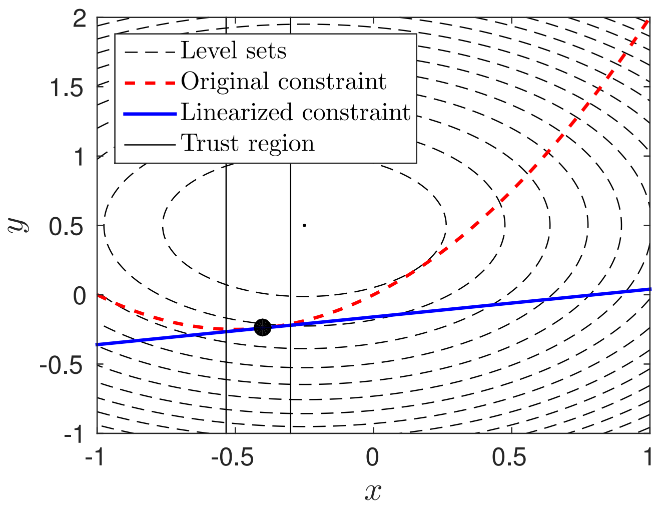


Figure 7: Design space for fully linearized constraint with added trust region. Feasible design space is now on the blue line between the vertical black lines.

$$\begin{tikzcd} \text{SP constraint} & & \text{GP approximations at } x^k \\ \boxed{p_1(x) = p_2(x)} \arrow[rr, "\text{Method D}"] & & \boxed{\begin{aligned} \frac{\hat{p}_1(x, x^k)}{\hat{p}_2(x, x^k)} &\leq 1 \\ \frac{\hat{p}_2(x, x^k)}{\hat{p}_1(x, x^k)} &\leq 1 \\ \frac{x}{x^k} &\leq \frac{1}{\alpha} \\ \frac{x^k}{x} &\leq \frac{1}{\alpha} \end{aligned}} \end{tikzcd}$$

Figure 8: Illustration of changes to the original SP constraints to form the GP subproblem using Method D for one posynomial equality constraint.

3.5 Xu's method

Xu proposes a global optimization approach for SPs which also relies on the monomialization of the inequality and equality constraints.[25] This algorithm applies several heuristic parameters including an added constant to the objective function and a vector of weights for auxiliary variables assigned to a set of equality constraints. The weights monotonically increase to force the auxiliary variables to unity and shift the feasible region of the relaxed problem closer to that of the original SP. Xu's algorithm is provided in Algorithm 2.

Continuing with the previous example from Eq. (9), we can illustrate the problem transformation following Xu's algorithm without converting into log-space. Note that the objective function for our example is always positive and hence the parameter M can be neglected[a]. The constraint is first transformed into two inequality constraints as follows:

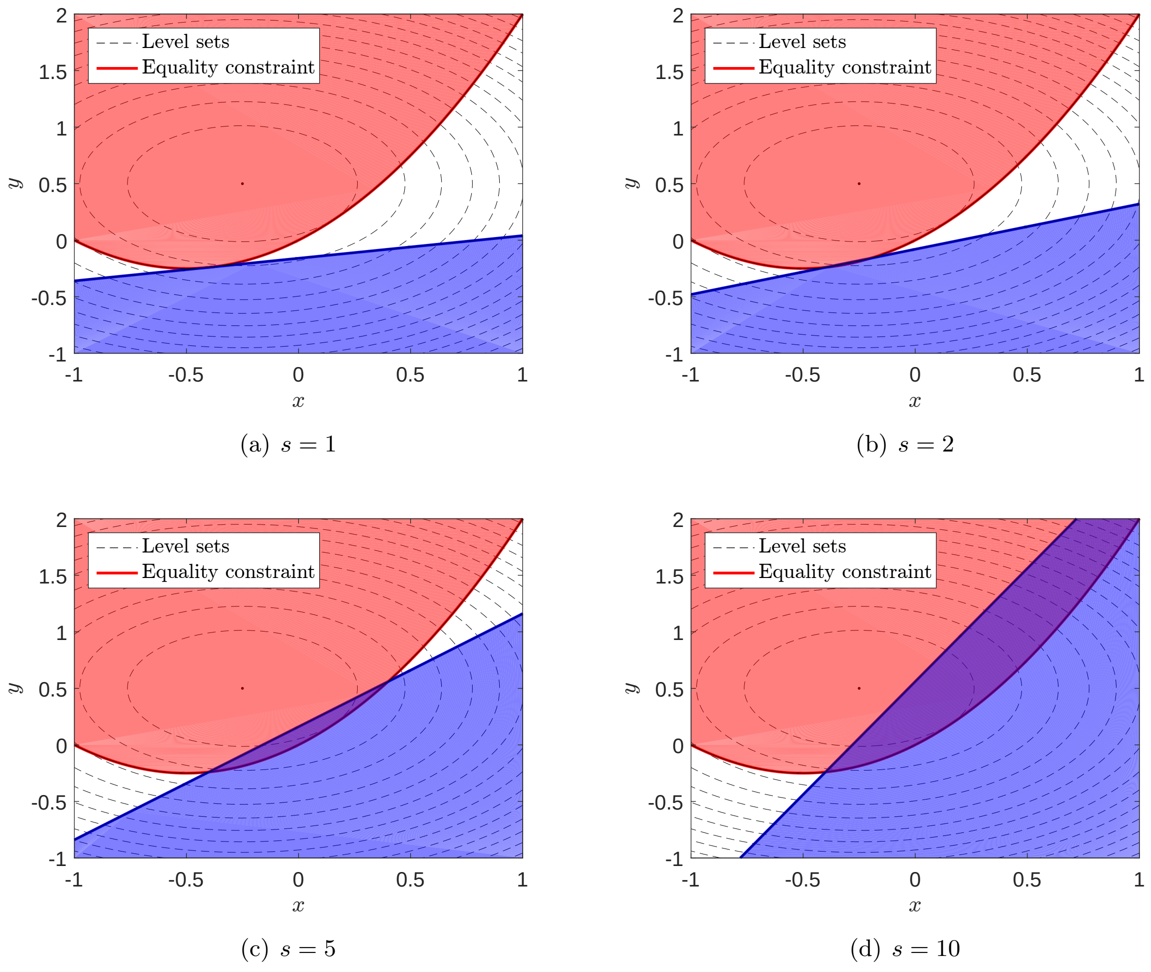
$$\begin{aligned} y &\geq x^2 + x \\ y &\leq s(2x + 1) \end{aligned} \tag{11}$$

[a]Xu adds a sufficiently large scalar M to the objective function to ensure the objective function is always positive.

