Optim Eng
DOI 10.1007/s11081-016-9332-3

Data fitting with geometric-programming-compatible softmax functions

Warren Hoburg[1] · **Philippe Kirschen**[1] · **Pieter Abbeel**[2]

Received: 11 March 2015 / Revised: 8 December 2015 / Accepted: 13 July 2016

Abstract Motivated by practical applications in engineering, this article considers the problem of approximating a set of data with a function that is compatible with geometric programming (GP). Starting with well-established methods for fitting max-affine functions, it is shown that improved fits can be obtained using an extended function class based on the softmax of a set of affine functions. The softmax is generalized in two steps, with the most expressive function class using an implicit representation that allows fitting algorithms to locally tune softness. Each of the proposed function classes is directly compatible with the posynomial constraint forms in GP. Max-monomial fitting and posynomial fitting are shown to correspond to fitting special cases of the proposed implicit softmax function class. The fitting problem is formulated as a nonlinear least squares regression, solved locally using a Levenberg–Marquardt algorithm. Practical implementation considerations are discussed. The article concludes with numerical examples from aerospace engineering and electrical engineering.

Keywords Convex optimization · Convex regression · Geometric programming

✉ Warren Hoburg
whoburg@mit.edu

Philippe Kirschen
kirschen@mit.edu

Pieter Abbeel
pabbeel@cs.berkeley.edu

[1] Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA

[2] Computer Science Department, University of California, Berkeley, CA 94720, USA

Published online: 04 August 2016

1 Introduction

This article presents methods for fitting functions that are compatible with geometric programming (GP) to data. To provide context, it begins with an overview of GP, followed by a motivation for fitting GP-compatible functions.

1.1 Overview of geometric programming

First introduced in 1967 by Duffin et al. (1967), a geometric program (GP)[1] is a specific type of constrained, nonlinear optimization problem that becomes convex after a logarithmic change of variables. Despite significant work on early applications in structural design, network flow, and optimal control (Beightler and Phillips 1976; Wilde 1978), reliable and efficient numerical methods for solving GPs were not available until the 1990s (Nesterov and Nemirovsky 1994). Geometric programming has recently undergone a resurgence as researchers have discovered promising applications in statistics (Boyd and Vandenberghe 2004), digital circuit design (Boyd et al. 2005), antenna optimization (Babakhani et al. 2010), communication systems (Chiang 2005), aircraft design (Hoburg and Abbeel 2014), and other engineering fields.

Modern GP solvers employ primal-dual interior point methods (Mehrotra 1992) and are extremely fast. A typical sparse GP with tens of thousands of decision variables and one million constraints can be solved on a desktop computer in minutes (Boyd et al. 2007). Furthermore, these solvers do not require an initial guess and converge to a *global* optimum, whenever a feasible solution exists.

These performance benefits are possible because geometric programs represent a restricted subset of nonlinear optimization problems. In particular, the objective and constraints can only be composed of *monomial* and *posynomial* functions.

Monomials In GP, a monomial is a function $h(\mathbf{u}) : \mathbb{R}^n_{++} \rightarrow \mathbb{R}_{++}$ of the form

$$h(\mathbf{u}) = c \prod_{j=1}^{n} u_j^{a_j}, \tag{1}$$

where $c \in \mathbb{R}_{++}$, $\mathbf{u} = (u_1, \ldots, u_n) \in \mathbb{R}^n_{++}$, and $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n$. Since the exponents a_j in (1) may be negative and non-integer, expressions like $\frac{u_1 u_2^{0.7} \sqrt{u_3}}{u_4}$ are monomials.

Posynomials Like monomials, posynomials are functions $g(\mathbf{u}) : \mathbb{R}^n_{++} \rightarrow \mathbb{R}_{++}$. A posynomial has the form

$$g(\mathbf{u}) = \sum_{k=1}^{K} c_k \prod_{j=1}^{n} u_j^{a_{jk}}, \tag{2}$$

where $c_k \in \mathbb{R}_{++}$, and $\mathbf{a}_k = (a_{1k}, \ldots, a_{nk}) \in \mathbb{R}^n$. Thus, a posynomial is simply a sum of monomial terms, and all monomials are also posynomials (with just one term).

[1] The "GP" acronym is overloaded, referring both to geometric programs—the class of optimization problem discussed in this article—and geometric programming—the practice of using such programs to model and solve optimization problems.

The expression $0.23 + u_1^2 + 0.1u_1u_2^{-0.8}$ is an example of a posynomial in $\mathbf{u} = (u_1, u_2)$, whereas $2u_1 - u_2^{1.5}$ is not a posynomial because negative leading coefficients c_k are not allowed.

1.1.1 Geometric programs in standard form

GPs are often written in the standard form

$$\begin{aligned} &\text{minimize} && g_0(\mathbf{u}) \\ &\text{subject to} && g_i(\mathbf{u}) \leq 1, i = 1, \ldots, n_i, \\ &&& h_i(\mathbf{u}) = 1, i = 1, \ldots, n_e, \end{aligned} \tag{3}$$

where each g_i is a posynomial, and each h_i is a monomial.

The expression *GP-compatible* is used in this article to refer to constraints that can be written in the form of either the monomial or posynomial constraints of Problem (3).

1.1.2 Geometric programs in convex form

GPs are of special interest because they become *convex* optimization problems under a logarithmic change of variables. In particular, the transformation

$$\mathbf{x} = \log \mathbf{u} \tag{4}$$

converts monomial equality constraints $h(\mathbf{u}) = 1$ to the form

$$\log h(e^{\mathbf{x}}) = \mathbf{a}^{\mathrm{T}}\mathbf{x} + b = 0, \tag{5}$$

where $b = \log c$, and posynomial inequality constraints $g(\mathbf{u}) \leq 1$ to the form

$$\log g(e^{\mathbf{x}}) = \log \sum_{k=1}^{K} \exp(\mathbf{a}_k^{\mathrm{T}}\mathbf{x} + b_k) \leq 0, \tag{6}$$

where $b_k = \log c_k$. By inspection, (5) is affine in $\mathbf{x}$, and it is straightforward to show that (6) is convex[2] in $\mathbf{x}$, since log-sum-exp functions are known to be convex, and convexity is preserved under affine transformations (Boyd and Vandenberghe 2004). Thus, (3) is transformed into a convex optimization problem by the variable change (4).

Posynomials are one example of a *log-log-convex* function class. A function $f(\mathbf{u}) : \mathbb{R}^n_{++} \to \mathbb{R}_{++}$ is log-log-convex if and only if $\log f$ is a convex function of $\log \mathbf{u}$, i.e. if and only if $\mathbf{x} \mapsto \log f(e^{\mathbf{x}})$ is convex in $\mathbf{x}$.

1.2 Motivation for fitting GP-compatible functions to data

Not all log–log–convex relationships can be expressed directly in monomial or posynomial form. Examples include analytic expressions that are log-log-convex

[2] A function $f : \mathcal{X} \to \mathbb{R}$ is convex if its domain $\mathcal{X}$ is a convex set and the property $f(\theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2) \leq \theta f(\mathbf{x}_1) + (1-\theta)f(\mathbf{x}_2)$ holds for all $0 \leq \theta \leq 1$ and for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$.

but not posynomial, such as $\exp(u)$ or $-\log(1-u)$, and empirical data sets or "black box" simulations that are well-approximated by log-log-convex functions. Although one could conceivably develop solution techniques for a more general class of log-log-convex functions, the existence of tested, reliable, and numerically stable commercial and open-source GP solvers creates an incentive to express or approximate relationships in the standard GP form (3).

1.3 Overview of GP-compatible fitting

In *GP-compatible fitting*, one is interested in modeling a set of data points

$$\mathcal{D} = \{(\mathbf{u}_i, w_i) \,|\, i = 1...m\},$$

with monomial or posynomial constraints on a GP. The multivariate data set $\mathcal{D}$ is the input to the fitting procedure and typically captures a relationship between some variables in the GP. There are two restrictions on the input data: 1) the data must be strictly positive because of the log transformation, and 2) the data should be well-approximated by a log-log-convex function.

