# A Tutorial on Geometric Programming

Stephen P. Boyd* Seung Jean Kim† Lieven Vandenberghe‡
Arash Hassibi§

Revised for *Optimization and Engineering*, 7/05

**Abstract**

A geometric program (GP) is a type of mathematical optimization problem characterized by objective and constraint functions that have a special form. Recently developed solution methods can solve even large-scale GPs extremely efficiently and reliably; at the same time a number of practical problems, particularly in circuit design, have been found to be equivalent to (or well approximated by) GPs. Putting these two together, we get effective solutions for the practical problems. The basic approach in GP modeling is to attempt to express a practical problem, such as an engineering analysis or design problem, in GP format. In the best case, this formulation is exact; when this isn't possible, we settle for an approximate formulation. This tutorial paper collects together in one place the basic background material needed to do GP modeling. We start with the basic definitions and facts, and some methods used to transform problems into GP format. We show how to recognize functions and problems compatible with GP, and how to approximate functions or data in a form compatible with GP (when this is possible). We give some simple and representative examples, and also describe some common extensions of GP, along with methods for solving (or approximately solving) them.

*Information Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford CA 94305 (boyd@stanford.edu)

†Information Systems Laboratory, Department of Electrical Engineering, Stanford University, Stanford,CA 94305 (sjkim@stanford.edu)

‡Department of Electrical Engineering, University of California, Los Angeles, CA 90095 (vandenbe@ucla.edu)

§Clear Shape Technologies, Inc., Sunnyvale, CA 94086 (arash@clearshape.com)

# Contents

# 1 The GP modeling approach

A *geometric program* (GP) is a type of mathematical optimization problem characterized by objective and constraint functions that have a special form. The importance of GPs comes from two relatively recent developments:

- New solution methods can solve even large-scale GPs extremely efficiently and reliably.
- A number of practical problems, particularly in electrical circuit design, have recently been found to be equivalent to (or well approximated by) GPs.

Putting these two together, we get effective solutions for the practical problems. Neither of these developments is widely known, at least not yet. Nor is the story over: Further improvements in GP solution methods will surely be developed, and, we believe, many more practical applications of GP will be discovered. Indeed, one of our principal aims is to broaden knowledge and awareness of GP among potential users, to help accelerate the hunt for new practical applications of GP.

The basic approach is to attempt to express a practical problem, such as an engineering analysis or design problem, in GP format. In the best case, this formulation is exact; when this isn't possible, we settle for an approximate formulation. Formulating a practical problem as a GP is called *GP modeling*. If we succeed at GP modeling, we have an effective and reliable method for solving the practical problem.

We will see that GP modeling is not just a matter of using some software package or trying out some algorithm; it involves some knowledge, as well as creativity, to be done effectively. Moreover, success isn't guaranteed: Many problems simply cannot be represented, or even approximated, as GPs. But when we do succeed, the results are very useful and impressive, since we can reliably solve even large-scale instances of the practical problem.

It's useful to compare GP modeling and modeling via general purpose nonlinear optimization (also called nonlinear programming, or NLP). NLP modeling is relatively easy, since the objective and constraint functions can be any nonlinear functions. In contrast, GP modeling can be much trickier, since we are rather constrained in the form the objective and constraint functions can take. Solving a GP is very easy; but solving a general NLP is far trickier, and always involves some compromise (such as accepting a local instead of a global solution). When we do GP modeling, we are limiting the form of the objective and constraint functions. In return for accepting this limitation, though, we get the benefit of extremely efficient and reliable solution methods, that scale gracefully to large-scale problems.

A good analogy can be made with linear programming (LP). A linear program is an optimization problem with an even stricter limitation on the form of the objective and constraint functions (*i.e.*, they must be linear). Despite what appears to be a very restrictive form, LP modeling is widely used, in many practical fields, because LPs can be solved with great reliability and efficiency. (This analogy is no accident — LPs and GPs are both part of the larger class of *convex optimization problems*.)

This tutorial paper collects together in one place the basic background material needed to do GP modeling. We start with the basic definitions and facts, and some methods used

to transform problems into GP format. We show how to recognize functions and problems compatible with GP, and how to approximate functions or data in a form compatible with GP (when this is possible). We give some simple and representative examples, and also describe some common extensions of GP, along with methods for solving (or approximately solving) them. This paper does not cover the detailed *theory* of GPs (such as optimality conditions or duality) or *algorithms* for solving GPs; our focus is on *GP modeling*.

This tutorial paper is organized as follows. In §2, we describe the basic form of a GP and some simple extensions, and give a brief discussion of how GPs are solved. We consider feasibility analysis, trade-off analysis, and sensitivity analysis for GPs in §3, illustrated with simple numerical examples. In §4, we give two longer examples to illustrate GP modeling, one from wireless communications, and the other from semiconductor device engineering. We move on to *generalized geometric programming* (GGP), a significant extension of GP, in §5, and give a number of examples from digital circuit design and mechanical engineering in §6. In §7, we describe several more advanced techniques and extensions of GP modeling, and in §8 we describe practical methods for fitting a function or some given data in a form that is compatible with GP. In §9 we describe some extensions of GP that result in problems that, unlike GP and GGP, are difficult to solve, as well as some heuristic and nonheuristic methods that can be used to solve them. We conclude the tutorial with notes and references in §10.

# 2 Basic geometric programming

## 2.1 Monomial and posynomial functions

Let $x_1, \ldots, x_n$ denote $n$ real positive variables, and $x = (x_1, \ldots, x_n)$ a vector with components $x_i$. A real valued function $f$ of $x$, with the form

$$f(x) = cx_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}, \tag{1}$$

where $c > 0$ and $a_i \in \mathbf{R}$, is called a *monomial function*, or more informally, a *monomial* (of the variables $x_1, \ldots, x_n$). We refer to the constant $c$ as the *coefficient* of the monomial, and we refer to the constants $a_1, \ldots, a_n$ as the *exponents* of the monomial. As an example, $2.3x_1^2 x_2^{-0.15}$ is a monomial of the variables $x_1$ and $x_2$, with coefficient 2.3 and $x_2$-exponent $-0.15$.

Any positive constant is a monomial, as is any variable. Monomials are closed under multiplication and division: if $f$ and $g$ are both monomials then so are $fg$ and $f/g$. (This includes scaling by any positive constant.) A monomial raised to any power is also a monomial:

$$f(x)^\gamma = (cx_1^{a_1} x_2^{a_2} \cdots x_n^{a_n})^\gamma = c^\gamma x_1^{\gamma a_1} x_2^{\gamma a_2} \cdots x_n^{\gamma a_n}.$$

The term 'monomial', as used here (in the context of geometric programming) is similar to, but differs from the standard definition of 'monomial' used in algebra. In algebra, a monomial has the form (1), but the exponents $a_i$ must be nonnegative integers, and the

coefficient $c$ is one. Throughout this paper, 'monomial' will refer to the definition given above, in which the coefficient can be any positive number, and the exponents can be any real numbers, including negative and fractional.

A sum of one or more monomials, *i.e.*, a function of the form

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

where $c_k > 0$, is called a *posynomial function* or, more simply, a *posynomial* (with $K$ terms, in the variables $x_1, \ldots, x_n$). The term 'posynomial' is meant to suggest a combination of 'positive' and 'polynomial'.

Any monomial is also a posynomial. Posynomials are closed under addition, multiplication, and positive scaling. Posynomials can be divided by monomials (with the result also a posynomial): If $f$ is a posynomial and $g$ is a monomial, then $f/g$ is a posynomial. If $\gamma$ is a nonnegative integer and $f$ is a posynomial, then $f^\gamma$ always makes sense and is a posynomial (since it is the product of $\gamma$ posynomials).

Let us give a few examples. Suppose $x$, $y$, and $z$ are (positive) variables. The functions (or expressions)

$$2x, \quad 0.23, \quad 2z\sqrt{x/y}, \quad 3x^2y^{-.12}z$$

are monomials (hence, also posynomials). The functions

$$0.23 + x/y, \quad 2(1+xy)^3, \quad 2x + 3y + 2z$$

are posynomials but *not* monomials. The functions

$$-1.1, \quad 2(1+xy)^{3.1}, \quad 2x + 3y - 2z, \quad x^2 + \tan x$$

are not posynomials (and therefore, not monomials).

## 2.2 Standard form geometric program

A *geometric program* (GP) is an optimization problem of the form

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m, \\ & g_i(x) = 1, & i = 1, \ldots, p, \end{array} \tag{3}$$

where $f_i$ are posynomial functions, $g_i$ are monomials, and $x_i$ are the optimization variables. (There is an implicit constraint that the variables are positive, *i.e.*, $x_i > 0$.) We refer to the problem (3) as a geometric program in *standard form*, to distinguish it from extensions we will describe later. In a standard form GP, the objective must be posynomial (and it must be minimized); the equality constraints can only have the form of a monomial equal to one, and the inequality constraints can only have the form of a posynomial less than or equal to one.

As an example, consider the problem

$$\begin{array}{ll} \text{minimize} & x^{-1}y^{-1/2}z^{-1} + 2.3xz + 4xyz \\ \text{subject to} & (1/3)x^{-2}y^{-2} + (4/3)y^{1/2}z^{-1} \leq 1, \\ & x + 2y + 3z \leq 1, \\ & (1/2)xy = 1, \end{array}$$

with variables $x$, $y$ and $z$. This is a GP in standard form, with $n = 3$ variables, $m = 2$ inequality constraints, and $p = 1$ equality constraints.

We can switch the sign of any of the exponents in any monomial term in the objective or constraint functions, and still have a GP. For example, we can change the objective in the example above to $x^{-1}y^{1/2}z^{-1} + 2.3xz^{-1} + 4xyz$, and the resulting problem is still a GP (since the objective is still a posynomial). But if we change the sign of any of the coefficients, or change any of the additions to subtractions, the resulting problem is not a GP. For example, if we replace the second inequality constraint with $x + 2y - 3z \leq 1$, the resulting problem is *not* a GP (since the lefthand side is no longer a posynomial).

The term *geometric program* was introduced by Duffin, Peterson, and Zener in their 1967 book on the topic [54]. It's natural to guess that the name comes from the many geometrical problems that can be formulated as GPs. But in fact, the name comes from the geometric-arithmetic mean inequality, which played a central role in the early analysis of GPs.

It is important to distinguish between *geometric programming*, which refers to the family of optimization problems of the form (3), and *geometric optimization*, which usually refers to optimization problems involving geometry. Unfortunately, this nomenclature isn't universal: a few authors use 'geometric programming' to mean optimization problems involving geometry, and vice versa.

## 2.3 Simple extensions of GP

Several extensions are readily handled. If $f$ is a posynomial and $g$ is a monomial, then the constraint $f(x) \leq g(x)$ can be handled by expressing it as $f(x)/g(x) \leq 1$ (since $f/g$ is posynomial). This includes as a special case a constraint of the form $f(x) \leq a$, where $f$ is posynomial and $a > 0$. In a similar way if $g_1$ and $g_2$ are both monomial functions, then we can handle the equality constraint $g_1(x) = g_2(x)$ by expressing it as $g_1(x)/g_2(x) = 1$ (since $g_1/g_2$ is monomial). We can maximize a nonzero monomial objective function, by minimizing its inverse (which is also a monomial).

As an example, consider the problem

$$\begin{array}{ll} \text{maximize} & x/y \\ \text{subject to} & 2 \leq x \leq 3, \\ & x^2 + 3y/z \leq \sqrt{y}, \\ & x/y = z^2, \end{array} \tag{4}$$

with variables $x,\ y,\ z \in \mathbf{R}$ (and the implicit constraint $x,\ y,\ z > 0$). Using the simple transformations described above, we obtain the equivalent standard form GP

$$\begin{array}{ll} \text{minimize} & x^{-1}y \\ \text{subject to} & 2x^{-1} \leq 1, \quad (1/3)x \leq 1, \\ & x^2y^{-1/2} + 3y^{1/2}z^{-1} \leq 1, \\ & xy^{-1}z^{-2} = 1. \end{array}$$

It's common to refer to a problem like (4), that is easily transformed to an equivalent GP in the standard form (3), also as a GP.

## 2.4 Example

Here we give a simple application of GP, in which we optimize the shape of a box-shaped structure with height $h$, width $w$, and depth $d$. We have a limit on the total wall area $2(hw + hd)$, and the floor area $wd$, as well as lower and upper bounds on the aspect ratios $h/w$ and $w/d$. Subject to these constraints, we wish to maximize the volume of the structure, $hwd$. This leads to the problem

$$\begin{array}{ll} \text{maximize} & hwd \\ \text{subject to} & 2(hw + hd) \leq A_{\text{wall}}, \quad wd \leq A_{\text{flr}}, \\ & \alpha \leq h/w \leq \beta, \quad \gamma \leq d/w \leq \delta. \end{array} \tag{5}$$

Here $d$, $h$, and $w$ are the optimization variables, and the problem parameters are $A_{\text{wall}}$ (the limit on wall area), $A_{\text{flr}}$ (the limit on floor area), and $\alpha,\ \beta,\ \gamma,\ \delta$ (the lower and upper limits on the wall and floor aspect ratios). This problem is a GP (in the extended sense, using the simple transformations described above). It can be transformed to the standard form GP

$$\begin{array}{ll} \text{minimize} & h^{-1}w^{-1}d^{-1} \\ \text{subject to} & (2/A_{\text{wall}})hw + (2/A_{\text{wall}})hd \leq 1, \quad (1/A_{\text{flr}})wd \leq 1, \\ & \alpha h^{-1}w \leq 1, \quad (1/\beta)hw^{-1} \leq 1, \\ & \gamma wd^{-1} \leq 1, \quad (1/\delta)w^{-1}d \leq 1. \end{array}$$

## 2.5 How GPs are solved

As mentioned in the introduction, the main motivation for GP modeling is the great efficiency with which optimization problems of this special form can be solved. To give a rough idea of the current state of the art, standard *interior-point* algorithms can solve a GP with 1000 variables and 10000 constraints in under a minute, on a small desktop computer (see [17]). For sparse problems (in which each constraint depends on only a modest number of the variables) far larger problems are readily solved. A typical sparse GP with 10000 variables and 1000000 constraints, for example, can be solved in minutes on a desktop computer. (For sparse problems, the solution time depends on the particular sparsity pattern.) It's also possible to optimize a GP solver for a particular application, exploiting special structure to gain even more efficiency (or solve even larger problems).

In addition to being fast, interior-point methods for GPs are also very robust. They require essentially no algorithm parameter tuning, and they require no starting point or initial guess of the optimal solution. They *always* find the (true, globally) optimal solution, and when the problem is infeasible (*i.e.*, the constraints are mutually inconsistent), they provide a certificate showing that no feasible point exists. General methods for NLP can be fast, but are not guaranteed to find the true, global solution, or even a feasible solution when the problem is feasible. An initial guess must be provided, and can greatly affect the solution found, as well as the solution time. In addition, algorithm parameters in general purpose NLP solvers have to be carefully chosen.

In the rest of this section, we give a brief description of the method used to solve GPs. This is not because the GP modeler needs to know how GPs are solved, but because some of the ideas will resurface in later discussions.

The main trick to solving a GP efficiently is to convert it to a nonlinear but *convex optimization problem*, *i.e.*, a problem with convex objective and inequality constraint functions, and linear equality constraints. Efficient solution methods for general convex optimization problems are well developed [17]. The conversion of a GP to a convex problem is based on a logarithmic change of variables, and a logarithmic transformation of the objective and constraint functions. In place of the original variables $x_i$, we use their logarithms, $y_i = \log x_i$ (so $x_i = e^{y_i}$). Instead of minimizing the objective $f_0$, we minimize its logarithm $\log f_0$. We replace the inequality constraints $f_i \leq 1$ with $\log f_i \leq 0$, and the equality constraints $g_i = 1$ with $\log g_i = 0$. This results in the problem

$$\begin{array}{lll} \text{minimize} & \log f_0(e^y) & \\ \text{subject to} & \log f_i(e^y) \leq 0, & i = 1, \ldots, m, \\ & \log g_i(e^y) = 0, & i = 1, \ldots, p, \end{array} \tag{6}$$

with variables $y = (y_1, \ldots, y_n)$. Here we use the notation $e^y$, where $y$ is a vector, to mean componentwise exponentiation: $(e^y)_i = e^{y_i}$.

This new problem (6) doesn't look very different from the original GP (3); if anything, it looks more complicated. But unlike the original GP, this transformed version is convex, and so *can be solved very efficiently*. (See [17] for convex optimization problems, including methods for solving them; §4.5 gives more details of the transformation of a GP to a convex problem.)

It's interesting to understand what it means for the problem (6) to be convex. We start with the equality constraints. Suppose $g$ is a monomial,

$$g(x) = c x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}.$$

Under the transformation above, it becomes

$$\begin{aligned} \log g(e^y) &= \log c + a_1 \log x_1 + \cdots + a_n \log x_n \\ &= \log c + a_1 y_1 + \cdots + a_n y_n, \end{aligned}$$

which is an *affine* function of variables $y_i$. (An affine function is a linear function plus a constant.) Thus, a monomial equality constraint $g = 1$ is transformed to a *linear equation*

in the new variables,

$$a_1 y_1 + \cdots + a_n y_n = -\log c.$$

(In a convex optimization problem, all equality constraint functions must be linear.)

The posynomial inequality constraints are more interesting. If $f$ is a posynomial, the function

$$F(y) = \log f(e^y)$$

is convex, which means that for any $y$, $\tilde{y}$, and any $\theta$ with $0 \leq \theta \leq 1$, we have

$$F(\theta y + (1-\theta)\tilde{y}) \leq \theta F(y) + (1-\theta)F(\tilde{y}). \tag{7}$$

The point $\theta y + (1-\theta)\tilde{y}$ is a (componentwise) weighted arithmetic mean of $y$ and $\tilde{y}$. Convexity means that the function $F$, evaluated at a weighted arithmetic mean of two points, is no more than the weighted arithmetic mean of the function $F$ evaluated at the two points. (For much more on convexity, see [17]).

In terms of the original posynomial $f$ and variables $x$ and $\tilde{x}$, the convexity inequality above can be stated as

$$f(x_1^\theta \tilde{x}_1^{1-\theta}, \ldots, x_n^\theta \tilde{x}_n^{1-\theta}) \leq f(x_1, \ldots, x_n)^\theta f(\tilde{x}_1, \ldots, \tilde{x}_n)^{1-\theta}. \tag{8}$$

The point with coefficients $x_i^\theta \tilde{x}_i^{1-\theta}$ is a weighted *geometric* mean of $x$ and $\tilde{x}$. The inequality (8) above means that the posynomial $f$, when evaluated at a weighted geometric mean of two points, is no more than the weighted geometric mean of the posynomial $f$ evaluated at the two points. This is a very basic property of posynomials, which we'll encounter later.

We emphasize that in most cases, the GP modeler does not need to know how GPs are solved. The transformation to a convex problem is handled entirely by the solver, and is completely transparent to the user. To the GP modeler, a GP solver can be thought of as a reliable black box, that solves any problem put in GP form. This is very similar to the way a numerical linear algebra subroutine, such as an eigenvalue subroutine, is used.

# 3 Feasibility, trade-off, and sensitivity analysis

## 3.1 Feasibility analysis

A basic part of solving the GP (3) is to determine whether the problem is feasible, *i.e.*, to determine whether the constraints

$$f_i(x) \leq 1, \quad i = 1, \ldots, m, \qquad g_i(x) = 1, \quad i = 1, \ldots, p \tag{9}$$

are mutually consistent. This task is called the *feasibility problem*. It is also sometimes called the *phase I problem*, since some methods for solving GPs involve two distinct phases: in the first, a feasible point is found (if there is one); in the second, an optimal point is found.

If the problem is infeasible, there is certainly no optimal solution to the GP problem (3), since there is no point that satisfies all the constraints. In a practical setting, this is disappointing, but still very useful, information. Roughly speaking, infeasibility means the

constraints, requirements, or specifications are too tight, and cannot be simultaneously met; at least one constraint must be relaxed.

When a GP is infeasible, it is often useful to find a point $\hat{x}$ that is as close as possible to feasible, in some sense. Typically the point $\hat{x}$ is found by minimizing some measure of infeasibility, or constraint violation. (The point $\hat{x}$ is not optimal for the original problem (3) since it is not feasible.) One very common method is to form the GP

$$\begin{array}{ll} \text{minimize} & s \\ \text{subject to} & f_i(x) \leq s, \quad i = 1, \ldots, m, \\ & g_i(x) = 1, \quad i = 1, \ldots, p, \\ & s \geq 1, \end{array} \tag{10}$$

where the variables are $x$, and a new scalar variable $s$. We solve this problem (which itself is always feasible, assuming the monomial equality constraints are feasible), to find an optimal $\bar{x}$ and $\bar{s}$. If $\bar{s} = 1$, then $\bar{x}$ is feasible for the original GP; if $\bar{s} > 1$, then the original GP is not feasible, and we take $\hat{x} = \bar{x}$. The value $\bar{s}$ tells us how close to feasible the original problem is. For example, if $\bar{s} = 1.1$, then the original problem is infeasible, but, roughly speaking, only by 10%. Indeed, $\bar{x}$ is a point that is within 10% of satisfying all the inequality constraints.

There are many variations on this method. One is based on introducing independent variables $s_i$ for the inequality constraints, and minimizing their product:

$$\begin{array}{ll} \text{minimize} & s_1 \cdots s_m \\ \text{subject to} & f_i(x) \leq s_i, \quad i = 1, \ldots, m, \\ & g_i(x) = 1, \quad i = 1, \ldots, p, \\ & s_i \geq 1, \quad i = 1, \ldots, m, \end{array} \tag{11}$$

with variables $x$ and $s_1, \ldots, s_n$. Like the problem above, the optimal $s_i$ are all one when the original GP is feasible. When the original GP is infeasible, however, the optimal $x$ obtained from this problem typically has the property that it satisfies most (but not all) of the inequality constraints. This is very useful in practice since it suggests which of the constraints should be relaxed to achieve feasibility. (For more on methods for obtaining points that satisfy many constraints, see [17, §11.4].)

