gpposy
A Matlab Solver for Geometric Programs in Posynomial Form

Kwangmoo Koh
deneb1@stanford.edu

Seungjean Kim
sjkim@stanford.edu

Almir Mutapcic
almirm@stanford.edu

Stephen Boyd
boyd@stanford.edu

May 22, 2006

1 Introduction

gpposy solves an optimization problem of the form

$$\begin{array}{ll} \text{minimize} & \sum_{k=1}^{K_0} b_k^{(0)} x_1^{a_{k1}^{(0)}} x_2^{a_{k2}^{(0)}} \cdots x_n^{a_{kn}^{(0)}} \\ \text{subject to} & \sum_{k=1}^{K_i} b_k^{(i)} x_1^{a_{k1}^{(i)}} x_2^{a_{k2}^{(i)}} \cdots x_n^{a_{kn}^{(i)}} \leq 1, \quad i = 1, \ldots, m, \\ & h_i x_1^{g_{i1}} x_2^{g_{i2}} \cdots x_n^{g_{in}} = 1, \quad i = 1, \ldots, p, \\ & l \preceq x \preceq u, \end{array} \tag{1}$$

where the optimization variable is the vector $x = (x_1, \ldots, x_n) \in \mathbf{R}_+^n$. The problem data are $a_{kj}^{(i)}, g_{ij} \in \mathbf{R}$, $b_k^{(i)}, h_i \in \mathbf{R}_+$, and $l, u \in \mathbf{R}_+^n$. Here $\preceq$ means componentwise inequality between vectors. This problem is called a *geometric program in posynomial form*. For more information about geometric programming, see [BV04, BKVH].

2 Calling sequences

The complete calling sequence of gpposy is

```
>> [x,status,lambda,nu] = gpposy(A,b,szs,G,h,l,u,quiet);
```

Input arguments represent the problem data of the problem (1). Output arguments are the optimal point (if feasible), sensitivity information (if feasible) and the solution status.

Input arguments

- `A`: matrix with n columns and $\sum_{i=0}^{m} K_i$ rows that specifies the exponents of the objective and equality constraints, *i.e.*,

$$A = \begin{bmatrix} A^{(0)} \\ \vdots \\ A^{(m)} \end{bmatrix}, \quad A^{(i)} = \begin{bmatrix} a_{11}^{(i)} & \cdots & a_{1n}^{(i)} \\ \vdots & \ddots & \vdots \\ a_{K_i 1}^{(i)} & \cdots & a_{K_i n}^{(i)} \end{bmatrix} \in \mathbf{R}^{K_i \times n}, \quad i = 0, \ldots, m.$$

  `A` can be in sparse format.

- `b`: vector of length $\sum_{i=0}^{m} K_i$ that specifies the coefficients of the objective and inequality constraints, *i.e.*,

$$b = \begin{bmatrix} b^{(0)} \\ \vdots \\ b^{(m)} \end{bmatrix}, \quad b^{(i)} = \begin{bmatrix} b_1^{(i)} \\ \vdots \\ b_{K_i}^{(i)} \end{bmatrix} \in \mathbf{R}_+^{K_i}, \quad i = 0, \ldots, m.$$

  All elements $b_k^{(i)}$ must be positive.

- `szs`: vector of length $m+1$ that specifies the number of terms in each objective and inequality constraints, *i.e.*, $(K_0, \ldots, K_m)$.

- `G`: matrix with n columns and p rows, that specifies the exponents of equality constraints, *i.e.*,

$$G = \begin{bmatrix} g_{11} & \cdots & g_{1n} \\ \vdots & \ddots & \vdots \\ g_{p1} & \cdots & g_{pn} \end{bmatrix} \in \mathbf{R}^{p \times n}.$$

  `G` can be in sparse format.

- `h`: p-vector that contains the coefficients of equality constraints, *i.e.*,

$$h = \begin{bmatrix} h_1 \\ \vdots \\ h_p \end{bmatrix} \in \mathbf{R}_+^{p}.$$

  All elements h_k must be positive.

- `l`: n-vector that specifies lower bounds on x. If not given, it will be set to the default lower bounds $(10^{-100}, \ldots, 10^{-100})$. All elements l_i must be positive.

- `u`: n-vector that specifies upper bounds on x. If not given, it will be set to the default upper bounds $(10^{100}, \ldots, 10^{100})$. All elements u_i must be positive.

- `quiet`: boolean. Suppresses print messages during execution if `true`. The default value is `false`.