The fitting procedure can be summarized in three steps:

1. Apply the log transformation $(\mathbf{x}_i, y_i) = (\log \mathbf{u}_i, \log w_i)$.
2. Fit a convex function f to the transformed data, such that $y_i \approx f(\mathbf{x}_i)$.
3. Relax the equality $y = f(\mathbf{x})$ into an inequality $y \geq f(\mathbf{x})$, corresponding to a GP-compatible posynomial constraint $g(\mathbf{u}, w) \leq 1$.

In the last step, the set $y = f(\mathbf{x})$ is replaced with the epigraph of f. This relaxed epigraph formulation is traditional in optimization (Boyd and Vandenberghe 2004), and indeed necessary to preserve the convex structure of the problem, but it requires care on the part of the GP modeler. In particular, if it is desired that equality between y and $f(\mathbf{x})$ be achieved at the optimum, then this places restrictions on the remainder of the GP. One sufficient, but not necessary, condition to guarantee that $g(\mathbf{u}, w) = 1$ holds at a global optimum is given in Boyd et al. (2007):

- w (the dependent variable in the fitting) does not appear in any monomial equality constraints.
- The GP objective and inequality constraint functions (other than the fitted posynomial constraint) are all monotone increasing in w.
- The fitted posynomial $g(\mathbf{u}, w)$ is monotone decreasing in w (this condition is satisfied for all function classes presented herein).

The output of the fitting procedure is a single multivariate GP-compatible constraint[3] $g(\mathbf{u}, w) \leq 1$. One can examine the corresponding fitted function, $\tilde{g}(\mathbf{u}) = \mathbf{u} \mapsto \{w \,|\, g(\mathbf{u}, w) = 1\}$. An important subtlety is that $\tilde{g}(\mathbf{u})$ need not be a posynomial in order to be GP-compatible. A central theme of this work is that more

[3] It is also possible to achieve GP-compatible fits via multiple functions, resulting in a set of constraints.

expressive function classes are possible by only requiring that the epigraph of $\tilde{g}(\mathbf{u})$ correspond to a posynomial constraint $g(\mathbf{u}, w) \leq 1$.

2 Current methods for fitting convex functions to data

To obtain GP-compatible function fits, we first address the problem of fitting functions that are convex but not necessarily GP-compatible. Consider the problem of fitting a multivariate function $f(\mathbf{x})$ to a set of m data points,

$$\{(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \mathbb{R}, i = 1 \ldots m\}.$$

With the restriction that f be convex, and choosing the sum of squared errors as a loss function, the fitting problem is

$$\underset{f \in \mathcal{F}}{\text{minimize}} \sum_{i=1}^{m} (f(\mathbf{x}_i) - y_i)^2, \tag{7}$$

where f is the function being fitted, and $\mathcal{F}$ is a set of convex functions. Clearly, the sum of squared errors cannot be small if the data is not well-approximated by any convex function.

As with any regression problem, the specific choice of function class $\mathcal{F}$ can dramatically influence the quality of the resulting fit with respect to the chosen loss function. One common choice is $\mathcal{F}_{\text{MA}}^{K}$, the set of *max-affine* functions with K terms:

$$f_{\text{MA}}(\mathbf{x}) = \max_{k=1 \ldots K} \left[b_k + \mathbf{a}_k^{\text{T}} \mathbf{x} \right], \tag{8}$$

where $b_k \in \mathbb{R}$ and $\mathbf{a}_k \in \mathbb{R}^n$ are parameters. The total number of parameters is $n_p = K(n+1)$. It is well known that the supremum over a set of affine functions is a convex function, thus (8) is convex in $\mathbf{x}$.

This choice is motivated by the fact that any closed convex function can be expressed as the point-wise supremum over a (generally infinite) set of affine functions (Boyd and Vandenberghe 2004). That is, $\mathcal{F}_{\text{MA}}^{\infty}$ can approximate *any* convex function to arbitrary precision. Max-affine functions are also appealing from a practical perspective: they transform to monomial constraint sets (compatible with GP) under the variable change (4).

Methods for fitting max-affine functions are well established. Magnani and Boyd (2009) proposed a least-squares partition algorithm that works well in practice. They also discussed a more general bi-affine function parameterization. Kim et al. (2010) used a similar method to fit max-monomial functions for circuit design. Hannah and Dunson (2011) fit max-affine functions using a statistically consistent adaptive partitioning approach that refines accuracy for increasing K by generating candidate partition splits and selecting splits greedily. They also describe an ensemble version of their method that avoids instability in max-monomial fitting (Hannah and Dunson 2012).

Although max-affine functions are a natural and popular choice, a large number of affine terms K may be necessary to achieve a desired level of accuracy. This

article argues for choosing f from a more general class of convex functions. Previous work on convex regression has improved fitting algorithms and guaranteed important statistical properties like consistency, often for the special case of max-affine functions. In contrast, the emphasis here is on the mathematical structure of the functions being fitted – the choice of $\mathcal{F}$.

3 Proposed convex function classes

This section introduces two convex function classes: the *softmax-affine* class, $\mathcal{F}^K_{\text{SMA}}$, and the *implicit softmax-affine* class, $\mathcal{F}^K_{\text{ISMA}}$. These can be thought of as successive generalizations of $\mathcal{F}^K_{\text{MA}}$, a relationship discussed later in this section.

3.1 Softmax-affine functions

The proposed softmax-affine class $\mathcal{F}^K_{\text{SMA}}$ consists of functions of the form

$$f_{\text{SMA}}(\mathbf{x}) = \frac{1}{\alpha}\log\sum_{k=1}^{K}\exp\big(\alpha(b_k + \mathbf{a}_k^{\text{T}}\mathbf{x})\big). \tag{9}$$

The addition of the scalar parameter $\alpha \in \mathbb{R}_{++}$ brings the total number of parameters to $n_p = K(n+1)+1$. Since log-sum-exp functions are convex and convexity is preserved under positive scaling and affine transformations (Boyd and Vandenberghe 2004), SMA functions are guaranteed to be convex in $\mathbf{x}$ for any $\alpha > 0$.

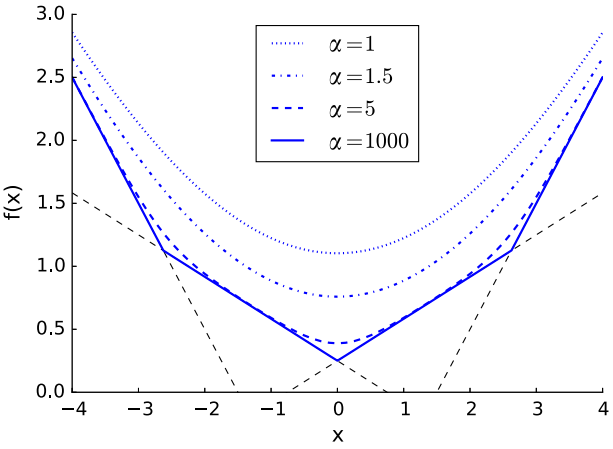


Fig. 1 Influence of α on softmax-affine functions. Each of the softmax-affine functions plotted above shares the same $K = 4$ affine terms (the *straight dashed lines*), but has a different α. The *solid curve* corresponds to a max-affine function; the *dotted curve* corresponds to a softmax-affine function with $\alpha = 1$, and one can interpolate smoothly among these by varying α. While this figure illustrates the situation in $\mathbb{R}^1$, the limiting cases extend to arbitrary dimensions

Limiting behavior As depicted in Fig. 1, one can view α as a smoothing parameter that controls the sharpness of the softmax over the K affine planes. In the limit of infinite sharpness,

$$\lim_{\alpha\to+\infty}\frac{1}{\alpha}\log\sum_{k=1}^{K}\exp\big(\alpha(b_k+\mathbf{a}_k^{\mathrm{T}}\mathbf{x})\big)=\max_{k=1\ldots K}\big[b_k+\mathbf{a}_k^{\mathrm{T}}\mathbf{x}\big]. \tag{10}$$

According to Eq. 10, softmax-affine functions become max-affine functions in the limit $\alpha\to+\infty$. This limiting behavior implies that for a given data set (X, Y) and number of affine terms K, there always exist SMA functions with at least as small a residual as the best possible max-affine fit.

