CVXOPT plugin for OpenOffice.org User's Guide

J. Dahl and L. Vandenberghe

October 18, 2008

1 Introduction

The OpenOffice.org plugin described in this document provides a spreadsheet interface to the basic optimization solvers in the Python convex optimization package CVXOPT.

2 License

3 Installation

The plugin is available from http://abel.ee.ucla.edu/cvxopt as a compressed zip-file. It requires OpenOffice.org 2.2.0 or later, Python, and CVXOPT version 1.1 or later[1]. After downloading and unpacking the plugin, open the OpenOffice.org spreadsheet and select Extension Manager from the Tools menu. In the extension manager window, choose Add and select the file cvxopt.uno.zip.

4 A small example

Before describing the optimization functions in more detail, we illustrate the basic usage with a simple linear programming problem,

$$\begin{array}{ll} \text{minimize} & -4x_1 - 5x_2 \\ \text{subject to} & 2x_1 + x_2 \leq 3 \\ & x_1 + 2x_2 \leq 3 \\ & x_1 \geq 0, \quad x_2 \geq 0. \end{array}$$

[1] The installation of CVXOPT must be in a location known to the OpenOffice.org spreadsheet. On a Linux system this corresponds to a regular "system-wide" installation. For other platforms installation may vary.

We assume that the OpenOffice.org spreadsheet has been started with an empty sheet[2].

1. *Enter the coefficients of the objective.* To specify the cost function, we enter the coefficients in adjacent cells. For example, enter `-4` in the cell `B1` and `-5` in the cell `C1`.

2. *Enter the constraints.* To enter the parameters of the constraint

$$2x_1 + x_2 \leq 3$$

   enter the values `2`, `1`, `3` in the cells `B3`, `C3` and `E3`, respectively, and enter the inequality string `<=` in cell `D3`. In a similar way, enter the coefficients of other three inequalities below the first one, *i.e.*, enter the values `1`, `2`, `3`, `1`, `0`, `0`, `0`, `1`, `0` in the cells `B4`, `C4`, `E4`, `B5`, `C5`, `E5`, `B6`, `C6`, and `E6`, respectively, and the inequality strings `<=`, `>=`, `>=` in cells `D4`, `D5`, `D6`.

3. *Solve the linear program.*
   Type

```
= CLP(B1:C1;B2:C2;B3:E6;A3:A6)
```

   in cell `B9`. The solver returns the optimal value `-9` in cell `B9`, the primal solution in cells `B2` and `C2` and the dual solution in cells `A3`–`A6`.

Apart from some cosmetic formatting, the resulting spreadsheet should look like figure 1. In the following sections we give a detailed description of the solver functions.

5 Cone linear programming

A *cone (linear) program* is an optimization problem of the form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Gx \preceq h \\ & Ax = b. \end{array}$$

The inequality is a generalized inequality with respect to a proper convex cone. The `CVXOPT` plugin accepts a combination of linear inequalities, second order cone inequalities, and linear matrix inequalities; see chapter 8 of the `CVXOPT` user's guide.

The format of the cone programming solver is

```
CLP(c; x; constraints; dvar)
```

where `c` is a cell-range with the location of the coefficients c of the objective, `x` is a cell-range in which the solution x will be returned, `constraints` is a cell-range with the location of the constraint parameters (*i.e.*, both the equality constraints $Ax = b$ and the inequality constraints $Gx \preceq h$), and `dvar` is a cell-range in which the dual variables will be returned.

If the problem is successfully solved the function returns the optimal value $c^T x$, and the (approximately) optimal primal and dual solutions. If the problem is found to be primal infeasible

[2] On Ubuntu 7.10, the Linux distribution used for developing and testing the plugin, the OpenOffice.org spreadsheet must be started from the `Applications` menu. For unknown reasons, the plugin will not work correctly if the spreadsheet is started from a terminal. Similar behaviour may be experienced on other platforms.

