CS 524: Introduction to Optimization
Lecture 39 : Geometric Programming

Michael Ferris

Computer Sciences Department
University of Wisconsin-Madison

December 6, 2023

Example: geometric programming

Example: We want to design a box of height h, width w, and depth d with maximum volume (hwd) subject to the limits:

- total wall area: $2(hw + hd) \leq A_{\text{wall}}$
- total floor area: $wd \leq A_{\text{flr}}$
- height-width aspect ratio: $\alpha \leq \frac{h}{w} \leq \beta$
- width-depth aspect ratio: $\gamma \leq \frac{d}{w} \leq \delta$

We can make some of the constraints linear, but not all of them. This appears to be a nonconvex optimization problem...

Monomials and posynomials

- $\mathrm{dom} f = \mathbb{R}_{++}$
- Monomial: $g(x) = c x_1^{a_1} \dots x_n^{a_n}$, $c \geq 0$
- Posynomial: $f(x) = \sum_k c_k x_1^{a_1 k} \dots x_n^{a_n k}$, $c_k \geq 0$
- Geometric Program:

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 1 \\ & g_j(x) = 1 \end{aligned}$$

with $x \in \mathbb{R}_{++}$ as an implicit constraint

Example: geometric programming

Example: We want to design a box of height h, width w, and depth d with maximum volume (hwd) subject to the limits:

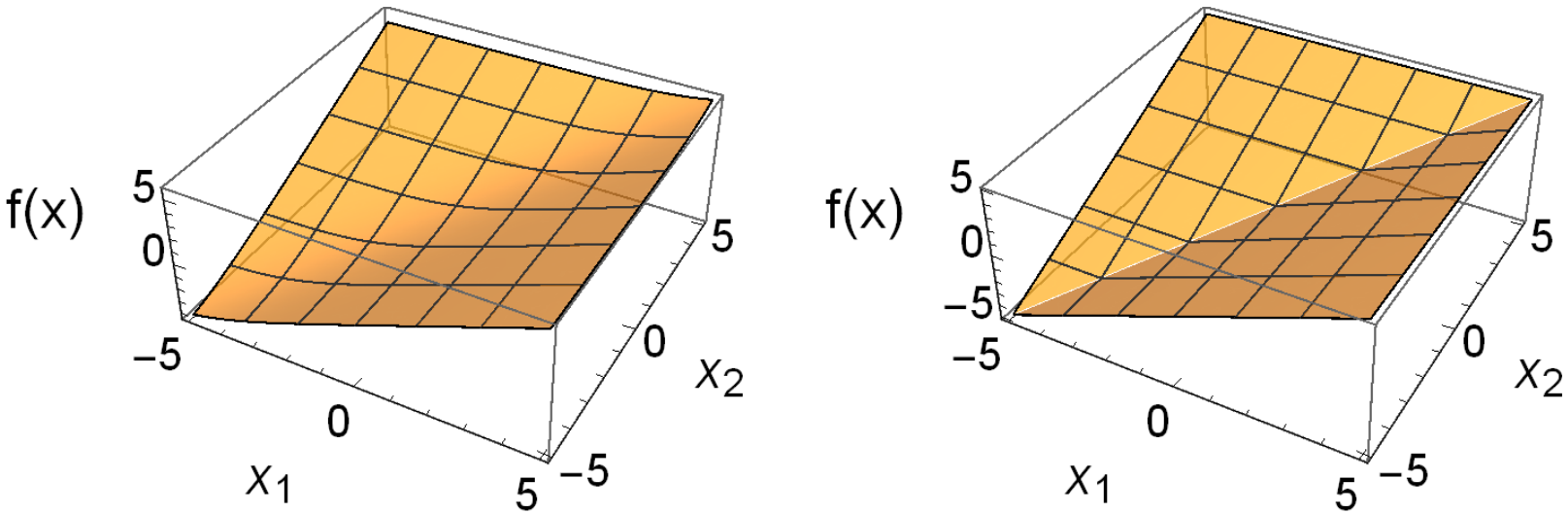
- total wall area: $2(hw + hd) \leq A_{\text{wall}}$
- total floor area: $wd \leq A_{\text{flr}}$
- height-width aspect ratio: $\alpha \leq \frac{h}{w} \leq \beta$
- width-depth aspect ratio: $\gamma \leq \frac{d}{w} \leq \delta$

$$
\begin{array}{lll}
\underset{h,w,d>0}{\text{minimize}} & h^{-1}w^{-1}d^{-1} & \\
\text{subject to:} & \frac{2}{A_{\text{wall}}}hw + \frac{2}{A_{\text{wall}}}hd \leq 1, & \frac{1}{A_{\text{flr}}}wd \leq 1 \\
& \alpha h^{-1}w \leq 1, & \frac{1}{\beta}hw^{-1} \leq 1 \\
& \gamma wd^{-1} \leq 1, & \frac{1}{\delta}w^{-1}d \leq 1
\end{array}
$$