Algorithm 2 Xu's algorithm for solving of signomial programs with the equality constraint $p_1 = p_2$.[25]

```
Input: Initial point $x^0$, positive constant $M$, initial slack variable weights $w_i^0$, and update
    parameter $\alpha$
Output: Approximate solution $x^*$ to the signomial program within specified tolerance $\epsilon$
 1: Initialization $k = 0$
 2: while $\frac{|f(x^k)-f(x^{k-1})|}{f(x^k)+f(x^{k-1})} > \epsilon$ do
 3:   $\bar{f}(t,s) = t + \sum ws$
 4:   $g_0(x^k) = \frac{f^+ + M}{f^- + t}$
 5:   if $p_1$ is a monomial then
 6:     $m_2(x^k) = \hat{p_2}(x^k)$
 7:     $g_1(x^k) = \frac{p_2(x^k)}{p_1(x^k)}$
 8:     $\hat{g}_2(x^k) = \frac{p_1(x^k)}{m_2(x^k)}$
 9:   else if $p_2$ is a monomial then
10:     $m_1(x^k) = \hat{p_1}(x^k)$
11:     $g_1(x^k) = \frac{p_1(x^k)}{p_2(x^k)}$
12:     $\hat{g}_2(x^k) = \frac{p_2(x^k)}{m_1(x^k)}$
13:   else
14:     $m_1(x^k) = \hat{p_1}(x^k)$
15:     $m_2(x^k) = \hat{p_2}(x^k)$
16:     $\hat{g}_1(x^k) = \frac{p_1(x^k)}{m_2(x^k)}$
17:     $\hat{g}_2(x^k) = \frac{p_2(x^k)}{m_1(x^k)}$
18:   end if
19:   $(x^*, t^*, s^*) = \arg\min_{t,s} \bar{f}(t,s)$
        subject to $g_0(x^k) \leq 1$ and $g_1(x^k) \leq 1$ and $\hat{g}_2(x^k) \leq s$ and $s \geq 1$
20:   $x^{k+1} = x^*$
21:   $w^{k+1} = \alpha w^k$
22:   $k = k + 1$
23: end while
```

where s is the slack variable. Note that the slack variable effectively loosens the equality constraint by shifting the linearized approximation. This approach is very similar to what is being proposed in Section 3.2 and is illustrated in Fig. 9. In addition, the original problem



(a) $s = 1$ (b) $s = 2$ (c) $s = 5$ (d) $s = 10$

Figure 9: Increasing the feasible space with slack variable s.

plotted in the $\{x, y\}$ design space is now mapped to the $\{s, t\}$ space where t is a new variable introduced to linearize the objective function. The mapping between the two spaces is illustrated in Fig. 10. Because the slack variable effectively changes the size of the feasible region, larger slack variables will allow for greater optimality at the expense of feasibility. Hence the Xu approximation should ultimately yield slack variables close to 1.0 in order to enforce the equality constraints effectively. As is shown in Fig. 10, $s = 1.0$ may not minimize the transformed problem at any given iteration in the Xu algorithm. However, by requiring the weights to monotonically increase, Xu's algorithm is continuously changing the objective landscape such that a slack variable of 1.0 will ultimately be optimal. This trend is illustrated in Fig. 11 where a larger weighting factor is applied to the linearized objective function. The shift in objective function level sets as the weighting factor w is increased indicates the smallest value of s (the lower bound of 1.0) will ultimately minimize the objective function.

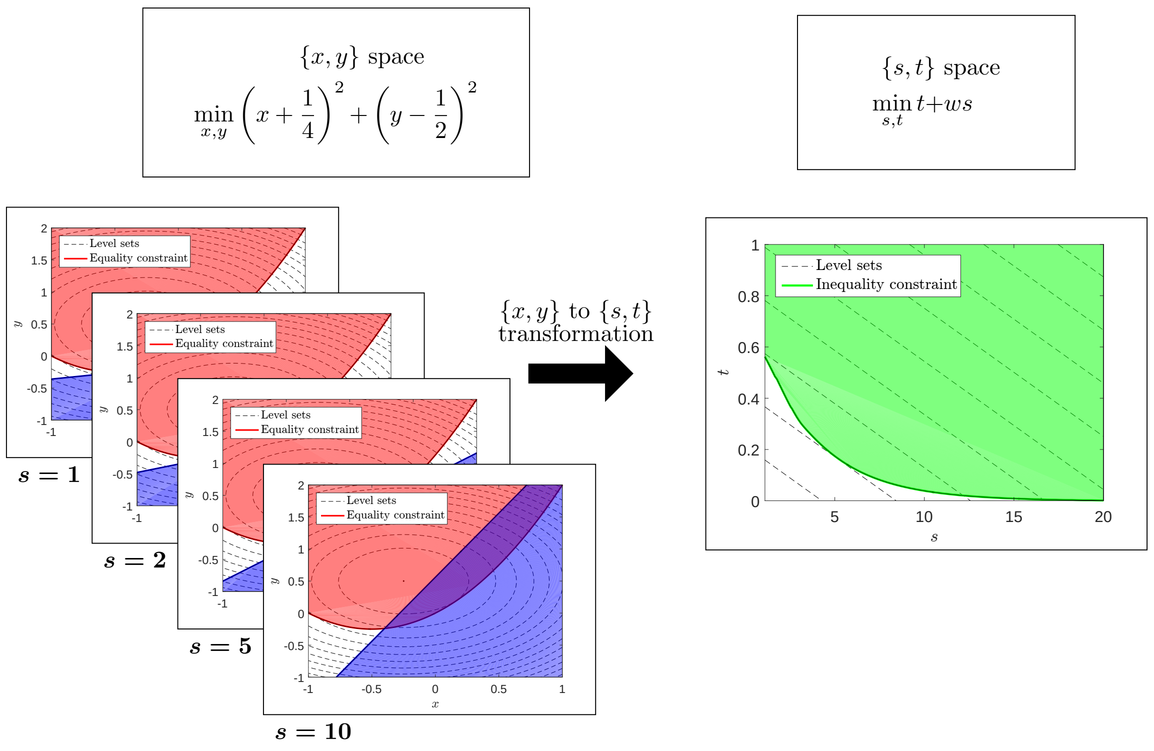
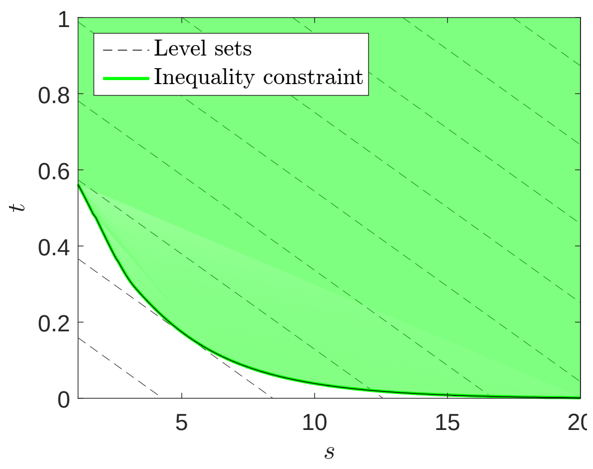
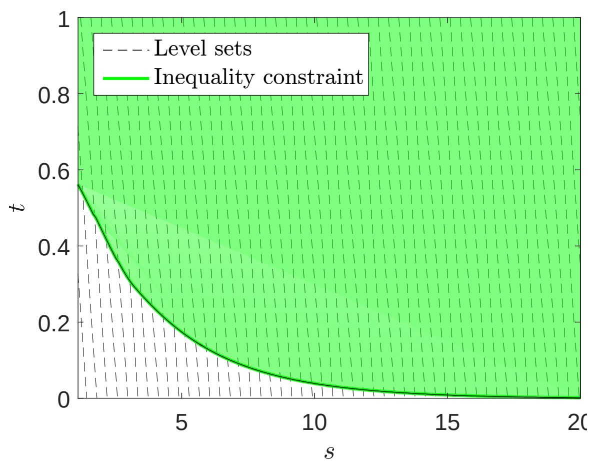