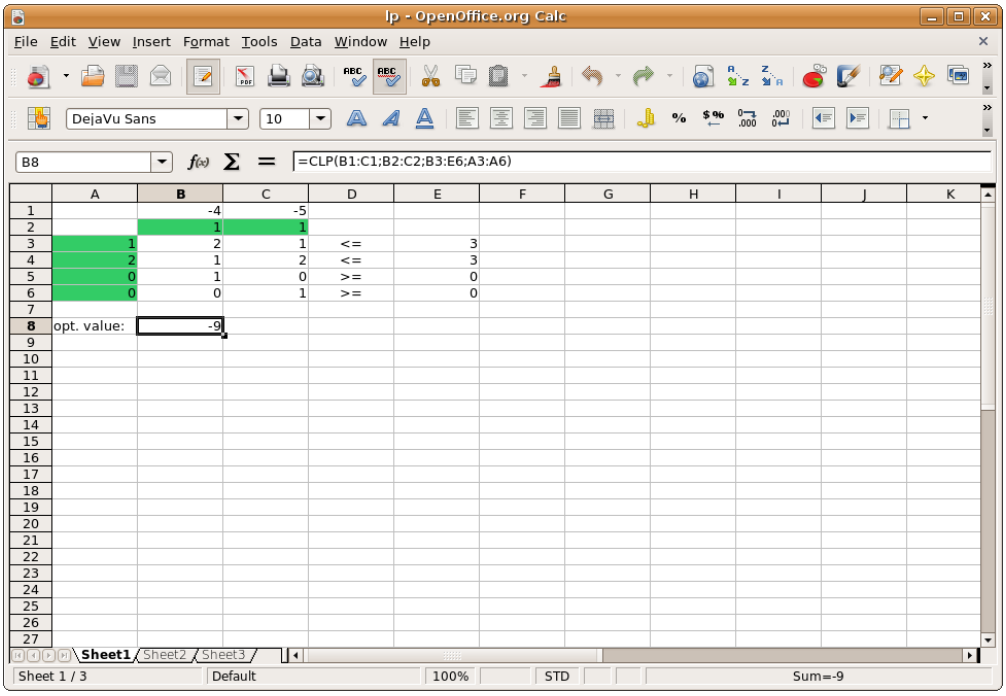


Figure 1: Screenshot of the linear programming example.

the function returns a status string `primal infeasible` and a certificate of primal infeasiblity in the location of the dual variables. If the problem is determined to be dual infeasible, the function returns a status string `dual infeasible` and a certificate of dual infeasibility in the location of the primal variables. If the solver is unable to determine the problem status, it returns a status string `unknown`.

5.1 Scalar linear inequalities

The simplest example of linear cone programs are problems with only scalar linear inequalities. As an example, let us return to the linear programming problem of §4:

$$\begin{array}{ll} \text{minimize} & -4x_1 - 5x_2 \\ \text{subject to} & 2x_1 + x_2 \leq 3 \\ & x_1 + 2x_2 \leq 3 \\ & x_1 \geq 0, \quad x_2 \geq 0. \end{array}$$

Figure 1 shows a screenshot of the spreadsheet for solving the linear program. The cost vector `c` is specified in cells `B1:C1`, the solution vector `x` is returned in `B2:C2`, the constraints are specified in `B3:E6`, and the dual variables are returned in `A3:A6`. The linear inequality constraints can be specified using either `<=`, `<`, `>=`, or `>`.

5.2 Second-order cone inequalities

A second-order cone inequality is specified by placing a `<q` or `<=q` specifier in the cell between the first row of the coefficient matrix and the first component of the righthand side. As an example,

we solve the second-order cone program from §8.5 of the `CVXOPT` documentation,

$$\begin{array}{ll} \text{minimize} & -2x_1 + x_2 + 5x_3 \\ \text{subject to} & \left\| \begin{bmatrix} -13x_1 + 3x_2 + 5x_3 - 3 \\ -12x_1 + 12x_2 - 6x_3 - 2 \end{bmatrix} \right\|_2 \leq -12x_1 - 6x_2 + 5x_3 - 12 \\ & \left\| \begin{bmatrix} -3x_1 + 6x_2 + 2x_3 \\ x_1 + 9x_2 + 2x_3 + 3 \\ -x_1 - 19x_2 + 3x_3 - 42 \end{bmatrix} \right\|_2 \leq -3x_1 + 6x_2 - 10x_3 + 27. \end{array}$$

Figure 2 shows a screenshot of the spreadsheet for the second-order cone programming example.