GP solvers unambiguously determine feasibility. But they differ in what point (if any) they return when a GP is determined to be infeasible. In any case, it is always possible to set up and solve the problems described above (or others) to find a potentially useful 'nearly feasible' point.

## 3.2 Trade-off analysis

In *trade-off analysis* we vary the constraints, and see the effect on the optimal value of the problem. This reflects the idea that in many practical problems, the constraints are not really set in stone, and can be changed, especially if there is a compelling reason to do so (such as a drastic improvement in the objective obtained).

Starting from the basic GP (3), we form a *perturbed GP*, by replacing the number one that appears on the righthand side of each constraint with a parameter:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & f_i(x) \leq u_i, \quad i = 1, \ldots, m, \\ & g_i(x) = v_i, \quad i = 1, \ldots, p. \end{array} \tag{12}$$

Here $u_i$ and $v_i$ are positive constants. When $u_i = 1$ and $v_i = 1$, this reduces to the original GP (3). We let $p(u, v)$ denote the optimal value of the perturbed problem (12) as a function of $u$ and $v$. Thus, the value $p(\mathbf{1}, \mathbf{1})$ (where $\mathbf{1}$ denotes a vector with all components one) is equal to the optimal value of the original GP (3).

If $u_i > 1$, then the $i$th inequality constraint for the perturbed problem,

$$f_i(x) \leq u_i,$$

is *loosened*, compared to the inequality in the standard problem,

$$f_i(x) \leq 1.$$

Conversely, if $u_i < 1$, then the $i$th inequality constraint for the perturbed problem is *tightened* compared to the $i$th inequality in the standard problem. We can interpret the loosening and tightening quantitatively: for $u_i > 1$, we can say that the $i$th inequality constraint has been loosened by $100(u_i - 1)$ percent; for $u_i < 1$, then we can say that the $i$th inequality constraint has been tightened by $100(1 - u_i)$ percent. Similarly, the number $v_i$ can be interpreted as a *shift* in the $i$th equality constraint.

It's important to understand what $p(u, v)$ means. It gives the optimal value of the problem, after we perturb the constraints, and then *optimize again*. When $u$ and $v$ change, so does (in general) the associated optimal point. There are several other perturbation analysis problems one can consider. For example, we can ask how sensitive a particular point $x$ is, with respect to the objective and constraint functions (*i.e.*, we can ask how much $f_i$ and $g_i$ change when we change $x$). But this perturbation analysis is unrelated to trade-off analysis.

In optimal trade-off analysis, we study or examine the function $p(u, v)$ for certain values of $u$ and $v$. For example, to see the optimal trade-off of the $i$th inequality constraint and the objective, we can plot $p(u, v)$ versus $u_i$, with all other $u_j$ and all $v_j$ equal to one. The resulting curve, called the *optimal trade-off curve*, passes through the point given by the optimal value of the original GP when $u_i = 1$. As $u_i$ increases above one, the curve must decrease (or stay constant), since by relaxing the $i$th constraint we can only improve the optimal objective. The optimal trade-off curve flattens out when $u_i$ is made large enough that the $i$th constraint is no longer relevant. When $u_i$ is decreased below one, the optimal value increases (or stays constant). If $u_i$ is decreased enough, the perturbed problem can become infeasible.

When multiple constraints are varied, we obtain an *optimal trade-off surface*. One common approach is to plot trade-off surfaces with two parameters as a set of trade-off curves

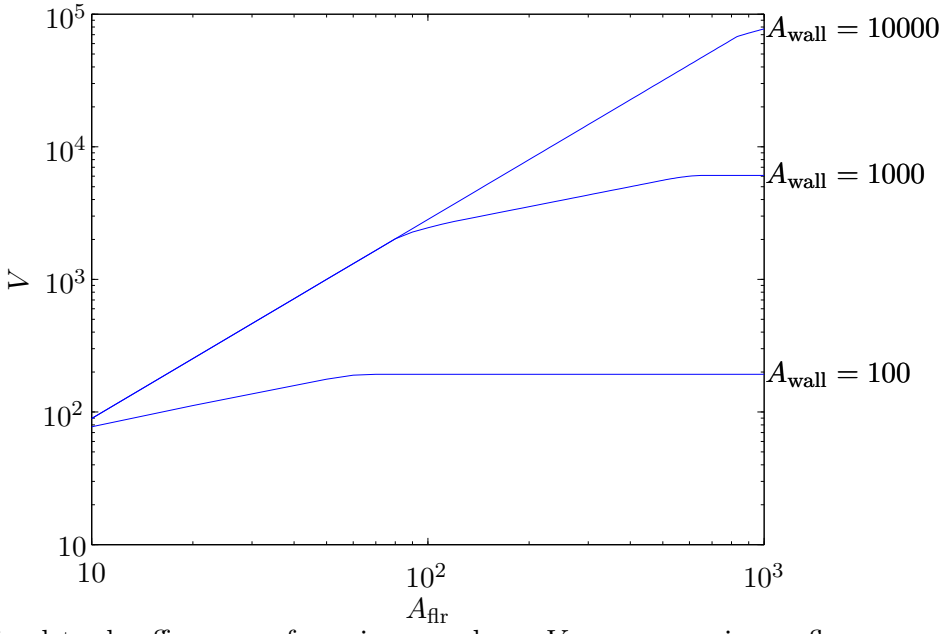


**Figure 1:** Optimal trade-off curves of maximum volume $V$ versus maximum floor area $A_{\text{flr}}$, for three values of maximum wall area $A_{\text{wall}}$.

for several values of the second parameter. An example is shown in figure 1, which shows optimal trade-off curves of optimal (maximum) volume versus $A_{\text{flr}}$, for three values of $A_{\text{wall}}$, for the simple example problem (5) given on page 8. The other problem parameters are $\alpha = 0.5$, $\beta = 2$, $\gamma = 0.5$, $\delta = 2$.

The optimal trade-off curve (or surface) can be found by solving the perturbed GP (12) for many values of the parameter (or parameters) to be varied. Another common method for finding the trade-off curve (or surface) of the objective and one or more constraints is the *weighted sum method*. In this method we *remove* the constraints to be varied, and add positive weighted multiples of them to the objective. (This results in a GP, since we can always add a positive weighted sum of monomials or posynomials to the objective.) For example, assuming that the first and second inequality constraints are to be varied, we would form the GP

$$\begin{array}{ll} \text{minimize} & f(x) + \lambda_1 f_1(x) + \lambda_2 f_2(x) \\ \text{subject to} & f_i(x) \leq 1, \quad i = 3, \ldots, m, \\ & g_i(x) = 1, \quad i = 1, \ldots, p. \end{array} \tag{13}$$

By solving the weighted sum GP (13), we always obtain a point on the optimal trade-off surface. To obtain the surface, we solve the weighted sum GP for a variety of values of the weights. This weighted sum method is closely related to *duality theory*, a topic beyond the scope of this tutorial; we refer the reader to [17] for more details.

## 3.3 Sensitivity analysis

*Sensitivity analysis* is closely related to trade-off analysis. In sensitivity analysis, we consider how *small* changes in the constraints affect the optimal objective value. In other words, we are interested in the function $p(u, v)$ for $u_i$ and $v_i$ near one; this means that the constraints in the perturbed GP (12) aren't very different from the constraints in the original GP (3). Assuming that the optimal objective value $p(u, v)$ is differentiable (which need not be the case, in general) at $u_i = 1$, $v_i = 1$, changes in optimal objective value with respect to small changes in $u_i$ can be predicted from the partial derivative

$$\left.\frac{\partial p}{\partial u_i}\right|_{u=1,\, v=1}.$$

This is nothing more than that slope of the trade-off curve as it passes through the point $u = 1$, $v = 1$.

It is more useful to work with a normalized derivative, that gives the fractional or relative change in the optimal objective, given a fractional or relative change in $u_i$. We define the *optimal sensitivity*, or just sensitivity, of the GP (3), with respect to the $i$th inequality constraint, as

$$S_i = \left.\frac{\partial \log p}{\partial \log u_i}\right|_{u=1,\, v=1} = \left.\frac{\partial \log p}{\partial u_i}\right|_{u=1,\, v=1}. \tag{14}$$

(This is the slope of the trade-off curve on a log-log plot.) The optimal sensitivities are also called the *optimal dual variables* for the problem; see [17]. The sensitivity $S_i$ gives the (approximate) fractional change in optimal value per fractional change in the righthand side of the $i$th inequality. Since the optimal value $p$ decreases when we increase $u_i$, we always have $S_i \leq 0$. If the $i$th inequality constraint is not tight at the optimum, then we have $S_i = 0$, which means that a small change in the righthand side of the $i$th constraint (loosening or tightening) has no effect on the optimal value of the problem.

As an example, suppose we have $S_1 = -0.2$ and $S_2 = -5.5$. This means that if we relax the first constraint by 1% (say), we would expect the optimal objective value to decrease by about 0.2%; if we tighten the first inequality constraint by 1%, we expect the optimal objective value to increase by about 0.2%. On the other hand if we relax the second inequality constraint by 1%, we expect the optimal objective value to decrease by the much larger amount 5.5%; if we tighten the first constraint by 1%, we expect the optimal objective value to increase by the much larger amount 5.5%. Roughly speaking, we can say that while both constraints are tight, the second constraint is much more tightly binding than the first.

For equality constraints we define the optimal sensitivity the same way:

$$T_i = \left.\frac{\partial \log p}{\partial \log v_i}\right|_{u=1,\, v=1} = \left.\frac{\partial \log p}{\partial v_i}\right|_{u=1,\, v=1}. \tag{15}$$

Here the sign of $T_i$ tells us whether a small increase in the righthand side of the $i$th equality constraint increases or decreases the optimal objective value. The magnitude tells us how sensitive the optimal value is to the righthand side of the $i$th equality constraint.

Optimal sensitivities can be very useful in practice. If a constraint is tight at the optimum, but has a small sensitivity, then small changes in the constraint won't affect the optimal value of the problem much. On the other hand, a constraint that is tight and has a large sensitivity is one that (for small changes) will greatly change the optimal value: if it is loosened (even just a small amount), the objective is likely to decrease considerably; if it is tightened, even just a little bit, the optimal objective value will increase considerably. Roughly speaking, a constraint with a large sensitivity can be considered more strongly binding than one with a small sensitivity.

The optimal sensitivities are also useful when a problem is infeasible. Assuming we find a point that minimizes some measure of infeasibility, the sensitivities associated with the constraints can be very informative. Each one gives the (approximate) relative change in the optimal infeasibility measure, given a relative change in the constraint. The constraints with large sensitivities are likely candidates for the ones to loosen (for inequality constraints), or modify (for equality constraints) to make the problem feasible.

One very important fact is that when we solve a GP, *we get the sensitivities of all constraints at no extra cost*. This is because modern methods for solving GPs solve both the primal (*i.e.*, original) problem, and its dual (which is related to the sensitivities) simultaneously. (See [17].)

To illustrate these ideas, we consider again the simple example problem (5) given on page 8. We solve the problem with parameters

$$A_{\text{flr}} = 1000, \quad A_{\text{wall}} = 200, \quad \alpha = 0.5, \quad \beta = 2, \quad \gamma = 0.5, \quad \delta = 2.$$

The associated maximum volume is $V = 5632$. The optimal sensitivities associated with the floor area constraint and the wall area constraint are

$$S_{\text{flr}} = 0.249, \qquad S_{\text{wall}} = 1.251.$$

(These are both positive since increasing the area limits *increases* the maximum volume obtained.) Thus we expect that a 1% increase in allowed floor space to result in around 0.25% increase in maximum volume, and a 1% increase in allowed wall space to result in around 1.25% increase in maximum volume.

To check these approximations, we change the two wall area constraints by various amounts, and compare the predicted change in maximum volume (from the sensitivities) with the actual change in maximum volume (found by forming and solving the perturbed GP). The results are shown in table 1. The sensitivities give reasonable predictions of the change in maximum volume when the constraints are tightened or loosened. One interesting pattern can be seen in the data: the maximum volume predicted by the sensitivities is always *more* than the actual maximum volume obtained. In other words, the prediction of the objective based on the sensitivities is always larger than the true optimal objective obtained. This is always the case, due to the convex optimization formulation; see [17, §5.6.2].

# 4 GP examples

In this section we give two simple examples of GP applications.

| $\Delta A_{\rm flr}$ | $\Delta A_{\rm wall}$ | $\Delta V$ | $\Delta V_{\rm pred}$ |
|---|---|---|---|
| 0% | 0% | 0% | 0% |
| 0% | +5% | +6.1% | +6.3% |
| 0% | −5% | −6.5% | −6.3% |
| +5% | 0% | +1.0% | +1.2% |
| −5% | 0% | −1.5% | −1.2% |
| +5% | −5% | −6.1% | −5.0% |

**Table 1:** Changes in maximum volume with changes in floor and wall area constraints. The first two columns give the change in the constraints. The third column gives the actual change in maximum volume, found by solving the GP with the new constraints, and the fourth column gives the change in maximum volume predicted by the optimal sensitivities.

## 4.1 Power control

Several problems involving power control in communications systems can be cast as GPs (see, *e.g.*, [83, 81, 62]). We consider a simple example here. We have $n$ transmitters, labeled $1, \ldots, n$, which transmit at (positive) power levels $P_1, \ldots, P_n$, which are the variables in our power control problem. We also have $n$ receivers, labeled $1, \ldots, n$; receiver $i$ is meant to receive the signal from transmitter $i$. The power received from transmitter $j$, at receiver $i$, is given by

$$G_{ij}P_j.$$

Here $G_{ij}$, which is positive, represents the path gain from transmitter $j$ to receiver $i$. The *signal power* at receiver $i$ is $G_{ii}P_i$, and the *interference power* at receiver $i$ is $\sum_{k\neq i} G_{ik}P_k$. The *noise power* at receiver $i$ is given by $\sigma_i$. The *signal to interference and noise ratio* (SINR) of the $i$th receiver/transmitter pair is given by

$$S_i = \frac{G_{ii}P_i}{\sigma_i + \sum_{k\neq i} G_{ik}P_k}. \tag{16}$$

We require that the SINR of any receiver/transmitter pair is at least a given threshold $S^{\rm min}$:

$$\frac{G_{ii}P_i}{\sigma_i + \sum_{k\neq i} G_{ik}P_k} \geq S^{\rm min}, \quad i = 1, \ldots, n.$$

We also impose limits on the transmitter powers,

$$P_i^{\rm min} \leq P_i \leq P_i^{\rm max}, \quad i = 1, \ldots, n.$$

The problem of minimizing the total transmitter power, subject to these constraints, can be expressed as

$$\begin{array}{ll} \text{minimize} & P_1 + \cdots + P_n \\ \text{subject to} & P_i^{\rm min} \leq P_i \leq P_i^{\rm max}, \quad i = 1, \ldots, n, \\ & G_{ii}P_i/(\sigma_i + \sum_{k\neq i} G_{ik}P_k) \geq S^{\rm min}, \quad i = 1, \ldots, n. \end{array}$$

This is not a GP, but is easily cast as a GP, by taking the inverse of the SINR constraints:

$$\frac{\sigma_i + \sum_{k\neq i} G_{ik}P_k}{G_{ii}P_i} \leq 1/S^{\min}, \quad i = 1, \ldots, n.$$

(The lefthand side is a posynomial.) This allows us to solve the power control problem via GP.

This simple version of the problem can also be solved using linear programming, by expressing the SINR constraints as linear inequalities,

$$\sigma_i + \sum_{k\neq i} G_{ik}P_k \leq G_{ii}P_i/S^{\min}, \quad i = 1, \ldots, n.$$

But the GP formulation allows us to handle the more general case in which the receiver interference power is any posynomial function of the powers. For example, interference contributions from intermodulation products created by nonlinearities in the receivers typically scale as polynomials in the powers. Third order intermodulation power from the first and second transmitted signals scales as $P_1P_2^2$ or $P_1^2P_2$; if terms like these are added to the interference power, the power allocation problem above is still a GP. Choosing optimal transmit powers in such cases is complex, but can be formulated as a GP.

## 4.2 Optimal doping profile

We consider a simple version of a problem that arises in semiconductor device engineering, described in more detail in [80]. The problem is to choose the *doping profile* (also called the *acceptor impurity concentration*) to obtain a transistor with favorable properties. We will focus on one critical measure of the transistor: its *base transit time*, which determines (in part) the speed of the transistor.

The doping profile, denoted $N_{\mathrm{A}}(x)$, is a positive function of a space variable $x$ over the interval $0 \leq x \leq W_{\mathrm{B}}$, where $W_{\mathrm{B}}$ is the *base width*. The base transit time, denoted $\tau_{\mathrm{B}}$, is determined by the doping profile. A simplified model for $\tau_{\mathrm{B}}$ is given by

$$\tau_{\mathrm{B}} = \int_0^{W_{\mathrm{B}}} \frac{n_i^2(x)}{N_{\mathrm{A}}(x)} \left( \int_x^{W_{\mathrm{B}}} \frac{N_{\mathrm{A}}(y)}{n_i^2(y)D_n(y)}\, dy \right) dx, \tag{17}$$

where $n_i$ is the intrinsic carrier concentration, and $D_n$ is the carrier diffusion coefficient. Over the region of interest, these can be well approximated as

$$D_n(x) = D_{n0}\Big(\frac{N_{\mathrm{A}}(x)}{N_{\mathrm{ref}}}\Big)^{-\gamma_1}, \qquad n_i^2(x) = n_{i0}^2\Big(\frac{N_{\mathrm{A}}(x)}{N_{\mathrm{ref}}}\Big)^{\gamma_2},$$

where $N_{\mathrm{ref}}$, $D_{n0}$, $n_{i0}$, $\gamma_1$, and $\gamma_2$ are (positive) constants. Using these approximations we obtain the expression

$$\tau_{\mathrm{B}} = \kappa \int_0^{W_{\mathrm{B}}} N_{\mathrm{A}}(x)^{\gamma_2-1} \left( \int_x^{W_{\mathrm{B}}} N_{\mathrm{A}}(y)^{1+\gamma_1-\gamma_2}\, dy \right) dx, \tag{18}$$

where $\kappa$ is a constant.

The basic optimal doping profile design problem is to choose the doping profile to minimize the base transit time, subject to some constraints:

$$\begin{array}{ll} \text{minimize} & \tau_{\mathrm{B}} \\ \text{subject to} & N_{\min} \leq N_{\mathrm{A}}(x) \leq N_{\max} \text{ for } 0 \leq x \leq W_{\mathrm{B}}, \\ & |N_{\mathrm{A}}'(x)| \leq \alpha N_{\mathrm{A}}(x) \text{ for } 0 \leq x \leq W_{\mathrm{B}}, \\ & N_{\mathrm{A}}(0) = N_0, \quad N_{\mathrm{A}}(W_{\mathrm{B}}) = N_c. \end{array} \tag{19}$$

Here $N_{\min}$ and $N_{\max}$ are the minimum and maximum values of the doping profile, $\alpha$ is a given maximum (percentage) doping gradient, and $N_0$ and $N_c$ are given initial and final values for the doping profile. The problem (19) is an infinite dimensional problem, since the optimization variable is the doping profile $N_{\mathrm{A}}$, a function of $x$.

To solve the problem we first discretize with $M+1$ points uniformly spaced in the interval $[0, W_{\mathrm{B}}]$, *i.e.*, $x_i = iW_{\mathrm{B}}/M$, $i = 0, \ldots, M$. We then have the approximation

$$\hat{\tau}_{\mathrm{B}} = \kappa \frac{W_{\mathrm{B}}}{M+1} \sum_{i=0}^{M} v_i^{\gamma_2 - 1} \left( \frac{W_{\mathrm{B}}}{M+1} \sum_{j=i}^{M} v_j^{1+\gamma_1-\gamma_2} \right), \tag{20}$$

where $v_i = N_{\mathrm{A}}(x_i)$. This shows that $\hat{\tau}_{\mathrm{B}}$ is a posynomial function of the variables $v_0, \ldots, v_M$. We can approximate the doping gradient constraint $|N_{\mathrm{A}}'(x)| \leq \alpha N_{\mathrm{A}}(x)$ as

$$(1 - \alpha W_{\mathrm{B}}/(M+1))v_i \leq v_{i+1} \leq (1 + \alpha W_{\mathrm{B}}/(M+1))v_i, \quad i = 0, \ldots, M-1.$$

(We can assume $M$ is large enough that $1 - \alpha/M > 0$.)

Using these approximations, the optimal doping profile problem reduces to the GP

$$\begin{array}{ll} \text{minimize} & \hat{\tau}_{\mathrm{B}} \\ \text{subject to} & N_{\min} \leq v_i \leq N_{\max}, \quad i = 0, \ldots, M, \\ & (1 - \alpha W_{\mathrm{B}}/(M+1))v_i \leq v_{i+1} \leq (1 + \alpha W_{\mathrm{B}}/(M+1))v_i, \quad i = 0, \ldots, M-1, \\ & v_0 = N_0, \quad v_M = N_c, \end{array}$$

with variables $v_0, \ldots, v_M$.

Now that we have formulated the problem as a GP, we can consider many extensions and variations. For example, we can use more accurate (but GP compatible) expressions for the base transit time, a more accurate (but GP compatible) approximation for the intrinsic carrier concentration and the carrier diffusion coefficient, and we can add any other constraints that are compatible with GP.

# 5 Generalized geometric programming

In this section we first describe some extensions of GP that are less obvious than the simple ones described in §2.3. This leads to the idea of *generalized posynomials*, and an extension of geometric programming called *generalized geometric programming*.

## 5.1 Fractional powers of posynomials

We have already observed that posynomials are preserved under positive integer powers. Thus, if $f_1$ and $f_2$ are posynomials, a constraint such as $f_1(x)^2 + f_2(x)^3 \leq 1$ is a standard posynomial inequality, once the square and cube are expanded. But the same argument doesn't hold for a constraint such as

$$f_1(x)^{2.2} + f_2(x)^{3.1} \leq 1, \tag{21}$$

which involves fractional powers, since the lefthand side is not a posynomial. Nevertheless, we can handle the inequality (21) in GP, using a trick.