Figure 10: Transformation from $\{x, y\}$ space to $\{s, t\}$ space using Xu's algorithm with a slack variable weight $w = 0.05$.



(a) Weighting factor $w = 0.05$



(b) Weighting factor $w = 1.0$

Figure 11: Increasing the weighting factor shifts the linearized objective function such that smaller slack variables are desired.

4 Results

The methods from Section 3 are applied to several test problems, as well as one application problem in aircraft design.

Each method from Section 3 is implemented in a geometric programming modeling package in Python known as *GPkit*.[5] GPkit is focused on making GP and SP solution techniques accessible to engineering designers. It interfaces with open-source and commercially-available

Table 1: Convergence results for Example 1 with an initial point $x^0 = (1, 1)$.

Method	A	B	B	C	D	D	Xu[†]
Trust region, α	n/a	99.9%	99 %	n.a.	90%	50%	n/a
# GP iterations	8	4	4	3	12	5	3
CPU time, [s]	0.235	0.162	0.158	0.112	0.313	0.152	0.145
Objective value	5.000	4.9995	4.95	5.000	5.000	5.000	5.000
Absolute objective error	6.2e-10	0.005	0.05	8.88e-16	8.88e-16	8.88e-16	1.171e-09

[†] an initial weighting of $w^0 = 1$ and update parameters $\alpha = 1.1$ was used for all slack variables in the linearized objective function

Table 2: Convergence results for Example 1 with an initial point $x^0 = (5, 10)$.

Method	A	B	B	C	D	D	Xu[†]
Trust region, α	n/a	99.9%	99%	n/a	90%	50%	n/a
# GP iterations	100	8	4	3	11	4	3
CPU time, [s]	−−	0.295	0.161	0.0853	0.308	0.111	0.139
Objective value	−−	4.995	4.950	5.000	5.000	5.000	5.000
Absolute objective error	−−	0.005	0.05	8.88e-16	8.88e-16	8.88e-16	1.170e-09

[†] an initial weighting of $w^0 = 1$ and update parameters $\alpha = 1.1$ was used for all slack variables in the linearized objective function.
[††] "−−" denotes that the problem could not be solved in 100 iterations.

interior-point solvers. The following example problems were solved using the solver MOSEK[19] with a free academic license. All tests were run on a dual-core 3.1GHz Intel Core i7 processor.

4.1 Example 1

In the first example we consider a small two-dimensional linear problem

$$\begin{aligned} \min_{x_1, x_2} \quad & x_1 \\ \text{subject to} \quad & x_1 = x_2 + 1 \\ & x_2 \geq 4. \end{aligned} \tag{12}$$

It is straightforward to see that the optimum solution exists at $\bar{x}_1 = 5$, $\bar{x}_2 = 4$. Of course, in practice one would not solve this using signomial programming, but this example illustrates some important characteristics of the solution methods proposed.

Tables 1 and 2 provide a general comparison between the performance of all five methods discussed in Section 3. Table 1 provides convergence characteristics using an infeasible initial design point $x^0 = (1, 1)$ and Table 2 provides the same data where a feasible initial design point $x^0 = (5, 10)$ is chosen. For the algorithms that require tuning parameters (i.e Methods B, D and Xu), we also check the convergence of these methods for different values of these parameters.

We see that except for Method A with starting point $x^0 = (5, 10)$, all methods do converge to some value. The reason Method A converges when given an initial infeasible design point is because the first solve is actually a feasibility solve. A feasibility solve will push x_2 close to the inequality constraint $x_2 \geq 4$. The solver then iterates around the new feasible point for a few steps before converging. When the initial point is feasible, the algorithm cannot converge because there is only one feasible point in each approximated geometric program and as such the only way the design point can progress is due to floating point rounding errors. Method

Table 3: Convergence results for Example 1 using Methods A and C with 100 random initial points.