Example: geometric programming

The **log-sum-exp** function (shown left) is convex:

$$f(x) := \log\left(\sum_{k=1}^{n} \exp x_k\right)$$



It's a smoothed version of $\max\{x_1, \ldots, x_k\}$ (shown right)

Example: geometric programming

Suppose we have positive decision variables $x_i > 0$, and constraints of the form (with each $c_j > 0$ and $\alpha_{jk} \in \mathbb{R}$):

$$\sum_{j=1} c_j x_1^{\alpha_{j1}} x_2^{\alpha_{j2}} \cdots x_n^{\alpha_{jn}} \leq 1$$

Then by using the substitution $y_i := \log(x_i)$, we have:

$$\log \left(\sum_{j=1}^{n} \exp\left(a_{j0} + a_{j1} y_1 + \cdots + a_{jn} y_n\right) \right) \leq 0$$

(where $a_{j0} := \log c_j$). This is a log-sum-exp function composed with an affine function (convex!)

Convex reformulation (see box.gms)

- $y_i = \log x_i$, $x_i = \exp y_i$, deals with $\mathbb{R}_{++}$ nicely
- Monomial: $g(x) = \exp(a^T y + \log c)$
- Posynomial: $f(x) = \sum_k \exp(a_k^T y + \log c_k)$
- Geometric Program:

$$\begin{aligned} \min \quad & \tilde{f}_0(y) = \log(\sum_k \exp(a_{0k}^T y + \log c_{0k})) \\ \text{s.t.} \quad & \tilde{f}_i(y) = \log(\sum_k \exp(a_{ik}^T y + \log c_{ik})) \leq 0 \\ & \tilde{g}_j(y) = a_j^T y + \log c_j = 0 \end{aligned}$$

Example: geometric programming

$$
\begin{aligned}
\underset{h,w,d>0}{\text{minimize}} \quad & h^{-1}w^{-1}d^{-1} \\
\text{subject to:} \quad & \tfrac{2}{A_{\text{wall}}}hw + \tfrac{2}{A_{\text{wall}}}hd \le 1, \quad \tfrac{1}{A_{\text{flr}}}wd \le 1 \\
& \alpha h^{-1}w \le 1, \quad \tfrac{1}{\beta}hw^{-1} \le 1 \\
& \gamma wd^{-1} \le 1, \quad \tfrac{1}{\delta}w^{-1}d \le 1
\end{aligned}
$$

- Define: $x := \log h$, $y := \log w$, and $z := \log d$.
- Express the problem in terms of the new variables x, y, z.
  Note: h, w, d are positive but x, y, z are unconstrained.

Example: geometric programming

$$
\begin{aligned}
\underset{x,y,z}{\text{minimize}} \quad & \log(e^{-x-y-z}) \\
\text{subject to:} \quad & \log(e^{\log(2/A_{\text{wall}})+x+y} + e^{\log(2/A_{\text{wall}})+x+z}) \leq 0 \\
& \log(e^{\log(1/A_{\text{flr}})+y+z}) \leq 0 \\
& \log(e^{\log\alpha - x + y}) \leq 0, \qquad \log(e^{-\log\beta + x - y}) \leq 0 \\
& \log(e^{\log\gamma + y - z}) \leq 0, \qquad \log(e^{-\log\delta - y + z}) \leq 0
\end{aligned}
$$

- this is a convex model, but it can be simplified!
- most of the constraints are actually linear.

Example: geometric programming

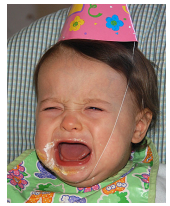
$$
\begin{aligned}
\underset{x,y,z}{\text{minimize}} \quad & -x - y - z \\
\text{subject to:} \quad & \log\left(e^{\log(2/A_{\text{wall}})+x+y} + e^{\log(2/A_{\text{wall}})+x+z}\right) \leq 0 \\
& y + z \leq \log A_{\text{flr}} \\
& \log \alpha \leq x - y \leq \log \beta \\
& \log \gamma \leq z - y \leq \log \delta
\end{aligned}
$$

- This is a convex optimization problem.

Extensions

- We can maximize a nonzero monomial objective function, by minimizing its inverse (which is also a monomial).
- 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).
- See Enzyme Cost Minimization on NEOS.

My Model Doesn't Solve



How to deal with bad models, or models that don't solve.

1. Formulation: see the examples from other lectures.
2. Try different solvers, CONOPT, KNITRO
3. Starting points, maybe use loop to set level (`.l`)
4. Update bounds on variables to restrict search.
5. Introduce the notion of scaling
6. Introduce the notion of intermediates
7. Try using a global solver (to find global solutions) on a smaller instance
   - Multistart
   - Global optimization software.

Initial Values

Most recommendations will be useful for any nonlinear solver. Good initial values are important for many reasons.

- Initial values that satisfy or closely satisfy many of the constraints reduces the work involved in finding a first feasible solution.
- Initial values that in addition are close to the optimal ones also reduce the distance to the final point and therefore indirectly the computational effort.
- The progress of the optimization algorithm is based on good directional information and therefore on good derivatives. The derivatives in a nonlinear model depend on the current point, and the initial point in which the initial derivatives are computed is therefore again important.
- Non-convex models may have multiple solutions, but the modeler is looking for one in a particular part of the search space; an initial point in the right neighborhood is more likely to return the desired solution.

