Linear Programming and Game Theory Library for Xcas and the HP Prime

Nikolaus Henderson (ftneek)

2023 - 2024-12-06

Overview

This library contains a linear program solver (**simplex()**) capable of solving mixed constraint problems with integer variables, binary variables, and unrestricted variables through the use of the two-phase Simplex, Dual Simplex, and Gomory Plane Cutting algorithms, as well as game theory commands capable of solving two-person zero-sum games (**solveGame()**).

Contents

1 Installation and Verification

1. Download the attached zip file.
2. - To use in Xcas [1]: Click File >Open >File and select the simplex.xws file. You may need to click 'OK' in the 3 program editor cells.
   - To use on an HP Prime: Use the Connectivity Kit to transfer the three .hpprgm files (in the hpprgm folder) to the HP Prime.
3. Verify everything is working correctly by running the **test_simplex()** command.

test_simplex()

Solves a set of linear programming problems and returns a list of 1's or 0's (true or false) depending on whether or not the corresponding test's output matched the expected result.

Example:

```
test_simplex()
```

$$[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

Note: 1 test (for the game theory commands) failed in an Xcas web session, but everything works in Xcas and on the HP Prime.

2 Linear Programming

simplex(a, [dir], [integers], [binary], [unrestricted])

Solves a linear program by using the Simplex Algorithm or Gomory's Plane Cutting Algorithm. Accepts 1-5 arguments:

- **a**: The linear program as an augmented matrix of the form $\left[\begin{array}{c|c} A & b \\ \hline c & -z_0 \end{array}\right]$, where A is the constraint matrix, b is the right hand side of the constraints as a column, c is the objective function row, and z_0 is the constant coefficient of the objective function. Any $=$ constraints should be the first rows of the matrix. Any $\leq$ constraints should be the next rows. Any $\geq$ constraints should be the last rows. The objective function is always the final row, with z_0 negated. This means that you should put the constraints in this order to create the augmented matrix: $=, \leq, \geq,$ objective function.
- **dir**: A list of 2 items; the number of $=$ constraints and the number of $\geq$ constraints. Uses maximization if the first value is positive, and minimization if it is negative. If there are no $=$ constraints, you can use $\pm$inf for min or max. If there are no $\geq$ constraints, you can omit the list delimiters and provide only the first value.
- **integers**: a list of integer variable indices.
- **binary**: a list of binary variable indices.

- **unrestricted**: a list of variable indices without nonnegative restriction.

Returns [z, m, bv, P, X]. z is the optimal value, m is the final matrix tableau, bv is the list of final basic variable indices, P is the tally of pivot1 operations, X is a matrix whose columns are the vertices of the basic feasible solution.

Example: $\min 2x+5y$ subject to $3x-y=1, x-y\leq 5$, where x, y are nonnegative and integer.

```
a:=[[3,-1,1],[1,-1,5],[2,5,0]];
dir:=[-1,0]; // (or we can use dir:=-1 as a shortcut)
integers:=[1,2];
simplex(a,dir,integers)
```

$$[12, \begin{bmatrix} 1 & 0 & 0 & -1 & 1 & 1 \\ 0 & 0 & 1 & -2 & 1 & 6 \\ 0 & 1 & 0 & -3 & 2 & 2 \\ 0 & 0 & 0 & 17 & -12 & -12 \end{bmatrix}, [1,3,2], 2, \begin{bmatrix} 1 \\ 2 \\ 6 \\ 0 \\ 0 \end{bmatrix}].$$

Therefore, the minimum value of 12 occurs at $x = 1$ and $y = 2$.

Notes: **simplex()** uses default settings of minimizing the objective function and all constraints are $\leq$ unless specified by **dir**, therefore, you can omit the **dir** arguement for problems aligning with the default settings. You can transform constraints from $\geq$ to $\leq$ and vice versa by multiplying the constraint by -1 to change the problem's form and still arrive at the same solution. The indices stored in **integers**, **binary**, and **unrestricted** start from 1 (variable labeling starts from x_1 instead of x_0). If a variable is **binary**, it is not necessary to indicate it as **integer** (this is done automatically). Currently, using one of the optional arguments requires you to provide all arguments that come before. For example, to enter **unrestricted** variables, you should provide values for **dir**, **integers** and **binary** (even if it is the default value or an empty list). In addition, using **unrestricted** variables currently requires an additional manual step after the final iteration is returned to obtain the final vertex.

simplex_core(a, bv, art, ign, P)

Solves a linear program in canonical form by using the Simplex Algorithm. Accepts 5 arguments:

- **a**: a matrix contains a linear program in canonical form.
- **bv**: a list of basic variable indices.
- **art**: the number of (new or unused) = constraints in the program.
- **ign**: the number of (old or used) = constraints in the linear program.
- **P**: the tally of pivot1 operations used so far.