3.2 Implicit softmax-affine functions

The proposed implicit softmax-affine class, $\mathcal{F}_{\mathrm{ISMA}}^{K}$, expresses the relationship between $\mathbf{x}$ and y via the zero of an *implicit* function

$$\tilde{f}_{\mathrm{ISMA}}(\mathbf{x},y)=\log\sum_{k=1}^{K}\exp\big(\alpha_k(b_k+\mathbf{a}_k^{\mathrm{T}}\mathbf{x}-y)\big), \tag{11}$$

where each $\alpha_k\in\mathbb{R}_{++}$.

Proposition 1 For all $\mathbf{x}$, there exists a unique y such that $\tilde{f}_{\mathrm{ISMA}}(\mathbf{x},y)=0$.

Proof For all $\mathbf{x}$, $y\mapsto\tilde{f}_{\mathrm{ISMA}}(\mathbf{x},y)$ is a continuous monotone strictly decreasing function of y, since increasing y decreases every term in the K-term sum. Moreover, since every $b_k+\mathbf{a}_k^{\mathrm{T}}\mathbf{x}$ is finite and every $\alpha_k>0$, there exists some γ^- such that $\tilde{f}_{\mathrm{ISMA}}(\mathbf{x},\gamma^-)>0$, and some γ^+ such that $\tilde{f}_{\mathrm{ISMA}}(\mathbf{x},\gamma^+)<0$. Thus by the intermediate value theorem the function has a unique zero crossing between γ^- and γ^+. □

Based on Proposition 1, f_{ISMA} is defined such that for all $\mathbf{x}$,

$$\tilde{f}_{\mathrm{ISMA}}(\mathbf{x},f_{\mathrm{ISMA}}(\mathbf{x}))=0. \tag{12}$$

That is, the predicted value $\hat{y}=f_{\mathrm{ISMA}}(\mathbf{x})$ is the unique value $\hat{y}$ such that $\tilde{f}_{\mathrm{ISMA}}(\mathbf{x},\hat{y})=0$.

Proposition 2 The function $\mathbf{x}\mapsto f_{\mathrm{ISMA}}(\mathbf{x})$ is convex.

Proof Consider any two points $(\mathbf{x}_1,y_1)$ and $(\mathbf{x}_2,y_2)$ that both solve $\tilde{f}_{\mathrm{ISMA}}(\mathbf{x},y)=0$. By convexity of the log-sum-exp function and preservation of convexity under affine mappings, $\tilde{f}_{\mathrm{ISMA}}(\theta\mathbf{x}_1+(1-\theta)\mathbf{x}_2,\theta y_1+(1-\theta)y_2)\le 0\ \ \forall\theta\in[0,1]$. But by Proposition 1, for some $\hat{y}$, $\tilde{f}_{\mathrm{ISMA}}(\theta\mathbf{x}_1+(1-\theta)\mathbf{x}_2,\hat{y})=0$. Since $\tilde{f}_{\mathrm{ISMA}}$ is monotone decreasing in y, $\hat{y}\le\theta y_1+(1-\theta)y_2$. Thus the function value $\hat{y}$ at any weighted combination of $\mathbf{x}$ points is less than or equal to a weighted combination of the y values – exactly the definition of convexity. □

ISMA functions have individual softness parameters $\alpha_k > 0$ for each of the K affine terms, bringing the total number of parameters to $n_p = K(n+2)$. Figure 2 illustrates the influence of these softness parameters, and Fig. 3 shows how they can improve fitting performance.

Setting all the α_k parameters to the same value α, one recovers the softmax-affine function class.

$$\tilde{f}_{\text{ISMA}} = \log \sum_{k=1}^{K} \exp\big(\alpha(b_k + \mathbf{a}_k^{\text{T}}\mathbf{x} - y)\big) \tag{13}$$

$$0 = \log \sum_{k=1}^{K} \exp(-\alpha y) \exp\big(\alpha(b_k + \mathbf{a}_k^{\text{T}}\mathbf{x})\big) \tag{14}$$

$$y = \frac{1}{\alpha} \log \sum_{k=1}^{K} \exp\big(\alpha(b_k + \mathbf{a}_k^{\text{T}}\mathbf{x})\big) = f_{\text{SMA}} \tag{15}$$

This implies that the implicit softmax-affine class subsumes the softmax-affine class, and therefore also the max-affine class. As a result, for a given fitting problem and choice of K, there always exists some setting of the α_k parameters for which the ISMA class performs as well as or better than the best fit from each of the other function classes.

Evaluating ISMA functions No explicit formula is available for evaluating ISMA functions. Instead, Algorithm 1 gives a Newton–Raphson (Ypma 1995) method for solving $\tilde{f}_{\text{ISMA}}(\mathbf{x}, y) = 0$.

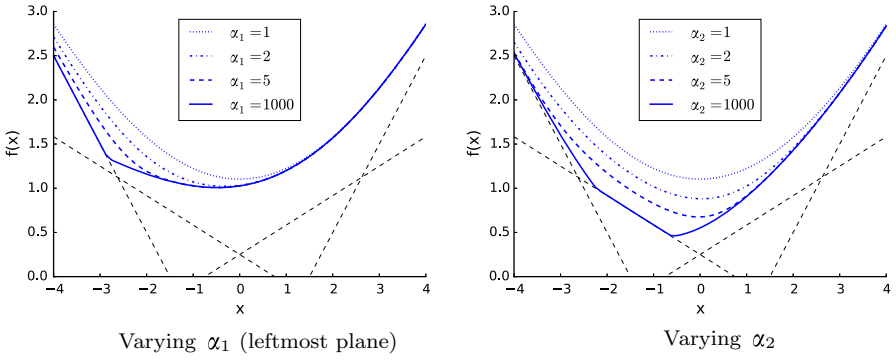


Fig. 2 Influence of individual softness parameters α_k on ISMA functions. Each of the functions above shares the same $K = 4$ affine terms (the *thin dashed lines*). Setting all of the softness parameters α_k to 1 results in the *top curve* in each figure. Varying just one of the four softness parameters then gives intuition about its effect. This figure illustrates the situation in $\mathbb{R}^1$, but the qualitative behavior extends to arbitrary dimensions

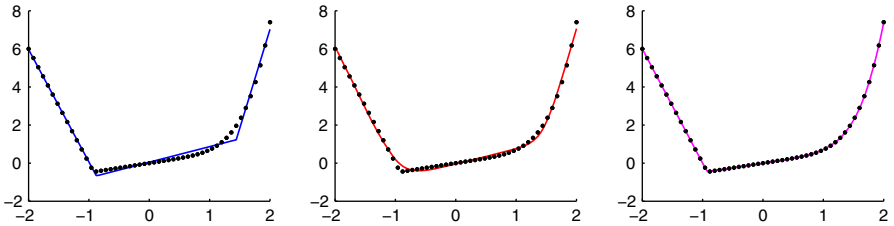


Fig. 3 This toy fitting problem illustrates how ISMA functions can significantly outperform SMA (and therefore also MA) functions. All fits used $K = 3$ affine terms. Here the data are samples of the convex function $y = \max\left(-6x - 6, \frac{1}{2}x, \frac{1}{5}x^5 + \frac{1}{2}x\right)$, which has a sharp kink near $x = -0.8$, but gradual curvature elsewhere. The best fit is an ISMA function, which can conform to the data by adjusting softness locally

Algorithm 1 Evaluate $y = f_{\text{ISMA}}(\mathbf{x})$

$z_k \leftarrow b_k + \mathbf{a}_k^{\mathrm{T}}\mathbf{x}, \;\; k = 1 \ldots K$
$s \leftarrow \max_k z_k$
$t \leftarrow 0$
repeat
  $f \leftarrow \log \sum_{k=1}^{K} \exp\left(\alpha_k(z_k - s - t)\right)$
  $J \leftarrow -\frac{\sum_k \alpha_k \exp(\alpha_k(z_k - s - t))}{\sum_k \exp(\alpha_k(z_k - s - t))}$
  $t \leftarrow t - f/J$
until $|f| <$ `machine_precision`
return $y = s + t$

The breakdown $y = s + t$ avoids numerical issues associated with computing ratios and logarithms of large exponentials. In practice, Algorithm 1 reliably converges to machine precision in approximately 5-10 Newton iterations.

