The Power of Log Transformation: A Comparison of Geometric and Signomial Programming with General Nonlinear Programming Techniques for Aircraft Design Optimization

Philippe G. Kirschen[*] and Warren W. Hoburg [†]
Massachusetts Institute of Technology, Cambridge, Massachusetts, 02139, USA

Geometric and signomial programming are emerging as promising methods for aircraft design optimization, having both been demonstrated to reliably and quickly find optimal solutions to aircraft design problems. To better understand how they perform compared with more conventional alternatives, this work presents a direct comparison with a general nonlinear programming approach. The crux of geometric programming, and by extension signomial programming, is in the formulation and the logarithmic transformation that makes the problem convex. Starting with the same problem formulation we assess the difference in speed and effectiveness achieved by performing the transformation. Two relatively small aircraft design problems, one a geometric program, the other a signomial program, are solved using the interior point and sequential quadratic programming alogrithms implemented in MATLAB's fmincon function, both with and without performing the log transformation first. Results show that performing the log transformation consistently yields the same optimal solution, independent of initial guess, whereas applying a general nonlinear programming technique directly to the un-transformed problem, at best, takes significantly longer and, at worst, terminates at an infeasible solution. The results also show that the general approach is highly sensitive to the initial guess whereas geometric and signomial programming approaches are not.

Nomenclature

$A_{C_{D_0}}$	fuselage drag area	S_{wet}/S	wing wetted area ratio	$W_{w_{surf}}$	wing weight, surface
C_D	drag coefficient	V	cruise velocity	Æ	aspect ratio
$C_{L_{max}}$	maximum lift coefficient	V_f	fuel volume, total	μ	viscosity of air
C_L	lift coefficient	$V_{f_{avail}}$	available fuel volume	ρ	density of air
$C_{W_{w,1}}$	wing weight coefficent 1	$V_{f_{fuse}}$	available fuselage volume	ρ_f	density of fuel
$C_{W_{w,2}}$	wing weight coefficent 2	$V_{f_{wing}}$	available wing volume	τ	airfoil thickness-to-chord ratio
C_f	skin friction coefficient	V_{min}	takeoff velocity		
D	total drag force	W	total aircraft weight	c_T	thrust specific fuel consumption
L/D	lift-to-drag ratio	W_0	aircraft weight excluding wing		
N_{ult}	ultimate load factor			e	Oswald efficiency factor
R	aircraft range	W_f	fuel weight	g	gravitational acceleration
Re	Reynold's number	W_w	wing weight, total	k	form factor
S	total wing area	$W_{w_{strc}}$	wing weight, structure	t	flight time

I. Introduction

Previous work has demonstrated that simple aircraft design problems can be solved quickly, effectively and easily using Geometric Programming (GP)[a] [1], and that conceptual aircraft design problems comprising a more general set of mathematical relationships can be solved quickly, effectively and easily using Signomial Programming (SP) [2]. These claims of speed, effectiveness and ease have limited value, however, without the benchmark of an alternative technique. An important conclusion from both of these works is that aircraft conceptual design models fit naturally into the geometric and signomial programming frameworks, with little to no sacrifice in fidelity. The purpose of this work is to compare GP and SP with general Nonlinear Programming (NLP) techniques, in the context of aircraft design optimization. Using two different aircraft design problems as test cases, comparisons are made based on two metrics: computational cost, as measured by the number of solver iterations, and solution quality, as measured by the final objective value. Before this, we begin with brief introductions to geometric and signomial programming.

[*]Graduate Student, Department of Aeronautics and Astronautics; currently Optimization Engineer, Hyperloop One.
[†]Assistant Professor, Department of Aeronautics and Astronautics; currently Astronaut Candidate, NASA.
[a]The "GP" acronym is overloaded, referring both to geometric programs and geometric programming. The same is true of the "SP" acronym.

A. Geometric Programming

A Geometric Program (GP) is a special form of nonlinear optimization problem that can be transformed into a convex optimization problem through a logarithmic change of variables. To have this property, GPs are limited to using two special classes of function.

A *monomial* is a function of the form

$$m(\boldsymbol{x}) = c \prod_{j=1}^{n} x_j^{a_j}, \tag{1}$$

where $a_j \in \mathbb{R}$, $c \in \mathbb{R}_{++}$, and $x_j \in \mathbb{R}_{++}$. For instance, the familiar expression for lift, $\frac{1}{2}\rho V^2 C_L S$, is a monomial with $\boldsymbol{x} = (\rho, V, C_L, S)$, $c = 1/2$, and $\boldsymbol{a} = (1, 2, 1, 1)$.

A *posynomial* is a function of the form

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} c_k \prod_{j=1}^{n} x_j^{a_{jk}}, \tag{2}$$

where $\boldsymbol{a_k} \in \mathbb{R}^n$, $c_k \in \mathbb{R}_{++}$, and $x_j \in \mathbb{R}_{++}$. Thus, a posynomial is simply a sum of monomial terms, and all monomials are also posynomials (with just one term).