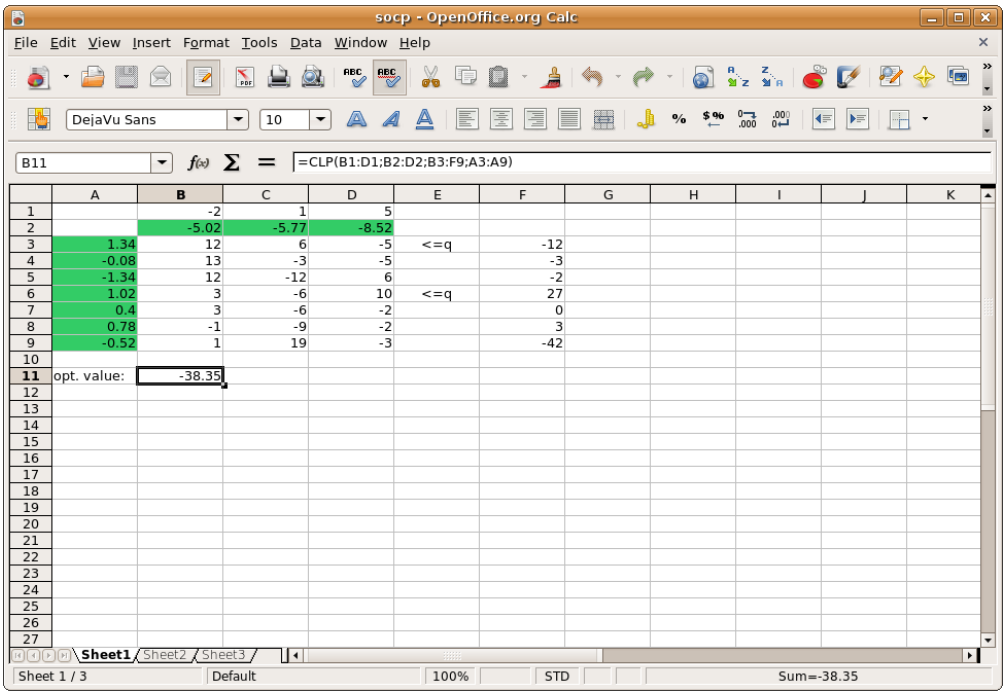


Figure 2: Screenshot of the second-order cone programming example.

The cost vector `c` is specified in `B1:D1`. and the second-order conic inequalities in `B3:F9`. The primal solution `x` is returned in `B2:D2`, and the dual solution in `A3:A9`.

5.3 Linear matrix inequalities

A linear matrix inequality is specified by placing a `<s` or `<=s` specifier in the cell between the first row of the coefficient matrix and the first component of the righthand side. As an example we solve the semidefinite program in §8.6 of the `CVXOPT` documentation,

$$\begin{array}{ll} \text{minimize} & x_1 - x_2 + x_3 \\ \text{subject to} & x_1 \begin{bmatrix} -7 & -11 \\ -11 & 3 \end{bmatrix} + x_2 \begin{bmatrix} 7 & -18 \\ -18 & 8 \end{bmatrix} + x_3 \begin{bmatrix} -2 & -8 \\ -8 & 1 \end{bmatrix} \preceq \begin{bmatrix} 33 & -9 \\ -9 & 26 \end{bmatrix} \\ & x_1 \begin{bmatrix} -21 & -11 & 0 \\ -11 & 10 & 8 \\ 0 & 8 & 5 \end{bmatrix} + x_2 \begin{bmatrix} 0 & 10 & 16 \\ 10 & -10 & -10 \\ 16 & -10 & 3 \end{bmatrix} + x_3 \begin{bmatrix} -5 & 2 & -17 \\ 2 & -6 & -7 \\ -17 & 8 & 6 \end{bmatrix} \preceq \begin{bmatrix} 14 & 9 & 40 \\ 9 & 91 & 10 \\ 40 & 10 & 15 \end{bmatrix}. \end{array}$$

Figure 3 shows the spreadsheet. The cost vector `c` is specified in `B1:D1` and the linear matrix