4 Transformation to GP-compatible functions

Currently, a common approach to fit GP-compatible functions is to approximate y as a max-affine function of $\mathbf{x}$, and convert the resulting affine functions back to equivalent monomial constraints on the GP. Since GPs also admit posynomial constraints, another approach is to directly fit w as a posynomial function of $\mathbf{u}$. This approach has been used by a number of authors (Daems et al. 2003; Li et al. 2004; Oliveros et al. 2008). Typically mixed results are reported, with max-monomial functions performing better on data with sharp kinks, and posynomial functions performing better on smoother data (Boyd et al. 2007; Magnani and Boyd 2009). As discussed shortly, one benefit of SMA and ISMA functions is the unification of these two approaches.

4.1 GP interpretation of proposed convex function classes

Each of the convex function classes in this article has a parallel interpretation as a GP-compatible function. This makes SMA and ISMA functions natural choices for GP-compatible fitting.

4.1.1 Max-affine functions as max-monomial functions

Recall that under the GP log transformation, monomial functions become affine. Thus, a max-affine approximation for y,

$$\hat{y} = f_{\text{MA}}(\mathbf{x}), \tag{16}$$

corresponds exactly to a max-monomial approximation for w,

$$\hat{w} = \max_{k=1\ldots K}\left[e^{b_k}\prod_{i=1}^{n} u_i^{a_{ik}}\right], \tag{17}$$

whose epigraph is a GP-compatible set of K monomial inequality constraints on $(\mathbf{u}, w)$,

$$\frac{e^{b_k}}{w}\prod_{i=1}^{n} u_i^{a_{ik}} \leq 1, k = 1\ldots K. \tag{18}$$

Because they can be fit using established max-affine methods, max-monomial functions are currently a common choice for GP modeling (Hannah and Dunson 2012; Kim et al. 2010).

4.1.2 SMA functions as posynomial functions

Consider the GP log transformation applied to (13). An approximation,

$$\hat{y} = f_{\text{SMA}}(\mathbf{x}), \tag{19}$$

corresponds to a posynomial approximation[4] for w^α,

$$w^\alpha = \sum_{k=1}^{K} e^{\alpha b_k}\prod_{i=1}^{n} u_i^{\alpha a_{ik}}, \tag{20}$$

whose epigraph is a posynomial constraint,

$$\frac{1}{w^\alpha}\sum_{k=1}^{K} e^{\alpha b_k}\prod_{i=1}^{n} u_i^{\alpha a_{ik}} \leq 1. \tag{21}$$

[4] Equation (20) can also be interpreted as a generalized posynomial (Boyd et al. 2007; Kasamsetty et al. 2000) constraint on w, $w = \left(\sum_{k=1}^{K} e^{\alpha b_k}\prod_{i=1}^{n} u_i^{\alpha a_{ik}}\right)^{\frac{1}{\alpha}}$.

For the special case $\alpha = 1$, SMA functions reduce to posynomials in convex form [see Eq. (6)]. That is, the constraint $\hat{y} \geq f_{\text{SMA}}(\mathbf{x}; \alpha = 1)$ corresponds to a posynomial constraint on $(\mathbf{u}, w)$,

$$\frac{1}{w} \sum_{k=1}^{K} e^{b_k} \prod_{i=1}^{n} u_i^{a_{ik}} \leq 1. \tag{22}$$

While the distinction between (21) and (22) appears subtle, it has important implications for GP modeling. In particular, in existing literature, posynomial fitting often refers to fitting functions of the form $\hat{w} = g(\mathbf{u})$, which corresponds to searching for softmax-affine functions, but with the restriction $\alpha = 1$. Better fits can be obtained by searching for functions of a more expressive posynomial form $\hat{w}^{\alpha} = g(\mathbf{u})$.

The softmax-affine function class includes posynomial functions as a special case $(\alpha = 1)$, and max-affine functions as a limiting case $(\alpha \to +\infty)$. Softmax-affine functions thereby unify max-monomial and posynomial fitting, eliminating the need to search over multiple function classes.

4.1.3 ISMA functions as implicit posynomial functions

Consider the GP log transformation applied to (9). An ISMA fit,

$$\hat{y} = f_{\text{ISMA}}(\mathbf{x}), \tag{23}$$

corresponds exactly to an implicit posynomial,

$$\sum_{k=1}^{K} \frac{e^{\alpha_k b_k}}{w^{\alpha_k}} \prod_{i=1}^{n} u_i^{\alpha_k a_{ik}} = 1, \tag{24}$$

whose epigraph corresponds to a posynomial constraint on $(\mathbf{u}, w)$,

$$\sum_{k=1}^{K} \frac{e^{\alpha_k b_k}}{w^{\alpha_k}} \prod_{i=1}^{n} u_i^{\alpha_k a_{ik}} \leq 1. \tag{25}$$

By allowing a different α_k in each monomial term, ISMA functions are even more expressive than SMA or MA functions.

5 Fitting model parameters

This section addresses the problem of fitting the proposed function classes to data. Data fitting and nonlinear regression are well-established fields supported by an extensive literature. Nevertheless, there is value in describing a practical end-to-end fitting procedure for the specific cases of SMA and ISMA functions.

5.1 Fitting approach overview

Given m data points $(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \mathbb{R}$, a least squares fitting objective is

$$\underset{\beta}{\text{minimize}} \sum_{i=1}^{m} (f(\mathbf{x}_i; \beta) - y_i)^2, \tag{26}$$

where f is an instance of one of the convex function classes already presented, and $\beta \in \mathbb{R}^{n_p}$ is a vector that contains the parameters a, b, and α for the chosen function class. In general, (26) is a nonlinear least squares problem, which is non-convex and may have multiple local optima. The quantity $\mathbf{r}(\beta) = f(X; \beta) - Y$ is the vector of residuals, and the goal is to find a set of parameters β for which $\mathbf{r}(\beta)^{\mathrm{T}}\mathbf{r}(\beta)$ is minimized.

Consider some initial set of parameters β_0, and corresponding residual $\mathbf{r}(\beta_0)$. For a small change in parameters δ, the new residual is approximately

$$\mathbf{r}(\beta_0 + \delta) \approx \mathbf{r}(\beta_0) + \mathbf{J}\delta, \tag{27}$$

where $\mathbf{J} \in \mathbb{R}^{m \times n_p}$ is $\partial f / \partial \beta$, the Jacobian of f. This suggests a first-order approximation of (26), rewritten in terms of the parameter update δ:

$$\begin{aligned} \underset{\delta}{\text{minimize}}\ & \delta^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{J}\delta + 2\delta^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{r} + \mathbf{r}^{\mathrm{T}}\mathbf{r} \\ & \text{subject to } \delta \in \Delta, \end{aligned} \tag{28}$$

where Δ represents a trust region, imposed to keep the approximation (27) valid. Most popular algorithms for nonlinear least squares alternate between solving some form of the subproblem (28), and updating β, $\mathbf{r}$, and $\mathbf{J}$ for those steps that achieve an acceptable improvement in residual.

5.2 Parameter update subproblem

This section describes two specific formulations of the subproblem (28).

5.2.1 Gauss–Newton update

Gauss–Newton methods (Nocedal and Wright 2006) find the δ that minimizes the quadratic objective (28), but with no trust region bounds. The optimal step is the solution to a set of linear equations,

$$\mathbf{J}^{\mathrm{T}}\mathbf{J}\delta = -\mathbf{J}^{\mathrm{T}}\mathbf{r}. \tag{29}$$

If the computed step does not achieve a satisfactory reduction in residual, a line search is typically used to refine the step length. The least squares partition algorithm due to Magnani and Boyd (2009) can be viewed as a Gauss–Newton update for the specific case of max-affine fitting.

5.2.2 Levenberg–Marquardt algorithms

Levenberg–Marquardt (LM) algorithms (Marquardt 1963; Moré 1978; Press et al. 2007) are similar to Gauss–Newton methods, but with trust region bounds on the parameter update. Instead of constraining δ to lie within the bounds of an explicit

trust region, LM algorithms construct the trust region implicitly through a quadratic penalty on δ. The advantage of this formulation is that the step is simply the solution to a linear system,

$$\left(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda \mathrm{diag}(\mathbf{J}^{\mathrm{T}}\mathbf{J})\right)\delta = -\mathbf{J}^{\mathrm{T}}\mathbf{r}, \tag{30}$$

where λ controls the magnitude of the quadratic penalty on δ. Various update schemes for λ exist; in general λ should be increased when the step fails to decrease the residual sufficiently, kept constant when the penalty term is not appreciably affecting the solution, and decreased otherwise.