In plain English, a GP minimizes a posynomial objective function, subject to monomial equality constraints and posynomial inequality constraints. The standard form of a geometric program in mathematical notation is as follows:

$$\begin{aligned} \text{minimize} \quad & p_0(\boldsymbol{x}) \\ \text{subject to} \quad & p_i(\boldsymbol{x}) \leq 1, \quad i = 1, ..., n_p, \\ & m_i(\boldsymbol{x}) = 1, \quad i = 1, ..., n_m, \end{aligned} \tag{3}$$

where the p_i are posynomial (or monomial) functions, the m_i are monomial functions, and $\boldsymbol{x} \in \mathbb{R}^n_{++}$ are the decision variables.

Whereas a *geometric program* is just a type of optimization problem with a certain mathematical structure, for the purpose of this work, *geometric programming* specifically refers to solving a geometric program by transforming it into logarithmic space through a change of variables:

$$\boldsymbol{y} = \log \boldsymbol{x} \tag{4}$$

and then applying a reliable gradient-based optimization technique to the resulting (convex) problem:

$$\begin{aligned} \text{minimize} \quad & \log p_0(e^{\boldsymbol{y}}) \\ \text{subject to} \quad & \log p_i(e^{\boldsymbol{y}}) \leq 0, \quad i = 1, ..., n_p, \\ & \log m_i(e^{\boldsymbol{y}}) = 0, \quad i = 1, ..., n_m. \end{aligned} \tag{5}$$

Given that an un-transformed GP is a non-convex optimization problem, even applying the same gradient-based optimization technique to an un-transformed GP falls into the significantly more general classification of NLP, without any of the guarantees associated with geometric programming.

B. Signomial Programming

A Signomial Program (SP) is a closely related generalization of a GP that allows signomial constraints. A *signomial* is a function with the same form as a posynomial,

$$s(\boldsymbol{x}) = \sum_{k=1}^{K} c_k \prod_{j=1}^{n} x_j^{a_{jk}}, \tag{6}$$

except that the coefficients, $c_k \in \mathbb{R}$, can now be any real number. In particular, they can be negative. The 'difference of convex' formulation of a signomial program permits the objective function to be a ratio of posynomials and is given by:

$$\begin{aligned} \text{minimize} \quad & \frac{p_0(\boldsymbol{x})}{q_0(\boldsymbol{x})} \\ \text{subject to} \quad & s_i(\boldsymbol{x}) \leq 0, \quad i = 1, ..., n_s, \\ & p_i(\boldsymbol{x}) \leq 1, \quad i = 1, ..., n_p, \\ & m_i(\boldsymbol{x}) = 1, \quad i = 1, ..., n_m. \end{aligned} \tag{7}$$

Although (7) is standard form for a signomial program, signomial constraints often take the form $p_1(\boldsymbol{x}) \leq p_2(\boldsymbol{x})$ or $s(\boldsymbol{x}) \leq p(\boldsymbol{x})$, because these are often more intuitive, and both can easily be transformed into the standard form $s(\boldsymbol{x}) \leq 0$. This follows the geometric programming convention of using posynomial inequality constraints of the form $p(\boldsymbol{x}) \leq m(\boldsymbol{x})$ and monomial equality constraints of the form $m_1(\boldsymbol{x}) = m_2(\boldsymbol{x})$ [3].

The definition of *signomial programming* is slightly more nuanced than that for geometric programming, because whereas a geometric program is convex in log space, a signomial program is not. Solving a signomial program with a log transformation is, in principle, no more efficient than solving it without a log transformation, because neither option is convex. Signomial programming refers to the process of solving a signomial program by solving a sequence of GP approximations, each solved via the logarithmic transformation. Note that when a general NLP solver is applied directly to a signomial program, it may also use a sequence of convex approximations to find a solution. The key distinction with signomial programming is that the convex approximations are always geometric programs.

The majority of SP heuristics involve finding a local GP approximation to the SP about an initial guess, $\boldsymbol{x^0}$, solving this GP, and then repeating the process, using the previous iteration's optimal solution as the point about which to take the next GP approximation. The process is repeated until the solution converges [3]. The GP approximation is obtained by approximating each signomial constraint with a posynomial constraint.