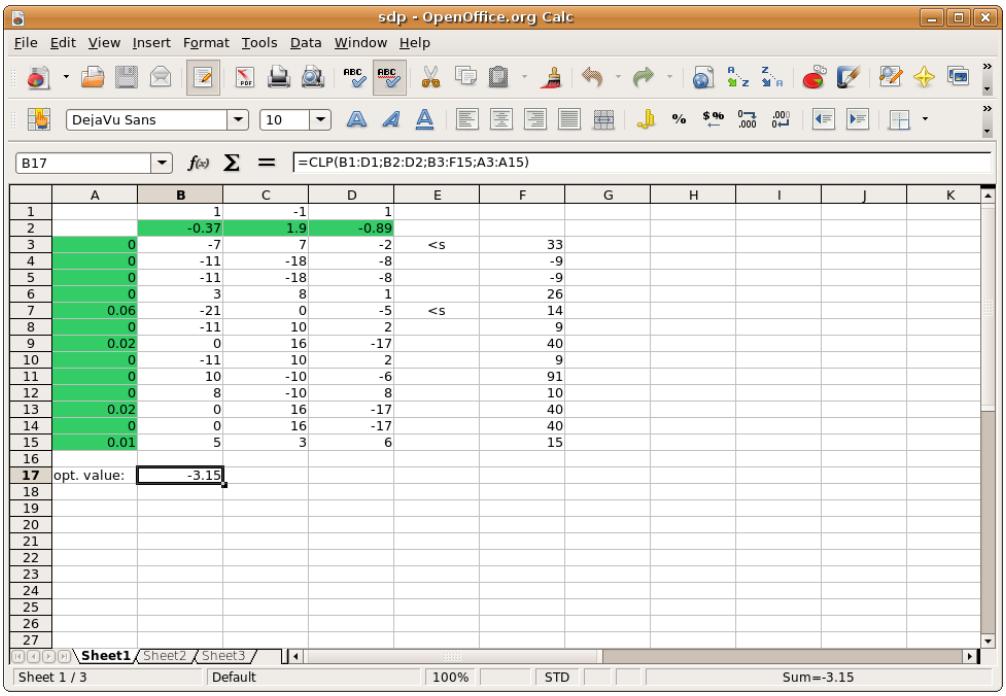


Figure 3: Screenshot of the semidefinite programming example.

inequalities in `B3:F15`. The symmetric matrix coefficients are entered in column major order. The righthand sides are also entered in column major order. The solution vector `x` is returned in `B2:D2`, and the dual variables in `A3:A15`.

5.4 Linear equalities

We can also include linear equality constraints, which are specified similarly using either `=` or `==`. Note, that to enter `==` into the spreadsheet, it is required to add a leading `'` character (*i.e.*, type `'==`) to differentiate the string from the logic comparison operator.

5.4.1 General example

More generally, we can have a combination of the three types of inequalities described above (*i.e.*, a combination of scalar linear inequalities, second-order cone inequalities, and linear matrix inequalities) and linear equality constraints. The different inequality and equality constraints can be entered in arbitrary order.

As an example we solve the problem

$$
\begin{array}{ll}
\text{minimize} & -6x_1 - 4x_2 - 5x_3 \\
\text{subject to} & 16x_1 - 14x_2 + 5x_3 \leq -3 \\
& 7x_1 + 2x_2 \leq 5 \\
& \left((8x_1 + 13x_2 - 12x_3 - 2)^2 + (-8x_1 + 18x_2 + 6x_3 - 14)^2 + (x_1 - 3x_2 - 17x_3 - 13)^2\right)^{1/2} \\
& \qquad \leq -24x_1 - 7x_2 + 15x_3 + 12 \\
& (x_1^2 + x_2^2 + x_3^2)^{1/2} \leq 10 \\
& \begin{bmatrix} 7x_1 + 3x_2 + 9x_3 & -5x_1 + 13x_2 + 6x_3 & x_1 - 6x_2 - 6x_3 \\ -5x_1 + 13x_2 + 6x_3 & x_1 + 12x_2 - 7x_3 & -7x_1 - 10x_2 - 7x_3 \\ x_1 - 6x_2 - 6x_3 & -7x_1 - 10x_2 - 7x_3 & -4x_1 - 28x_2 - 11x_3 \end{bmatrix} \preceq \begin{bmatrix} 68 & -30 & -19 \\ -30 & 99 & 23 \\ -19 & 23 & 10 \end{bmatrix}.
\end{array}
$$

Figure 4 shows a screenshot of the spreadsheet.