We introduce new variables $t_1$ and $t_2$, along with the inequality constraints

$$f_1(x) \leq t_1, \qquad f_2(x) \leq t_2, \tag{22}$$

which are both compatible with GP. The new variables $t_1$ and $t_2$ act as upper bounds on the posynomials $f_1(x)$ and $f_2(x)$, respectively. Now, we replace the (nonposynomial) inequality (21) with the inequality

$$t_1^{2.2} + t_2^{3.1} \leq 1, \tag{23}$$

which is a valid posynomial inequality.

We claim that we can replace the nonposynomial fractional power inequality (21) with the three inequalities given in (22) and (23). To see this, we note that if $x$ satisfies (21), then $x$, $t_1 = f_1(x)$, and $t_2 = f_2(x)$ satisfy (22) and (23). Conversely, if $x$, $t_1$, and $t_2$ satisfy (22) and (23), then $x$, $t_1 = f_1(x)$, and $t_2 = f_2(x)$ satisfy (22) and (23). Here we use the critical fact that if $t_1$ and $t_2$ satisfy (23), and we reduce them (for example, setting them equal to $f_1(x)$ and $f_2(x)$, respectively) then they still satisfy (22). This relies on the fact that the posynomial $t_1^{2.2} + t_2^{3.1}$ is an *increasing function* of $t_1$ and $t_2$.

More generally, we can see that this method can be used to handle any number of positive fractional powers occurring in an optimization problem. We can handle any problem which has the form of a GP, but in which the posynomials are replaced with positive fractional powers of posynomials. We will see later that positive fractional powers of posynomials are special cases of *generalized posynomials*, and a problem with the form of a GP, but with $f_i$ fractional powers of posynomials, is a *generalized GP*.

As an example, consider the problem

$$\begin{array}{ll} \text{minimize} & \sqrt{1+x^2} + (1+y/z)^{3.1} \\ \text{subject to} & 1/x + z/y \leq 1, \\ & (x/y + y/z)^{2.2} + x + y \leq 1, \end{array}$$

with variables $x$, $y$, and $z$. This problem is *not* a GP, since the objective and second inequality

constraint functions are not posynomials. Applying the method above, we obtain the GP

$$\begin{array}{ll} \text{minimize} & t_1^{0.5} + t_2^{3.1} \\ \text{subject to} & 1 + x^2 \leq t_1, \\ & 1 + y/z \leq t_2, \\ & 1/x + z/y \leq 1, \\ & t_3^{2.2} + x + y \leq 1, \\ & x/y + y/z \leq t_3. \end{array}$$

We can solve the problem above by solving this GP, with variables $x$, $y$, $z$, $t_1$, $t_2$, and $t_3$.

This method for handling positive fractional powers can be applied recursively. For example, a (nonposynomial) constraint such as

$$x + y + \left((1 + xy)^{1/2} + (1 + y/z)^{1/2}\right)^{3.1} \leq 1$$

can be replaced with

$$x + y + t_1^{3.1} \leq 1, \qquad t_2^{1/2} + t_3^{1/2} \leq t_1, \qquad 1 + xy \leq t_2, \qquad 1 + y/z \leq t_3,$$

where $t_1$, $t_2$, and $t_3$ are new variables.

The same idea also applies to other composite functions of posynomials. If $f_0$ is a posynomial of $k$ variables, with all its exponents positive (or zero), and $f_1, \ldots, f_k$ are posynomials, then the composition function inequality

$$f_0(f_1(x), \ldots, f_k(x)) \leq 1$$

can be handled by replacing it with

$$f_0(t_1, \ldots, t_k) \leq 1, \qquad f_1(x) \leq t_1, \ldots, f_k(x) \leq t_k,$$

where $t_1, \ldots, t_k$ are new variables. This shows that products, as well as sums, of fractional positive powers of posynomials can be handled.

## 5.2 Maximum of posynomials

In the previous section, we showed how positive fractional powers of posynomials, while not posynomials themselves, can be handled via GP by introducing a new variable and a bounding constraint. In this section we show how the same idea can be applied to the maximum of some posynomials. Suppose $f_1$, $f_2$, and $f_3$ are posynomials. The inequality constraint

$$\max\{f_1(x), f_2(x)\} + f_3(x) \leq 1 \tag{24}$$

is certainly not a posynomial inequality (unless we have $f_1(x) \geq f_2(x)$ for all $x$, or vice versa). Indeed, the maximum of two posynomials is generally not differentiable (where the two posynomials have the same value), whereas a posynomial is everywhere differentiable.

To handle (24) in GP, we introduce a new variable $t$, and two new inequalities, to obtain

$$t + f_3(x) \leq 1, \qquad f_1(x) \leq t, \qquad f_2(x) \leq t$$

(which are valid GP inequalities). The same arguments as above show that this set of constraints is equivalent to the original one (24).

The same idea applies to a maximum of more than two posynomials, by simply adding extra bounding inequalities. As with positive fractional powers, the idea can be applied recursively, and indeed, it can be mixed with the method for handling positive fractional powers. As an example, consider the problem

$$\begin{array}{ll} \text{minimize} & \max\{x+z, 1+(y+z)^{1/2}\} \\ \text{subject to} & \max\{y, z^2\} + \max\{yz, 0.3\} \leq 1, \\ & 3xy/z = 1, \end{array}$$

which is certainly not a GP. Applying the methods of this and the previous section, we obtain the equivalent GP

$$\begin{array}{ll} \text{minimize} & t_1 \\ \text{subject to} & x + z \leq t_1, \quad 1 + t_2^{1/2} \leq t_1, \\ & y + z \leq t_2, \\ & t_3 + t_4 \leq 1, \\ & y \leq t_3, \quad z^2 \leq t_3, \\ & yz \leq t_4, \quad 0.3 \leq t_4, \\ & 3xy/z = 1. \end{array}$$

## 5.3 Generalized posynomials

We say that a function $f$ of positive variables $x_1, \ldots, x_n$ is a *generalized posynomial* if it can be formed from posynomials using the operations of addition, multiplication, positive (fractional) power, and maximum.

Let us give a few examples. Suppose $x_1, x_2, x_3$ are positive variables. The function

$$\max\left\{1 + x_1, 2x_1 + x_2^{0.2} x_3^{-3.9}\right\}$$

is a generalized posynomial, since it is the maximum of two posynomials. The function

$$\left(0.1 x_1 x_3^{-0.5} + x_2^{1.7} x_3^{0.7}\right)^{1.5}$$

is a generalized posynomial, since it is the positive power of a posynomial. All of the functions appearing in the examples of the two previous sections, as the objective or on the lefthand side of the inequality constraints, are generalized posynomials.

As a more complex example, the function

$$h(x) = (1 + \max\{x_1, x_2\}) \left(\max\left\{1 + x_1, 2x_1 + x_2^{0.2} x_3^{-3.9}\right\} + \left(0.1 x_1 x_3^{-0.5} + x_2^{1.7} x_3^{0.7}\right)^{1.5}\right)^{1.7}$$

is a generalized posynomial. This can be seen as follows:

- $x_1$ and $x_2$ are variables, and therefore posynomials, so $h_1(x) = \max\{x_1, x_2\}$ is a generalized posynomial.
- $1 + x_1$ and $2x_1 + x_2^{0.2}x_3^{-3.9}$ are posynomials, so $h_2(x) = \max\left\{1 + x_1, 2x_1 + x_2^{0.2}x_3^{-3.9}\right\}$ is a generalized posynomial.
- $0.1x_1x_3^{-0.5} + x_2^{1.7}x_3^{0.7}$ is a posynomial, so $h_3(x) = \left(0.1x_1x_3^{-0.5} + x_2^{1.7}x_3^{0.7}\right)^{1.5}$ is a generalized posynomial.
- $h$ can be expressed as $h(x) = (1 + h_1(x))\,(h_2(x) + h_3(x))^{1.7}$ (*i.e.*, by addition, multiplication, and positive power, from $h_1$, $h_2$, and $h_3$) and therefore is a generalized posynomial.

Generalized posynomials are (by definition) closed under addition, multiplication, positive powers, and maximum, as well as other operations that can be derived from these, such as division by monomials. They are also closed under composition in the following sense. If $f_0$ is a generalized posynomial of $k$ variables, for which no variable occurs with a negative exponent, and $f_1, \ldots, f_k$ are generalized posynomials, then the composition function

$$f_0(f_1(x), \ldots, f_k(x))$$

is a generalized posynomial.

A very important property of generalized posynomials is that they satisfy the convexity property (7) that posynomials satisfy. If $f$ is a generalized posynomial, the function

$$F(y) = \log f(e^y)$$

is a convex function: for any $y$, $\tilde{y}$, and any $\theta$ with $0 \leq \theta \leq 1$, we have

$$F(\theta y + (1-\theta)\tilde{y}) \leq \theta F(y) + (1-\theta)F(\tilde{y}).$$

In terms of the original generalized posynomial $f$ and variables $x$ and $\tilde{x}$, we have the inequality

$$f(x_1^\theta \tilde{x}_1^{1-\theta}, \ldots, x_n^\theta \tilde{x}_n^{1-\theta}) \leq f(x_1, \ldots, x_n)^\theta f(\tilde{x}_1, \ldots, \tilde{x}_n)^{1-\theta},$$

for any $\theta$ with $0 \leq \theta \leq 1$.

## 5.4 Generalized geometric program

A *generalized geometric program* (GGP) is an optimization problem of the form

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m, \\ & g_i(x) = 1, & i = 1, \ldots, p, \end{array} \tag{25}$$

where $g_1, \ldots, g_p$ are monomials and $f_0, \ldots, f_m$ are generalized posynomials. Since any posynomial is also a generalized posynomial, any GP is also a GGP.

While GGPs are much more general than GPs, *they can be mechanically converted to equivalent GPs* using the transformations described in §5.1 and §5.2. As a result, GGPs can be solved very reliably and efficiently, just like GPs. The conversion from GGP to GP can be done automatically by a parser, that automatically carries out the transformations described in §5.1 and §5.2 as it parses the expressions describing the problem. In particular, the GP modeler (or user of a GGP parser) *does not need to know the transformations* described in §5.1 and §5.2. The GP modeler only needs to know the rules for forming a valid GGP, which are very simple to state. There is no need for the user to ever see, or even know about, the extra variables introduced in the transformation from GGP to GP.

Unfortunately, the name 'generalized geometric program' has been used to refer to several different types of problems, in addition to the one above. For example, some authors have used the term to refer to what is usually called a *signomial program* (described in §9.1), a very different generalization of a GP, which in particular cannot be reduced to an equivalent GP, or easily solved.

Once we have the basic idea of a parser that scans a problem description, verifies that it is a valid GGP and transforms it to GP form (for numerical solution), we can add several useful extensions. The parser can also handle inequalities involving negative terms in expressions, negative powers, minima, or terms on the righthand side of inequalities, in cases when they can be transformed to valid GGP inequalities. For example, the inequality

$$x + y + z - \min\{\sqrt{xy}, (1 + xy)^{-0.3}\} \leq 0 \tag{26}$$

(which is certainly not a valid generalized posynomial inequality) could be handled by a parser by first replacing the minimum with a variable $t_1$ and two *upper bounds*, to obtain

$$x + y + z - t_1 \leq 0, \qquad t_1 \leq \sqrt{xy}, \qquad t_1 \leq (1 + xy)^{-0.3}.$$

Moving terms around (by adding or multiplying) we obtain

$$x + y + z \leq t_1, \qquad t_1 \leq \sqrt{xy}, \qquad t_1 t_2^{0.3} \leq 1, \qquad 1 + xy \leq t_2,$$

which is a set of posynomial inequalities. (Of course we have to be sure that the transformations are valid, which is the case in this example.)

The fact that a parser can recognize an inequality like (26) and transform it to a set of valid posynomial constraints is a double-edged sword. The ability to handle a wider variety of constraints makes the modeling job less constrained, and therefore easier. On the other hand, few people would immediately recognize that the inequality (26) can be transformed, or understand *why* it is a valid inequality. A user who does not understand why it is valid will have no idea how to modify the constraint (*e.g.*, add terms, change coefficients or exponents) so as to maintain a valid inequality.

# 6 GGP examples

In this section we give simple examples of GGP applications.

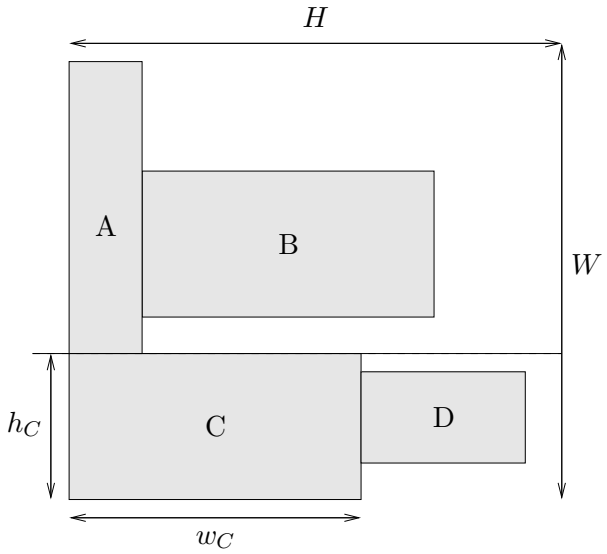


**Figure 2:** A floor planning example with four rectangles A, B, C, and D.

## 6.1 Floor planning

In a *floor planning problem*, we have some rectangles that are to be configured and placed in such a way that they do not overlap. The objective is usually to minimize the area of the *bounding box*, which is the smallest rectangle that contains the rectangles to be configured and placed. Each rectangle can be reconfigured, within some limits. For example we might fix the area of each rectangle, but not the width and height separately.

The constraint that the rectangles do not overlap makes the general floor planning problem a complicated combinatorial optimization or packing problem. However, if the *relative positioning* of the boxes is specified, the floor planning problem can be formulated as a GP, and therefore easily solved (see [17, §8.8], or the early paper [129]). For each pair of rectangles, we specify that one is left of, or right of, or above, or below the other. This information can be given in many ways, such as a table, a pair of graphs (one for horizontal and one for vertical relations), or a *slicing tree*; see [17, §8.8] or [141].

Instead of describing the general case, we will consider a specific example with 4 rectangles, labeled A, B, C, and D, shown in figure 2. The relative positioning constraints are:

- A is to the left of B.
- C is to the left of D.
- A and B are above C and D.

These guarantee that the rectangles do not overlap.

The width of A and B together is $w_A+w_B$, and the width of C and D together is $w_C+w_D$. Therefore the width of the bounding box is

$$W = \max\{w_A + w_B, w_C + w_D\}.$$

The height of A and B together is $\max\{h_A, h_B\}$, and the height of C and D together is $\max\{h_C, h_D\}$, so the height of the bounding box is

$$H = \max\{h_A, h_B\} + \max\{h_C, h_D\}.$$

The bounding box area is

$$WH = \max\{w_A + w_B, w_C + w_D\} \left(\max\{h_A, h_B\} + \max\{h_C, h_D\}\right).$$

This expression looks complicated, but we recognize it as a generalized posynomial of the variables $w_A, \ldots, w_D$ and $h_A, \ldots, h_D$.

The problem of minimizing bounding box area, with given rectangle areas and limits on aspect ratios, is

$$\begin{array}{ll} \text{minimize} & \max\{w_A + w_B, w_C + w_D\} \left(\max\{h_A, h_B\} + \max\{h_C, h_D\}\right) \\ \text{subject to} & h_A w_A = a, \quad h_B w_B = b, \quad h_C w_C = c, \quad h_D w_D = d, \\ & 1/\alpha_{\max} \leq h_A/w_A \leq \alpha_{\max}, \ldots, 1/\alpha_{\max} \leq h_D/w_D \leq \alpha_{\max}. \end{array}$$

This is a GGP, with variables $w_A, \ldots, w_D$ and $h_A, \ldots, h_D$. The parameters $a$, $b$, $c$, and $d$ are the given rectangle areas, and $\alpha_{\max}$ is the maximum allowed aspect ratio. The more general case (*i.e.*, with many more rectangles, and more complex relative positioning constraints) is also easily posed as a GGP.

Let's look at a specific numerical instance of our simple example, with areas

$$a = 0.2, \qquad b = 0.5, \qquad c = 1.5, \qquad d = 0.5.$$

Figure 3 shows optimal trade-off curve of minimum bounding box area versus the maximum aspect ratio $\alpha_{\max}$. Naturally, the minimum area obtained decreases as we relax (*i.e.*, loosen) the aspect ratio constraint. At the upper left is the design corresponding to $\alpha_{\max} = 1$. In this case each of the rectangles is fixed to be square. The flat portion at the right part of the trade-off curve is also easily understood. With a loose enough constraint on aspect ratio, the rectangles can configure themselves so that they give a perfect packing; there is no unused space inside the bounding box. In this case, the bounding box area is just the sum of the rectangle areas (which are given), which is clearly optimal. The trade-off plot shows that this perfect packing occurs for $\alpha_{\max} \geq 2.86$.

## 6.2 Digital circuit gate sizing

We consider a digital circuit consisting of a set of *gates* (that perform logic functions), interconnected by wires. For simplicity we'll assume that each gate has a single *output*, and

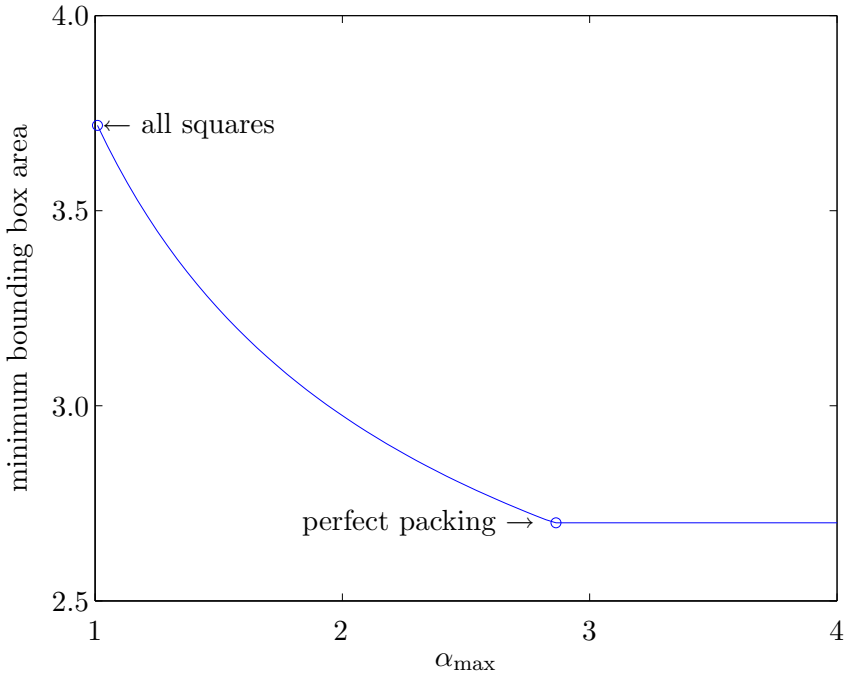


**Figure 3:** Optimal trade-off curves of minimum bounding box area versus maximum aspect ratio $\alpha_{\max}$.

one or more *inputs*. Each output is connected via a wire to the inputs of one or more other gates, or some external circuitry. Each input is connected to the output of another gate, or to some external circuitry. A *path* through the circuit is a sequence of gates, with each gate's output connected to the following gate's input. We assume that the circuit topology has no loops, *i.e.*, no paths start and end at the same gate.

Each gate is to be *sized*. In the simplest case, each gate has a variable $x_i \geq 1$ associated with it, which gives a scale factor for the size of the transistors it is made from. The scale factors of the gates, which are the optimization variables, affect the total circuit area, the power consumed by the circuit, and the speed of the circuit (which determines how fast the circuit can operate).

The area of a scaled gate is proportional to the scale factor $x_i$, so total circuit area has the form

$$A = \sum_{i=1}^{n} a_i x_i,$$

where $a_i$ is the area of gate $i$ with unit scaling. Similarly, the energy lost when a gate transitions (*i.e.*, its output changes logic value) is also approximately proportional to the scale factor. The total power consumed by the circuit has the form

$$P = \sum_{i=1}^{n} f_i e_i x_i,$$

where $f_i$ is the frequency of transition of the gate, and $e_i$ is the energy lost when the gate

transitions. Total circuit power and area are both posynomial functions of the scale factors.

Each gate has an *input capacitance* $C_i$, which is an affine function of its scale factor:

$$C_i = \alpha_i + \beta_i x_i,$$

and a *driving resistance* $R_i$, which is approximately inversely proportional to the scale factor:

$$R_i = \gamma_i / x_i.$$

The *delay* $D_i$ of a gate is the product of its driving resistance, and the sum of the input capacitances of the gates its output is connected to (if it is not an output gate) or its load capacitance (if it is an output gate):

$$D_i = \begin{cases} R_i \sum_{j \in F(i)} C_j & \text{for } i \text{ not an output gate} \\ R_i C_i^{\text{out}} & \text{for } i \text{ an output gate.} \end{cases}$$

(Here $F(i)$ is the set of gates whose input is connected to the output of gate $i$ and $C_i^{\text{out}}$ represents the load capacitance of an output gate.) Combining these three formulas, we see that the gate delay $D_i$ is a posynomial function of the scale factors.

We measure the speed of the circuit using its maximum or worst-case delay $D$, which is the maximum total delay along any path through the circuit. Since the delay of each gate is posynomial, the total delay along any path is also posynomial (since it is a sum of gate delays). The maximum path delay, over all possible paths, is a generalized posynomial, since it is the maximum of a set of posynomials.

We can pose the digital circuit gate scaling problem as the problem of choosing the scale factors to give minimum delay, subject to limits on the total area and power:

$$\begin{array}{ll} \text{minimize} & D \\ \text{subject to} & P \leq P^{\max}, \qquad A \leq A^{\max}, \\ & x_i \geq 1, \quad i = 1, \ldots, n. \end{array}$$

(Here $P^{\max}$ and $A^{\max}$ are given limits on the total power and area.) Since $D$ is a generalized posynomial, this is a GGP.