The first step, if it has not already been done, is to express each signomial, $s_i(\boldsymbol{x})$, as a difference of posynomials, $p_i(\boldsymbol{x})$ and $q_i(\boldsymbol{x})$, and rearrange them to the form of a posynomial less than or equal to another posynomial.

$$s_i(\boldsymbol{x}) \leq 0 \tag{8}$$

$$p_i(\boldsymbol{x}) - q_i(\boldsymbol{x}) \leq 0 \tag{9}$$

$$p_i(\boldsymbol{x}) \leq q_i(\boldsymbol{x}) \tag{10}$$

Although (10) is not a GP-compatible constraint, it can be made into a GP constraint if posynomial $q_i(\boldsymbol{x})$ is replaced with its local monomial approximation, $\hat{q}_i(\boldsymbol{x};\boldsymbol{x^0})$, because a posynomial divided by a monomial is also a posynomial.

$$p_i(\boldsymbol{x}) \leq \hat{q}_i(\boldsymbol{x};\boldsymbol{x^0}) \tag{11}$$

$$\frac{p_i(\boldsymbol{x})}{\hat{q}_i(\boldsymbol{x};\boldsymbol{x^0})} \leq 1 \tag{12}$$

Finding a monomial approximation to a posynomial is equivalent to finding a local affine approximation to a non-linear function in log space. The best-possible local monomial approximation to a posynomial $q(\boldsymbol{x})$ at the point $\boldsymbol{x^0}$ is given by [3]:

$$\hat{q}_i(\boldsymbol{x})\big|_{\boldsymbol{x^0}} = q_i(\boldsymbol{x^0}) \prod_{n=1}^{N} \left(\frac{x_n}{x_n^0}\right)^{a_n} \tag{13}$$

where x_n are the elements of $\boldsymbol{x}$ and:

$$a_n = \frac{x_n^0}{q_i(\boldsymbol{x^0})} \frac{\partial q_i}{\partial x_n}. \tag{14}$$

C. Choice of Solvers

The MATLAB® function `fmincon` is used for this analysis because it is a popular, commercially available general-purpose optimization tool for finding the minima of constrained nonlinear problems and comprises a number of different types of solver, including both an Interior Point (IP) method solver and a Sequential Quadratic Programming (SQP) solver [4]. Most importantly, it can be used to solve both the transformed and un-transformed versions of the optimization problems solved in this work.

Interior point methods are a powerful class of algorithm that are the method of choice for solving log-transformed GPs, and therefore, by extension, SPs [3]. Though originally derived for solving linear programs, and most frequently used for convex optimization problems, interior point algorithms have also been developed and successfully applied to solving nonlinear, non-convex problems [5]. The `fmincon` implementation of an interior point method is based on work presented in [6], [7] and [8], and is designed to be robust to non-convexity.

SQP is one of the most effective methods for general nonlinearly constrained optimization [9] and is widely used in many fields, including aircraft design [10]. The `fmincon` implementation of the SQP algorithm is largely based on the algorithm described in [9].

The `fmincon` function has a number of options designed to improve its performance. For example, it is possible to provide analytical gradients of the objective and constraints as inputs. When provided to `fmincon` in the experiments presented below, the analytical gradients are computed as follows. Recalling that monomials and posynomials are specific classes of signomial, the gradient of a signomial:

$$s(\boldsymbol{u}) = \sum_{k=1}^{K} c_k \prod_{j=1}^{n} u_j^{a_{jk}} \tag{15}$$

is computed as:

$$\frac{\partial s(\boldsymbol{u})}{\partial u_i} = \sum_{k=1}^{K} c_k a_{ik} u_i^{a_{ik}-1} \prod_{j=1, j\neq i}^{n} u_j^{a_{jk}} \tag{16}$$

and checked using the `CheckGradients` feature in `fmincon`.

The `fmincon` function requires an initial guess for each design variable, regardless of which solution algorithm is used. Note that many interior point methods used to solve geometric and signomial programs do not require the user to provide an initial guess, although signomial programming heuristics typically require an initial guess for variables that appear in signomial constraints [3]. Initial guesses used in this paper are based on solutions found a priori using GPkit [11] with MOSEK [12] (academic license) as the backend solver.

In the next section, two test cases are solved: a relatively small GP model for UAV design introduced in [1], and a slightly larger SP extension of the same model introduced in [13]. To obtain a meaningful comparison, both test cases are solved using a range of different initial guesses, as well as both with and without analytical gradients. All tests are performed on a laptop computer with a 2.4 GHz Intel Core i5 processor, using MATLAB 9.3.0.