Method		A			C	
	min.	mean	max.	min.	mean	max.
% Successful runs		35%			100%	
# GP iterations	2	8.5	74	2	2.9	3
CPU time, [s]	0.045	0.21	1.8	0.028	0.049	0.088
Objective value	5.000	5.056	5.611	5.000	5.000	5.000
Absolute objective error	6.21e-10	0.0561	0.611	8.88e-16	8.88e-16	8.88e-16
Absolute constraint error	2.76e-09	1.15e-08	5.39e-08	2.22e-16	2.22e-16	2.22e-16

B does converge for all conditions in Tables 1 and 2, but has a relatively large error which seems to correlate with the size of the trust region. This makes sense intuitively since one of the inequality constraints (derived from the equality constraint) remains tight in each GP subproblem, but that constraint is exactly the one that is relaxed from the original equality constraint. Method C performs the best of all methods considering that it solves for both initial conditions and only requires 3 iterations to converge while taking the least CPU time. Moreover the solutions achieved by Method C are accurate up to machine precision. Method D is similar to Method C in terms of accuracy, but requires more iterations (and hence more CPU time) to converge. This result is expected since each successive step is bounded by the predefined trust region. Xu's method also performs comparably well to Method C. For both initial points Xu's method requires only 3 iterations to converge and is only marginally slower than Method C. The accuracy, however, is orders of magnitude lower than Method C which may be attributed to the fact that the slack variable never exactly reaches its lower bound of 1.0. Thus the equality constraints will always have some finite slack and the feasibility of the design point can never be guaranteed to be within machine precision.

Next, each algorithm is tested on a set of 100 randomly chosen initial design points to determine how sensitive the results are to the initial condition. This analysis is also meant to assess the global optimality of these methods. Table 3 compares the performance of Methods A and C. Likewise, Table 4 compares the performance of Methods B and D respectively as both require a trust region sizing parameter that is also varied. And finally the results for the Xu algorithm are provided in Table 5. The randomly generated initial variables x_1 and x_2 have a uniform distribution from 1 to 11 (i.e. $x_1^0 \sim \mathcal{U}[1, 11]$, $x_2^0 \sim \mathcal{U}[1, 11]$) and the convergence tolerance is set to 10^{-7}. Note that the initial points may be feasible or infeasible. An unsuccessful solve means that either convergence was not achieved in 100 iterations or the solver returned an error.

From Table 3, Method A is successful in 35% of the runs, which makes sense because only roughly 35% of the runs have an infeasible starting point (see previous discussion above). Method C on the other hand works 100% of the time and needs only 2 to 3 iterations to converge to a solution which is accurate to machine precision. Note that in all these results, the reported CPU time does not include failed runs.

In Table 4 Methods B and D are compared using only the averaged values of all 100 runs. Both methods perform adequately well and both achieve 100% success rates. However, as previously mentioned, Method B suffers from lower accuracy whereas, Method D requires more iterations. Both methods are clearly outperformed by Method C.

Finally in Table 5, the Xu algorithm is tested using both a small initial slack variable weight $w^0 = 1$ and small update factor $\alpha = 1.1$ as well as a larger weight $w^0 = 10$ and update factor $\alpha = 2$. Recall that α multiplies the slack variable weight to drive the slack variable to 1.0. Essentially, the inequality constraints are tightened more quickly with larger w^0 and α to match the equality constraints.

Table 4: Convergence results for Example 1 using Methods B and D with 100 random initial points.

Method	B	B	B	D	D	D
Trust region, α	90%	99%	99.9%	50%	70%	90%
% Successful runs	100%	100%	100%	100%	100%	100%
# GP iterations	2.0	3.0	5.6	3.5	4.7	8.1
Objective value	4.500	4.950	4.995	5.000	5.000	5.000
Absolute objective error	0.50	0.50	0.005	8.88e-16	8.88e-16	8.88e-16
Absolute constraint error	0.50	0.50	0.005	8.88e-16	8.88e-16	8.88e-16

Table 5: Convergence results for Example 1 using Xu's Algorithm with 100 random initial points.

Intial weights w^0		$w^0 = 1$			$w^0 = 10$	
Update factor α		$\alpha = 1.1$			$\alpha = 2$	
	min.	mean	max.	min.	mean	max.
% Successful runs		100%			100%	
# GP Iterations	2	2.98	3	2	2.98	3
Objective value	4.999	4.999	4.999	5.000	5.000	5.000
Absolute objective error	5.83e-8	1.31e-7	3.72e-6	8.45e-7	2.27e-5	2.31e-5
Absolute constraint error	5.65e-8	5.84e-8	1.51e-7	2.93e-8	7.53e-7	7.68e-7

4.2 Example 2

The next example problem is used by Xu[25] (example 4 in his work) – originally from Rountree and Rigler[21] – and is put in standard SP form here

$$\begin{aligned} &\min_{t,x_1,x_2} \quad t \\ &\text{subject to} \quad x_1^2 + x_2^2 + 5 - (t + 4x_1 + 2x_2) \leq 0 \\ &\qquad \frac{1}{4}x_1^2 + x_2^2 \leq 1 \\ &\qquad 2x_2 - (x_1 + 1) = 0, \end{aligned} \tag{13}$$

where the global optimum is $\bar{x}_1 = \left(\sqrt{7} - 1\right)/2$, $\bar{x}_2 = \left(\sqrt{7} + 1\right)/4$ with an objective value of 1.393.

Again each of the five methods from Section 3 are compared to one another in Tables 6 and 7. Table 6 provides all test results for the initial infeasible point $x^0 = (2, 1)$ and Table 7 provides all test results for the initial feasible point $x^0 = (1/2, 1)$.

In this particular example only Method C and the Xu algorithm can solve for the infeasible starting point within the 100 iteration limit. For the feasible starting point, all methods do converge (except Method D when the trust region size is 50%). However, Methods A and B have low accuracy compared to Methods C and D. Other variations in the number of iterations and CPU time are small. The Xu algorithm again performs comparably well to Method C which performs best amongst all these algorithms for this specific problem and initial conditions.

As before, all algorithms are run for this problem with 100 different randomly chosen initial points and their performance is compared between one another. The results of this comparison are shown in Table 8 for method A and C, in Table 9 for method B and D, and Table 10 for Xu's method. Note that the convergence tolerance is lowered to 10^{-4} in order for methods A, B and D to yield success rates above 0%. Clearly Method A does not work well, even if the initial point is infeasible. Both Method A – if it is successful – and Method C are accurate to within

Table 6: Convergence results for all methods for Example 2 with an initial point $x^0 = (2, 1)$.

Method	A	B	B	C	D	D	Xu[†]
Trust region, α	n/a	99.9%	99%	n.a.	90%	50%	n/a
# GP iterations	−−	−−	−−	4	−−	−−	3
CPU time, [s]	−−	−−	−−	0.168	−−	−−	0.214
Objective value	−−	−−	−−	1.393	−−	−−	1.393
Absolute objective error	−−	−−	−−	7.42e-11	−−	−−	7.036e-10

[†] an initial weighting of $w^0 = 1$ and update parameters $\alpha = 1.1$ was used for all slack variables in the linearized objective function.
[††] "−−" denotes that the problem could not be solved in 100 iterations.