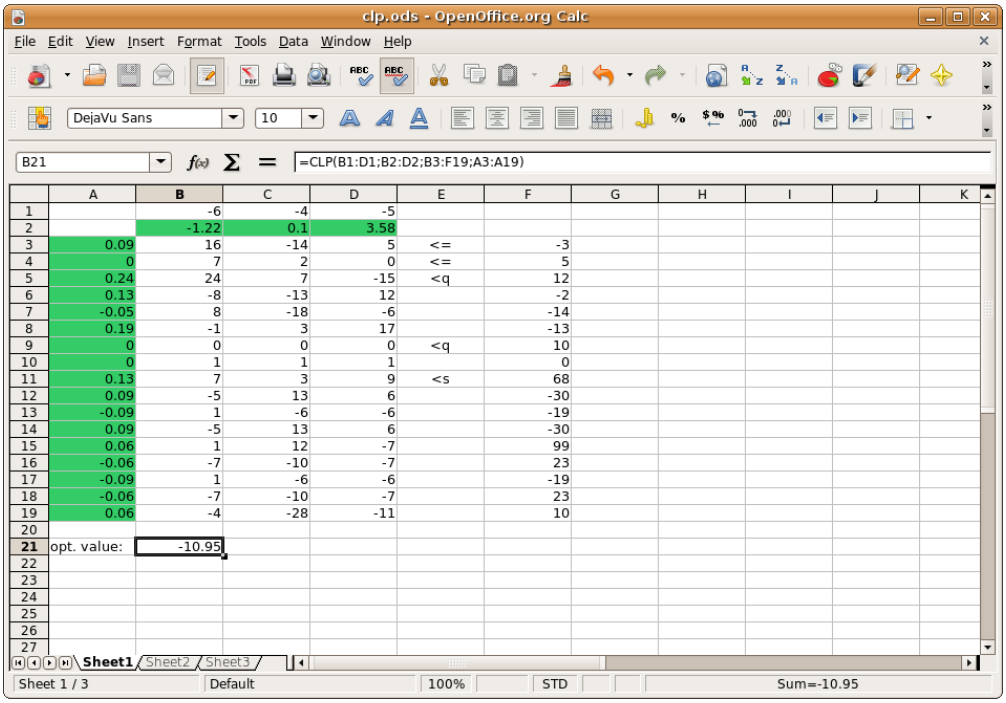


Figure 4: Screenshot of the general conic programming example.

6 Cone quadratic programming

A *cone (quadratic) program* is an optimization problem of the form

$$
\begin{array}{ll}
\text{minimize} & (1/2)x^T P x + q^T x \\
\text{subject to} & Gx \preceq h \\
& Ax = b
\end{array}
$$

where P is a symmetric positive semidefinite matrix, and the inequality denotes a generalized inequality as in §5. From the spreadsheet, it is solved using the `CQP` routine,

```
CQP(P; q; x; constraints; dvar)
```

where `P` is a cell-range specifying the quadratic term P of the objective function, `q` is a cell-range specifying the linear term q of the objective function, `x` is a cell-range in which the solution vector x is returned, `constraints` is a cell-range specifying the constraints (generalized inequalities $Gx \preceq h$ and equalities $Ax = b$), and `dvar` is a cell-range in which the dual optimal dual variables are returned. The function either returns the optimal solution $(1/2)x^T Px + q^T x$ or a status string `unknown`.

As a simple example we solve a regular quadratic programming problem (*i.e.*, one including only linear constraints),

$$\begin{array}{ll} \text{minimize} & x_1^2 + x_2^2 - 4x_1 - 5x_2 \\ \text{subject to} & 2x_1 + x_2 \leq 3 \\ & x_1 + 2x_2 \leq 3 \\ & x_1 \geq 0, \quad x_2 \geq 0 \\ & x_1 + x_2 = 1 \end{array}$$

using the spreadsheet in figure 5. The matrix `P` is specified in `B3:C4`, the vector `q` in `B1:D1`, and