As a specific example, consider the digital circuit shown in figure 4. This small circuit has 7 gates, and only 7 paths. The worst-case delay is given by

$$\begin{aligned} D \;=\; & \max\{D_1 + D_4 + D_6,\ D_1 + D_4 + D_7,\ D_2 + D_4 + D_6, \\ & \quad D_2 + D_4 + D_7,\ D_2 + D_5 + D_7,\ D_3 + D_5 + D_6,\ D_3 + D_7\}. \end{aligned} \tag{27}$$

In larger circuits, the number of paths can be very large, and it is not practical to form an expression like (27) for $D$, by listing the paths. A simple recursion for $D$ can be used to avoid enumerating all paths. We define $T_i$ as the maximum delay over all paths that end with gate $i$. (We can interpret $T_i$ as the latest time at which the output of gate $i$ can transition, assuming the input signals transition at $t = 0$.) We can compute $T_i$ via a recursion, as follows.

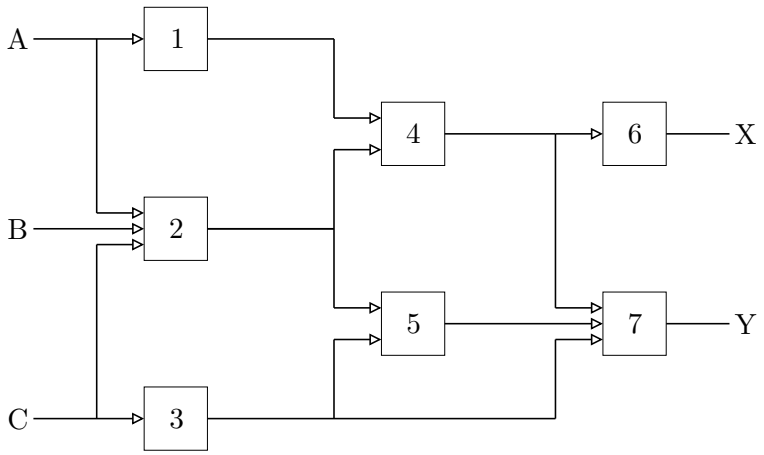


**Figure 4:** Digital circuit with 7 gates.

- For gates whose inputs are not connected to other gates, we take $T_i = D_i$.
- For other gates, we take $T_i = \max T_j + D_i$, where the maximum is over all gates that drive gate $i$.

Finally, we can express the delay $D$ as the maximum over all $T_i$, over all output gates. Note that this recursion shows that each $T_i$ is a generalized posynomial of the gate scaling factors.

For the particular example described above, the recursion is

$$\begin{aligned} T_i &= D_i, \quad i = 1, 2, 3, \\ T_4 &= \max\{T_1, T_2\} + D_4, \\ T_5 &= \max\{T_2, T_3\} + D_5, \\ T_6 &= T_4 + D_6, \\ T_7 &= \max\{T_3, T_4, T_5\} + D_7, \\ D &= \max\{T_6, T_7\}. \end{aligned}$$

(For this small example, the recursion gives no real savings over the expression (27) above based on enumeration of the paths, but in larger circuits the savings can be dramatic.)

We now consider a specific instance of the problem, with parameters

$$\begin{aligned} &a_i = 1, \quad \alpha_i = 1, \quad \beta_i = 1, \quad \gamma_i = 1, \quad i = 1, \ldots, 7, \\ &f_1 = 1, \quad f_2 = 0.8, \quad f_3 = 1, \quad f_4 = 0.7, \quad f_5 = 0.7, \quad f_6 = 0.5, \quad f_7 = 0.5, \\ &e_1 = 1, \quad e_2 = 2, \quad e_3 = 1, \quad e_4 = 1.5, \quad e_5 = 1.5, \quad e_6 = 1, \quad e_7 = 2, \end{aligned}$$

and load capacitances $C_6^{\text{out}} = 10$, $C_7^{\text{out}} = 10$. Figure 5 shows optimal trade-off curves of the minimum delay versus maximum allowed power, for three values of maximum area.

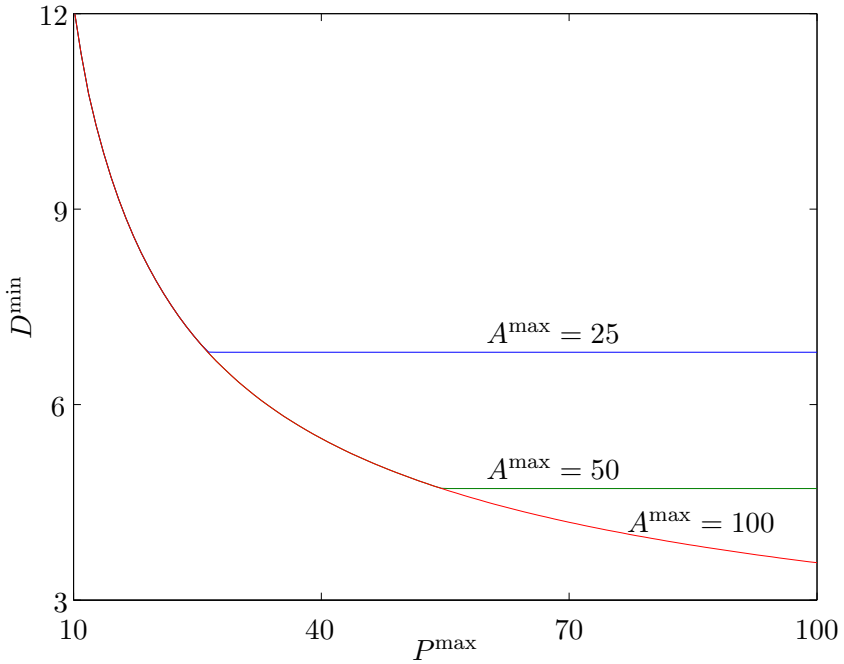


**Figure 5:** Optimal trade-off curves of minimum delay $D^{\min}$ versus maximum power $P^{\max}$, for three values of maximum area $A^{\max}$.

## 6.3 Truss design

We consider a simple mechanical design problem, expanded from an example given in [9, p8]. Figure 6 shows a two-bar truss with height $2h$ and width $w$. The two bars are cylindrical tubes with inner radius $r$ and outer radius $R$. We are interested in determining the values of $h$, $w$, $r$, and $R$ that minimize the weight of the truss subject to a number of constraints.

The cross-sectional area $A$ of the bars is given by

$$A = 2\pi(R^2 - r^2),$$

and the weight of the truss is proportional to the total volume of the bars, which is given by

$$2A\sqrt{w^2 + h^2}.$$

This is the cost function in the design problem. Since $A = 2\pi(R^2 - r^2)$, it is *not* a generalized posynomial in the original design variables $h$, $w$, $r$, and $R$. We will address this issue later.

The structure should be strong enough for two loading scenarios. In the first scenario a vertical force $F_1$ is applied to the node; in the second scenario the force is horizontal with magnitude $F_2$. The constraint that the truss should be strong enough to carry the load $F_1$ means that the stress caused by the external force $F_1$ must not exceed a given maximum value. To formulate this constraint, we first determine the forces in each bar when the structure is subjected to the vertical load $F_1$. From force equilibrium and the geometry of

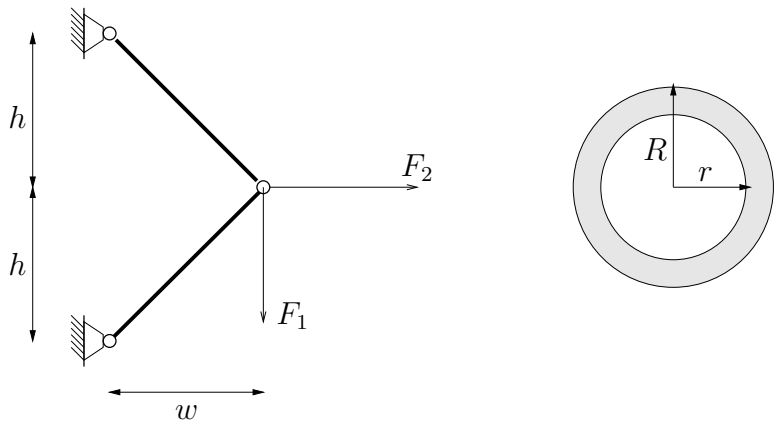


**Figure 6:** *Left.* Two-bar truss. *Right.* Cross section of the bars.

the problem we can determine that the magnitudes of the forces in two bars are equal and given by

$$\frac{\sqrt{w^2+h^2}}{2h}F_1.$$

The maximum force in each bar is equal to the cross-sectional area times the maximum allowable stress $\sigma$ (which is a given constant). This gives us the first constraint:

$$\frac{\sqrt{w^2+h^2}}{2h}F_1 \leq \sigma A.$$

When $F_2$ is applied, the magnitudes of the forces in two bars are again equal and given by

$$\frac{\sqrt{w^2+h^2}}{2w}F_2,$$

which gives us the second constraint:

$$\frac{\sqrt{w^2+h^2}}{2w}F_2 \leq \sigma A.$$

We also impose limits

$$w_{\min} \leq w \leq w_{\max}, \qquad h_{\min} \leq h \leq h_{\max}$$

on the width and the height of the structure, and limits

$$1.1r \leq R \leq R_{\max}$$

on the outer bar radius.

The design problem is:

$$\begin{array}{ll} \text{minimize} & 2A\sqrt{w^2+h^2} \\ \text{subject to} & F_1\sqrt{w^2+h^2}/h \le \sigma A, \\ & F_2\sqrt{w^2+h^2}/w \le \sigma A, \\ & w_{\min} \le w \le w_{\max}, \qquad h_{\min} \le h \le h_{\max}, \\ & 1.1r \le R \le R_{\max}. \end{array}$$

This is *not* a GGP in the variables $h$, $w$, $r$, and $R$, since $A = 2\pi(R^2 - r^2)$ is not a monomial function of the variables. But a simple change of variables converts it to a GGP. We will use $A$ as a *variable*, instead of $R$. For $R$, we use

$$R = \sqrt{A/(2\pi) + r^2}, \tag{28}$$

which a generalized posynomial in $r$ and $A$. Using the variables $h$, $w$, $r$, and $A$, the problem above is almost a GGP. The only constraint that doesn't fit the required form is

$$1.1r \le R = \sqrt{A/(2\pi) + r^2}.$$

But we can express this as $1.1^2 r^2 \le A/(2\pi) + r^2$, *i.e.*,

$$0.21r^2 \le A/(2\pi),$$

which is compatible with GGP.

In summary, the truss design problem can be expressed as the GGP

$$\begin{array}{ll} \text{minimize} & 2A\sqrt{w^2+h^2} \\ \text{subject to} & F_1\sqrt{w^2+h^2}/h \le \sigma A, \\ & F_2\sqrt{w^2+h^2}/w \le \sigma A, \\ & w_{\min} \le w \le w_{\max}, \qquad h_{\min} \le h \le h_{\max}, \\ & \sqrt{A/(2\pi) + r^2} \le R_{\max}, \\ & 0.21r^2 \le A/(2\pi), \end{array}$$

with variables $h$, $w$, $r$, and $A$. (After solving this GGP, we can recover $R$ from $A$ using (28).)

## 6.4 Wire sizing

We consider the problem of determining the widths $w_1, \ldots, w_n$ of $n$ wire segments in an interconnect network in an integrated circuit. The interconnect network forms a tree; its root is driven by the input signal (which is to be distributed), which is modeled as a voltage source and a series resistance. Each wire segment has a given capacitive load $C_i$ connected to it. A simple interconnect network is shown in figure 7.

We will use a simple $\pi$ model for each wire segment, as shown in figure 8. The wire resistance and capacitances are given by

$$R_i = \alpha_i \frac{l_i}{w_i}, \qquad \overline{C}_i = \beta_i l_i w_i + \gamma_i l_i,$$

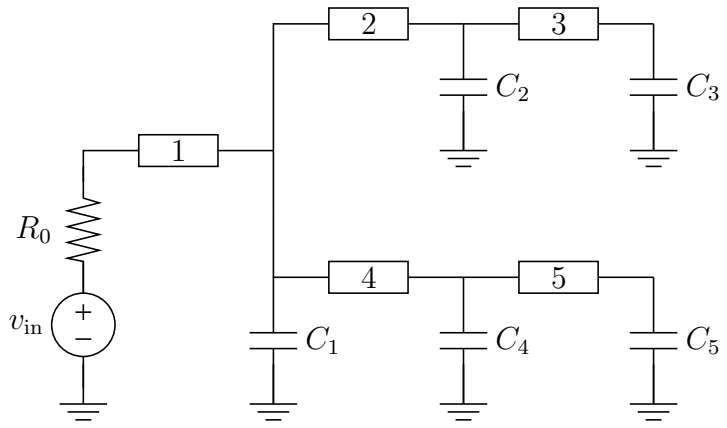


**Figure 7:** An interconnect network consisting of an input (the voltage source and resistance) driving a tree of 5 wire segments (shown as boxes labeled $1, \ldots, 5$) and capacitive loads $C_1, \ldots, C_5$.

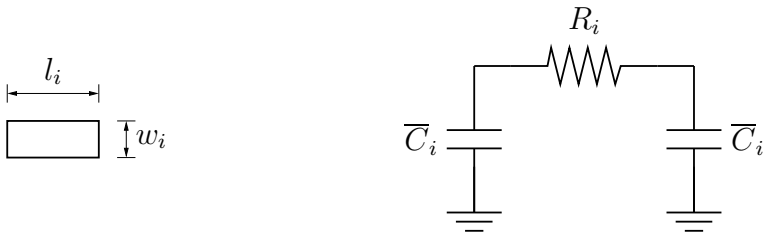


**Figure 8:** Wire segment with length $l_i$ and width $w_i$ (left) and its $\pi$ model (right).

where $l_i$ and $w_i$ are the length and width of the wire segment, and $\alpha_i$, $\beta_i$, and $\gamma_i$ are positive constants that depend on the physical properties of the routing layer of the wire segment. The wire segment resistance and capacitance are both posynomial functions of the wire widths $w_i$, which will be our design variables.

Substituting this $\pi$ model for each of the wire segments, the interconnect network becomes a resistor-capacitor (RC) tree. Each branch has the associated wire resistance. Each node has several capacitances to ground: the load capacitance, the capacitance contributed by the upstream wire segment, and the capacitances contributed by each of the downstream wire segments. The capacitances at each node can be added together. The resulting RC tree has resistances and capacitances which are posynomial functions of the wire segment widths $w_i$. As an example, the network in figure 7 yields the RC tree shown in figure 9, where

$$\begin{aligned}
\tilde{C}_0 &= \overline{C}_1, \\
\tilde{C}_1 &= C_1 + \overline{C}_1 + \overline{C}_2 + \overline{C}_4, \\
\tilde{C}_2 &= C_2 + \overline{C}_2 + \overline{C}_3, \\
\tilde{C}_4 &= C_4 + \overline{C}_4 + \overline{C}_5, \\
\tilde{C}_5 &= C_5 + \overline{C}_5.
\end{aligned}$$

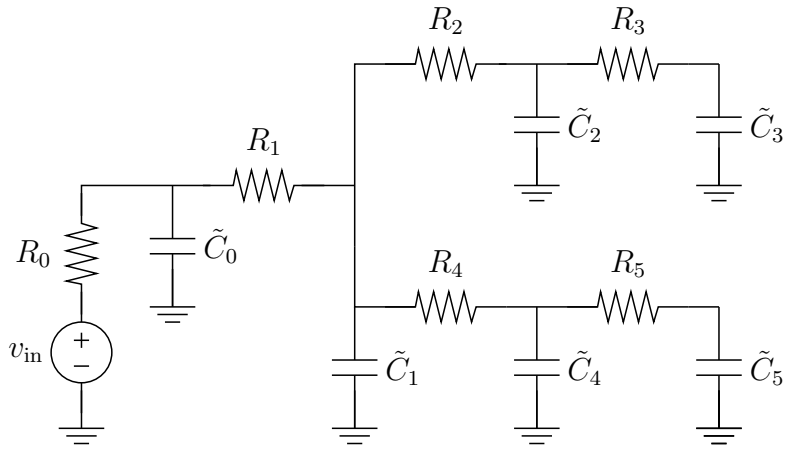


**Figure 9:** RC tree model of the interconnect network in figure 7 using the $\pi$ model.

When the voltage source switches from one value to another, there is a delay before the voltage at each capacitor converges to the new value. We will use the *Elmore delay* to measure this. (The Elmore delay to capacitor $i$ is the area under its voltage versus time plot, for $t \geq 0$, when the input voltage source switches from 1 to 0 at $t = 0$.) For an RC tree circuit, the Elmore delay $D_k$ to capacitor $k$ is given by the formula

$$D_k = \sum_{i=1}^{n} \tilde{C}_i \left( \sum R\text{'s upstream from capacitors } k \text{ and } i \right).$$

The Elmore delay is a sum of products of $\tilde{C}_i$'s and $R_j$'s, and therefore is a posynomial function of the wire widths (since each of these is a posynomial function of the wire widths). The *critical delay* of the interconnect network is the largest Elmore delay to any capacitor in the network:

$$D = \max\{D_1, \ldots, D_n\}.$$

(This maximum always occurs at the leaves of the tree.) The critical delay is a generalized posynomial of the wire widths, since it is a maximum of a set of posynomials.

As an example, the Elmore delays to (leaf) capacitors 3 and 5 in the RC tree in figure 9 are given by

$$\begin{aligned} D_3 &= \tilde{C}_3(R_0 + R_1 + R_2 + R_3) + \tilde{C}_2(R_0 + R_1 + R_2) + \tilde{C}_0 R_0 + \tilde{C}_1(R_0 + R_1) \\ &\quad + \tilde{C}_4(R_0 + R_1) + \tilde{C}_5(R_0 + R_1) + \tilde{C}_6(R_0 + R_1), \\ D_5 &= \tilde{C}_5(R_0 + R_1 + R_4 + R_5) + \tilde{C}_4(R_0 + R_1 + R_4) + \tilde{C}_0 R_0 + \tilde{C}_1(R_0 + R_1) \\ &\quad + \tilde{C}_2(R_0 + R_1) + \tilde{C}_3(R_0 + R_1) + \tilde{C}_6(R_0 + R_1). \end{aligned}$$

For this interconnect network, the critical delay is $D = \max\{D_3, D_5\}$.

Now we can formulate the wire sizing problem, *i.e.*, the problem of choosing the wire segment widths $w_1, \ldots, w_n$. We impose lower and upper bounds on the wire widths,

$$w_i^{\min} \leq w_i \leq w_i^{\max},$$

as well as a limit on the total wire area,

$$l_1 w_1 + \cdots + l_n w_n \leq A^{\max}.$$

Taking critical delay as objective, we obtain the problem

$$\begin{array}{ll} \text{minimize} & D \\ \text{subject to} & w_i^{\min} \leq w_i \leq w_i^{\max}, \quad i = 1, \ldots, n \\ & l_1 w_1 + \cdots + l_n w_n \leq A^{\max}, \end{array} \tag{29}$$

with variables $w_1, \ldots, w_n$. This is a GGP.

This formulation can be extended to use more accurate models of wire segment resistance and capacitance, as long as they are generalized posynomials of the wire widths. Wire sizing using Elmore delay goes back to Fishburn and Dunlop [60]; for some more recent work on wire (and device) sizing via Elmore delay, see, *e.g.*, [142, 138, 135].

# 7 More transformations

In this section we describe a few more advanced techniques used to express problems in GP (or GGP) form.

## 7.1 Function composition

In §5.4 we described methods for handling problems whose objective or constraint functions involve composition with the positive power function or the maximum function. It's possible to handle composition with many other functions. As a common example, we consider the function $1/(1-z)$. Suppose we have the constraint

$$\frac{1}{1-q(x)} + f(x) \leq 1, \tag{30}$$

where $q$ and $f$ are generalized posynomials, and we have the implicit constraint $q(x) < 1$. We replace the first term by a new variable $t$, along with the constraint $1/(1-q(x)) \leq t$, which can be expressed as the generalized posynomial inequality $q(x) + 1/t \leq 1$, to obtain

$$t + f(x) \leq 1, \quad q(x) + 1/t \leq 1.$$

This pair of inequalities is equivalent to the original one (30) above, using the same argument as in §5.4.

A more general variation on this transformation can be used to handle a constraint such as

$$\frac{p(x)}{r(x)-q(x)} + f(x) \leq 1,$$

where $r$ is monomial, $p$, $q$, and $f$ are generalized posynomials, and we have the implicit constraint $q(x) < r(x)$. We replace this inequality constraint with

$$t + f(x) \leq 1, \qquad q(x) + p(x)/t \leq r(x),$$

where $t$ is a new variable.

The idea that composition of a generalized posynomial with $1/(1-z)$ can be handled in GP can be guessed from its Taylor series,

$$\frac{1}{1-z} = 1 + z + z^2 + \cdots,$$

which is a limit of polynomials with positive coefficients. This analysis suggests that we can handle composition of a generalized posynomial with any function whose series expansion has no negative coefficients, at least approximately, by truncating the series. In some cases (such as $1/(1-z)$), the composition can be handled exactly.

Another example is the exponential function. Since the Taylor series for the exponential has all coefficients positive, we can guess that the exponential of a generalized posynomial can by handled as it were a generalized posynomial. One good approximation is

$$e^{f(x)} \approx (1 + f(x)/a)^a,$$

where $a$ is large. If $f$ is a generalized posynomial, the righthand side is a generalized posynomial. This approximation is good for small enough $f(x)$; if $f(x)$ is known to be near, say, the number $b$, we can use the approximation

$$e^{f(x)} = e^b e^{f(x)-b} \approx e^b (1 + (f(x) - b)/a)^a,$$

which is a generalized posynomial provided $a > b$.