Table 7: Convergence results for all methods for Example 2 with an initial point $x^0 = (0.5, 1)$.

Method	A	B	B	C	D	D	Xu[†]
Trust region, α	n/a	99.9%	99%	n/a	90%	50%	n/a
# GP iterations	5	8	5	5	−−	4	5
CPU time, [s]	0.222	0.274	0.350	0.184	−−	0.166	0.187
Objective value	2.311	1.391	1.364	1.393	−−	1.393	1.393
Absolute objective error	0.918	0.0029	0.0290	3.75e-11	−−	2.11e-9	8.16e-10

[†] an initial weighting of $w^0 = 1$ and update parameters $\alpha = 1.1$ was used for all slack variables in the linearized objective function.
[††] "−−" denotes that the problem could not be solved in 100 iterations.

10^{-9} for both the objective and constraint functions, however Method C not only demonstrates a 100% success rate, but also proves to be slightly faster in mean CPU time required.

Next, comparing methods B and D in Table 9, the same conclusion can be made that Method B has poor relative accuracy compared to the other methods. Furthermore, neither B or D are 100% successful and typically they fail more than 50% of the time. A similar trend still exists that a larger the trust region size usually results in a higher convergence success rate and a lower accuracy. Method D is again not nearly as robust as Method B, but is able to achieve orders of magnitude higher accuracy to compete with Methods A and C.

Finally, Xu's algorithm also performs well for this problem. Again narrowly defeated by Method C, Xu's algorithm is able to perform the solves in roughly the same number of iterations, with a 100% success rate and only 2 to 3 orders of magnitude reduction in accuracy for both

Table 8: Convergence results for Example 2 using Methods A and C with 100 random initial points.

Method	A			C		
	min.	mean	max.	min.	mean	max.
% Successful runs		4%			100%	
# GP iterations	5	5.8	7	5	6.4	9
CPU time, [s]	0.155	0.188	0.206	0.11	0.16	0.33
Objective value	1.42	1.66	1.88	1.393	1.393	1.393
Absolute objective error	0.028	0.27	0.49	2.86e-11	2.88e-11	2.92e-11
Absolute constraint error	3.93e-10	1.31e-09	1.67e-09	1.86e-10	1.87e-10	1.90e-10

Table 9: Convergence results for Example 2 using Methods B and D with 100 random initial points.

Method	B	B	B	D	D	D
Trust region, α	90%	99%	99.9%	50%	70%	90%
% Successful runs	53%	13%	12%	13%	3%	1%
# GP iterations	5.7	5.8	6.8	4.5	4.3	7.0
Objective value	1.109	1.364	1.391	1.395	1.393	1.393
Absolute objective error	0.284	0.0290	0.00291	1.99e-9	1.80e-9	5.90e-9
Absolute constraint error	0.195	0.0184	0.00182	3.59e-10	4.35e-10	9.54e-10

Table 10: Convergence results for Example 2 using Xu's Algorithm with 100 random initial points.

Intial weights w^0		$w^0 = 1$			$w^0 = 10$	
Update factor α		$\alpha = 1.1$			$\alpha = 2$	
	min.	mean	max.	min.	mean	max.
% Successful runs		100%			100%	
# GP Iterations	6	6	6	7	7	7
Objective value	1.393	1.393	1.393	1.393	1.393	1.393
Absolute objective error	3.664e-7	3.664e-7	3.664e-7	6.296e-8	6.296e-8	6.296e-8
Absolute constraint error	4.470e-8	4.470e-8	4.470e-8	7.261e-9	7.261e-9	7.261e-9

the objective and constraint functions. Another significant find is that the heuristic parameters of Xu's method – the initial slack variable weights w^0 and the update factor α – can have a wide variation and still yield comparably good results for this specific problem.

4.3 Example 3

Next we investigate the performance for Example 5 in Xu's work.[25] This problem was originally proposed in Ref. 10 and 16 for the optimal design of a sequence of two CSTR reactors

$$\begin{aligned}
\max_{x_1,x_2,x_3,x_4,x_5,x_6} \quad & x_4 \\
\text{subject to} \quad x_1 + k_1 x_1 x_5 &= 1 \\
x_2 - x_1 + k_2 x_2 x_6 &= 0 \\
x_1 + x_3 + k_3 x_3 x_5 &= 1 \\
x_4 - x_3 + x_2 - x_1 + k_4 x_4 x_6 &= 0 \\
\sqrt{x_5} + \sqrt{x_6} &\le 4 \\
(x_1, x_2, x_3, x_4, x_5, x_6) &\le (1, 1, 1, 1, 16, 16) \\
(x_5, x_6) &\ge \left(10^{-5}, 10^{-5}\right),
\end{aligned} \tag{14}$$

where $k_1 = 0.09755988$, $k_2 = 0.99k_1$, $k_3 = 0.0391908$, and $k_4 = 0.9k_3$.

For this problem, we consider only the 100 random initial point test. The results between Methods A and C, and Methods B and D are presented in Table 11 and Table 12 respectively. Again the same trend continues: Method A is not robust, Method B is relatively slow and inaccurate, and not robust. Finally, Method D is not robust if the trust region is too small since a larger number of iterations are required. Again, there does not seem to be a benefit of using Method D over Method C for this particular problem.

Xu's algorithm again performs comparably well for this example. However, from Table 13 it is clear that the rate of convergence is significantly affected by the initial weights and update

Table 11: Convergence results for Example 3 using 100 random initial points.

Method	A			C		
	min.	mean	max.	min.	mean	max.
% Successful runs		3%			100%	
# GP iterations	8	9.3	10	2	7.7	23
CPU time, [s]	0.25	0.30	0.34	0.076	0.27	0.81
Objective value	0.386	0.388	0.389	1.393	1.393	1.393
Absolute error	4.457e-08	8.452e-4	0.00255	2.86e-11	1.64e-9	1.82e-8
Absolute constraint error	3.92e-10	1.58e-09	3.32e-09	5.55e-17	3.27e-10	5.03e-9

Table 12: Convergence results for Example 3 using Methods B and D with 100 random initial points.