5.2.3 Comments on scaling to large problems

From a computational perspective, solving (28) for δ is the dominant cost in solving a nonlinear least squares problem (Agarwal and Mierle 2010). The overall cost of fitting one of the proposed functions to a data set scales as the cost of solving (28), times the number of iterations needed for convergence, times the number of random restarts used to avoid poor local optima. The number of iterations and necessary number of random restarts are user-chosen parameters of the fitting algorithm for which rigorous bounds are not available.

The computational cost of each LM iteration is comparable to solving a linear least squares problem involving $\mathbf{J}$, which grows as $\mathcal{O}(mn_p^2)$ (Boyd and Vandenberghe 2004). The number of parameters n_p is $\mathcal{O}(Kn)$ for all three function classes, thus the total computational cost of each LM iteration is $\mathcal{O}(mK^2n^2)$.

Although the authors have not attempted to improve upon this scaling, for large problems better performance may be possible by exploiting sparsity. In particular, even for the smoothed SMA and ISMA function classes, the parameters $\mathbf{a}_k, b_k$ of each affine plane typically have relatively little effect on the function value at many of the data points, which introduces a nearly-sparse structure in $\mathbf{J}$. The limiting case is the max-affine function class, for which the fitting problems for each affine plane actually decouple. The Magnani-Boyd partition algorithm leverages this by solving K independent least squares problems, each with an average of m / K data points, for a total cost of $\mathcal{O}(mn^2)$. Thus iterative methods that can exploit near-sparsity in $\mathbf{J}$ may be capable of significantly reducing the effect of K on overall computational cost, even for SMA and ISMA functions.

5.3 Jacobians

This section lists the partial derivatives needed to construct Jacobians for all the convex function classes presented in Sect. 3. The parameterization in terms of $\log \alpha$ instead of α implicitly constrains $\alpha > 0$ and results in better numerical conditioning.

5.3.1 Max-affine functions

The partial derivatives for max-affine functions are defined assuming that there are no ties in the maximum that defines $f_{\mathrm{MA}}(x)$ (Magnani and Boyd 2009). For a practical implementation it suffices to break ties arbitrarily.

$$\frac{\partial f_{\mathrm{MA}}}{\partial b_i} = \begin{cases} 1, i = \operatorname{argmax}_k b_k + \mathbf{a}_k^{\mathrm{T}}\mathbf{x} \\ 0, \text{otherwise} \end{cases} \tag{31}$$

$$\frac{\partial f_{\mathrm{MA}}}{\partial \mathbf{a}_i} = \begin{cases} \mathbf{x}^{\mathrm{T}}, i = \operatorname{argmax}_k b_k + \mathbf{a}_k^{\mathrm{T}}\mathbf{x} \\ 0, \text{otherwise} \end{cases} \tag{32}$$

5.3.2 Softmax-affine functions

$$\frac{\partial f_{\mathrm{SMA}}}{\partial b_i} = \frac{\exp(\alpha(b_i + \mathbf{a}_i^{\mathrm{T}}\mathbf{x}))}{\sum_{k=1}^{K}\exp(\alpha(b_k + \mathbf{a}_k^{\mathrm{T}}\mathbf{x}))} \tag{33}$$

$$\frac{\partial f_{\mathrm{SMA}}}{\partial \mathbf{a}_i} = \frac{\mathbf{x}^{\mathrm{T}} \cdot \exp(\alpha(b_i + \mathbf{a}_i^{\mathrm{T}}\mathbf{x}))}{\sum_{k=1}^{K}\exp(\alpha(b_k + \mathbf{a}_k^{\mathrm{T}}\mathbf{x}))} \tag{34}$$

$$\frac{\partial f_{\mathrm{SMA}}}{\partial(\log\alpha)} = \frac{\sum_{k=1}^{K}(b_k + \mathbf{a}_k^{\mathrm{T}}\mathbf{x})\exp(\alpha(b_k + \mathbf{a}_k^{\mathrm{T}}\mathbf{x}))}{\sum_{k=1}^{K}\exp(\alpha(b_k + \mathbf{a}_k^{\mathrm{T}}\mathbf{x}))} - f_{\mathrm{SMA}} \tag{35}$$

5.3.3 Implicit softmax-affine functions

$$\frac{\partial f_{\mathrm{ISMA}}}{\partial b_i} = \frac{\alpha_i \exp(\alpha_i(b_i + \mathbf{a}_i^{\mathrm{T}}\mathbf{x} - f_{\mathrm{ISMA}}))}{\sum_{k=1}^{K}\alpha_k\exp(\alpha_k(b_k + \mathbf{a}_k^{\mathrm{T}}\mathbf{x} - f_{\mathrm{ISMA}}))} \tag{36}$$

$$\frac{\partial f_{\mathrm{ISMA}}}{\partial \mathbf{a}_i} = \frac{\mathbf{x}^{\mathrm{T}}\alpha_i \cdot \exp(\alpha_i(b_i + \mathbf{a}_i^{\mathrm{T}}\mathbf{x} - f_{\mathrm{ISMA}}))}{\sum_{k=1}^{K}\alpha_k\exp(\alpha_k(b_k + \mathbf{a}_k^{\mathrm{T}}\mathbf{x} - f_{\mathrm{ISMA}}))} \tag{37}$$

$$\frac{\partial f_{\mathrm{ISMA}}}{\partial(\log\alpha_i)} = \frac{\alpha_i(b_i + \mathbf{a}_i^{\mathrm{T}}\mathbf{x} - f_{\mathrm{ISMA}})\exp(\alpha_i(b_i + \mathbf{a}_i^{\mathrm{T}}\mathbf{x} - f_{\mathrm{ISMA}}))}{\sum_{k=1}^{K}\alpha_k\exp(\alpha_k(b_k + \mathbf{a}_k^{\mathrm{T}}\mathbf{x} - f_{\mathrm{ISMA}}))} \tag{38}$$

5.4 Parameter initialization

This section describes suitable initial parameter vectors β_0 for each of the convex function classes. To avoid convergence to a poor local optimum, the nonlinear least squares problem is randomly initialized multiple times, following the flowdown process shown in Fig. 4.

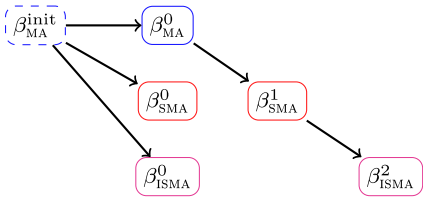


Fig. 4 Parameter initialization flowdown. Each $\beta^{\text{init}}_{\text{MA}}$ is randomly drawn as described in Sect. 5.4.1 and initializes a max-affine fit β^{0}_{MA}. SMA fits β^{0}_{SMA} and β^{1}_{SMA} are initialized from $\beta^{\text{init}}_{\text{MA}}$ and β^{0}_{MA} as described in Sect. 5.4.2, and the best fit is chosen. Finally, ISMA fits β^{0}_{SMA} and β^{2}_{SMA} are initialized from $\beta^{\text{init}}_{\text{MA}}$ and β^{1}_{SMA} as described in Sect. 5.4.3. Initializing additional ISMA fits from β^{0}_{MA} and β^{0}_{SMA} is sensible, but not done in the present implementation so that SMA and ISMA comparisons have an equal number of starting points

5.4.1 Max-affine initialization

The parameter initialization for a max-affine function follows Magnani and Boyd (2009). They pick K data points $\{\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \ldots, \bar{\mathbf{x}}_K\}$ at random without replacement, and partition the data set according to which $\bar{\mathbf{x}}$ is closest (in Euclidean distance, for example). Each partition is then fit locally using least squares, thereby forming an initial set of max-affine parameters.

In practice, one or more of the randomly-sampled partitions may contain very few data points, resulting in a singular local fitting problem. In the present implementation, rank-deficient initial partitions are expanded until all the local fits become well-posed.