It's also possible to handle exponentials of posynomials exactly, *i.e.*, without approximation. We replace a term of the form $e^{f(x)}$ with a new variable $t$, which can be used anywhere a posynomial can be, and we add the constraint $e^{f(x)} \leq t$ to our problem. This results in a problem that would be a GP, except for the exponential constraint $e^{f(x)} \leq t$. To solve the problem, we carry out the same logarithmic transformation used to solve GPs. First we take the logarithm of the original variables $x$ as well as the new variable $t$, so our variables become $y = \log x$ and $s = \log t$, and then we take the logarithm of both sides of each constraint. The posynomial objective, posynomial inequality, and monomial equality constraints transform as usual to a convex objective and inequality constraints, and linear equality constraints. The exponential constraint becomes

$$\log e^{f(e^y)} \leq \log e^s,$$

*i.e.*, $f(e^y) \leq s$, which is a convex constraint on $y$ and $s$, since $f(e^y)$ is a convex function of $y$. Thus, the logarithmic transformation yields a convex problem, which is not quite the same as the one obtained from a standard GP, but is still easily solved.

In summary, exponential terms can be handled exactly, but with the disadvantage of requiring software that can handle a wider class of problems than GP. For this reason, the most common approach is to use an approximation such as the one described above.

## 7.2 Additive log terms

In the preceding section, we saw that the exponential of a generalized posynomial can be well approximated as a generalized posynomial, and therefore used in the objective or inequality constraints, anywhere a generalized posynomial can. It turns out that the logarithm of a generalized posynomial can also be (approximately) incorporated in the inequality constraints, but in more restricted ways. Suppose we have a constraint of the form

$$f(x) + \log q(x) \leq 1,$$

where $f$ and $q$ are generalized posynomials. This constraint is a bit different from the ones we have seen so far, since the lefthand side can be *negative*. To (approximately) handle this constraint, we use the approximation

$$\log u \approx a(u^{1/a} - 1),$$

valid for large $a$, to obtain

$$f(x) + a(q(x)^{1/a} - 1) \leq 1.$$

This inequality is *not* a generalized posynomial inequality, but it can be expressed as the generalized posynomial inequality

$$f(x) + aq(x)^{1/a} \leq 1 + a.$$

Like exponentials, additive log terms can be also be handled exactly, again with the disadvantage of requiring specialized software. Using the variables $y = \log x$, we can write $f(x) + \log q(x) \leq 1$ as

$$f(e^y) + \log q(e^y) \leq 1.$$

This is a convex constraint, and so can be handled directly.

## 7.3 Mixed linear geometric programming

In generalized geometric programming, the righthand side of any inequality constraint must be a monomial. The righthand side of an inequality constraint can never be a sum of two or more terms (except in trivial cases, such as when one of the terms is a multiple of the other). There is one special case, however, when it is possible to handle constraints in which the righthand side is a sum of terms. Consider a problem of the form

$$\begin{array}{ll} \text{minimize} & f_0(x) + h_0(z) \\ \text{subject to} & f_i(x) \leq h_i(z), \quad i = 1, \ldots, m, \\ & g_i(x) = 1, \quad i = 1, \ldots, p, \end{array} \tag{31}$$

where the variables are $x \in \mathbf{R}^n$ and $z \in \mathbf{R}^k$, $f_0, \ldots, f_m$ are generalized posynomials, $g_1, \ldots, g_p$ are monomials, and $h_0, \ldots, h_m$ are affine, *i.e.*, linear functions plus a constant. This problem form can be thought of as a combination or hybrid of a GGP and a linear program (LP):

without the variable $x$, it reduces to an LP; without the $z$ variable (so the affine functions reduce to constants), it reduces to a GGP. For this reason the problem (31) is called a *mixed linear geometric program*. In a mixed linear geometric program, we partition the optimization variables into two groups: the variables $x_1, \ldots, x_n$, which appear in the posynomials and monomials, and the variables $z_1, \ldots, z_k$, which appear in the affine function part of the objective, and on the righthand side of the inequality constraints.

Mixed linear geometric programs can be solved very efficiently, just like GGPs. The method uses the usual logarithmic transform for the variables $x_1, \ldots, x_n$, but keeps the variables $z_1, \ldots, z_k$ as they are; moreover, we do not take the logarithm of both sides of the mixed constraints, as we do for the standard posynomial inequality or monomial equality constraints. This transformation yields

$$f_i(e^y) \leq h_i(z), \quad i = 1, \ldots, m,$$

which are convex constraints in $y$ and $z$, so the mixed linear geometric program becomes a convex optimization problem. This problem is not the same as would be obtained from a GP or GGP, but nevertheless is easily solved.

## 7.4 Generalized posynomial equality constraints

In a GGP (or GP), the only equality constraints allowed involve monomials. In some special cases, however, it is possible to handle generalized posynomial equality constraints in GGP. This idea can be traced back at least to 1978 [153].

We first describe the method for the case with only one generalized posynomial equality constraint (since it is readily generalizes to the case of multiple generalized posynomial equality constraints). We consider the problem

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m, \\ & g_i(x) = 1, & i = 1, \ldots, p, \\ & h(x) = 1, & \end{array} \tag{32}$$

with optimization variables $x_1, \ldots, x_n$, where $g_i$ are monomials, and $f_i$ and $h$ are generalized posynomials. Without the last generalized posynomial equality constraint, this is a GGP. With the last constraint, however, the problem is very difficult to solve, at least globally. (It is a special case of a *signomial program*, discussed in §9.)

We first form the *GGP relaxation* of the problem (32),

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m, \\ & g_i(x) = 1, & i = 1, \ldots, p, \\ & h(x) \leq 1, & \end{array} \tag{33}$$

by replacing the generalized posynomial equality with an inequality. This problem is a GGP and therefore easily solved. It is called a relaxation since we have relaxed (or loosened) the last equality constraint, by allowing $h(x) < 1$ as well as $h(x) = 1$.

Let $\bar{x}$ be an optimal solution of the relaxed problem (33). If we have $h(\bar{x}) = 1$, then $\bar{x}$ is also an optimal solution of the original problem (32), and we are done. Of course this need not happen; we can have $h(\bar{x}) < 1$, in which case $\bar{x}$ is not feasible for the original problem. In some cases, though, we can modify the point $\bar{x}$ so that it remains optimal for the relaxation (33), but also satisfies the generalized posynomial equality constraint (and therefore is optimal for the original problem (32)).

Suppose we can find a variable $x_k$ with the following properties:

- The variable $x_k$ does not appear in any of the monomial equality constraint functions.
- The objective and inequality constraint functions $f_0, \ldots, f_m$ are all *monotone decreasing* in $x_k$, *i.e.*, if we increase $x_k$ (holding all other variables constant), the functions $f_0, \ldots, f_m$ decrease, or remain constant.
- The generalized posynomial function $h$ is *monotone strictly increasing* in $x_k$, *i.e.*, if we increase $x_k$ (holding all other variables constant), the function $h$ increases.

Now suppose we start with the point $\bar{x}$, and increase $x_k$, *i.e.*, we consider the point

$$\tilde{x} = (\bar{x}_1, \ldots, \bar{x}_{k-1}, \bar{x}_k + u, \bar{x}_{k+1}, \ldots, \bar{x}_n),$$

where $u$ is a scalar that we increase from $u = 0$. By the first property, the monomial equality constraints are unaffected, so the point $\tilde{x}$ satisfies them, for any value of $u$. By the second property, the point $\tilde{x}$ continues to satisfy the inequality constraints, since increasing $u$ decreases (or keeps constant) the functions $f_1, \ldots, f_m$. The same argument tells us than the point $\tilde{x}$ has an objective value that is the same, or better than, the point $\bar{x}$. As we increase $u$, $h(\tilde{x})$ increases. Now we simply increase $u$ until we have $h(\tilde{x}) = 1$ ($h$ can be increased as much as we like, as a consequence of the convexity of $\log h(e^y)$, where $x = e^y$). The resulting $\tilde{x}$ is an optimal solution of the problem (32). Increasing $x_k$ until the generalized posynomial equality constraint is satisfied is called *tightening*.

The same method can be applied when the monotonicity properties are reversed, *i.e.*, if $f_0, \ldots, f_m$ are monotone increasing functions of $x_k$, and $h$ is strictly monotone decreasing in $x_k$. In this case we tighten by decreasing $x_k$ until the generalized posynomial constraint is satisfied.

As with the other tricks and transformations described, this one can be automated. Starting with a problem, with the objective and constraint functions given as expressions involving variables, powers, sum, and maximum, it is easy to check the monotonicity properties of the functions, and to determine if a variable $x_k$ with the required monotonicity properties exists.

The same idea can be used to solve a problem with multiple generalized posynomial equality constraints,

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m, \\ & g_i(x) = 1, & i = 1, \ldots, p, \\ & h_i(x) = 1, & i = 1, \ldots, k, \end{array} \tag{34}$$

where $f_i$ and $h_i$ are generalized posynomials, and $g_i$ are monomials. We form and solve the GGP relaxation

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m, \\ & g_i(x) = 1, & i = 1, \ldots, p, \\ & h_i(x) \leq 1, & i = 1, \ldots, k, \end{array} \tag{35}$$

and let $\bar{x}$ denote an optimal solution. In this case, we need a different variable for each generalized posynomial equality constraint, with the monotonicity properties given above. We re-order the variables so that $x_1$ is the variable we increase to cause $h_1(x)$ to increase to one, $x_2$ is the variable we increase to cause $h_2(x)$ to increase to one, and so on. The simple method of increasing or decreasing one of the variables until the equality constraint is satisfied cannot be used in this case, since increasing $x_1$ to make $h_1(x)$ increase to one can *decrease* $h_2(x)$ (and vice versa). But we can find a common adjustment of the variables $x_1, \ldots, x_k$ that results in all the equality constraints being tightened simultaneously, by forming and solving an auxiliary GGP:

$$\begin{array}{lll} \text{maximize} & x_1 \cdots x_k & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m \\ & g_i(x) = 1, & i = 1, \ldots, p \\ & h_i(x) \leq 1, & i = 1, \ldots, k \\ & f_0(x) \leq f^\star, & \end{array} \tag{36}$$

where $f^\star$ is the optimal value of the relaxed problem (35). The feasible set of this problem is the optimal set of the relaxed problem (35); the objective in the auxiliary problem is to maximize the product of the variables used to tighten the equality constraints. Any optimal solution of this auxiliary problem (36) is an optimal solution of the original problem (34). (We can use any objective that puts pressure on the variables $x_1, \ldots, x_k$ to increase; for example, we can minimize $1/x_1 + \cdots + 1/x_k$.)

As in the case of a single generalized posynomial constraint, it is easy to automate the task of finding a set variables used to tighten the equality constraints, and form the auxiliary problem (36).

# 8 Approximation and fitting

In this section we first address some fundamental theoretical questions:

- What functions can be approximated by monomials or generalized posynomials?
- When can an optimization problem be approximated as a GGP?

We then discuss practical methods for approximating a given function, or some given data, by a monomial or generalized posynomial function. These fitting and approximation methods can be used to derive GP compatible approximate expressions and problem formulations.

## 8.1 Theory

Suppose $f$ is a positive function of positive variables $x_1, \ldots, x_n$. We consider the question: When can $f$ be approximated by a monomial or generalized posynomial function?

At least in principle, the answer is simple. We first form the logarithmically transformed function

$$F(y) = \log f(e^y)$$

(which is used to transform a GP to a convex problem). Then we have the following:

- $f$ can be approximated by a monomial if and only if $F$ can be approximated by an affine function, *i.e.*, a constant plus a linear function.
- $f$ can be approximated by a generalized posynomial if and only if $F$ can be approximated by a convex function.

Here we are being informal about what exactly we mean by 'can be approximated', but the statements can easily be made precise.

We have already seen that when $f$ is a monomial, the logarithmically transformed function $F$ is affine, and that when $f$ is a generalized posynomial, $F$ is convex. It's easy to show the converse for monomials: if $F$ is affine, then $f$ is a monomial. The interesting part here is the converse for generalized posynomials, *i.e.*, the observation that if $F$ can be approximated by a convex function, then $f$ can be approximated by a generalized posynomial.

To show this, suppose $F \approx \phi$, where $\phi$ is a convex function. A basic result of convex analysis is that any convex function can be arbitrarily well approximated by a piecewise linear convex function, expressed as a maximum of a set of affine functions; see, *e.g.*, [17]. Thus, we have $b_{ki}$ for which

$$F(y) \approx \phi(y) \approx \max_{i=1,\ldots,p} (b_{0i} + b_{1i}y_1 + \cdots + b_{ni}y_n).$$

Taking the exponential of both sides, and changing variables back to $x$, we have

$$f(x) \approx \max_{i=1,\ldots,p} e^{b_{0i}} x^{b_{1i}} \cdots x^{b_{ni}}.$$

The righthand side is the maximum of $p$ monomials, and is therefore a generalized posynomial. (Such a function is sometimes called a *max-monomial*.)

It can be difficult to determine whether a function of many variables is convex (or nearly convex), but there are a number of techniques that can be used to verify or disprove convexity (see [17] for much more on this topic). For the special case $n = 1$, convexity is readily determined by simply plotting the function $F(y)$, and checking that it has positive (upward) curvature. This is the same as plotting $f(x)$ on a log-log plot, and checking that the resulting graph has positive curvature. For functions of many variables, we can use the fact that a function is convex only if it is convex when restricted to any line. Thus, we can plot $F(y_0+tv)$ versus $t$, for various values of $y_0$ and $v$. If any of these plots reveal negative (downward) curvature, then $F$ is not convex.

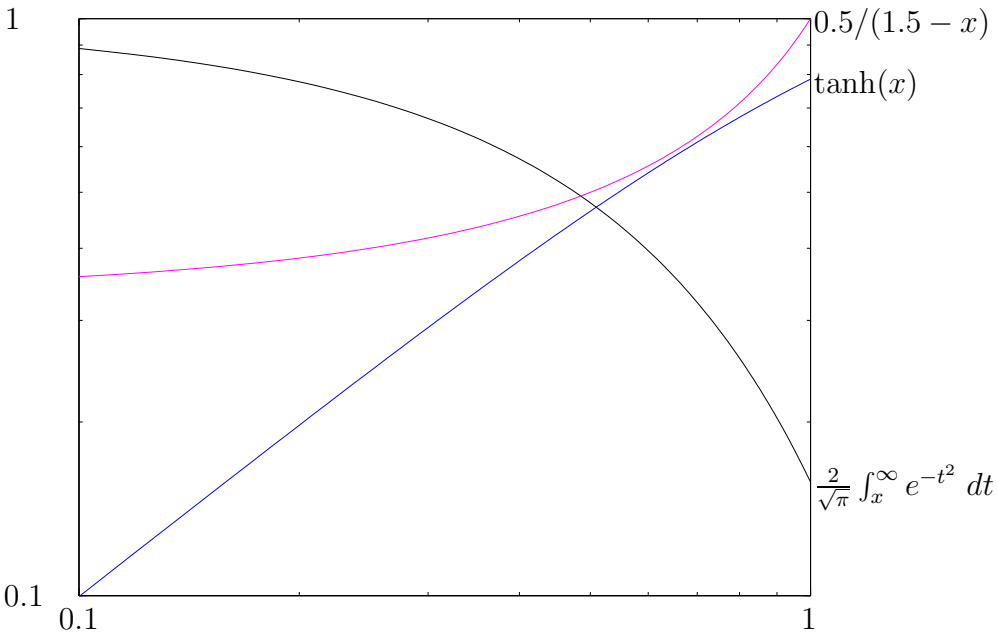


**Figure 10:** The three functions given in (37), on a log-log plot.

As an example, we consider the three functions of one variable,

$$\tanh(x), \qquad \frac{0.5}{1.5-x}, \qquad \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}\,dt, \tag{37}$$

and ask whether each can be approximated by a monomial or generalized posynomial, over the range $0.01 \leq x \leq 1$. These functions are plotted on a log-log plot in figure 10. From the figure we can see that the first function can be reasonably well fit by a monomial, since its graph is relatively straight. The second function can be fit by a generalized posynomial, since its graph is convex, *i.e.*, curves upward, on this log-log plot. The third function cannot be fit very well by a generalized posynomial, since its graph exhibits substantial downward curvature. (Note, however, that the reciprocal of the third function has upward curvature, and therefore can be fit by a generalized posynomial.)

Convexity of $F$ can be stated in terms of $f$, our original function. It means that $f$ must satisfy the inequality (8),

$$f(x_1^\theta \tilde{x}_1^{1-\theta}, \ldots, x_n^\theta \tilde{x}_n^{1-\theta}) \leq f(x_1, \ldots, x_n)^\theta f(\tilde{x}_1, \ldots, \tilde{x}_n)^{1-\theta},$$

for any $\theta$ with $0 \leq \theta \leq 1$. In other words, when $f$ is evaluated at a weighted geometric mean of two points, it cannot be more than the weighted geometric mean of the function $f$ evaluated at the two points. In §2.5, we noted that this inequality is satisfied by posynomials; what we know now is that its approximate satisfaction is the necessary and sufficient condition for a function to have a generalized posynomial approximation.

We conclude this section by asking: When can an optimization problem of the form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 1, \quad i = 1, \ldots, m, \\ & g_i(x) = 1, \quad i = 1, \ldots, p, \end{array} \tag{38}$$

with positive variables $x_1, \ldots, x_n$, be approximated by a GGP? From the discussion above, we see that the answer is:

- The transformed objective and inequality constraint functions $F_i(y) = \log f_i(e^y)$ must be nearly convex.
- The transformed equality constraint functions $G_i(y) = \log G_i(e^y)$ must be nearly affine.

## 8.2 Local monomial approximation

We consider the problem of finding a monomial approximation of a differentiable positive function $f$ near a point $x$ (with $x_i > 0$). Asking for a monomial approximation of $f(x)$ near $x$ corresponds to asking for an affine approximation of $F(y) = \log f(e^y)$ near $y = \log x$. But such an approximation is provided by the first order Taylor approximation of $F$:

$$F(z) \approx F(y) + \sum_{i=1}^{n} \frac{\partial F}{\partial y_i}(z_i - y_i), \tag{39}$$

which is valid for $z \approx y$. (The derivatives here are evaluated at $y$.)

We have

$$\frac{\partial F}{\partial y_i} = \frac{1}{f(e^y)} \, \frac{\partial f}{\partial x_i} \, e^{y_i} = \frac{x_i}{f(x)} \frac{\partial f}{\partial x_i},$$

where the partial derivatives of $f$ are evaluated at $x$. Using this formula, and taking the exponential of (39), we get

$$f(e^z) \approx f(x) \prod_{i=1}^{n} \exp\left( \frac{x_i}{f(x)} \frac{\partial f}{\partial x_i}(z_i - y_i) \right)$$

Defining $w_i = e^{z_i}$, we can express this as

$$f(w) \approx f(x) \prod_{i=1}^{n} \left( \frac{w_i}{x_i} \right)^{a_i}, \tag{40}$$

where

$$a_i = \frac{x_i}{f(x)} \frac{\partial f}{\partial x_i}$$

(evaluated at $x$). The approximation (40) is valid for $w \approx x$; the righthand side is the best local monomial approximation of $f$ near $x$.

## 8.3 Monomial fitting

Suppose we are given data points

$$(x^{(i)}, f^{(i)}), \quad i = 1, \ldots, N,$$

where $x^{(i)} \in \mathbf{R}^n$ are positive vectors and $f^{(i)}$ are positive constants. The goal is to fit the data with a monomial $f(x) = cx_1^{a_1} \cdots x_n^{a_n}$. In other words, we want to find $c > 0$ and $a_1, \ldots, a_n$ so that

$$f(x^{(i)}) \approx f^{(i)}, \qquad i = 1, \ldots, N.$$

This monomial fitting task can be done several ways.

The simplest approach follows the method used for monomial function approximation. We let $y^{(i)} = \log x^{(i)}$, and replace $f(x^{(i)}) \approx f^{(i)}$ with $\log f(e^{y^{(i)}}) \approx \log f^{(i)}$, to get

$$\log c + a_1 y_1^{(i)} + \cdots + a_n y_n^{(i)} \approx \log f^{(i)}, \qquad i = 1, \ldots, N.$$

This is a set of linear (approximate) equalities, in the unknowns $\log c$, $a_1, \ldots, a_n$. We can find an approximate solution via least-squares, by choosing $\log c$ and $a_1, \ldots, a_n$ to minimize the sum of the squared errors,

$$\sum_{i=1}^{N} \left( \log c + a_1 y_1^{(i)} + \cdots + a_n y_n^{(i)} - \log f^{(i)} \right)^2.$$

(In other words, we use simple linear regression to find $\log c$ and $a_1, \ldots, a_n$, given the data $y^{(1)}, \ldots, y^{(N)}$ and $f^{(1)}, \ldots, f^{(N)}$.)

This simple least-squares method can be modified and extended in several ways. We can introduce weights in the least-squares objective, in order to give more or less weight to the individual errors. We can also *regularize*, by adding a term

$$\lambda \sum_{i=1}^{n} a_i^2$$

to the least-squares objective, where $\lambda$ is a positive constant. By adding this term to the least-squares objective (which penalizes the approximation error) we add a penalty for large values of the exponents $a_i$. The parameter $\lambda$ is chosen to give a trade-off between good fit and small values of the exponents. We can also regularize with a term of the form

$$\lambda \sum_{i=1}^{n} |a_i|,$$

which results in a problem that can be posed and solved as a quadratic program (QP). This regularization tends to find good monomial fits with many, or at least several, of the coefficients $a_i$ equal to zero. (See [17, §6.5].) In other words, we tend to fit the data using monomials that (where possible) depend on only a few of the variables.

Another useful modification of the basic least-squares method is to add constraints (such as lower and upper bounds) on the unknowns $\log c$ and $a_1, \ldots, a_n$. (The resulting problem can be solved using quadratic programming.) For more on fitting methods, see [17, Ch.6].