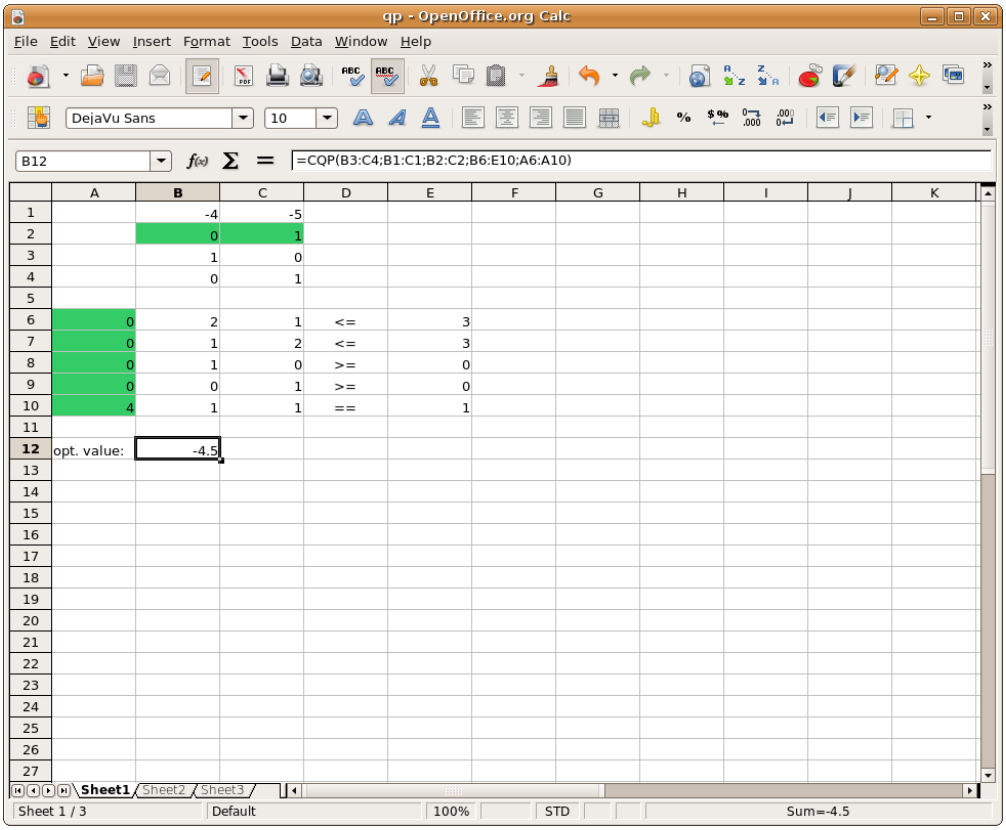


Figure 5: Screenshot of the quadratic programming example.

the constraints in `B6:E10`. The solution vector `x` is returned in `B2:D2`, and the dual variables in `A6:A10`. For a more advanced examples including nonlinear inequalities, see §8.

7 Geometric programming

The CVXOPT plugin also accepts geometric programs in posynomial form,

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq \alpha_i, \quad i = 1, \ldots, m \\ & h_i(x) = \beta_i, \quad i = 1, \ldots, p. \end{array}$$

The functions $f_i(x)$ are posynomial functions,

$$f_i(x) = \sum_{k=1}^{K_i} c_k^{(i)} x_1^{a_{k1}^{(i)}} x_2^{a_{k2}^{(i)}} \cdots x_n^{a_{kn}^{(i)}},$$

with $c_k^{(i)} > 0$. The functions $h_i(x)$ are monomial functions,

$$h_i(x) = d_i x_1^{b_{i1}} x_2^{b_{i2}} \cdots x_n^{b_{in}},$$

with $d_i > 0$. The righthand sides α_i and β_i are positive.

The geometric programming solver is called using the `GP` function

```
GP(obj; x; F; dvar)
```

where `obj` is a cell-range with the data for the objective function, `x` is a cell-range in which the solution vector x is returned, `F` is a cell-range with the constraint parameters, and `dvar` is a cell-range in which dual optimal solution is returned. Posynomial inequalities are specified using the `<` or `<=` strings, and monomial equalities are specified using the `=` or `==` strings. The function either returns the optimal value $f_0(x)$ or a status string `unknown`. The coefficients of the posynomials f_i are entered as matrices with $n+1$ columns and as many rows as there are terms in the posynomial. The first column contains the coefficients $c_k^{(i)}$, the second column the coefficients $a_{k1}^{(i)}$, the third column the coefficients $a_{k2}^{(i)}$, et cetera.