II. Test Cases

A. Geometric Program

The first test case is the simple UAV design problem from [1]. This problem is a GP with 10 free variables and 8 constraints.

$$\begin{aligned}
\text{minimize} \quad & D && (17)\\
\text{subject to} \quad & D = \frac{1}{2}\rho V^2 S C_D && (18)\\
& C_D \geq \frac{A_{C_{D_0}}}{S} + kC_f\frac{S_{wet}}{S} + \frac{C_L^2}{\pi A\!\!\!R e} && (19)\\
& C_f \geq 0.074 Re^{-0.02} && (20)\\
& Re \leq \frac{\rho V\sqrt{S/A\!\!\!R}}{\mu} && (21)\\
& W \leq \frac{1}{2}\rho V^2 S C_L && (22)\\
& W \geq W_0 + W_w && (23)\\
& W_w \geq C_{W_{w,1}} S + C_{W_{w,2}} \frac{N_{ult} A\!\!\!R^{1.5}\sqrt{W_0 W S}}{\tau} && (24)\\
& W \leq \frac{1}{2}\rho V_{min}^2 S C_{L_{max}} && (25)
\end{aligned}$$

In geometric (and by extension, signomial) programming there is an implicit constraint $\boldsymbol{x} > 0$. To prevent the solver finding non-physical solutions in the un-transformed case, this constraint is explicitly included, adding a further 10 (very simple) constraints to the problem.

The problem is solved using both the IP and SQP algorithms, with and without the log transformation, using different initial guesses. The un-transformed problem is solved without providing analytical gradients to the solver.

The exact initial guesses used are listed in Table 1. Option A is an arbitrary initial guess of one for every variable. Without any understanding of a problem, let alone prior knowledge of the solution, this might represent the best initial guess a user can provide. It is also an appealing initial guess because it requires effectively no effort from the user.

Initial guess B is, at the opposite extreme, the globally optimal solution rounded to one significant figure, where the optimal solution is taken from the known GP solution. This is, of course, an unrealistic initial guess – a user does

Table 1: Initial guesses used for simple UAV design case

Variable	Optimal value from GP solution	A Ones	B Almost exact solution	C Order of magnitude (floor)	D Order of magnitude (round)	E Order of magnitude (mix)
Æ	8.46	1	8	1	10	10
C_D	0.02059	1	0.02	0.01	0.01	0.1
C_f	0.003599	1	0.004	0.001	0.01	0.01
C_L	0.4988	1	0.5	0.1	1	1
D	303.1	1	300	100	100	100
Re	3.675×10^6	1	4×10^6	1×10^6	1×10^7	1×10^6
S	16.44	1	20	10	10	10
V	38.15	1	40	10	100	10
W	7341	1	7000	1000	10,000	10,000
W_w	2401	1	2000	1000	1000	1000

not typically know the globally optimal solution before solving – but it tests the optimizer on what should be an easy problem.

Initial guesses C, D and E are based on the order of magnitude of the optimal solution. These are intended to be intermediate guesses between options A and B, and represent initial guesses that a user might be able to provide in a practical context.

Two different interpretations of the order of magnitude are used: the "floor" and "round" estimates. Using mathematical notation for the floor and round operators, initial guesses C and D are given by

$$x_C^0 = 10^{\lfloor \log_{10}(x^*) \rfloor} \tag{26}$$

and

$$x_D^0 = 10^{\left[\log_{10}(x^*)\right]}, \tag{27}$$

respectively, where x^* is the optimal value, as obtained from the GP solution.

Initial guess E is a combination of initial guesses C and D, with one perturbed value (C_D). These three fairly similar variants are intended to test how sensitive the solution is to slight changes in the assumed order of magnitude of the final solution.

The results of the comparison tests are listed in Table 2. A few observations can be made from the results:

1. As expected, solving with the log transformation consistently yields the (provably global) optimum in a fraction of a second, independent of solver algorithm and initial guess.

2. By contrast, the speed and effectiveness of solving without the log transformation are highly dependent on solver algorithm and initial guess.

3. Applying the IP solver to the un-transformed problem yields a close-to-globally-optimal solution for three of the five initial guesses, although it takes between 10 and 600 times as many iterations compared with solving the transformed problem. Interestingly, the IP solver performs better with the most naive initial guess than with all of the order-of-magnitude guesses.

4. The SQP solver also, surprisingly, performs best on the un-transformed problem with the most naive initial guess, finding the optimal solution in the same time and with only five times as many iterations as needed for the transformed problem. The SQP solver finds notably sub-optimal (10%) local optima for all three order-of-magnitude initial guesses.

5. There are significant differences in performance between the three order-of-magnitude initial guesses for both solver algorithms. For example, the IP solver returns an optimal solution for initial guess D, an infeasible solution for initial guess C, and a supposedly-feasible solution that actually exploits constraint violation tolerances for initial guess E.

6. Providing analytical gradients to the solvers yields mixed results. In most cases, it has no impact on the final solution and limited effect on the iteration count. Interestingly, it results in a significantly worse local optimum for the IP solver with initial guess D, but simultaneously enables the solver to find an optimal solution for initial guess E, which it does not find with finite difference gradients. For the SQP solver, analytical gradients substantially reduce the number of iterations for initial guess D, whereas, they actually increase the number of iterations for initial guess A.