5.4.2 Softmax-affine initialization

Since softmax-affine functions represent max-affine functions in the limit $\alpha \to +\infty$, max-affine functions provide a convenient initialization for SMA fitting. In particular, given a max-affine fit $\beta_{\text{MA}} = (\mathbf{a}_{\text{MA}}, \mathbf{b}_{\text{MA}})$, a natural softmax-affine initialization might be

$$\beta_{\text{SMA}} = (\mathbf{a}_{\text{MA}}, \mathbf{b}_{\text{MA}}, \alpha_0 = 100).$$

Too large an initial α_0 will cause the α-derivative (35) to be nearly zero for all data points. Therefore, a line search over α is implemented to find an α_0 that has non-zero $\partial f_{\text{SMA}}/\partial\alpha$ at some of the data points. This search manages the tradeoff between a poorly conditioned Jacobian (large α_0) and initial residuals significantly larger than those of the max-affine fit (small α_0).

5.4.3 Implicit softmax-affine initialization

Given the parameters $\beta_{\text{SMA}} = (\mathbf{a}_{\text{SMA}}, \mathbf{b}_{\text{SMA}}, \alpha_{\text{SMA}})$ of a SMA fit, a suitable ISMA initialization is

$$\beta_{\text{ISMA}} = (\mathbf{a}_{\text{SMA}}, \mathbf{b}_{\text{SMA}}, [\alpha_{\text{SMA}}, \alpha_{\text{SMA}}, \ldots, \alpha_{\text{SMA}}]).$$

Alternatively, one can initialize ISMA fitting directly from a MA fit:

$$\beta_{\text{ISMA}} = (\mathbf{a}_{\text{MA}}, \mathbf{b}_{\text{MA}}, [100, 100, ..., 100]).$$

5.5 Avoiding numerical overflow

Many `exp` terms appear in both the convex function definitions and their Jacobians. When exponentiated, a modestly large argument can quickly overwhelm the representation capability of floating point arithmetic. One must therefore use caution when implementing these equations in software. Two common situations occur:

Ratios of sums of exponentials To handle these, note that

$$\frac{ce^{\alpha p}}{\sum_{i=1}^{N} e^{\alpha p_i}} = \frac{ce^{\alpha(p-s)}}{\sum_{i=1}^{N} e^{\alpha(p_i-s)}}, \tag{39}$$

i.e. one can simply subtract some s from all the exponents in the numerator and denominator, such that the largest argument to `exp` is small - say, 0.

Log sum exp terms To handle these, note that

$$\frac{1}{\alpha}\log\sum_{i=1}^{N} e^{\alpha p_i} = s + \frac{1}{\alpha}\log\sum_{i=1}^{N} e^{\alpha(p_i-s)}, \tag{40}$$

for some s chosen to keep the maximum exponential small.

6 Numerical examples and comparisons

Throughout this section, RMS error on the residuals, $\hat{y} - y$, is defined as

$$\text{RMS error} \equiv \sqrt{\frac{1}{m}\sum_{i=1}^{m}(f(\mathbf{x}_i) - y_i)^2}. \tag{41}$$

Absolute error on $(\mathbf{x}, y)$ is related to relative error on $(\mathbf{u}, w)$, since

$$\frac{\hat{w} - w}{w} \approx \log\frac{\hat{w}}{w} = \hat{y} - y. \tag{42}$$

6.1 Example: local convex fitting of a scalar function

Suppose that the scalar relationship

$$w(u) = \frac{u^2 + 3}{(u+1)^2}, 1 \le u \le 3, \tag{43}$$

expresses an important relationship between two variables (u, w) in a GP. Can this relationship be encoded as a GP-compatible constraint?

Importantly, the expression $(u^2 + 3)/(u + 1)^2$ is not log-log-convex for all $u > 0$ (a simple log-log plot verifies this). Thus there is no hope of algebraically

manipulating (43) into a posynomial constraint. Nevertheless, (43) *is* log-log-convex over the domain of interest, $1 \leq u \leq 3$. This suggests sampling the function and fitting one of the proposed convex function classes to the 'data'. For this problem a value of $K = 2$ is used. Figure 5 shows the resulting fits, with SMA and ISMA functions significantly outperforming MA functions. The fitted functions are listed in Table 1.

6.2 Performance on randomly-generated convex functions

To test the proposed function classes and fitting algorithms on a wider range of data sets, a convex function generator was created to produce random instances of convex functions using a stochastic grammar of convexity-preserving operations and composition rules. This generator was used to produce 20 convex functions $\mathbb{R}^2 \rightarrow \mathbb{R}$, and 20 more convex functions $\mathbb{R}^5 \rightarrow \mathbb{R}$. For example, the first $\mathbb{R}^2 \rightarrow \mathbb{R}$ function drawn was

$$
\begin{aligned}
f_1(x_1, x_2) = \max(&\log(\exp(-0.32724x_1 + 0.89165x_2) + \exp(0.44408x_1 - 0.91182x_2 \\
&+ \ldots \max(-0.10307x_1 - 2.799x_2 + (0.62101x_1 - 1.5075x_2)^2 + 0.2417x_1 \\
&+ \cdots 0.54935x_2 - 0.2298x_1 - 0.57922x_2 + (0.42162x_1 + 1.0877x_2)^2, \\
&\log(\exp((0.17694x_1 + 3.4663x_2)^2) + \exp(\max(\log(\exp((-0.4385x_1 \\
&+ \ldots 0.34322x_2)^2) + \exp(\max(-0.83768x_1 - 1.3075x_2, 0.64915x_1 \\
&- 0.83147x_2))), 1.5667x_1 + 0.8465x_2))) - 0.39754x_1 + 0.25429x_2))), \\
&(1.2951x_1 + 2.7681x_2)^2).
\end{aligned}
$$

Input data $\mathbf{x}$ for each function consisted of 1000 points drawn from a multivariate Gaussian with zero mean and identity covariance. For each of the fitting problems, the fitting process was restarted from ten random initializations, and the best fit was chosen. Figure 6 and Tables 2 and 3 report the average fitting error and time across the 20 randomly-generated fitting problems considered in $\mathbb{R}^2$ and $\mathbb{R}^5$ respectively.

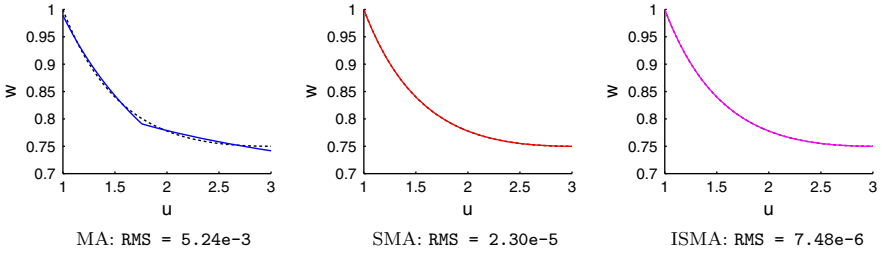


Fig. 5 Generalized posynomial fitting of the function $w = (u^2 + 3)/(u + 1)^2$ over $1 \leq u \leq 3$. The function was sampled at 501 logarithmically-spaced points, which were log-transformed and fitted with MA, SMA, and ISMA functions, each with $K = 2$ affine terms. Converting these functions back to to the original (u, w) space turned them into max-monomial and posynomial functions

Table 1 Convex fits for the fitting problem in Sect. 6.1

Function class	Fitted function	RMS log error
MA	$\hat{w} = \max(0.846u^{-0.12}, 0.989u^{-0.397})$	5.24×10^{-3}
SMA	$\hat{w}^{3.44} = 0.154u^{0.584} + 0.847u^{-2.15}$	2.30×10^{-5}
ISMA	$1 = 0.0658\frac{u^{0.958}}{\hat{w}^{3.79}} + 0.934\frac{u^{-1.22}}{\hat{w}^{2.03}}$	7.48×10^{-6}

6.3 Example: power modeling for circuit design

Consider a power dissipation model,

$$P = V_{\text{dd}}^2 + 30V_{\text{dd}}e^{-(V_{\text{th}}-0.06V_{\text{dd}})/0.039}, \tag{44}$$

which is relevant for GP-based circuit design. This example comes from Boyd et al. (2005), and is also studied in Hannah and Dunson (2012). The power relationship (44) is to be modeled as a posynomial constraint for the domain $1.0 \leq V_{\text{dd}} \leq 2.0$ and $0.2 \leq V_{\text{th}} \leq 0.4$.