Initial values in GAMS

- The initial values used by GAMS are by default the value zero projected on the bounds. I.e. if a variable is free or has a lower bound of zero, then its default initial value is zero.
- Unfortunately, zero is in many cases a bad initial value for a nonlinear variable. An initial value of zero is especially bad if the variable appears in a product term since the initial derivative becomes zero, and it appears as if the function does not depend on the variable.
- CONOPT will warn you and ask you to supply better initial values if the number of derivatives equal to zero is larger than 20 percent.
- If a variable has a small positive lower bound, for example because it appears as an argument to the Log function or as a denominator, then the default initial value is this small lower bound and it is also bad since this point will have very large first and second derivatives.
- You should therefore supply as many sensible initial values as possible by making assignment to the level value, var.L, in GAMS.

Use the model, duh!

An easy possibility is to initialize all variables to 1, or to the scale factor if you use GAMS' scaling option. A better possibility is to select reasonable values for some important variables, and then use some of the equations of the model to derive values for other variables. A model may contain the following equation:

```
xDef(i) .. x(i) =e= y(i)*z*(1 + m(i)) ;
```

where x, y, and z are variables and m is a parameter. The following assignment statements use the equation to derive consistent initial values for x from sensible initial values for y and z:

```
z.l = 1; y.l(i) = 1;
x.l(i) = y.l(i)*z.l*(1 + m(i)) ;
```

Bounds

Bounds have two purposes in nonlinear models.

- Some bounds represent constraints on the reality that is being modeled, e.g. a variable must be positive. These bounds are called model bounds.
- Other bounds help the algorithm by preventing it from moving far away from any optimal solution and into regions with singularities in the nonlinear functions or unreasonably large function or derivative values. These bounds are called algorithmic bounds.

Model bounds have natural roots and do not cause any problems.

Algorithmic bounds

Algorithmic bounds require a closer look at the functional form of the model. The argument of a Log should be greater than say 1.e-3, the argument of an Exp should be less than 5 to 8, and a denominator should be greater than say 1.e-2. These recommended lower bounds of 1.e-3 and 1.e-2 may appear to be unreasonably large. However, both Log(x) and 1/x are extremely nonlinear for small arguments. The first and second derivatives of Log(x) at x=1.e-3 are 1.e+3 and -1.e6, respectively, and the first and second derivatives of 1/x at x=1.e-2 are -1.e+4 and 2.e+6, respectively.

If the argument of a Log or Exp function or a denominator is an expression then it may be advantageous to introduce a bounded intermediate variable as discussed below.

Simple Expressions

The following model component

```
Parameter mu(i);
Variable  x(i), s(i), obj;
Equation  objDef;
objDef .. obj =e= Exp( Sum(i, Sqr( x(i)-mu(i) ) / s(i)));
```

can be re-written in the slightly longer but simpler form

```
Parameter mu(i);
Pariable  x(i), s(i), obj, inTerm;
Equation  intDef, objDef;
intDef .. inTerm =e= Sum(i, Sqr( x(i)-mu(i) ) / s(i));
objDef .. obj    =e= Exp( inTerm );
```

The first formulation has very complex derivatives because Exp is taken of a long expression. The second formulation has much simpler derivatives; Exp is taken of a single variable, and the variables in intDef appear in a sum of simple independent terms.

Intermediate variables often help

- Try to avoid nonlinear functions of expressions, divisions by expressions, and products of expressions, especially if the expressions depend on many variables.
- Define intermediate variables that are equal to the expressions and apply the nonlinear function, division, or product to the intermediate variable.
- The model will become larger, but the increased size is taken care of by the solvers sparse matrix routines, and it is compensated by the reduced complexity.
- If the model is solved with CONOPT using explicit second derivatives then simple expressions will result in sparser second derivatives that are both faster to compute and to use.

Reduction in complexity can be significant

If an intermediate expression is linear, then the following model fragment:

```
Variable x(i), y;  Equation yDef;
yDef ..  y =e= 1 / Sum(i, x(i) );
```

should be written as

```
Variable x(i), xSum, y;  Equation xSumDef, yDef;
xSumDef .. xSum =e= Sum(i, x(i) );
yDef    .. y =e= 1 / xSum;
xSum.lo = 1.e-2;
```

for three reasons.

1. The number of nonlinear derivatives is reduced in number and complexity.
2. The lower bound on the intermediate result will bound the search away from the singularity at xSum $= 0$.
3. The matrix of second derivatives for the last model only depends on xSum while it depends on all x in the first model.

Important Safety Tips

- a^x is evaluated as $e^{\log(a^x)} = e^{x \log(a)}$
- This is bad[1] when a is negative
- If x is an integer, a^x should not be bad
- Use `power(a,x)`. `x` should be an integer, though.
- There is also a `sqr` function
- Some functions are safer than others: See the GAMS user guide for details.

[1] not defined