Table 2: Comparison of objective value, $f(x)$, solution time, t, and iteration count, n, for three different formulations of the simple UAV design problem: (1) the original formulation as shown above, (2) the original formulation with analytical objective and constraint gradients provided to the solver, and (3) the formulation resulting from a logarithmic change of variables. Costs marked with (i) indicate that the solver returned an infeasible solution. Costs marked with (e) indicate that the solver exceeded 300,000 iterations without reaching an optimum.

		Without log transformation						With log transformation		
		No analytical gradients			Analytical gradients			No analytical gradients		
Solver type	Initial guess	$f(x)$ [N]	t [s]	n [-]	$f(x)$ [N]	t [s]	n [-]	$f(x)$ [N]	t [s]	n [-]
IP	A	303.14	9.8	2725	1.2802e-06(e)	1436.8	300000	303.07	0.2	28
IP	B	303.14	0.2	105	303.14	0.2	90	303.07	0.2	14
IP	C	0.0001601(i)	852.2	227857	0.00016007(e)	1225.3	300000	303.07	0.2	19
IP	D	303.14	53.2	11562	593.76	37.7	10530	303.07	0.1	20
IP	E	9.9955e-07	70.0	24621	303.14	17.4	5039	303.07	0.1	19
SQP	A	303.14	0.1	94	303.14	0.1	274	303.07	0.1	20
SQP	B	304.95	0.0	23	304.95	0.0	23	303.07	0.1	9
SQP	C	337.79	0.2	83	337.79	0.0	83	303.07	0.1	12
SQP	D	438.66	1.2	653	438.66	0.1	83	303.07	0.1	11
SQP	E	337.85	0.1	72	337.85	0.0	72	303.07	0.1	12

The different optimal solutions are shown in Table 3. Solutions (I) and (II) are virtually identical. Though not hugely different, solution (III) represents an aircraft that flies slightly slower with a larger wing and a higher lift coefficient. Solution (IV), however, represents a distinctly different aircraft that travels significantly faster with a wing that is simultaneously larger and lighter, due its much smaller aspect ratio. This design results in an aircraft that has almost 50% higher drag than the true optimal design. Solution (V) is an even worse local optimum, with an even larger wing and higher cruise velocity, resulting in a drag that is almost 100% higher than the true optimal design.

Table 3: A more detailed look at the feasible solution values for the GP test case

Variable	Units	Solution corresponding to objective value of...				
		303.07 (I)	303.14 (II)	337.79 (III)	438.66 (IV)	593.76 (V)
$Æ$	$[-]$	8.46	8.452	8.731	2.506	3.496
C_D	$[-]$	0.02059	0.02059	0.02668	0.01359	0.01318
C_f	$[-]$	0.003599	0.003598	0.004669	0.002946	0.002947
C_L	$[-]$	0.4988	0.4985	0.5896	0.1883	0.2194
D	[N]	303.1	303.1	337.8	438.7	593.8
Re	$[-]$	3.675×10^6	3.678×10^6	1.000×10^6	1.000×10^7	1.000×10^7
S	[m^2]	16.44	16.46	16.74	19.33	23.54
V	[m/s]	38.15	38.14	35.07	52.10	55.78
W	[N]	7341	7341	7466	6077	9880
W_w	[N]	2401	2401	2526	1137	1684

The key take-away from this example is that, even for a relatively small GP, when the problem is not transformed into log space, both solvers are sensitive to the choice of initial guess and can get stuck in local optima that are substantially worse than the true optimum. Furthermore, even when the solvers are able to find the correct optimum, on average it takes substantially more time and iterations than with the transformed problem.

B. Signomial Program

The second test case is an extension of the first, with the addition of 10 more constraints, including a signomial constraint. It is important to note that adding just one signomial constraint to a geometric program makes it a signomial program, and thus the guarantee of finding a global optimum, if one exists, is lost.

The new constraints capture fuel weight and volume relationships. The fuel can be carried in both the wing and the fuselage and this is captured in the signomial constraint (44). For this example, the objective is to minimize fuel weight. The full signomial program has 20 free variables and 38 constraints, 20 of which are of the form $x_j > 0$.

$$\text{minimize} \quad W_f \tag{28}$$

$$\text{subject to} \quad W_f \geq c_T t D \tag{29}$$

$$t \geq \frac{R}{V} \tag{30}$$

$$D \geq \frac{1}{2}\rho V^2 S C_D \tag{31}$$

$$C_D \geq \frac{A_{C_{D_0}}}{S} + kC_f\frac{S_{wet}}{S} + \frac{C_L^2}{\pi Æ e} \tag{32}$$