Starting with 1000 samples of $\mathbf{u} = (V_{\text{dd}}, V_{\text{th}})$, uniformly drawn from the region of interest, each sample is associated with the corresponding $w = P(\mathbf{u})$. After applying the log transformation $(\mathbf{x}, y) = (\log \mathbf{u}, \log w)$, MA, SMA, and ISMA functions are fit to the transformed data set. Table 4 gives the resulting RMS log errors. Of course, any of the resulting fits is easily converted to a posynomial constraint on V_{dd}, V_{th}, and P. For example, the $K = 3$ SMA fit corresponds to a posynomial constraint,

$$P^{2.14} \geq 1.09V_{\text{dd}}^{4.27}V_{\text{th}}^{0.112} + 7.79 \times 10^{-5}\frac{V_{\text{dd}}^{4.75}}{V_{\text{th}}^{6.44}} + 1.36 \times 10^{-7}\frac{V_{\text{dd}}^{8.94}}{V_{\text{th}}^{8.89}}, \tag{45}$$

which is readily substituted directly into any GP.

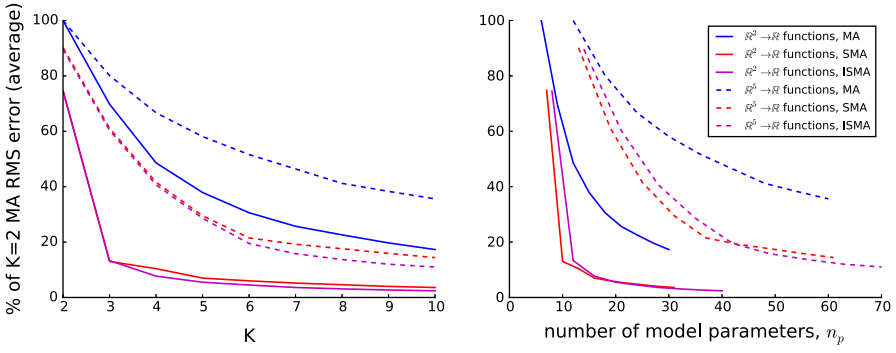


Fig. 6 Average fitting error on the 20 randomly generated functions from Sect. 6.2, plotted against the number of terms (*left*) and against the number of model parameters (*right*). SMA and ISMA functions provide a consistent benefit over MA functions. For some fitting problems, SMA functions can achieve a desired residual with fewer model parameters than ISMA functions (but not with fewer terms)

Table 2 Fitting error comparison for 20 randomly-generated functions $\mathbb{R}^2 \rightarrow \mathbb{R}$. Results are reported across the 20 fitting problems considered. The code ran on a quad-core Intel Core i7 CPU at 2.8GHz with 4GB memory

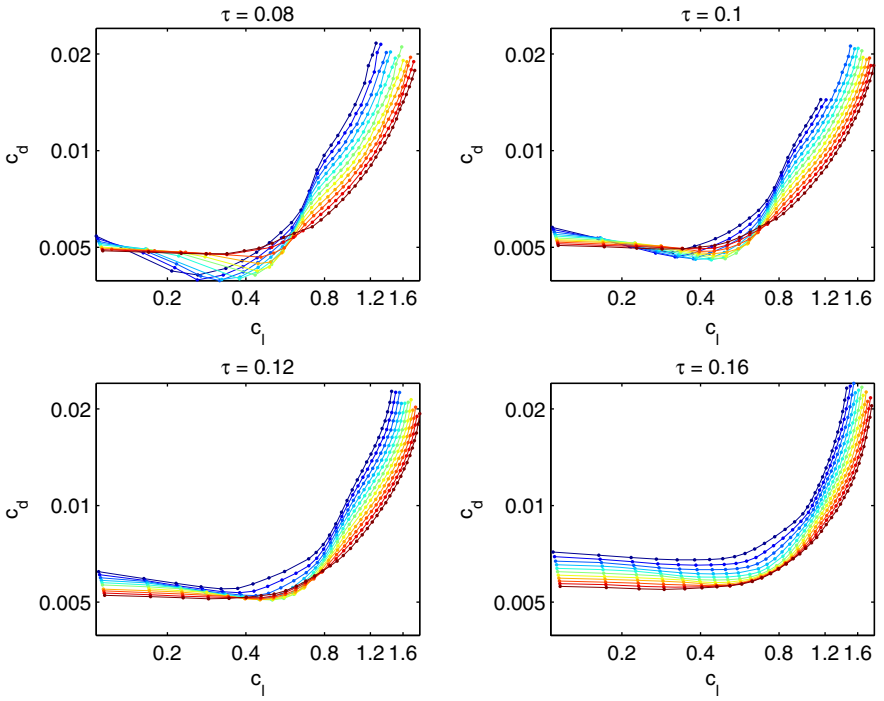
K	RMS error as percentage of MA error with K = 2 (worst-case, average, best-case)			Average fitting time (s) per random restart		
	MA	SMA	ISMA	MA	SMA	ISMA
2	(100.0, 100.0, 100.0)	(93.4, 74.8, 56.0)	(93.0, 74.5, 55.8)	0.18	0.28	0.48
3	(77.0, 69.8, 59.1)	(24.3, 13.0, 5.4)	(24.1, 13.3, 6.9)	0.23	0.36	0.90
4	(53.6, 48.6, 39.3)	(17.2, 10.4, 4.9)	(10.7, 7.7, 3.7)	0.24	0.55	0.95
5	(42.9, 37.9, 26.7)	(10.3, 7.0, 3.5)	(9.0, 5.5, 3.0)	0.26	0.54	1.10
6	(36.0, 30.6, 22.1)	(9.0, 6.0, 3.3)	(7.1, 4.5, 2.7)	0.28	0.53	1.56
7	(29.0, 25.7, 18.9)	(8.5, 5.2, 2.8)	(6.4, 3.6, 1.6)	0.29	0.92	1.44
8	(26.0, 22.6, 17.3)	(7.7, 4.6, 2.4)	(5.5, 3.1, 1.6)	0.32	0.85	1.27
9	(23.6, 19.7, 16.1)	(6.4, 4.0, 2.3)	(5.0, 2.7, 1.5)	0.38	1.12	1.55
10	(19.6, 17.3, 14.0)	(5.9, 3.6, 2.1)	(4.3, 2.4, 1.1)	0.38	1.22	1.66

Table 3 Fitting error comparison for 20 randomly-generated functions $\mathbb{R}^5 \rightarrow \mathbb{R}$

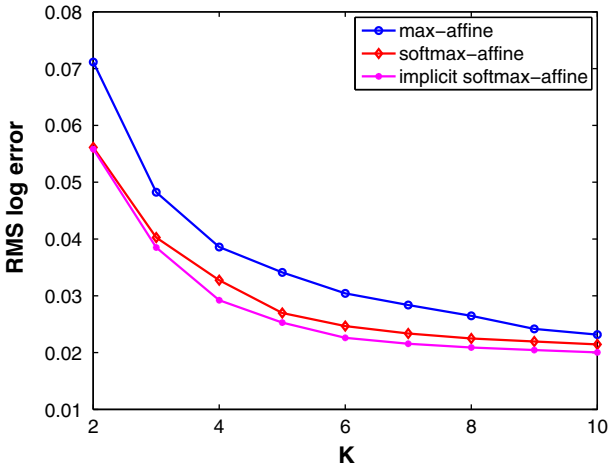
K	RMS error as percentage of MA error with K = 2 (worst-case, average, best-case)			Average fitting time (s) per random restart		
	MA	SMA	ISMA	MA	SMA	ISMA
2	(100.0, 100.0, 100.0)	(98.3, 90.2, 58.6)	(97.6, 89.6, 58.3)	0.54	0.66	1.40
3	(86.2, 80.1, 60.3)	(78.2, 61.1, 32.5)	(77.5, 60.4, 32.3)	0.52	0.74	2.04
4	(76.9, 66.7, 45.3)	(56.9, 41.6, 21.6)	(56.2, 40.7, 21.1)	0.62	0.91	2.07
5	(67.6, 58.1, 40.4)	(41.8, 29.5, 16.7)	(41.1, 28.6, 14.7)	0.86	0.99	1.98
6	(59.6, 51.6, 33.7)	(33.6, 21.5, 13.7)	(30.2, 19.5, 13.1)	0.86	1.43	2.38
7	(55.5, 46.4, 26.8)	(31.9, 19.2, 12.6)	(22.7, 15.8, 10.7)	1.15	1.60	2.89
8	(52.1, 41.2, 19.6)	(28.6, 17.6, 12.1)	(18.7, 13.7, 9.0)	1.23	1.78	3.00
9	(48.3, 38.3, 17.1)	(24.1, 15.9, 10.8)	(17.0, 12.0, 7.9)	1.38	2.39	4.24
10	(43.7, 35.6, 15.7)	(21.8, 14.4, 9.8)	(15.8, 11.0, 6.9)	1.56	3.23	5.05