One drawback of using a simple fitting method applied to the logarithm of the data is that the error or residual is implicitly taken to be

$$|\log f(x^{(i)}) - \log f^{(i)}|.$$

While this residual clearly is small when $f(x^{(i)}) \approx f^{(i)}$, it may not be exactly the residual we care about. For example, we might care about the absolute error, $|f(x^{(i)}) - f^{(i)}|$, or the relative error,

$$r_i = \frac{|f(x^{(i)}) - f^{(i)}|}{f^{(i)}}.$$

The relative error is closely related to the logarithmic residual, but not exactly the same. For $f(x^{(i)}) > f^{(i)}$, we have

$$|\log f(x^{(i)}) - \log f^{(i)}| = \log(1 + r_i),$$

but for $f(x^{(i)}) < f^{(i)}$, we have

$$|\log f(x^{(i)}) - \log f^{(i)}| = \log\left(1 + \frac{|f(x^{(i)}) - f^{(i)}|}{f(x^{(i)})}\right) > \log(1 + r_i).$$

Thus, the logarithmic residual $|\log f(x^{(i)}) - \log f^{(i)}|$ puts more cost on an approximation that underestimates the data, compared to the relative error $r_i$.

If we do care about the relative error (as opposed to the logarithmic residual implicitly used in the regression method described above) we can choose the unknowns to minimize

$$\sum_{i=1}^{N} r_i^2 = \sum_{i=1}^{N} \left(\frac{f(x^{(i)}) - f^{(i)}}{f^{(i)}}\right)^2.$$

This is a *nonlinear least-squares problem*, which can be solved (usually) using methods such as the Gauss-Newton method [13, 102, 116]. Unlike linear least-squares methods, nonlinear least-squares methods are not guaranteed to converge to the global minimizer. But these methods work well in practice, if the initial guess is found using logarithmic regression (as described above).

One problem involving the relative error is readily solvable, without resorting to nonlinear least-squares methods. Consider the problem of choosing $c$ and $a_1, \ldots, a_n$ to minimize

$$\max_{i=1,\ldots,N} \frac{|f(x^{(i)}) - f^{(i)}|}{f^{(i)}},$$

the *maximum relative fitting error*. We can solve this problem exactly, as follows. We select a target value $t$ of maximum relative error, and will try to determine if it can be achieved by some choice of $c$ and $a_1, \ldots, a_n$. This is the same as determining whether the inequalities

$$\frac{|f(x^{(i)}) - f^{(i)}|}{f^{(i)}} \leq t, \qquad i = 1, \ldots, N$$

are feasible (by choice of $c$ and $a_1, \ldots, a_n$). (We can take $0 < t < 1$.) We can write these inequalities as

$$f^{(i)}(1-t) \leq f(x^{(i)}) \leq f^{(i)}(1+t), \qquad i = 1, \ldots, N.$$

Now we use the (by now, standard) method of taking logarithms, and using $y^{(i)} = \log x^{(i)}$, to express these as

$$\log(f^{(i)}(1-t)) \leq \log c + a_1 y_1 + \cdots + a_n y_n \leq \log(f^{(i)}(1+t)).$$

This is a set of linear inequalities in the unknowns $\log c$ and $a_1, \ldots, a_n$, and is easily solved using linear programming. We can use a bisection method to find the smallest value of $t$ for which the set of inequalities is feasible. (See [17].)

To illustrate these ideas, we consider the function

$$f(x) = \sqrt{1 - (x-1)^2}, \tag{41}$$

over the interval $[0.1, 1]$. The best local monomial approximation to $f$, at the point $x = 0.5$, is

$$\widehat{f}_{\rm loc}(x) = 1.0911 x^{0.3333}.$$

This monomial agrees with $f$ at $x = 0.5$, and gives a very good approximation near $x = 0.5$.

To give a fit over the whole interval $[0.1, 1]$, we first generate 100 data points $(x^{(i)}, f(x^{(i)}))$, with $x^{(i)}$ uniformly spaced over the interval $[0.1, 1]$. The monomial obtained by a least-squares fit of an affine function to the logarithmically transformed data $(\log x^{(i)}, \log f(x^{(i)}))$ is

$$\widehat{f}_{\rm ls}(x) = 1.0695 x^{0.3549}.$$

The monomial that minimizes the maximum relative error (for the given data points) is

$$\widehat{f}_{\rm maxrel}(x) = 1.0539 x^{0.3606}.$$

The three monomial approximations $\widehat{f}_{\rm loc}$, $\widehat{f}_{\rm ls}$, and $\widehat{f}_{\rm maxrel}$ are plotted in figure 11, along with the original function $f$. The associated fractional error distributions (for the 100 data points used) are shown in figure 12. As expected, the local approximation gives a very good fit near $x = 0.5$, with the end-points $x = 0.1$ and $x = 1$ giving relative errors around 16%. The least-squares logarithmic approximation gives better fit across the interval, with most errors under 3%, but maximum relative error exceeding 8%. The minimax relative error approximation has a very tight error distribution, with maximum relative error only 5.3%.

One useful extension of monomial fitting is to include a constant offset, *i.e.*, to fit the data $(x^{(i)}, f^{(i)})$ to a model of the form

$$f(x) = b + c x_1^{a_1} \cdots x_n^{a_n},$$

where $b \geq 0$ is another model parameter. Here $f$ is a posynomial, but with a very special form: a monomial plus a constant. To find such a fit, we simply use monomial modeling to

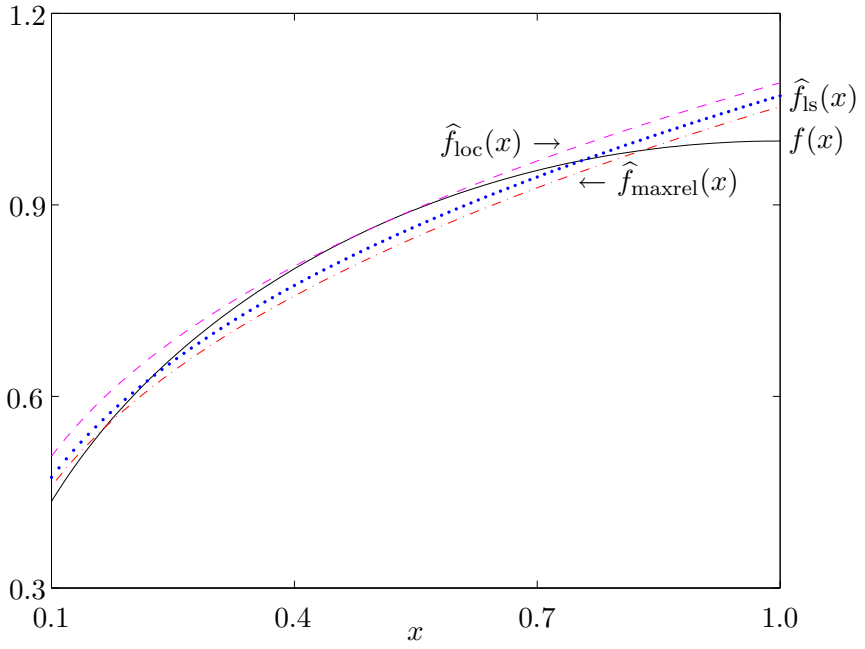


**Figure 11:** The solid curve is $f(x) = \sqrt{1-(x-1)^2}$. $\widehat{f}_{\rm loc}(x)$: best monomial approximation near $x = 0.5$. $\widehat{f}_{\rm ls}(x)$: monomial approximation obtained via least-squares fitting to logarithmically transformed data. $\widehat{f}_{\rm maxrel}(x)$: monomial approximation that minimizes maximum relative error.

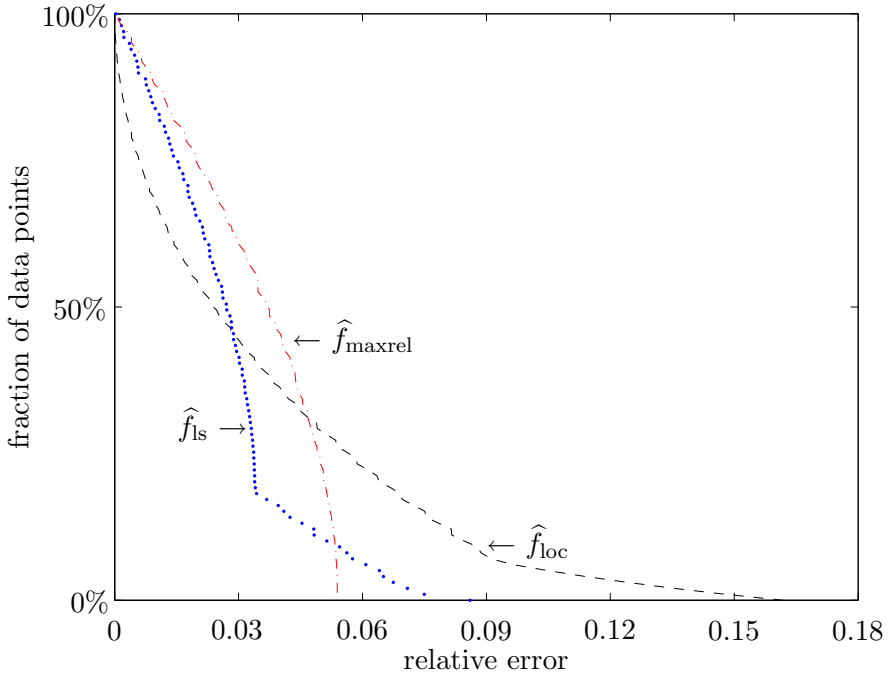


**Figure 12:** Relative error distributions for the three monomial approximations $\widehat{f}_{\rm loc}$, $\widehat{f}_{\rm ls}$, and $\widehat{f}_{\rm maxrel}$.

fit the modified data points $(x^{(i)}, f^{(i)} - b)$, for various values of $b$, and choose the one with the best fit.

The idea of using linear regression and fitting techniques on logarithmically transformed data can be applied to several problems beyond monomial fitting. As an example, suppose we have some positive vectors

$$x^{(1)}, \ldots, x^{(N)} \in \mathbf{R}^n,$$

and wish to find (approximate) monomial relations that the vectors satisfy, *i.e.*, we wish to find monomials $g_1, \ldots, g_k$ for which

$$g_j(x^{(i)}) \approx 1, \quad j = 1, \ldots, k, \quad i = 1, \ldots, N. \tag{42}$$

This can be done using the singular value decomposition (SVD), or canonical correlations, applied to logarithmically transformed data. We let $y^{(i)} = \log x^{(i)}$, and form the matrix

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(N)} \end{bmatrix} \in \mathbf{R}^{n \times N}.$$

The monomial relations (42) correspond to vectors $a_1, \ldots, a_k \in \mathbf{R}^{n+1}$ for which

$$[Y^T \ \mathbf{1}]a_j \approx 0, \quad j = 1, \ldots, k, \tag{43}$$

where $\mathbf{1}$ denotes a vector with all components one. To find such vectors (and also the number $k$), we compute the SVD of the matrix $[Y^T \ \mathbf{1}]$. The number of small singular values gives us the number of monomial approximate relations in the data, and the vectors $a_j$ are the left singular vectors associated with small singular values.

## 8.4 Max-monomial fitting

We consider the problem of fitting given data points $(x^{(i)}, f^{(i)})$, $i = 1, \ldots, N$, with a max-monomial function, *i.e.*, one of the form

$$f(x) = \max_{k=1,\ldots,K} f_k(x),$$

where $f_1, \ldots, f_K$ are monomials. When the number of terms $K$ is not specified, this problem can be solved exactly by the usual logarithmic transform, which reduces it to the problem of finding the best convex piecewise linear fit to some data. This problem can be posed as a large quadratic program and therefore solved; see [17, §6.5.5]. Unfortunately, this approach often leads to a number of terms that is very large, and therefore not practical. On the other hand, it can be very useful to know what the best max-monomial fit to the data is, when there is no limit on the number of terms.

We now concentrate on the case when the number of terms $K$ (or an upper bound on it) is fixed, and presumably not too large. Unlike the monomial fitting problem (which corresponds to the case $K = 1$), finding the best $K$-term max-monomial fit to the data is not a simple one. But there is a relatively simple method, based on monomial fitting and data point clustering, that works well in practice [103].

The algorithm is closely related to the $K$-means clustering algorithm [68], a well known method for clustering or partitioning a set of points into $K$ subsets, so as to minimize a sum-of-squares criterion. The general algorithm is outlined below:

*Simple max-monomial data fitting algorithm.*

given initial monomials $f_1(x), \ldots, f_K(x)$
**repeat**
  **for** $k = 1, \ldots, K$
  find all data points $x^{(j)}$ for which $f_k(x^{(j)}) = f(x^{(j)})$
    (*i.e.*, data points at which $f_k$ is the largest of the monomials)
  update $f_k$ by carrying out a monomial fit to these data points
**until** no further improvement occurs

This algorithm first clusters the data points into groups, for which the different monomials give the maximum value. Then, for each cluster of data points, the associated monomial is updated with a monomial fit to the data points. This is repeated until convergence. The algorithm is not guaranteed to converge, but with real data almost always does. The final max-monomial approximation can depend on the initial max-monomial, so the algorithm can be run from several initial max-monomials, and the best final fit taken. During the algorithm, it can happen that no points are assigned to one of the monomials. In this case we can just eliminate this monomial, and proceed with $K - 1$ terms. For this reason, the final max-monomial approximation can have fewer than $K$ terms.

One simple method for generating an initial max-monomial from which to start the algorithm is as follows. We first fit a single monomial to the data (using the methods described above), and then form $K$ versions of it by randomly perturbing the exponents. We express the monomial fit as

$$f_{\mathrm{mon}}(x) = c(x_1/\overline{x}_1)^{a_1} \cdots (x_n/\overline{x}_n)^{a_n},$$

where

$$\overline{x}_i = \left( \prod_{i=1}^{N} x_i^{(1)} \right)^{1/N}$$

is the geometric mean of the $i$th component of the data points (*i.e.*, the average value of the $i$th component on a logarithmic scale). We then form our monomials as

$$f_i(x) = c(x_1/\overline{x}_1)^{a_1+\delta_{i,1}} \cdots (x_n/\overline{x}_n)^{a_n+\delta_{i,n}},$$

where $\delta_{i,j}$ are independent random variables. This generates $K$ monomials, all near the original one, that are slightly different.

To illustrate this method for max-monomial fitting, we consider a numerical example. The data are generated as samples of the function

$$f(x) = e^{(\log x_1)^2 + (\log x_2)^2},$$

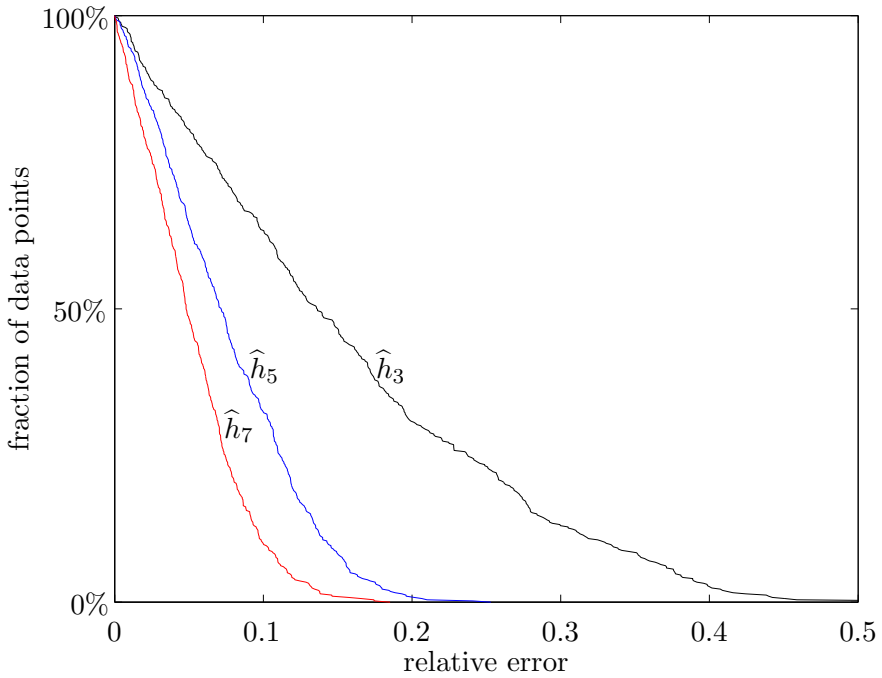


**Figure 13:** Relative error distributions for max-monomial approximations, with three terms ($\widehat{h}_3$), five terms ($\widehat{h}_5$), and seven terms ($\widehat{h}_7$).

with $N = 500$ data points drawn from a uniform distribution over the region $0.1 \leq x_i \leq 1$. The corresponding logarithmically transformed function is

$$F(y) = \log f(e^y) = y_1^2 + y_2^2,$$

which is a convex quadratic function of $y$. According to the theory, then, $f$ can be approximated arbitrarily well by a max-monomial with sufficiently many terms. (But we note that max-monomial fitting works when the data *do not* come from a function that can be arbitrarily well approximated by a max-monomial; in this case, the fitting error does not decrease to zero as the number of monomial terms increases.)

We use the algorithm above, with the simple least-squares logarithmic approximation method for monomial fitting, to find $K$-term max-monomial approximations, $\widehat{h}_K(x)$, for $K = 3$, 5, and 7. For this example, the final max-monomial found is (nearly) independent of the initial max-monomial, chosen randomly as described above. The associated residual error distributions for the three max-monomial approximations are shown in figure 13.

## 8.5 Posynomial fitting

General nonlinear least-squares methods can be used to find the coefficients and exponents in a posynomial with $K$ terms,

$$\widehat{f}(x) = \sum_{k=1}^{K} c_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_3^{a_{3k}},$$

that minimize, *e.g.*, the sum of squares of the relative errors,

$$\sum_{i=1}^{N} r_i^2 = \sum_{i=1}^{N} \left( \frac{f^{(i)} - f(x^{(i)})}{f^{(i)}} \right)^2$$

(along with the constraint $c_k \geq 0$). Methods for this type of problem, such as the Gauss-Newton method, or sequential quadratic programming, are local, like the max-monomial fitting method described above. This means that they cannot guarantee convergence to the true minimum; indeed, the final posynomial approximation obtained can depend on the initial one. But these methods often work well in practice, when initialized with reasonable starting posynomials. Because the final posynomial found depends on the starting posynomial, it is common to run the algorithm from several different starting points, and to take the best final posynomial found.

We can find a starting posynomial using a variation on the method used to initialize the max-monomial fitting algorithm. We first find a monomial fit of the data,

$$f_{\mathrm{mon}}(x) = c(x_1/\overline{x}_1)^{a_1} \cdots (x_n/\overline{x}_n)^{a_n}.$$

We express this as a sum of $K$ identical monomials,

$$f_{\mathrm{mon}}(x) = \sum_{k=1}^{K} (c/K)(x_1/\overline{x}_1)^{a_1} \cdots (x_n/\overline{x}_n)^{a_n}.$$

Now we perturb the exponents in the monomial terms to give a posynomial with $K$ different terms:

$$\widehat{f}(x) = \sum_{k=1}^{K} (c/K)(x_1/\overline{x}_1)^{a_1+\delta_{i,1}} \cdots (x_n/\overline{x}_n)^{a_n+\delta_{i,n}},$$

where $\delta_{i,j}$ are independent random variables. We use this posynomial as the starting point for a nonlinear least-squares method for fitting.

We illustrate posynomial fitting using the same data points as those used in the max-monomial fitting example given in §8.4. We used a Gauss-Newton method to find $K$-term posynomial approximations, $\widehat{h}_K(x)$, for $K = 3,\ 5,\ 7$, which (locally, at least) minimize the sum of the squares of the relative errors. In this case, the final posynomial obtained does depend substantially on the starting posynomial, so for each $K$, we solved 20 nonlinear fitting problems with different initial exponents $\delta_{i,n}$ drawn from normal distributions with zero mean and standard deviation $0.2|a_i|$, and take as the final posynomial approximation the best among the 20. The associated residual error distributions (for the 500 data points used) for the three posynomial approximations $\widehat{h}_K(x)$, $K = 3, 5, 7$ are shown in figure 14.

Comparing these error distributions to the ones obtained from max-monomial fitting, we see that posynomial fitting gives a much better fit for the same number of monomial terms. This is often the case when the original data or function is smooth. But the opposite can also occur: for some data, a max-monomial fit can be much better than a posynomial fit with the same number of terms.

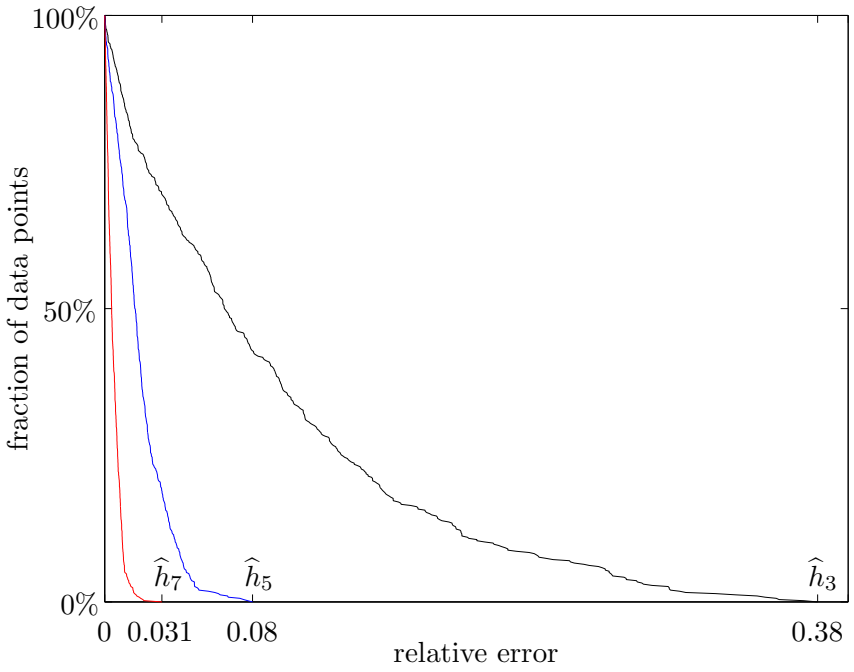


**Figure 14:** Relative error distributions for the posynomial approximations obtained via nonlinear least-squares fitting. $\widehat{h}_3$: 3-term posynomial fitting. $\widehat{h}_5$: 5-term posynomial fitting. $\widehat{h}_7$: 7-term posynomial fitting.