The dual variables are the dual multipliers for the equivalent convex problem

$$\begin{array}{ll} \text{minimize} & \log f_0(e^{y_1}, \ldots, e^{y_n}) \\ \text{subject to} & \log f_i(e^{y_1}, \ldots, e^{y_n}) \leq \log \alpha_i, \quad i = 1, \ldots, m \\ & \log h_i(e^{y_1}, \ldots, e^{y_n}) = \log \beta_i, \quad i = 1, \ldots, p. \end{array}$$

As an example we solve the geometric program in §9.3 of the `CVXOPT` documentation,

$$\begin{array}{ll} \text{minimize} & w^{-1}h^{-1}d^{-1} \\ \text{subject to} & 2hw + 2hd \leq 100 \\ & wd \leq 1000 \\ & 0.5wh^{-1} \leq 1 \\ & hw^{-1} \leq 2 \\ & 0.5wd^{-1} \leq 1 \\ & dw^{-1} \leq 2 \end{array}$$

with variables h, w, d (see figure 6). The objective function is specified in `B1:E1` and the constraints in `B3:G9`. The solution vector `x` is returned in `C2:E2`, and the dual optimal solution in `A3:A9`.

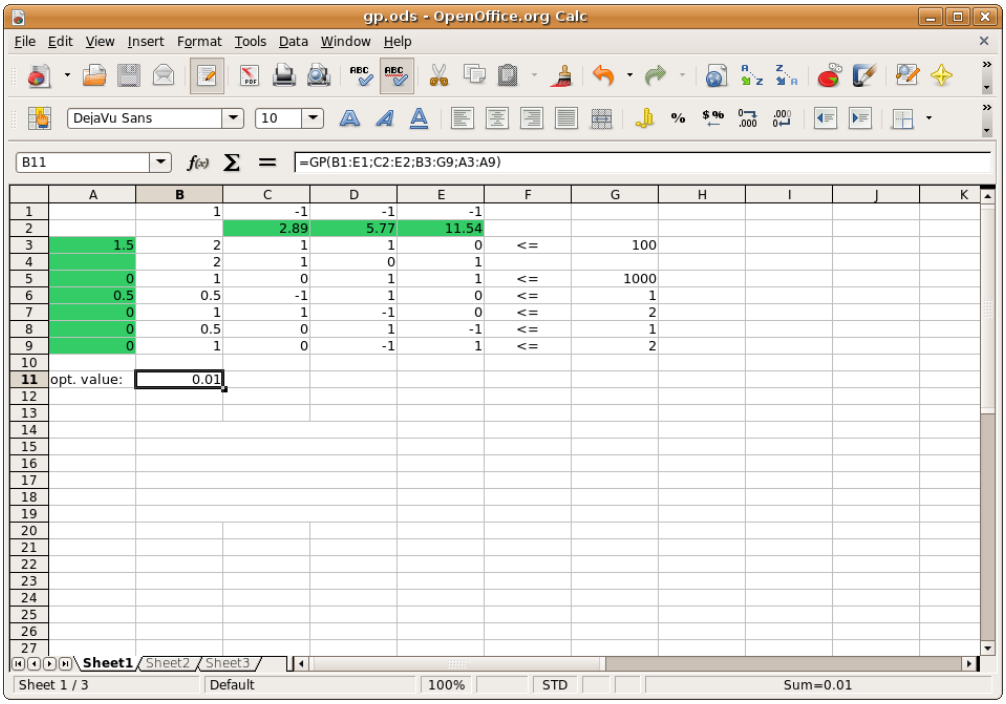


Figure 6: Screenshot of the geometric programming example.

8 Constrained least-squares problems

In this section we illustrate more advanced features of `CVXOPT` and OpenOffice.org spreadsheets, solving a constrained least-squares problem from §8.2 of the `CVXOPT` manual,

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|_2^2 \\ \text{subject to} & x \succeq 0 \\ & \|x\|_2 \leq 1 \end{array}$$