Method	B	B	B	D	D	D
Trust region	10%	1%	0.1%	50%	30%	10%
% Successful runs	100%	94%	74%	100%	3%	1%
# GP iterations	38.7	73.7	78.7	24.9	28.2	23.4
Objective value	0.466	0.396	0.390	0.389	0.389	0.389
Absolute objective error	0.0774	0.00804	0.000805	7.38e-6	3.77e-6	3.27e-7
Absolute constraint error	0.164	0.0175	0.00177	2.70e-10	2.76e-7	5.03e-6

parameter. For small values of α and w^0 it is evident that the constraints are not satisfied properly, resulting in as much as 20% error in satisfying the constraints.

4.4 Example 4

Method C performed well over the previous three examples. However, we can construct an optimization problem where the algorithm would never converge because it is caught in an infinite sequence jumping between two points in the design space. Consider the following problem,

$$\begin{aligned} \min_{x_1,x_2} \quad & x_2 \\ \text{subject to} \quad & x_2\,(1+x_1) = x_1^2\,(1+x_1) + 100 \\ & x_1 \geq 10^{-3} \\ & x_1 \leq 100, \end{aligned} \tag{15}$$

where the last two constraints are added to bound the problem. The design space of this problem is shown in Fig. 12. When a linearization is performed at any $x_1 < \bar{x}_1$, the algorithm will push the design point to the upper bound $x_1 = 100$. Likewise, monomializing the constraint to the

Table 13: Convergence results for Example 3 using Xu's Algorithm with 100 random initial points.

Intial weights w^0		$w^0 = 1$			$w^0 = 10$	
Update factor α		$\alpha = 1.1$			$\alpha = 2$	
% Successful runs		100%			100%	
# GP Iterations	5	27.2	40	3	5.48	12
Objective value	0.383	0.389	0.476	0.347	0.383	0.3888
Absolute objective error	2.23e-05	0.00426	0.0871	2.06e-10	1.36e-8	6.02e-8
Absolute constraint error	1.24e-07	0.00445	0.183	3.83e-12	4.78e-10	3.15e-7

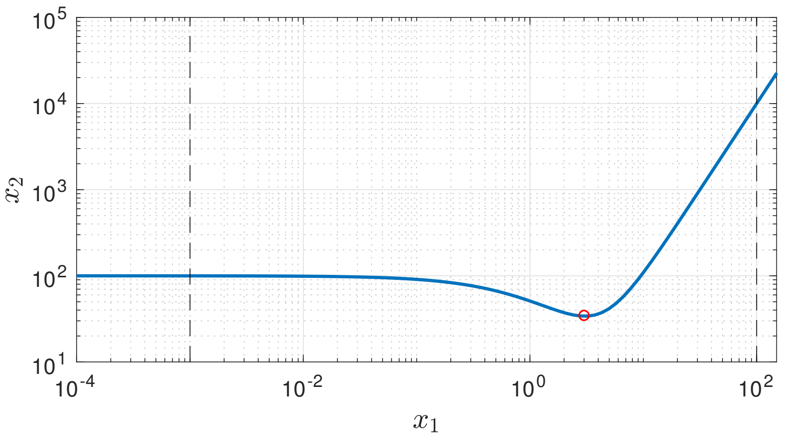


Figure 12: Design space for Example 4.

Table 14: Convergence results for Example 4 using Methods B and D with 100 random initial points.

Method	B	B	B	D	D	D
Trust region	10%	1%	0.1%	−−	−−	−−
% Successful runs	100%	100%	100%	−−	−−	−−
# GP iterations	5.02	5.02	5.08	−−	−−	−−
Objective value	34.0	34.0	34.0	−−	−−	−−
Absolute objective error	8.71e-5	8.72e-5	8.71e-5	−−	−−	−−
Absolute constraint error	2.11e-7	1.63e-7	1.15e-7	−−	−−	−−

"−−" denotes that the problem could not be solved in 100 iterations.

right of $\bar{x}_1$ will push the design point into the lower bound $x_1 = 0.001$. Method C is therefore expected to endlessly switch back and forth between those two bounds.

Firstly we use Methods A and C on this problem. Method A is not able to solve the problem within 100 iterations, irrespective of the initial condition. This is for the same reason as before: the linearization renders the initial condition as the only feasible point. Method C also does not work because it keeps switching between $x_1 = 10^{-3}$ and $x_1 = 100$ for the reason explained above, regardless of the initial condition.

The results for methods B and D is shown in Table 14. Method B performs well, and is able to converge in roughly 5 GP iterations for most runs. Method D on the other hand cannot solve the problem. This is due to the fact that the algorithm keeps switching two non-optimal points. The distance between those two points is determined by the size of the trust region.

Finally, we apply Xu's algorithm to this problem. We see that for $w^0 = 1$, and $\alpha = 1.1$, the algorithm is struggling, requiring 39 iterations to converge. When we run the algorithm for $w^0 = 10$, and $\alpha = 2$, the algorithm requires on average only 7 iterations. Method B, however, still performs better on this problem.

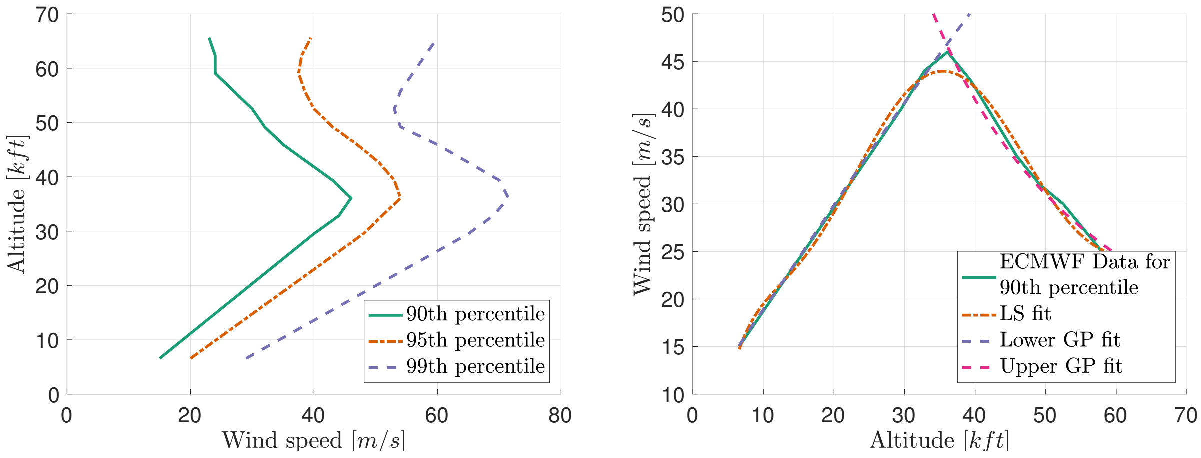
Note that this example is contrived. First, we can easily solve the problem using signomial programming by relaxing the equality constraint to an inequality constraint because the objective function x_2 will force the inequality constraint to be tight. Second, if we change the objective function to for instance $x_1 x_2$, Method C works again and the other methods also work better. Nonetheless, such a situation can arise in practice and in that case it could help to employ Method B for that one constraint.