We mention one more method for posynomial (and generalized posynomial) fitting. Suppose that we choose a set of $r$ basis functions $\phi_1, \ldots, \phi_r$, which are themselves generalized posynomials. These could be monomials, for example, whose exponents are chosen on the basis of some prior knowledge of how the variables affect the function value. Once we have fixed the basis functions, we can find the coefficients $c_k$ for the approximation

$$\hat{f}(x) = c_1\phi_1(x) + \cdots + c_r\phi_r(x),$$

that minimizes the sum of squares of relative errors, subject to the constraint $c_k \geq 0$, as a quadratic programming problem. (This we can solve exactly; we do not have to worry about convergence to a local minimum as in max-monomial or general posynomial fitting.)

One interesting variant on this approach is to take a very large number of basis functions, more than we would consider using in the final approximation. We then add to the objective a regularization term of the form $\lambda(c_1 + \cdots + c_r)$, with $\lambda > 0$. This tends to find solutions with many of the $c_k$ equal to zero. Thus, the quadratic program effectively chooses a small subset of the basis functions. (See [17, §6.5].)

# 9 Extensions

In this section we describe some extensions of GP.

## 9.1 Signomial programming

A *signomial* is a function with the same form as a posynomial (*i.e.*, (2)), where the coefficients $c_j$ are allowed to be negative. A *signomial program* (SGP) is a generalization of a geometric program, which has the form of a GP, but the objective and constraint functions can be signomials:

$$\begin{array}{lll} \text{minimize} & f_0(x) & \\ \text{subject to} & f_i(x) \leq 1, & i = 1, \ldots, m, \\ & g_i(x) = 1, & i = 1, \ldots, p, \end{array} \tag{44}$$

where $f_i$ and $g_i$ are signomial functions. (As in GP, there is an implicit constraint that the variables are positive, *i.e.*, $x_i > 0$.) Unfortunately some authors use the term 'generalized geometric program' to refer to a signomial program.

From a computational point of view, there is a huge difference between a GP (or a GGP) and an SGP. While the globally optimal solution of a GP can always be found efficiently, only a *locally optimal* solution of an SGP can be computed efficiently. (It's possible to compute the globally optimal solution of an SGP, but this can require prohibitive computation, even for relatively small problems.)

We describe one general method for finding a local solution of an optimization problem that has the same form as a GP, but the objective and inequality constraint functions are not posynomials, and the equality constraint functions are not monomials. (SGP is a special case of this.) At each step we have a current guess $x^{(k)}$. For the objective and each

constraint function, we find the best local monomial approximation near the current guess $x^{(k)}$, using the formula (40). We replace the original functions in the optimization problem with these monomial approximations, which results in a GP (in fact, a linear program after the logarithmic transformation). Of course, we cannot trust these approximations for points that are not close to the current guess $x^{(k)}$, so we add some *trust region constraints* to the problem, which limit how much the variables are allowed to differ from the current guess. We might, for example, form the problem

$$\begin{array}{ll} \text{minimize} & \hat{f}_0(x) \\ \text{subject to} & \hat{f}_i(x) \leq 1, \quad i = 1, \ldots, m, \\ & \hat{g}_i(x) = 1, \quad i = 1, \ldots, p, \\ & (1/1.1)x_i^{(k)} \leq x_i \leq 1.1x_i^{(k)}, \quad i = 1, \ldots, n, \end{array} \tag{45}$$

where $\hat{f}_i$ is the local monomial approximation of $f_i$ at $x^{(k)}$, and $\hat{g}_i$ is the local monomial approximation of $g_i$ at $x^{(k)}$. The problem (45) is a GP, and therefore can be solved efficiently. Its solution is taken as the next iterate $x^{(k+1)}$. The algorithm consists of repeating this step until convergence.

This method is local: It need not converge (but very often does), and can converge to a point that is not the global solution. But if the problem is not too far from a GP, and the starting guess $x^{(0)}$ is good, it can work well in practice.

There are many variations on this method. For example, we do not need to form a monomial approximation of any constraint that is already in GP form in the original problem; if $f_i$ is a posynomial in the original problem (44), then we can leave $f_i$ in the problem (45) used to generate the next iterate. This idea can be taken a step further, when the original problem is an SGP, by forming a monomial approximation only of the offending terms in each constraint, *i.e.*, those that have negative coefficients. A general signomial inequality constraint can be expressed as

$$f(x) = p(x) - q(x) \leq 1,$$

where $p$ and $q$ are both posynomials. (Here $p$ consists of the terms in the signomial $f$ with positive coefficients, and $-q$ consists of the terms in $f$ with negative coefficients.) We write the signomial inequality as

$$p(x) \leq 1 + q(x),$$

and then form a monomial approximation of $1 + q(x)$, say, $r(x)$, at the current point $x^{(k)}$, and substitute the constraint

$$p(x) \leq r(x)$$

(which is GP compatible) in the GP (45) used to form the next iterate.

## 9.2 Mixed-integer geometric programming

In a *mixed-integer GP*, we have a GP with the additional constraint that some of the variables lie in some discrete set, such as the integers:

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 1, \quad i = 1, \ldots, m, \\ & g_i(x) = 1, \quad i = 1, \ldots, p, \\ & x_i \in \mathbf{N}, \quad i = 1, \ldots, k, \end{array} \tag{46}$$

where $\mathbf{N}$ denotes the set of natural numbers, *i.e.*, positive integers.

Mixed-integer GPs are in general very hard to solve, and all methods for solving them make some compromise, compared to methods for GP.

- *Heuristic methods* attempt to find good approximate solutions quickly (say, taking about the same time a similar size GP would), but cannot guarantee that the solution found is optimal.
- *Global methods* always find the global solution, but can take an extremely long time to run.

The simplest heuristic method is to ignore the integer constraints in (46), and solve the resulting GP (which can always be done fast). This problem is called the *GP relaxation* of the MIGP (46), since we have relaxed the integer constraints. The optimal value of the relaxed GP is a lower bound on the optimal value of the MIGP, since when we relax we expand the feasible set (and therefore reduce the optimal value). To obtain an approximate solution of the MIGP, we simply round each of the variables $x_1, \ldots, x_k$ towards the nearest integer. Then, we fix the integer variables (*i.e.*, treat them as constants) and solve the resulting GP. This second step allows the continuous variables to move around a bit to make up for the rounding. Although this method can work well for some problems, it often works poorly. For example, it often yields a point that is infeasible. One way to deal with this is to first tighten all constraints by some amount, such as 5%, prior to relaxing. This gives the relaxed solution some margin, so the rounded approximate solution has a greater chance of being feasible.

A more sophisticated method is to carry out the rounding in a sequence of steps. First we solve the relaxed GP. Then for each discrete variable that is within, say, 0.1 of an integer, we round it, and fix its value. We then form a new MIGP, which consists of the original MIGP, but with the integer variables that were fixed substituted with their fixed values. We then repeat the process, until all integer variables have been fixed. The idea here is that we only round the integer variables in cases where the rounding does little harm (it is hoped).

The most common global methods are *branch and bound* algorithms [95, 112]. They are nonheuristic, in the sense that they maintain a provable upper and lower bound on the (globally) optimal objective value, and ultimately find the true global solution. Branch and bound algorithms can be (and often are) slow, however. In the worst case they require effort

that grows exponentially with problem size, but in some cases we are lucky, and the methods converge with much less effort.

A typical branch and bound methods works as follows. We solve the relaxed GP, which gives us a lower bound on the optimal value of the MIGP, and we also use some heuristic (such as rounding) to find a locally optimal approximate solution. If it is feasible, and its objective value is close enough to the lower bound, we can take it as a guaranteed nearly optimal solution, and quit. If this is not the case, we choose an integer variable $x_i$ and *branch*. To do this we choose some integer $e$, and observe that if $x_i^*$ is the optimal value of $x_i$ (for the MIGP) then either $x_i^* \leq e$ or $x_i^* \geq e+1$. We form two *children MIGPs*, from the original one, by adding these two constraints to the original problem. We then repeat on each of these children. Eventually, the constraints on the integer variables become equality constraints, at which point we have GPs, and the relaxation yields the exact global solution. The hope is that this happens within a reasonable number of steps. Branch and bound algorithms differ in the heuristics used to choose a variable to branch on, and on which children MIGPs to process at each step.

As an example of mixed integer geometric programming, we revisit the digital circuit gate scaling problem considered in §6.2, with an additional twist: we require that the scale factors must take integer values. The gate scaling problem is then the MIGP

$$\begin{array}{ll} \text{minimize} & D \\ \text{subject to} & P \leq P^{\max}, \qquad A \leq A^{\max}, \\ & x_i \in \mathbf{N}, \quad i = 1, \ldots, n. \end{array} \tag{47}$$

(The requirement that the gate scaling factors are integers is natural, since the individual devices in the gates can be obtained by replicating an integer number of the devices appearing in the minimum sized gate.)

Now we consider a numerical example, the small digital circuit shown in figure 4, with the same parameters as those used in the numerical instance in §6.2, and maximum area $A^{\max} = 100$. This problem is small enough that we can easily find the global optimum of the MIGP (47) using the branch and bound method. Figure 15 compares the optimal trade-off curve of the minimum delay versus maximum allowed power for the design with discrete scale factors with that for the design with continuous scale factors. Adding the integer constraints reduces the feasible set, and so can only increase the minimum delay obtained. The plot shows that some values of $P^{\max}$ (such as 40) the penalty paid for integer scale factors isn't large; for other values of $P^{\max}$ (such as 20) the penalty paid for integer scale factors is much larger. (In the general case, of course, the penalty can be much larger, or even make the problem infeasible.)

# 10 Notes and references

### Origins of geometric programming

Geometric programming was introduced by Duffin, Peterson, and Zener in their 1967 book [54]. Two other early books on geometric programming are by Wilde and Breightler (1967) [154],

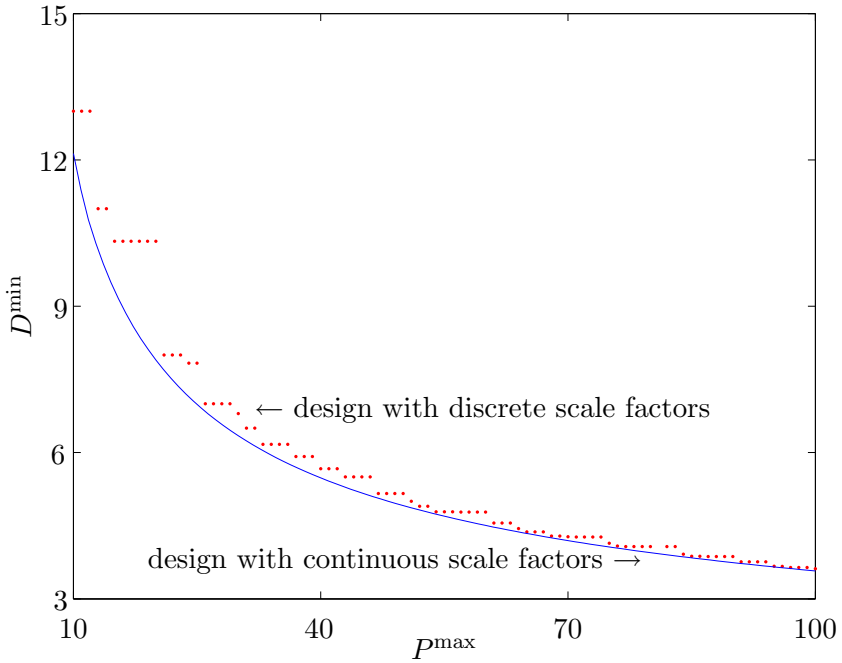


**Figure 15:** Optimal trade-off curves of minimum delay $D^{\min}$ versus maximum power $P^{\max}$, for the maximum area $A^{\max} = 100$, with and without the constraint that the gate scale factors are integers.

and Zener (1971) [161]. These early books contain much material, such as sensitivity analysis, monomial fitting, and many of the extensions described in this tutorial (fractional powers and exponentials of posynomials, and posynomial equality constraints).

Some of the ideas behind geometric programming can be traced back to earlier work. Although not discussed in this paper, GPs are very closely related via duality to problems involving entropy and cross entropy (see, *e.g.*, [17, §5.7]), which appeared earlier. For example, in the 1958 paper [48], Dantzig, Johnson, and White consider a chemical equilibrium problem in what is now recognized as geometric programming dual form. Dual form GPs were called linear-logarithmic programs in the 1960s (see, *e.g.*, [38]). A history of the development of geometric programming can be found in Peterson's survey [123].

Sensitivity analysis for geometric programming is discussed in [50, 52, 93, 94, 128]. Recent work on *robust geometric programming* (a variation on GP in which the coefficients in the posynomials are uncertain) can be found in [58, 75]. Some recent work on mixed-integer geometric programming and signomial programming can be found in the papers [4, 33, 34, 105, 140, 148, 160, 157].

### Algorithms and software

In the early work by Duffin, Peterson, Zener, and Wilde, GPs were solved analytically via the dual problem (which is possible only for very small problems). Numerical methods for

(computer) solution of GPs were developed in the 1970s; see, *e.g.*, Duffin [53], Avriel et al. [7], and Rajpogal and Bricker [126]. These methods were based on solving a sequence of linear programs.

The first interior-point method for GP was described by Nesterov and Nemirovsky in 1994, who also established polynomial time complexity of the method [114]. More recent work on interior-point methods for GP include Kortanek, Xu, and Ye [91] and Andersen and Ye [5]. For a general introduction to interior-point methods for convex optimization (and GP in particular) see part III of Boyd and Vandenberghe [17]. A high quality implementation of a primal-dual interior-point method for GP is available in the MOSEK [113] software package and the TOMLAB software package [57]. Both packages include a simple interface to the MathWorks' MATLAB.

The power of GGP can be utilized conveniently only with a parser that automatically converts a GGP to a GP. A few simple computer programs that automate the transformation from a text description of a GP or GGP into a form suitable for a solver are already available. An example is YALMIP [100], which has a simple interface that recognizes some GGPs, and automatically forms and solves the resulting GPs. While the current generation of GP and GGP software is for experts only, we can expect in the next few years rapid progress in GP and GGP solution speed, as well as user-friendly interfaces for specifying (and debugging) GPs and GGPs.

### Applications

Since its inception, GP has been closely associated with applications in engineering. Early applications of geometric programming include nonlinear network flow problems, optimal control, optimal location problems, and chemical equilibrium problems. These are described in the books [6, 10, 54, 115, 153, 154, 161] and survey papers [56, 122].

Geometric programming has been used in a variety of problems in digital circuit design; see [16, 136] for more on GP-based digital circuit sizing and optimization. In 1985 Fishburn and Dunlop [60] used geometric programming for transistor and wire sizing in digital integrated circuit design. Since then many digital circuit design problems have been formulated as GPs or related problems:

- gate sizing [37, 119, 46, 85, 107, 106, 121, 142, 134, 137, 138]
- wire sizing [41, 42, 43, 44, 23, 22, 64, 86, 97, 99, 135]
- buffering and wire sizing [35, 36]
- simultaneous gate and wire sizing [21, 79]
- sizing and placement [25, 101]
- routing [15]
- design with multiple supply voltages [92]
- yield maximization [89, 120]
- parasitic reduction [124]
- clock skew optimization [139]

These are all based on gate delay models that are compatible with geometric programming; see [85, 131, 146, 130, 1] for more on such models. Some of the papers listed above develop custom methods for solving the resulting GPs, instead of using general purpose interior-point methods (see, *e.g.*, [35, 37, 77, 159]). See [16, 136] for more on GP-based digital circuit sizing.

In 1997, Hershenson, Boyd, and Lee applied GP to analog integrated circuit design [72]. GP is applied to RF circuit design in [73, 111, 110], operational amplifier design in [49, 70, 71, 104, 149], mixed signal IC design in [40, 67, 69], switching regulator design in [96], and configurable analog/RF circuit design in [98, 156]. GP compatible models for analog circuit design are discussed in [47, 88, 87]. Geometric programming has also been used for floorplanning in the context of circuit design [109, 129, 159].

Applications of geometric programming in other fields include:

- chemical engineering [39, 132, 133, 152]
- environment quality control [143, 65]
- resource allocation in communication and network systems [14, 28, 29, 30, 31, 55, 65, 81, 83, 117]
- information theory [11, 12, 84, 90, 29]
- probability and statistics [108, 151, 59, 63, 76, 18, 8, 145]
- structural design [3, 24, 19, 51, 66, 150]
- computer system architecture design [147]
- inventory control [2, 82]
- production system optimization [32]
- mechanical engineering [78, 144]
- transportation engineering [155]
- management science [45]
- computational finance [125]
- systems and control theory [20, 158]

Some recent books [118, 74, 127, 61, 17] include engineering applications of geometric programming. We refer the reader to the tutorial paper [27] for an extensive discussion on applications of geometric programming in communications and information theory.

## Acknowledgments

This material is supported in part by the National Science Foundation under grants #0423905 and (through October 2005) #0140700, by the Air Force Office of Scientific Research under grant #F49620-01-1-0365, by MARCO Focus center for Circuit & System Solutions contract #2003-CT-888, and by MIT DARPA contract #N00014-05-1-0700. The authors thank Dennis Bricker, Mung Chiang, Lungying Fong, Kan-Lin Hsiung, Siddharth Joshi, Kwangmoo Koh, Johan Löfberg, Tony Luk, Almir Mutapcic, Sachin Sapatnekar, and Tamás Terlaky for helpful comments and suggestions.

# References


[1] A. Abou-Seido, B. Nowak, and C. Chu. Fitted Elmore delay: A simple and accurate interconnect delay model. *IEEE Transactions on VLSI Systems*, 12(7):691–696, 2004.

[2] M. Abuo-El-Ata, H. Fergany, and M. El-Wakeel. Probabilistic multi-item inventory model with varying order cost under two restrictions: A geometric programming approach. *International Journal of Production Economics*, 83(3):223–231, 2003.

[3] H. Adeli and O. Kamal. Efficient optimization of space trusses. *Computers and Structures*, 24:501–511, 1986.

[4] J. Alejandre, A. Allueva, and J. Gonzalez. A general alternative procedure for solving negative degree of difficulty problems in geometric programming. *Computational Optimization and Applications*, 27(1):83–93, 2004.

[5] E. Andersen and Y. Ye. A computational study of the homogeneous algorithm for large-scale convex optimization. *Computational Optimization and Applications*, 10(3):243–269, 1998.

[6] M. Avriel, editor. *Advances in Geometric Programming*, volume 21 of *Mathematical Concept and Methods in Science and Engineering*. Plenum Press, New York, 1980.

[7] M. Avriel, R. Dembo, and U. Passy. Solution of generalized geometric programs. *International Journal for Numerical Methods in Engineering*, 9:149–168, 1975.

[8] H. El Barmi and R. Dykstra. Restricted multinomial maximum likelihood estimation based upon Fenchel duality. *Statistics and Probability Letters*, 21(2):121–130, 1994.

[9] M. Bazaraa, C. Shetty, and H. Sherali. *Non-linear Programming: Theory and Algorithms*. John Wiley and Sons, Inc., 1993.

[10] C. Beightler and D. Phillips. *Applied Geometric Programming*. Wiley, New York, 1976.

[11] A. Ben-Tal and M. Teboulle. Rate distortion theory with generalized information measures via convex programming duality. *IEEE Transactions on Information Theory*, 32:630–641, 1986.

[12] A. Ben-Tal, M. Teboulle, and A. Charnes. The role of duality in optimization problems involving entropy functionals. *Journal of Optimization Theory and Applications*, 58:209–223, 1988.

[13] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 1999.

[14] H. Boche and S. Stańczak. Optimal QoS tradeoff and power control in CDMA systems. In *Proceedings of the 23rd IEEE INFOCOM*, pages 477–486, March 2004.

[15] M. Borah, R. Owens, and M. Irwin. A fast algorithm for minimizing the Elmore delay to identified critical sinks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(7):753–759, 1997.

[16] S. Boyd, S.-J. Kim, D. Patil, and M. Horowitz. Digital circuit optimization via geometric programming. To appear in *Operations Research*. Available from `www.stanford.edu/~boyd/gp_digital_ckt.html`.

[17] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[18] D. Bricker, K. Kortanek, and L. Xui. Maximum likelihood estimates with order restrictions on probabilities and odds ratios: A geometric programming approach. *Journal of Applied Mathematics and Decision Sciences*, 1(1):53–65, 1997.

[19] A. Chan and E. Turlea. An approximate method for structural optimisation. *Computers and Structures*, 8:357–363, 1978.

[20] D. Chandra, V. Singh, and H. Kar. Improved Routh-Padé approximants: A computer-aided approach. *IEEE Transactions on Automatic Control*, 49(2):292–296, February 2004.

[21] C.-P. Chen, C. Chu, and D. Wong. Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(7):1014–1025, 1999.

[22] C.-P. Chen and D. Wong. Greedy wire-sizing is linear time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(4):398–405, 1999.

[23] T.-C. Chen, S.-R. Pan, and Y.-W. Chang. Timing modeling and optimization under the transmission line model. *IEEE Transactions on Very Large Scale Integration Systems*, 12(1):28–41, January 2004.

[24] T.-Y. Chen. Structural optimization using single-term posynomial geometric programming. *Computers and Structures*, 45:911–918, 1992.

[25] W. Chen, C.-T. Hseih, and M. Pedram. Simultaneous gate sizing and placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(2):206–214, February 2000.

[26] H. Cheng, S.-C. Fang, and J. Lavery. Univariate cubic $L_1$ splines—A geometric programming approach. *Mathematical Methods of Operations Research*, 56:197–229, 2002.

[27] M. Chiang. Geometric programming for communication systems, 2004. To be submitted to Trends and Foundations in Communication and Information Theory.