$$C_f \geq 0.074 Re^{-0.02} \tag{33}$$

$$Re \leq \frac{\rho V \sqrt{S/Æ}}{\mu} \tag{34}$$

$$\frac{1}{2}\rho V^2 S C_L \geq W_0 + W_w + \frac{1}{2}W_f \tag{35}$$

$$\frac{1}{2}\rho V_{min}^2 S C_{L_{max}} \geq W \tag{36}$$

$$W \geq W_0 + W_w + W_f \tag{37}$$

$$W_w \geq W_{w_{surf}} + W_{w_{strc}} \tag{38}$$

$$W_{w_{surf}} \geq C_{W_{w,1}} S \tag{39}$$

$$W_{w_{strc}} \geq C_{W_{w,2}} \frac{N_{ult} Æ^{\frac{3}{2}} \sqrt{(W_0 + V_{f_{fuse}} g \rho_f) W S}}{\tau} \tag{40}$$

$$\left(\frac{L}{D}\right) = \frac{C_L}{C_D} \tag{41}$$

$$V_f \leq V_{f_{avail}} \tag{42}$$

$$V_f = \frac{W_f}{g\rho_f} \tag{43}$$

$$V_{f_{avail}} \leq V_{f_{wing}} + V_{f_{fuse}} \tag{44}$$

$$V_{f_{wing}}^2 \leq 0.0009 \frac{S^3}{Æ} \tau^2 \tag{45}$$

$$V_{f_{fuse}} \leq A_{C_{D_0}} 10[m] \tag{46}$$

This model is first solved by direct application of the IP and SQP solvers, both with and without analytical gradients, and then solved using the signomial programming approach described in section I, again with both the IP and SQP solvers. The results are presented in Table 4.

Both the IP and SQP solvers are used with the first four types of initial guess from the previous example, this time based on a previously found optimal solution of the SP . It should be noted that this SP solution was found using an initial guess of one for every variable that appears in a signomial constraint, in this case, $V_{f_{fuse}}$ and $V_{f_{wing}}$. This choice impacts all of the results obtained in this section.

For a signomial program, the initial guess not only determines the starting point for the solvers, but also, in the case of the log transformation approach, the point about which the first GP approximation is made. Subsequent GP approximations are taken about the optimal solution from the previous iteration, until convergence is achieved.

Table 4: Comparison of objective value, $f(x)$, solution time, t, and iteration count, n, for three different formulations of the SP test case: (1) the original formulation, (2) the original formulation with analytical objective and constraint gradients provided to the solver, and (3) the formulation resulting from a logarithmic change of variables. Costs marked with (i) indicate that the solver returned an infeasible solution.

		Without log transformation						With log transformation		
		No analytical gradients			Analytical gradients			No analytical gradients		
Solver type	Initial guess	$f(x)$ [N]	t [s]	n [-]	$f(x)$ [N]	t [s]	n [-]	$f(x)$ [N]	t [s]	n [-]
IP	A	0.00029284(i)	316.6	73654	9.5991e-05(e)	1232.5	300000	4536.2	0.4	103
IP	B	4543.6	10.0	1939	4543.6	5.8	1694	4536.2	0.3	61
IP	C	4543.6	9.7	2006	4543.6	28.4	5025	4536.2	0.4	97
IP	D	4543.6	300.5	55821	11062	429.5	115141	4536.2	0.3	68
SQP	A	21.3(i)	0.1	13	-2.5512e-05(i)	0.1	32	4536.2	0.0	51
SQP	B	4547.9	0.1	34	4547.9	0.1	48	4536.2	0.0	25
SQP	C	3.4751e+06	3.5	772	1.0339e+06	1.0	486	4536.2	0.1	43
SQP	D	5132.1	0.4	136	5619.9(i)	0.0	2	4536.2	0.1	30

For this signomial program the solution took either 3 or 4 GP iterations to converge, depending on the initial guess. The time and iteration counts in the log transformation column of Table 4 are summations of the time and number of solver iterations required for each GP. The breakdowns of these iterations for the IP and SQP solutions are presented in Table 5 and Table 6, respectively.

Table 5: Convergence of the objective value with the solver time and iteration count per GP iteration for the IP solver

Initial Guess	GP Iteration	$f(x)$ [N]	t [s]	n [-]
A	1	5717.1	0.1	40
	2	4538.3	0.1	23
	3	4536.2	0.1	20
	4	4536.2	0.1	20
B	1	4538.2	0.1	22
	2	4536.2	0.1	19
	3	4536.2	0.1	20
C	1	5717.1	0.1	34
	2	4538.3	0.1	23
	3	4536.2	0.1	20
	4	4536.2	0.1	20
D	1	4594.2	0.1	28
	2	4536.5	0.1	20
	3	4536.2	0.1	20

Table 6: Convergence of the objective value with the solver time and iteration count per GP iteration for the SQP solver

Initial Guess	GP Iteration	$f(x)$ [N]	t [s]	n [-]
A	1	5717.1	0.0	25
	2	4538.3	0.0	12
	3	4536.2	0.0	8
	4	4536.2	0.0	6
B	1	4538.2	0.0	11
	2	4536.2	0.0	8
	3	4536.2	0.0	6
C	1	5717.1	0.0	17
	2	4538.3	0.1	12
	3	4536.2	0.0	8
	4	4536.2	0.0	6
D	1	4594.2	0.0	12
	2	4536.5	0.1	10
	3	4536.2	0.0	8