Table 15: Convergence results for Example 4 using Xu's Algorithm with 100 random initial points.

Intial weights w^0		$w^0 = 1$			$w^0 = 10$	
Update factor α		$\alpha = 1.1$			$\alpha = 2$	
	min.	mean	max.	min.	mean	max.
% Successful runs		100%			100%	
# GP Iterations	39	39	39	4	6.9	13
Objective value	34.0	34.0	34.0	34.0	34.0	34.0
Absolute objective error	8.71e-05	8.71e-05	8.71e-05	1.23e-07	3.15e-04	5.24e-03
Absolute constraint error	5.13e-08	5.13e-08	5.13e-08	1.87e-08	1.91e-06	1.03e-05

4.5 Solar UAV design problem

To demonstrate our algorithm, we apply it to a practical engineering problem, specifically the design of a solar *Unmanned Aerial Vehicle* (UAV). This UAV is intended to be used as an atmospheric satellite and should stay stationary with respect to the ground, i.e. fly at the same speed as the wind, or circle around if the wind speed is below the optimum endurance speed. The UAV design problem – without including an atmosphere model – can be written as a geometric program, following similar modeling strategies as in Ref. 11. The atmosphere model, however, couples with the UAV design problem to transform the problem into a signomial program with posynomial equality constraints.



(a) 90th, 95th, 99th percentile wind speeds averaged over the year 2015 as a function of altitude.

(b) Signomial and posynomial fitting to the 90th percentile wind speed relation.

Figure 13: Absolute wind speed versus altitude, averaged over the globe for the year 2015.

The atmosphere model is necessary to determine at which altitude the UAV should fly, beyond the requirement to stay above 15,000 ft for coverage requirements. The model uses the *International Standard Atmosphere* (ISA) and ideal gas law to relate altitude to atmospheric pressure, density, and temperature. The constraints that are added for the atmosphere model are,

$$\frac{p_{\text{atm}}}{p_0} = \left(\frac{T_{\text{atm}}}{T_0}\right)^{-\frac{g}{R_{\text{air}}L}} \tag{16}$$

$$\rho_{\text{atm}} = \frac{p_{\text{atm}}}{R_{\text{air}}T_{\text{atm}}} \tag{17}$$

$$T_0 = T_{\text{atm}} + Lh, \tag{18}$$

where p_{atm}, ρ_{atm}, and T_{atm} are the pressure, density, and temperature at the altitude h, respectively. p_0 and T_0 are the pressure and temperature at sea level. L is the temperature lapse rate, g is the gravitational acceleration and R_{air} is the gas constant for air. Because both T_{atm} and h are design variables, Eq. 18 is in fact a posynomial equality constraint.

Furthermore, we use a wind model based on data from the *European Centre for Medium-Range Weather Forecasts* (ECMWF).[7] We use this data to relate wind speed to altitude. Because of the constraint to remain stationary with respect to the ground, the mission air speed needs to be higher than the wind speed. The mission required for this particular UAV is that it must be able to fly faster than the 90th percentile wind speeds averaged over the whole world during the year 2015. The variation of wind speeds with altitude is shown in Fig. 13(a), where we clearly see the jet stream between 20,000 ft and 50,000 ft.

In order to integrate the wind data in the aircraft design model, the mapping between wind speed and altitude must be translated into a posynomial or signomial form. Unfortunately, the data is concave in log-space so fitting a posynomial over the whole range would not work. Instead, one can fit a polynomial to the data – which inevitably will have negative coefficients – using least squares regression. The result is shown in 13(b). In the signomial program this constraint would then be implemented as a signomial inequality constraint.

However, we can also see that the relation up to about 35,000 ft is almost linear. Furthermore, the relation above 35,000 ft looks like a convex function. Therefore, we can also fit posynomials to these altitude subsets separately. We use *GPfit* for this purpose, which fits posynomials to data.[12] Using these two approximations, we have to solve the signomial program twice: once for low altitudes and once for high altitudes. We compare the results of both methods – least squares regression and fitting two separate polynomials – when we run the problem.

We attempt to solve the resulting program using the methods from Section 3. Xu's algorithm is however not included because the number of constraints in the problem are too large to manually pick each constraint to belong to one of four different sets, as is required by the Xu algorithm. The Xu approach is therefore not scalable to large design problems. Our objective function is the weight of the UAV. In total the problem has 19 design variables – such as span, area, battery weight, lift coefficient, etc. – and 22 constraints. The results are shown in Table 16, where we show some characteristic design variables, as well as the number of GP iterations and CPU time. We only show the results for Method C, the other methods showed similar results to earlier examples: Method A is not able to solve within a reasonable time, and Methods B and D are able to solve the problem but take more iterations than Method C. Additionally, we see that fitting two posynomials to the data and solving the problem twice is much easier to solve. This highlights the fact that signomial (in)equalities should be avoided whenever possible, because they do make the problem harder to solve.

Table 16: Results for Solar UAV examples using the GP fit (for higher altitudes) and the SP fit.

	GP fit	SP fit
Overall weight, $[lbf]$	92.84	102.3
Span, $[ft]$	86.57	71.44
Altitude, $[ft]$	65,617	59,130
Air density, $[kg/m^3]$	0.0954	0.1328
Cruise velocity, $[m/s]$	22.89	24.82
# GP iterations	5	226
CPU time, $[s]$	0.395	17.4

Concluding, Method C does not seem to have any issue solving this optimization problem for a real application, and requires minimal effort to implement – both for implementing the algorithm, as well as setting up the problem. This extends the applicability of the original GP model substantially and allows for finding the optimal cruise altitude for the UAV.