[28] M. Chiang. Balancing transport and physical layers in wireless multihop networks: Jointly optimal congestion control and power control. To appear in *IEEE Journal on Selected Areas in Communications*, 2005. Available from `www.princeton.edu/~chiangm/publications.html`.

[29] M. Chiang and S. Boyd. Geometric programming duals of channel capacity and rate distortion. *IEEE Transactions on Information Theorey*, 50(2):245–258, February 2004.

[30] M. Chiang, B. Chan, and S. Boyd. Convex optimization of output link scheduling and active queue management in QoS constrained packet sitches. In *Proceedings of the IEEE International Conference on Communications*, pages 2126–2130, New York, USA, April 2002.

[31] M. Chiang, A. Sutivong, and S. Boyd. Efficient nonlinear optimization of queuing systems. In *Proceedings of the IEEE GLOBECOM*, Taipei, ROC, November 2002.

[32] J. Choi and D. Bricker. Effectiveness of a geometric programming algorithm for optimization of machining economics models. *Computers and Operations Research*, 23(10):957–961, 1996.

[33] J.-C. Choi and D. Bricker. Geometric programming with several discrete variables: Algorithms employing generalized benders. *Engineering Optimization*, 3:201–212, 1995.

[34] J.-C. Choi and D. Bricker. A heuristic procedure for rounding posynomial geometric programming solutions to discrete value. *Computers and Industrial Engineering*, 30(4):623–629, 1996.

[35] C. Chu and D.Wong. Closed form solutions to simultaneous buffer insertion/sizing and wire sizing. *ACM Transactions on Design Automation of Electronic Systems*, 6(3):343–371, 2001.

[36] C. Chu and D. Wong. An efficient and optimal algorithm for simultaneous buffer and wire sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(9):1297–1304, 1999.

[37] C. Chu and D. Wong. VLSI circuit performance optimization by geometric programming. *Annals of Operations Research*, 105:37–60, 2001.

[38] R. Clasen. The linear logarithmic programming problem. Rand Corp. Memo. RM-37-7-PR, June 1963.

[39] R. Clasen. The solution of the chemical equilibrium programming problem with generalized benders decomposition. *Operations Research*, 32(1):70–79, 1984.

[40] D. Colleran, C. Portmann, A. Hassibi, C. Crusius, S. Mohan, S. Boyd, T. Lee, and M. Hershenson. Optimization of phase-locked loop circuits via geometric programming. In *Proceedings of the Custom Integrated Circuits Conference (CICC)*, pages 326–328, September 2003.

[41] J. Cong and L. He. Optimal wire sizing for interconnects with multiple sources. *ACM Transactions on Design Automation of Electronic Systems*, 1(4):478–511, 1996.

[42] J. Cong and C.-K. Koh. Simultaneous driver and wire sizing for performance and power optimization. *IEEE Transactions on Very Large Scale Integration Systems*, 2(4):408–423, 1994.

[43] J. Cong and K.-S. Leung. Optimal wiresizing under Elmore delay model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(3):321–336, 1995.

[44] J. Cong and Z. Pan. Wire width planning for interconnect performance optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(3):319–329, 2002.

[45] M. Corstjens and P. Doyle. Channel optimization in complex marketing systems. *Management Science*, 25(10):1014–1025, 1979.

[46] O. Coudert, R. Haddad, and S. Manne. New algorithms for gate sizing: A comparative study. In *Proceedings of 33rd IEEE/ACM Design Automation Conference*, pages 734–739, 1996.

[47] W. Daems, G. Gielen, and W. Sansen. Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(5):517–534, May 2003.

[48] G. Dantzig, S. Johnson, and W. White. A linear programming extension approach to the chemical equilibrium problem. *Management Science*, 5(1):38–43, 1958.

[49] J. Dawson, S. Boyd, M. Hershenson, and T. Lee. Optimal allocation of local feedback in multistage amplifiers via geometric programming. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48(1):1–11, January 2001.

[50] R. Dembo. Sensitivity analysis in geometric programming. *Journal of Optimization Theory and Applications*, 37:1–21, 1982.

[51] B. Dhillon and C. Kuo. Optimum design of composite hybrid plate girders. *Journal of Structural Engineering*, 117(7):2088–2098, 1991.

[52] J. Dinkel, M. Kochenberger, and S. Wong. Sensitivity analysis procedures for geometric programs: Computational aspects. *ACM Transactions on Mathematical Software*, 4(1):1–14, 1978.

[53] R. Duffin. Linearizing geometric programs. *SIAM Review*, 12(2):668–675, 1970.

[54] R. Duffin, E. Peterson, and C. Zener. *Geometric Programming–Theory and Application*. Wiley, New York, 1967.

[55] A. Dutta and D. Rama. An optimization model of communications satellite planning. *IEEE Transactions on Communications*, 40(9):1463–1473, 1992.

[56] J. Ecker. Geometric programming: Methods, computations and applications. *SIAM Review*, 22(3):338–362, 1980.

[57] M. Edvall and F. Hellman. *User's Guide for TOMLAP/GP*, 2005. Available from `http://tomlab.biz/docs/TOMLAB_GP.pdf`.

[58] A. Federowicz and J. Rajgopal. Robustness of posynomial geometric programming optima. *Mathematical Programming*, 85(2):421–431, 1999.

[59] P. Feigin and U. Passy. The geometric programming dual to the extinction probability problem in simple branching processes. *The Annals of Probability*, 9(3):498–503, 1981.

[60] J. Fishburn and A. Dunlop. TILOS: A posynomial programming approach to transistor sizing. In *IEEE International Conference on Computer-Aided Design: ICCAD-85. Digest of Technical Papers*, pages 326–328. IEEE Computer Society Press, 1985.

[61] C. Floudas. *Deterministic Global Optimization: Theory, Algorithms and Applications*. Kluwer Academic Publishers, 1999.

[62] G. Foschini and Z. Miljanic. A simple distributed autonomous power control algorithm and its convergence. *IEEE Transactions on Vehicular Technology*, 42(4):641–646, 1993.

[63] C.-D. Fuh and I. Hu. Asymptotically efficient strategies for a stochastic scheduling problem with order constraints. *The Annals of Statistics*, 28(6):1670–1695, 2000.

[64] Y. Gao and D. Wong. Optimal shape function for a bidirectional wire under Elmore delay model. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(7):994–999, 1999.

[65] H. Greenberg. Mathematical programming models for environmental quality control. *Operations Research*, 43(4):578–622, 1995.

[66] P. Hajela. Geometric programming strategies for large-scale structural synthesis. *AIAA Journal*, 24(7):1173–1178, 1986.

[67] A. Hassibi and M. Hershenson. Automated optimap design of switched-capacitor filters. In *Design, Autmation and Test in Europe Conference and Exhibition*, page 1111, Paris, France, March 2002.

[68] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[69] M. Hershenson. Design of pipeline analog-to-digital converters via geometric programming. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 317–324, San Jose, CA, November 2002.

[70] M. Hershenson. Analog design space exploration: Efficient description of the design space of analog circuits. In *Proceedings of the 40th Design automation Conference*, pages 970–973, Anaheim, CA, June 2003.

[71] M. Hershenson, S. Boyd, and T. Lee. GPCAD: A tool for CMOS op-amp synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 296–303, November 1998.

[72] M. Hershenson, S. Boyd, and T. Lee. Optimal design of a CMOS op-amp via geometric programming. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(1):1–21, January 2001.

[73] M. Hershenson, A. Hajimiri, S. Mohan, S. Boyd, and T. Lee. Design and optimization of LC oscillators. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 65–69, 1999.

[74] K. Hitomi. *Manufacturing Systems Engineering: A Unified Approach to Manufacturing Technology, Production Management, and Industrial Economics*. CRC Press, second edition, 1996.

[75] K.-L. Hsiung, S.-J. Kim, and S. Boyd. Robust geometric programming via piecewise linear approximation. Revised for publication in *Mathematical Programming*, April 2004. Available from `www.stanford.edu/~boyd/rgp.html`.

[76] I. Hu and C. Wei. Irreversible adaptive allocation rules. *The Annals of Statistics*, 17(2):801–823, 1989.

[77] Y. Ismail, E. Friedman, and J. Neves. Equivalent Elmore delay for RLC trees. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(7):83–97, 2000.

[78] N. Jha. A discrete data base multiple objective optimization of milling operation through geometric programming. *Journal of Engineering for Industry*, 112:368–374, 1990.

[79] I. Jiang, Y. Chang, and J. Jou. Crosstalk-driven interconnect optimization by simultaneous gate and wire sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(9):999–1010, 2000.

[80] S. Joshi, S. Boyd, and R. Dutton. Optimal doping profiles via geometric programming, 2005. Manuscript. Available from `www.stanford.edu/~boyd/opt_doping_profile.html`.

[81] D. Julian, M. Chiang, D. O'Neill, and S. Boyd. QoS and fairness constrained convex optimization of resource allocation for wireless cellular and ad hoc networks. In *Proceedings of the 21st IEEE INFOCOM*, pages 477–486, June 2002.

[82] H. Jung and C. Klein. Optimal inventory policies under decreasing cost functions via geometric programming. *European Journal of Operational Research*, 132(3):628–642, 2001.

[83] S. Kandukuri and S. Boyd. Optimal power control in interference-limited fading wireless channels with outage-probability specifications. *IEEE Transactions on Wireless Communications*, 1(1):46–55, January 2002.

[84] J. Karlof and Y. Chang. Optimal permutation codes for the gaussian channel. *IEEE Transactions on Information Theory*, 43(1):356–358, January 1997.

[85] K. Kasamsetty, M. Ketkar, and S. Sapatnekar. A new class of convex functions for delay modeling and its application to the transistor sizing problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(7):779–788, July 2000.

[86] R. Kay and L. Pileggi. EWA: Efficient wiring-sizing algorithm for signal nets and clock nets. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(1):40–49, January 1998.

[87] T. Kiely and G. Gielen. Performance modeling of analog integrated circuits using least-squares support vector machines. In *Design, Autmation and Test in Europe Conference and Exhibition*, pages 16–20, Paris, France, February 2004.

[88] J. Kim, J. Lee, L. Vandenberghe, and K. Yang. Techniques for improving the accuracy of geometric-programming based analog circuit design optimization. To be presented at IEEE International Conference on Computer-Aided Design: ICCAD-2004, September 2004.

[89] S.-J. Kim, S. Boyd, S. Yun, D. Patil, and M. Horowitz. A heuristic for optimizing stochastic activity networks with applications to statistical digital circuit sizing. Submitted, February 2005. Available from `www.stanford.edu/~boyd/heur_san_opt.html`.

[90] E. Klafszky, J. Mayer, and T. Terlaky. A geometric programming approach to the channel capacity problem. *Engineering Mathematics*, 19:115–130, 1992.

[91] K. Kortanek, X. Xu, and Y. Ye. An infeasible interior-point algorithm for solving primal and dual geometric progams. *Mathematical Programming*, 76(1):155–181, January 1996.

[92] R. Krishnamurthy and L. Carley. Exploring the design space of mixed swing quadrail for low-power digital circuits. *IEEE Transactions on Very Large Scale Integration Systems*, 5(4):389–400, 1997.

[93] J. Kyparsis. Sensitivity analysis in posynomial geometric programming. *Journal of Optimization Theory and Applications*, 57:85–121, 1988.

[94] J. Kyparsis. Sensitivity analysis in geometric programming: Theory and computations. *Annals of Operations Research*, 27:39–64, 1990.

[95] E. Lawler and D. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14:699–719, 1966.

[96] J. Lee, G. Hatcher, L. Vandenbergh, and K. Yang. Evaluation of fully-integrated switching regulators for CMOS process technologies. In *Proceedings of the International Symposium on System-on-Chip*, pages 155–158, Tampere, Finland, November 2003.

[97] Y.-M. Lee, C. Chen, and D. Wong. Optimal wire-sizing function under the Elmore delay model with bounded wire sizes. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 49(11):1671–1677, 2002.

[98] X. Li, P. Gopalakrishnan, Y. Xu, and T. Pileggi. Robust analog/RF circuit design with projection-based posynomial modeling. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 855–862, September 2004.

[99] T. Lin and L. Pileggi. RC(L) interconnect sizing with second order considerations via posynomial programming. In *Proceedings of the ACM/SIGDA International Symposium on Physical Design*, pages 16–21, Sonoma, CA, June 2001.

[100] J. Löfberg. *YALMIP. Yet Another LMI Parser. Version 2.4*, 2003. Available from `http://control.ee.ethz.ch/~joloef/yalmip.php`.

[101] J. Lou, W. Chen, and M. Pedram. Concurrent logic restructuring and placement for timing closure. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 31–35, November 1999.

[102] D. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, second edition, 1984.

[103] A. Magnani and S. Boyd. Convex piecewise linear fitting, 2005. Manuscript. Available from `www.stanford.edu/~boyd/cvx_pwl_fitting.html`.

[104] P. Mandal and V. Visvanathan. CMOS op-amp sizing using a geometric programming formulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(1):22–38, January 2001.

[105] C. Maranas and C. Floudas. Global optimization in generalized geometric programming. *Computers and Chemical Engineering*, 21(4):351–369, 1997.

[106] D. Marković, V. Stojanović, B. Nikolić, M. Horowitz, and R. Brodersen. Methods for true energy-performance optimization. *IEEE Journal of Solid-State Circuits*, 39(8):1282–1293, 2004.

[107] M. Matson and L. Glasser. Macromodeling and optimization of digital MOS VLSI circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 5(4):659–678, 1986.

[108] M. Mazumdar and T. Jefferson. Maximum likelihood estimates for multinomial probabilities via geometric programming. *Biometrika*, 70(1):257–261, April 1983.

[109] T.-S. Moh, T.-S. Chang, and S. Hakimi. Globally optimal floorplanning for a layout problem. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 43(29):713–720, September 1996.

[110] S. Mohan, M. Hershenson, S. Boyd, and T. Lee. Simple accurate expressions for planar spiral inductances. *IEEE Journal of Solid-State Circuit*, 34(10):1419–1424, October 1999.

[111] S. Mohan, M. Hershenson, S. Boyd, and T. Lee. Bandwidth extension in CMOS with optimized on-chip inductors. *IEEE Journal of Solid-State Circuits*, 35(3):346–355, March 2000.

[112] R. Moore. Global optimization to prescribed accuracy. *Computers and Mathematics with Applications*, 21(6/7):25–39, 1991.

[113] MOSEK ApS. *The MOSEK Optimization Tools Version 2.5. User's Manual and Reference*, 2002. Available from `www.mosek.com`.

[114] Y. Nesterov and A. Nemirovsky. *Interior-Point Polynomial Methods in Convex Programming*, volume 13 of *Studies in Applied Mathematics*. SIAM, Philadelphia, PA, 1994.

[115] P. Nijhamp. *Planning of Industrial Complexes by Means of Geometric Programming*. Rotterdam Univ. Press, Rotterdam Netherlands, 1972.

[116] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, New York, 1999.

[117] D. Palomar, J. Cioffi, and M. Lagunas. Joint Tx-Rx beamforming design for multicarrier MIMO channels: A unified framework for convex optimization. *IEEE Transactions on Signal Processing*, 51(9):2381–2401, September 2003.

[118] P. Papalambros and D. Wilde. *Principles of Optimal Design*. Cambridge University Press, 1988.

[119] U. Passy. Theory and algorithm of local refinement based optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(4):406–420, 1998.

[120] D. Patil, Y. Yun, S.-J. Kim, S. Boyd, and M. Horowitz. A new method for robust design of digital circuits. In *Proceedings of the International Symposium on Quality Electronic Design (ISQED)*, pages 676–681, March 2005.

[121] M. Pattanaik, S. Banerjee, and B. Bahinipati. GP based transistor sizing for optimal design of nanoscale CMOS inverter. In *IEEE Conference on Nanotechnology*, pages 524–527, August 2003.

[122] E. Peterson. Geometric programming. *SIAM Review*, 18(1):1–51, 1976.

[123] E. Peterson. Investigation of path-following algorithms for signomial geometric programming problems. *Annals of Operations Research*, 105:15–19, 2001.

[124] Z. Qin and C.-K. Cheng. Realizable parasitic reduction using generalized Y-$\Delta$ transformation. In *Proceedings of the 40th IEEE/ACM Design Automation Conference*, pages 220–225, June 2003.

[125] J. Rajasekera and M. Yamada. Estimating the firm value distribution function by entropy optimization and geometric programming. *Annals of Operations Research*, 105:61–75, 2001.

[126] J. Rajpogal and D. Bricker. Posynomial geometric programming as a special case of semi-infinite linear programming. *Journal of Optimization Theory and Applications*, 66:444–475, September 1990.

[127] S. Rao. *Engineering Optimization: Theory and Practice*. Wiley-Intersicence, third edition, 1996.

[128] M. Rijckaert and E. Walraven. Geometric programming: Estimation of Lagrange multipliers. *Operations Research*, 33(1):85–93, 1985.

[129] E. Rosenberg. Optimization module sizing in VLSI floorplanning by nonlinear programming. *ZOR-Methods and Models of Operations Research*, 33(2):131–143, 1989.

[130] J. Rubenstein, P. Penfield, and M. Horowitz. Signal delay in RC tree networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2(3):202–211, July 1983.

[131] T. Sakurai. Approximation of wiring delay in MOSFET LSI. *IEEE Journal of Solid-State Circuits*, 18(4):418–426, 1988.

[132] H. Salomone and O. Iribarren. Posynomial modeling of batch plants: A procedure to include process decision variables. *Computers and Chemical Engineering*, 16:173–184, 1993.

[133] H. Salomone, J. Montagna, and O. Iribarren. Dynamic simulations in the design of batch processes. *Computers and Chemical Engineering*, 18:191–204, 1994.

[134] P. Sancheti and S. Sapatnekar. Optimal design of macrocells for low power and high speed. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(9):1160–1166, September 1996.

[135] S. Sapatnekar. Wire sizing as a convex optimization problem: Exploring the area-delay tradeoff. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15:1001–1011, August 1996.

[136] S. Sapatnekar. *Timing*. Kluwer Academic Publishers, 2004.

[137] S. Sapatnekar and W. Chuang. Power-delay optimization in gate sizing. *ACM Transactions on Design Automation of Electronic Systems*, 5(1):98–114, 2000.

[138] S. Sapatnekar, V. Rao, P. Vaidya, and S. Kang. An exact solution to the transistor sizing problem for CMOS circuits using convex optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(11):1621–1634, November 1993.

[139] H. Sathyamurthy, S. Sapatnekar, and J. Fishburn. Speeding up pipelined circuits through a combination of gate sizing and clock skew optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(2):173–182, 1998.

[140] H. Sherali. Global optimization of nonconvex polynomial programming problems having rational exponents. *Journal of Global Optimization*, 12(3):267–283, 1998.

[141] N. Sherwani. *Algorithms for VLSI Design Automation*. Kluwer Academic Publishers, third edition, 1999.

[142] J. Shyu, A. Sangiovanni-Vincetelli, J. Fishburn, and A. Dunlop. Optimization-based transistor sizing. *IIEEE Journal of Solid-State Circuits*, 23(2):400–409, April 1988.

[143] Y. Smeers and D. Tyteca. A geometric programming model for the optimal design of wastewater treatment plants. *Operations Research*, 32(2):314–342, 1984.

[144] A. Sonmez, A. Baykasoglu, T. Dereli, and I. Flz. Dynamic optimization of multipass milling operations via geometric programming. *International Journal of Machine Tools and Manufacture*, 39(2):297–320, 1999.

[145] R. Stark and M. Machida. Chance design costs-A distributional limit. In *Proceedings of the Second International Symposium on Uncertainty Modeling and Analysis*, pages 269–270, June 1993.

[146] I. Sutherland, B. Sproull, and D. Harris. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[147] K. Trivedi and T. Sigmon. Optimal design of linear storage hierarchies. *Journal of the ACM*, 28(2):270–288, 1981.

[148] J.-F. Tsai, H.-L. Li, and N.-Z. Hu. Global optimization for signomial discrete programming problems in engineering design. *Engineering Optimization*, 34(6):613–622, 2002.

[149] J. Vanderhaegen and R. Brodersen. Automated design of operational transconductance amplifiers using reversed geometric programming. In *Proceedings of the 41st IEEE/ACM Design Automation Conference*, pages 133–138, San Diego, CA, June 2004.

[150] G. Vanderplaats. *Numerical Optimization Techniques for Engineering Design*. McGraw-Hill, 1984.

[151] Y. Vardi. Empirical distributions in selection bias models. *The Annals of Statistics*, 13(1):178–203, 1985.

[152] T. Wall, D. Greening, and R. Woolsey. Solving complex chemical equilibria using a geometric-programming based technique. *Operations Research*, 34(3):345–355, 1986.

[153] D. Wilde. *Globally Optimal Design*. John Wiley and Sons Inc., New York, 1978.

[154] D. Wilde and C. Beightler. *Foundations of Optimization*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.

[155] D. Wong. Maximum likelihood, entropy maximization, and the geometric programming approaches to the calibration of trip distribution models. *Transportation Research Part B: Methodological*, 15(5):329–343, 1981.

[156] Y. Xu, L. Pileggi, and S. Boyd. ORACLE: Optimization with recourse of analog circuits including layout extraction. In *Proceedings of the 41st IEEE/ACM Design Automation Conference*, pages 151–154, San Diego, CA, June 2004.

[157] H.-H. Yang and D. Bricker. Investigation of path-following algorithms for signomial geometric programming problems. *European Journal of Operational Research*, 103(1):230–241, 1997.

[158] H. Yazarel and G. Pappas. Geometric programming relaxations for linear system reachability. In *Proceedings of the 2004 American Control Conference*, pages 553–559, Boston, MA, USA, June 2004.

[159] F. Young, C. Chu, W. Luk, and Y. Wong. Handling soft modules in general nonslicing floorplan using Lagrangian relaxation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(5):687–629, May 2001.

[160] K. Yun and C. Xi. Second-order method of generalized geometric programming for spatial frame optimization. *Computer Methods in Applied Mechanics and Engineering*, 141:117–123, 1997.

[161] C. Zener. *Engineering Design by Geometric Programming*. Wiley, New York, 1971.