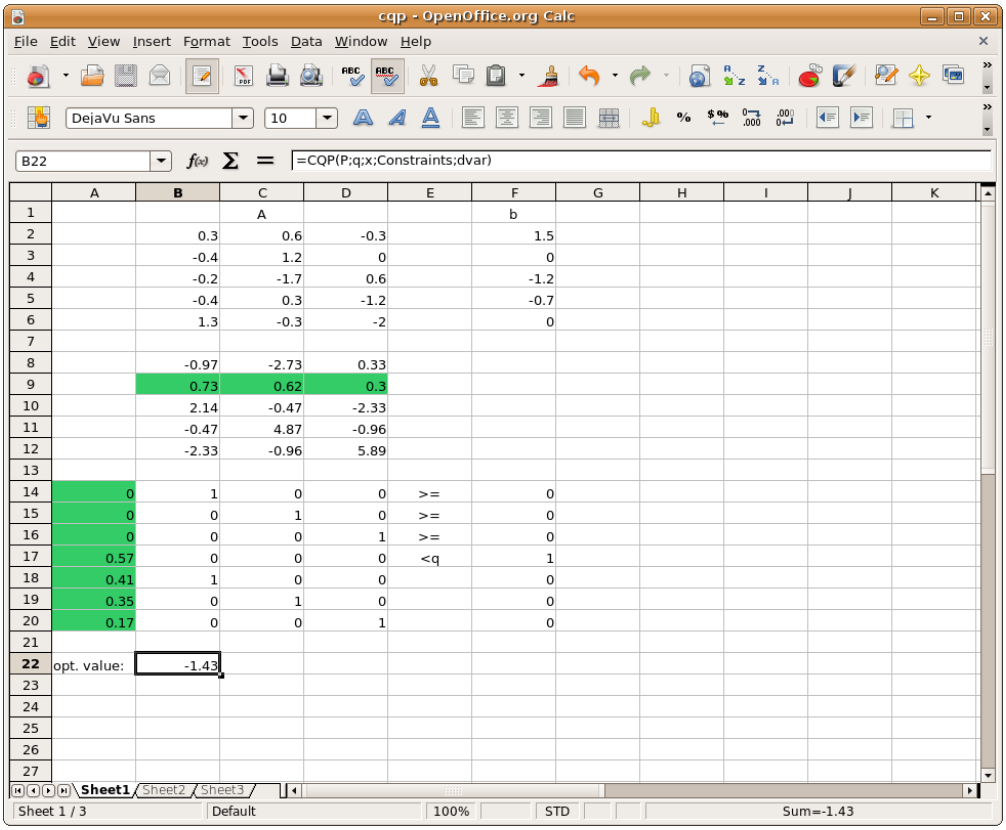
with

$$A = \begin{bmatrix} 0.3 & 0.6 & -0.3 \\ -0.4 & 1.2 & 0.0 \\ -0.2 & -1.7 & 0.6 \\ -0.4 & 0.3 & -1.2 \\ 1.3 & -0.3 & -2.0 \end{bmatrix}, \qquad b = \begin{bmatrix} 1.5 \\ 0.0 \\ -1.2 \\ -0.7 \\ 0.0 \end{bmatrix}.$$

For tutorial value, we summarize below the necessary steps in creating the spreadsheet shown in Fig 7.

1. *Specifying A and b.*
   We first enter the `A` and `b` matrices in cell-ranges `B2:C6` and `F2:F6`, respectively. We then give those two cell-ranges symbolic names `A` and `b` using `Insert->Names->Define` (or `CTRL-F3`).

2. *Calculating P and q.*
   We next compute $P = A^T A$ by writing `=MMULT(TRANSPOSE(A);A)` followed by `CTRL-SHIFT-ENTER`



	A	B	C	D	E	F	G	H	I	J	K
1			A			b					
2		0.3	0.6	-0.3		1.5					
3		-0.4	1.2	0		0					
4		-0.2	-1.7	0.6		-1.2					
5		-0.4	0.3	-1.2		-0.7					
6		1.3	-0.3	-2		0					
7											
8		-0.97	-2.73	0.33							
9		0.73	0.62	0.3							
10		2.14	-0.47	-2.33							
11		-0.47	4.87	-0.96							
12		-2.33	-0.96	5.89							
13											
14	0	1	0	0	>=	0					
15	0	0	1	0	>=	0					
16	0	0	0	1	>=	0					
17	0.57	0	0	0	<q	1					
18	0.41	1	0	0		0					
19	0.35	0	1	0		0					
20	0.17	0	0	1		0					
21											
22	opt. value:	-1.43									
23											
24											
25											
26											
27											

Figure 7: Screenshot of the cone quadratic programming example.

into cell `B10`. Similarly, we compute $q = -A^T b$ by writing `=-TRANSPOSE(MMULT(TRANSPOSE(A);b)` followed by `CTRL-SHIFT-ENTER` into cell `B8`. We then assign the symbolic names `P` and `q` to the two computed matrices.

3. *Specifying solution variables.*
   We assign the symbolic names `x` to `B9:D9`, `dvar` to `A14:A20`, and `Constraint` to `B14:F20`.

4. *Solving the problem.*
   We solve the problem by writing `=CQP(P;q;x;Constraints;dvar)` into cell `B22`. The resulting spreadsheet should look similar to Fig. 7 apart form cosmetic changes.