A few observations can be made from the results in Table 4:

1. All of the solutions obtained using the SP (log transformation) approach are identical, with only slight variations in the number of iterations taken to arrive there. The SQP algorithm takes fewer iterations than the IP algorithm.
2. For the un-transformed formulation with the IP solver, all initial guesses, except for the most naive guess, yield an optimal solution that is close to the solution obtained from the SP approach. However, these solutions take between 20 and 820 times as many iterations as the SP approach, depending on the choice of initial guess.

3. When applying the SQP solver directly to the un-transformed problem, only the initial guess closest to the SP optimum results in a final solution that is close to the SP optimum. Initial guess D (order-of-magnitude, round) finds a local optimum that is notably worse (13%) than the SP optimum. Initial guess A (ones) results in an infeasible solution, whereas initial guess C (order-of-magnitude, floor) finds a local optimum that is very far from a global optimum.

4. Once again, providing analytical gradients to the solver yields mixed results. The number of iterations needed to find an optimum is only reduced in one case and there are several cases where analytical gradients either increase the number of iterations or lead to an infeasible solution where the finite difference gradients find a feasible optimum. For the IP solver with initial guess D, using analytical gradients results in a new, significantly worse local optimum.

Table 7: A more detailed look at the feasible solution values for the SP test case

Variable	Units	Solution corresponding to objective value of...			
		4536.2 (I)	4543.6 (II)	5132.1 (III)	11062 (IV)
$\mathcal{R}$	[−]	11.96	11.96	4.716	3.938
$A_{C_{D_0}}$	[m^2]	0.04613	0.04630	0.04813	2.599×10^{-9}
C_D	[−]	0.01335	0.01337	0.01161	0.009552
C_f	[−]	0.00342	0.003418	0.002946	0.002948
C_L	[−]	0.3179	0.3181	0.1718	0.1648
D	[N]	463.4	464.0	701.2	999.6
(L/D)	[−]	23.81	23.79	14.79	17.26
Re	[−]	4.760×10^6	4.759×10^6	1.000×10^7	9.96×10^6
S	[m^2]	21.63	21.64	21.04	83.36
t	[hr]	16.31	16.32	12.20	18.44
V	[m/s]	51.08	51.06	68.32	45.18
V_f	[m^3]	0.5660	0.5680	0.6415	1.383
$V_{f_{avail}}$	[m^3]	0.5660	0.5680	0.6415	1.383
$V_{f_{fuse}}$	[m^3]	0.4613	0.4630	0.4813	1.761×10^{-8}
$V_{f_{wing}}$	[m^3]	0.1047	0.1050	0.1602	1.383
W	[N]	13300	13310	12940	22780
W_f	[N]	4536	4544	5132	11062
W_w	[N]	2515	2517	1558	5470
$W_{w_{strc}}$	[N]	1217	1218	295.3	468.2
$W_{w_{surf}}$	[N]	1298	1299	1262	5002

The four interesting feasible solutions are shown in Table 7. Solution (I), found using the log transformation (signomial programming) approach, regardless of initial guess and solver choice, has reasonable values for each physical quantity in the solution, given the parameters of the problem.

Solution (II), found by direct application of the interior point method to the un-transformed problem for certain initial guesses, is very similar to solution (I), with the exception of a slightly larger fuselage resulting in a slightly higher drag.

Solution (III) is another local optimum. As with the distinct local optima found in the first example, the aircraft designed in this case has a significantly lower (but not unrealistic) aspect ratio, a lower flight time and higher cruise velocity, and a lower lift-to-drag ratio.

Solution (IV) is an even worse local optimum. Although it also has a larger wing and lower aspect ratio, unlike other local optima this design suggests a slower velocity. The most unique aspect of this design is that has virtually no fuselage volume, choosing instead to put all of the fuel in the wings. The result is a design that consumes almost 250% more fuel.

Figure 1 shows the location of the first three solutions in the wing volume - fuselage volume design space. When these two variables are fixed, the problem becomes a geometric program. As can be seen, the two best optimal solutions appear to lie on a border to the left of which there is a region where there is no benefit to be gained from further decreasing the wing volume.