Table 4 RMS log errors for the circuit design power model in Sect. 6.3. For $K = 2$ through $K = 4$, SMA functions outperformed MA functions by a factor of 2.5–44

K	MA	SMA	ISMA
1	0.0904	0.0904	0.0904
2	0.0229	0.0092	0.0091
3	0.01260	0.00037	0.00034
4	0.00760	0.00017	0.00014



Profile drag data



GP-compatible fitting error

Fig. 7 Approximating profile drag for NACA 24xx airfoils. Here the data consists of 3073 samples of profile drag coefficient (c_d) as a function of lift coefficient (c_l), Reynolds number (Re), and airfoil thickness coefficient (τ). Each function class was fit for a range of K, with implicit softmax-affine functions achieving the best fit. For each K, 20 random restarts (initializations) were used, and, of these, the best fit achieved was chosen

6.4 Example: profile drag modeling for aircraft design

In aerospace engineering, *profile drag* on a wing arises due to viscous skin friction in the boundary layer surrounding the airfoil. For a given airfoil cross section shape, the profile drag coefficient (c_d) depends on Reynolds number (Re), airfoil thickness ratio (τ), and lift coefficient (c_l). No analytical expression is available for this relationship, but it can be simulated using the viscous airfoil analysis program XFOIL (Drela 2000).

The data for this example consists of 3073 samples, $((\mathrm{Re}, \tau, c_l), c_d)_i$, generated for Re ranging from 10^6 to 10^7, τ ranging from 8 to 16 %, and c_l ranging from 0.01 to stall. As before, a log transformation, $(\mathbf{x}, y) = (\log(\mathrm{Re}, \tau, c_l), \log c_d)$ is applied. MA, SMA, and ISMA functions are then fitted using LM for a range of K values.

As shown in Fig. 7, for a given number of terms K, an ISMA function provides the best fit, followed by SMA, and finally by MA. A related interpretation is that the ISMA function class can provide a desired level of maximum RMS error with no more, and often fewer, terms than the best MA fit. Each of the fitted functions is directly compatible with GP, and can therefore represent profile drag relations in practical aircraft design problems (Hoburg and Abbeel 2014).

7 Conclusions

This article discusses methods for fitting GP-compatible functions to data. Specifically, the focus of this work is on fitting special classes of convex functions to logarithmically transformed data. Two convex function classes are introduced: softmax-affine, and implicit softmax-affine. In Sect. 3, it is shown that these functions form a hierarchy, with the most general ISMA function class able to reproduce the other classes of convex functions for special parameter settings. One can therefore conclude that the best possible ISMA fit is always at least as good (and often better than) the best possible fit from the more commonly used max-affine class.

Both of the proposed function classes have parallel interpretations as GP-compatible posynomial epigraphs. Indeed, the proposed approach unifies max-affine fitting, max-monomial fitting, and posynomial fitting as special cases of SMA and ISMA fitting. The most general ISMA function class leverages the full expressive power of GP, by using an implicit representation corresponding to a posynomial constraint $g([\mathbf{u}, w]) \leq 1$.

While the focus of this article is on the form of the function classes and their connection to GP, the ingredients of a practical fitting algorithm that can be used to fit these functions to data are also presented.

Acknowledgments The authors thank Aude Hofleitner, Timothy Hunter, and several anonymous reviewers for their thorough and insightful comments on the draft. This work was supported by a National Science Foundation Graduate Research Fellowship.

References

Agarwal S, Mierle K (2010) Others: ceres solver. http://ceres-solver.org

Babakhani A, Lavaei J, Doyle J, Hajimiri A (2010) Finding globally optimum solutions in antenna optimization problems. In: IEEE international symposium on antennas and propagation

Beightler CS, Phillips DT (1976) Applied geometric programming. Wiley, New York

Boyd S, Kim SJ, Vandenberghe L, Hassibi A (2007) A tutorial on geometric programming. Optim Eng 8(1):67–127

Boyd S, Vandenberghe L (2004) Convex Optim. Cambridge University Press, New York

Boyd SP, Kim SJ, Patil DD, Horowitz MA (2005) Digital circuit optimization via geometric programming. Op Res 53:899–932

Chiang M (2005) Geometric programming for communication systems. Commun Inf Theory 2:1–154. doi:10.1516/0100000005. http://portal.acm.org/citation.cfm?id=1166381.1166382

Daems W, Gielen G, Sansen W (2003) Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits. IEEE Trans Comput Aided Des Integr Circuits Syst 22(5):517–534

Drela M (2000) Xfoil subsonic airfoil development system. Open source software. http://web.mit.edu/drela/Public/web/xfoil/

Duffin RJ, Peterson EL, Zener C (1967) Geometric programming: theory and application. Wiley, New York

Hannah L, Dunson D (2012) Ensemble methods for convex regression with applications to geometric programming based circuit design. arXiv preprint. http://arxiv.org/abs/1206.4645

Hannah LA, Dunson DB (2011) Multivariate convex regression with adaptive partitioning. arXiv preprint. http://arxiv.org/abs/1105.1924

Hoburg W, Abbeel P (2014) Geometric programming for aircraft design optimization. AIAA J 52:2414–2426

Kasamsetty K, Ketkar M, Sapatnekar SS (2000) A new class of convex functions for delay modeling and its application to the transistor sizing problem [cmos gates]. IEEE Trans Comput Aided Des Integr Circuits Syst 19(7):779–788

Kim J, Vandenberghe L, Yang CKK (2010) Convex piecewise-linear modeling method for circuit optimization via geometric programming. IEEE Trans Comput Aided Des Integr Circuits Syst 29(11):1823–1827. doi:10.1109/TCAD.2010.2053151

Li X, Gopalakrishnan P, Xu Y, Pileggi T (2004) Robust analog/rf circuit design with projection-based posynomial modeling. In: Proceedings of the 2004 IEEE/ACM international conference on computer-aided design, pp 855–862. IEEE computer society

Magnani A, Boyd SP (2009) Convex piecewise-linear fitting. Optim Eng 10:1–17. doi:10.1007/s11081-008-9045-3

Marquardt DW (1963) An algorithm for least-squares estimation of nonlinear parameters. J Soc Ind Appl Math 11(2):431–441. http://www.jstor.org/stable/2098941

Mehrotra S (1992) On the implementation of a primal-dual interior point method. SIAM J Optim 2(4):575–601

Moré JJ (1978) The Levenberg–Marquardt algorithm: implementation and theory. In: Watson GA (ed) Numerical analysis. Springer, Berlin, pp 105–116

Nesterov Y, Nemirovsky A (1994) Interior-point polynomial methods in convex programming, volume 13 of studies in applied mathematics. SIAM, Philadelphia

Nocedal J, Wright SJ (2006) Numerical optimization. Springer, New York

Oliveros J, Cabrera D, Roa E, Van Noije W (2008) An improved and automated design tool for the optimization of cmos otas using geometric programming. In: Proceedings of the 21st annual symposium on integrated circuits and system design, pp 146–151. ACM

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2007) Numerical recipes 3rd edition: the art of scientific computing. Cambridge University Press, Cambridge

Wilde D (1978) Globally optimal design. Wiley, New York. http://books.google.com/books?id=XYBRAAAAMAAJ

Ypma TJ (1995) Historical development of the Newton–Raphson method. SIAM Rev 37(4):531–551