5 Conclusion

The best overall performing algorithm was Method C. While its performance did rival that of Xu's algorithm, an important distinction here is that Method C is easier to implement. Xu's method requires each constraint to be individually categorized into one of the six categories of constraints defined by Xu in his algorithm.[25] For small problems this is not necessarily an issue, but for large optimization problems – such as the aircraft design problem from Kirschen et al.[14] which includes 238 design variables and 250 constraints – would have taken significantly longer to implement. However, it appears that the simplicity of Method C did not compromise performance when compared to Xu's algorithm. And the fact that Method C does not incorporate any heuristic parameters means that tuning of the algorithm, also a commonly time-consuming process, is not required. In summary, Method C provides the best overall performance when considering implementation complexity as as trade-off. However, it remains a heuristic, and we have seen that a relatively simple problem results in a failure of the algorithm. For that particular problem, Method B worked better.

We therefore suggest starting with Method C, which seems to work well for a large of variety of problems. If Method C does not converge, Method B could be used as a more robust alternative, albeit with slower convergence and less accurate results. Considering signomial programming uses heuristics to find a solution, we make no claims as to whether one of these methods can achieve global convergence, we can only assess convergence to a local optimum.

Acknowledgements

The authors would like to thank Edward Burnell and Philippe Kirschen for their help with GPkit. The first author acknowledges support of the NASA LEARN program under grant number NNX15AN97A.

References

[1] Avriel, M., Barrett, J.: Optimal Design of Pitched Laminated Wood Beams. Journal of Optimization Theory and Applications **26**(2), 291–303 (1978)

[2] Boyd, S.P.: Lecture notes in EE364b "Convex Optimization II", Stanford University (2015)

[3] Boyd, S.P., Kim, S.J., Patil, D.D., Horowitz, M.A.: Digital Circuit Optimization via Geometric Programming. Operations Research **53**(6), 899–932 (2005)

[4] Boyd, S.P., Kim, S.J., Vandenberghe, L., Hassibi, A.: A Tutorial on Geometric Programming. Optimization and Engineering **8**(1), 67–127 (2007)

[5] Burnell, E., Hoburg, W.: GPkit. https://github.com/convexopt/gpkit (2017). Version 0.5.3

[6] Chiang, M., Tan, C.W., Palomar, D.P., O'Neill, D., Julian, D.: Power Control by Geometric Programming. IEEE Transactions on Wireless Communications **6**(7), 2640–2651 (2007)

[7] Dee, D., Uppala, S., Simmons, A., Berrisford, P., Poli, P., Kobayashi, S., Andrae, U., Balmaseda, M., Balsamo, G., Bauer, P.: The ERA-Interim Reanalysis: Configuration and Performance of the Data Assimilation System. Quarterly Journal of the Royal Meteorological Society **137**(656), 553–597 (2011)

[8] Dembo, R.S., Avriel, M.: Optimal Design of a Membrane Separation Process using Signomial Programming. Mathematical Programming **15**(1), 12–25 (1978). DOI 10.1007/BF01608996. URL http://dx.doi.org/10.1007/BF01608996

[9] Duffin, R.J., Peterson, E.L., Zener, C.M.: Geometric Programming: Theory and Application. ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik **47**(8), 561–561 (1967)

[10] Floudas, C.A., Pardalos, P.M., Adjiman, C., Esposito, W.R., Gümüs, Z.H., Harding, S.T., Klepeis, J.L., Meyer, C.A., Schweiger, C.A.: Handbook of Test Problems in Local and Global Optimization, vol. 33. Springer Science & Business Media (2013)

[11] Hoburg, W., Abbeel, P.: Geometric Programming for Aircraft Design Optimization. AIAA Journal **52**(11), 2414–2426 (2014)

[12] Hoburg, W., Kirschen, P., Abbeel, P.: Data Fitting with Geometric-Programming-Compatible Softmax Functions. Optimization and Engineering **17**(4), 897–918 (2016)

[13] Jefferson, T.R., Scott, C.H.: Generalized Geometric Programming Applied to Problems of Optimal Control: I. Theory. Journal of Optimization Theory and Applications **26**(1), 117–129 (1978). DOI 10.1007/BF00933274. URL http://dx.doi.org/10.1007/BF00933274

[14] Kirschen, P.G., Burnell, E.E., Hoburg, W.W.: Signomial Programming Models for Aircraft Design. In: 54th AIAA Aerospace Sciences Meeting. San Diego, CA (2016)

[15] Lange, K., Zhou, H.: MM algorithms for Geometric and Signomial Programming. Mathematical Programming **143**(1), 339–356 (2014). DOI 10.1007/s10107-012-0612-1. URL https://doi.org/10.1007/s10107-012-0612-1

[16] Manousiouthakis, V., Sourlas, D.: A Global Optimization Approach to Rationally Constrained Rational Programming. Chemical Engineering Communications **115**(1), 127–147 (1992)

[17] Maranas, C.D., Floudas, C.A.: Global Optimization in Generalized Geometric Programming. Computers & Chemical Engineering **21**(4), 351–369 (1997)

[18] Mazumdar, M., Jefferson, T.: Maximum Likelihood Estimates for Multinomial Probabilities via Geometric Programming. Biometrika **70**(1), 257–261 (1983)

[19] Mosek ApS : The MOSEK C Optimizer API manual. Version 7.1 (Revision 41) (2015)

[20] Nesterov, Y., Nemirovski, A.: Interior-Point Polynomial Algorithms in Convex Programming. SIAM studies in applied mathematics. Society for Industrial and Applied Mathematics, Philadelphia (1994)

[21] Rountree, D., Rigler, A.: A Penalty Treatment of Equality Constraints in Generalized Geometric Programming. Journal of Optimization Theory and Applications **38**(2), 169–178 (1982)

[22] Singh, J., Nookala, V., Luo, Z.Q., Sapatnekar, S.: Robust Gate Sizing by Geometric Programming. In: Proceedings of the 42nd annual Design Automation Conference, pp. 315–320. ACM (2005)

[23] Wall, T.W., Greening, D., Woolsey, R.: Solving Complex Chemical Equilibria Using a Geometric-Programming Based Technique. Operations Research **34**(3), 345–355 (1986)

[24] Weeraddana, C., Codreanu, M., Latva-aho, M.: Cross-layer Resource Allocation for Wireless Networks via Signomial Programming. In: Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE, pp. 1–6. IEEE (2009)

[25] Xu, G.: Global Optimization of Signomial Geometric Programming Problems. European Journal of Operational Research **233**(3), 500–510 (2014)