Output arguments

- `x`: n-vector. `x` is the optimal point of the problem if the problem is feasible, and `x` is the last primal iterate of phase I if the problem is infeasible.

- `status`: string; possible values are `'Solved'`, `'Infeasible'` and `'Failed'`.

- `lambda`: vector of length $m + 2n$; the *optimal sensitivity* vector (see [BKVH, §3.3]) associated with inequality constraints if the problem is feasible. The first m elements, `lambda(1:m)`, are optimal sensitivities of the m inequality constraints, the next n elements, `lambda(m+1:m+n)`, are those of the lowerbound constraints ($l \preceq x$), and the last n elements, `lambda(m+n:m+2*n)`, are those of the upperbound constraints ($x \preceq u$). If the problem is feasible, `lambda` is a certificate of infeasibility (see [BKVH, §5.8.1,§11.4.3]).

- `nu`: p-vector; the optimal sensitivity vector (see [BKVH, §3.3]) associated with equality constraints ($Gx + h = 0$) if the problem is feasible. If infeasible, `nu` is a certificate of infeasibility (see [BKVH, §5.8.1,§11.4.3]).

Other calling sequences

Other calling sequences supported by `gpposy` are:

```
>> [x,status,lambda,nu] = gpposy(A,b,szs);
>> [x,status,lambda,nu] = gpposy(A,b,szs,G,h);
>> [x,status,lambda,nu] = gpposy(A,b,szs,G,h,l,u);
>> [x,status,lambda,nu] = gpposy(A,b,szs,G,h,l,u,quiet);
>> [x,status,lambda,nu] = gpposy(A,b,szs,[],[],l,u);
>> [x,status,lambda,nu] = gpposy(A,b,szs,[],[],l,u,quiet);
>> [x,status,lambda,nu] = gpposy(A,b,szs,[],[],[],[],quiet);
```

Caveats

- The equality constraint matrix, `G`, must be full rank.

- If your problem is large and sparse, be sure that `A` and `G` are in sparse format.

- Equality constraints should be explicitly specified as $Gx+h = 0$. You cannot represent an equality constraint as a pair of opposing inequality constraints.

3 Example

Consider the problem

$$\begin{array}{ll} \text{minimize} & x_1^{-1}x_2^{-1/2}x_3^{-1} + 2.3x_1x_3 + 4x_1x_2x_3 \\ \text{subject to} & (1/3)x_1^{-2}x_2^{-2} + (4/3)x_2^{1/2}x_3^{-1} \leq 1, \\ & 0.1x_1 + 0.2x_2 + 0.3x_3 \leq 1, \\ & (1/2)x_1x_2 = 1, \end{array}$$

with variables x_1, x_2 and x_3. This problem has the form (1) with

$$A^{(0)} = \begin{bmatrix} -1 & -0.5 & -1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad A^{(1)} = \begin{bmatrix} -2 & -2 & 0 \\ 0 & 0.5 & -1 \end{bmatrix}, \quad A^{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$b^{(0)} = \begin{bmatrix} 1 \\ 2.3 \\ 4 \end{bmatrix}, \quad b^{(1)} = \begin{bmatrix} 1/3 \\ 4/3 \end{bmatrix}, \quad b^{(2)} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix},$$

$$G = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}, \quad h = 0.5.$$

The Matlab code for solving this problem is as follows:

```
% Matlab script that solves the above problem

>> A0  = [  -1 -0.5   -1 ;...
             1    0    1 ;...
             1    1    1 ];
>> A1  = [  -2   -2    0 ;...
             0  0.5   -1 ];
>> A2  = [   1    0    0 ;...
             0    1    0 ;...
             0    0    1 ];
>> A   = [  A0;  A1;  A2 ];

>> b0  = [   1; 2.3;   4 ];
>> b1  = [ 1/3; 4/3 ];
>> b2  = [ 0.1; 0.2; 0.3 ];
>> b   = [  b0;  b1;  b2 ];

>> G   = [   1    1    0 ];
>> h   = [ 0.5 ];
>> szs = [ size(A0,1); size(A1,1); size(A2,1) ]; %i.e., [ 3; 2; 3 ]

>> [x,status,lambda,nu] = gpposy(A,b,szs,G,h);
```

After executing the code, you can see the result by typing `x` in Matlab.

```
>> x

ans =

    3.4783
    0.5750
    1.1030
```

References

[BKVH] S. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi. A tutorial on geometric programming. To appear in *Optimization and Engineering, 2005*. Available at `www.stanford.edu/~boyd/gp_tutorial.html`.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. Available at `www.stanford.edu/~boyd/cvxbook.html`.