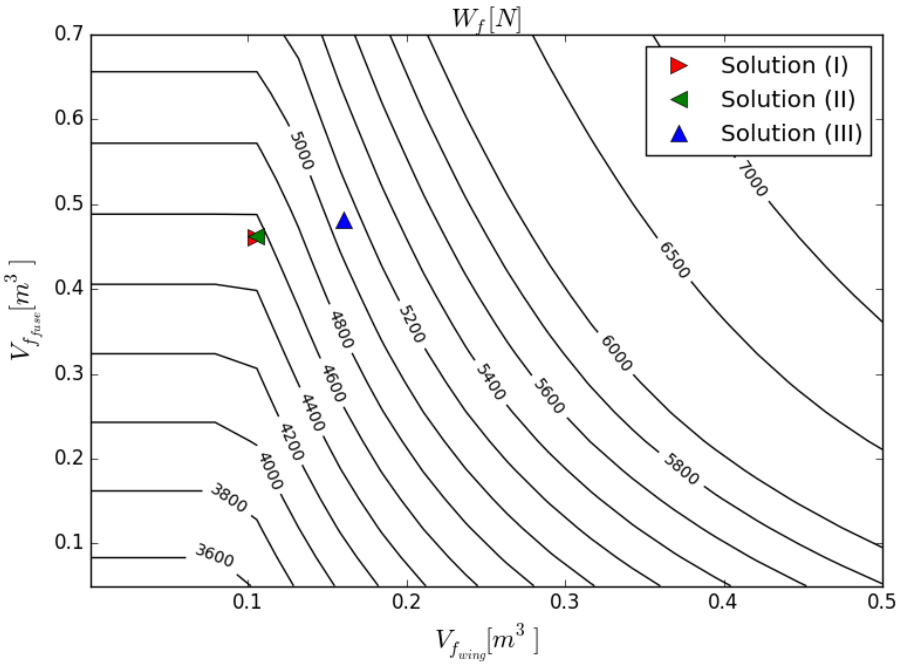

Figure 1: The three best solutions plotted in the wing volume vs. fuselage volume design space.

III. Conclusion

The results from this work demonstrate that using a logarithmic change of variables allows gradient-based solvers to more consistently and quickly find better solutions to aircraft design optimization problems, with more robustness to the choice of initial guess. Of course, this requires problems to be expressed as either geometric or signomial programs, and neglects any cost (i.e. time) associated with formulating the problem. However, previous work has demonstrated that many aircraft design relationships can be expressed naturally in a form compatible with either GP or SP, thus making this cost relatively low.

Combining these findings, there is a strong argument for formulating and solving aircraft design problems as geometric and signomial programs wherever possible, due to the speed and robustness advantages they hold over more general NLP approaches.

IV. Acknowledgments

This work was sponsored by a NASA Aeronautics Fellowship. The authors thank Berk Ozturk for providing the SP used in the second test case.

References

[1] Hoburg, W. and Abbeel, P., "Geometric programming for aircraft design optimization," *AIAA Journal*, Vol. 52, No. 11, 2014, pp. 2414–2426.

[2] Kirschen, P., York, M., Ozturk, B., and Hoburg, W., "Application of Signomial Programming to Aircraft Design," *AIAA Journal of Aircraft*, 2018, doi: 10.2514/1.C034378.

[3] Boyd, S., Kim, S.-J., Vandenberghe, L., and Hassibi, A., "A tutorial on geometric programming," *Optimization and engineering*, Vol. 8, No. 1, 2007, pp. 67–127.

[4] MATLAB, *version 9.3.0 (R2017b)*, The MathWorks Inc., Natick, Massachusetts, 2017.

[5] Vanderbei, R. J. and Shanno, D. F., "An interior-point algorithm for nonconvex nonlinear programming," *Computational Optimization and Applications*, Vol. 13, No. 1-3, 1999, pp. 231–252.

[6] Byrd, R. H., Gilbert, J. C., and Nocedal, J., "A trust region method based on interior point techniques for nonlinear programming," *Mathematical Programming*, Vol. 89, No. 1, 2000, pp. 149–185.

[7] Byrd, R. H., Hribar, M. E., and Nocedal, J., "An interior point algorithm for large-scale nonlinear programming," *SIAM Journal on Optimization*, Vol. 9, No. 4, 1999, pp. 877–900.

[8] Waltz, R. A., Morales, J. L., Nocedal, J., and Orban, D., “An interior algorithm for nonlinear optimization that combines line search and trust region steps,” *Mathematical programming*, Vol. 107, No. 3, 2006, pp. 391–408.

[9] Nocedal, J. and Wright, S., *Numerical optimization*, Springer Science & Business Media, 2006.

[10] Kroo, I., “PASS, program for aircraft synthesis studies,” *Software Package, Desktop Aeronautics, Palo Alto, CA*, 2005.

[11] Burnell, E. and Hoburg, W., “GPkit,” `https://github.com/convexopt/gpkit`, 2016, Version 0.4.0.

[12] ApS, M., *The MOSEK C optimizer API manual. Version 7.1 (Revision 41).*, 2015.

[13] Ozturk, B., *Title TBC*, Master’s thesis, Massachusetts Institute of Technology, 2018 (expected).