Returns [z, m, bv, P, X]. z is the optimal value, m is the final matrix tableau, bv is the list of final basic variable indices, P is the updated tally of pivot1 operations, X is a matrix whose columns are the vertices where the optimal value occurs.

simplex_int(a, bv, art, ign, P, integers)

Solves an (integer) linear program in canonical form by using Gomory's Plane Cutting Algorithm. Returns the same format as simpelx_core(). Accepts 6 arguments (see simplex_core() for 1-5):

- **integers**: a list of integer variable indices.

Note: **simplex_core()** and **simplex_int()** are used internally to perform the simplex and cutting plane algoritms. Since they are more complicated to set up, it is recommended to solve linear programs with the **simplex()** command.

basis_to_id(Basis, n)

Maps a basis to an ID. Accepts 2 arguments:

- **Basis**: a list of basic variable indices.
- **n**: the total number of variables in the system.

Example:

```
basis_to_id([3,4,5],5)
```

$$9$$

Therefore, 9 represents the basis $[x_3, x_4, x_5]$ in a system with 5 variables.

id_to_basis(ID, n, m)

Returns the basis mapped to the given ID. Accepts 3 arguments:

- **ID**: an integer representing a unique basis.
- **n**: the total number of variables in the system.
- **m**: the number of constraints in the system (number of variables in the target basis).

Example:

```
id_to_basis(9,5,3)
```

$$[3, 4, 5]$$

For a system with 5 variables and 3 constraints, the basis corresponding to an ID of 9 is $[x_3, x_4, x_5]$.

3 Game Theory

solveGame(p)

Solves a two-person zero sum game by incorporating multiple strategies including pure strategies, two-by-two matrix shortcut, dominant reduction, and Simplex Algorithm. Accepts 1 argument:

- **p**: a payoff matrix for a two-person zero sum game.

Returns $[v, X, Y]$.
v is value of the game.
A column of X is a strategy (x) for Player 1, and a column of Y is a strategy (y) for Player 2. For a given set of strategies, x and y, x_i and y_j are the respective probabilities that, for every play of the game, Player 1 should play s_i and Player 2 should play t_j.

Example 9.6.1 [2]:

```
solveGame([[0,1,-2],[-1,0,1],[2,-1,0]])
```

$$[0, \begin{bmatrix} \frac{1}{4} \\ \frac{1}{2} \\ \frac{1}{4} \end{bmatrix}, \begin{bmatrix} \frac{1}{4} \\ \frac{1}{2} \\ \frac{1}{4} \end{bmatrix}].$$

Therefore, the value of the game is 0, meaning neither player is expected to win in the long term (as the number of games approaches infinity). Player 1 and Player 2 should extend 1 finger with probability $\frac{1}{4}$, 2 fingers with probability $\frac{1}{2}$, or 3 fingers with probability $\frac{1}{4}$.

Notes: s_i are actions that can be taken by Player 1, t_j are actions that can be taken by Player 2. Each set of (s_i, t_j) is a strategy pair. As the number of games approaches infinity, the average payoff per game for Player 1 converges to (v) the value of the game (assuming both players always play optimally). Therefore, a positive value of the game indicates, in the long-term average, Player 1 wins v per game (Player 2 loses v per game), while a negative value of the game means, in the long term average, Player 1 loses v per game (Player 2 wins v per game). If the value of the game is 0, neither player is expected to come out ahead in the long run.

pureCheck(p)

Checks a payoff matrix for pure strategies. Accepts 1 argument:

- **p**: a payoff matrix for a two-person zero sum game.

Returns $[v, X, Y]$ for pure strategies or $[u_1, u_2]$ for no pure strategies. v is value of the game. A column of X or Y is a pure strategy for Player 1 or Player 2, respectively. u_1 is the security level for Player 1, u_2 is the security level for Player 2. One way to check for pure strategies by doing:

```
r:=pureCheck(p); if dim(r(2)) != 1 then // pure strategies exist
```

Example 9.3.1a [2]:

```
pureCheck
   ([[10,5,5,20,3],[10,15,10,17,25],[7,12,8,9,8],[5,12,9,10,5]])
```

$$[10, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}]$$

Indicates the value of the game is 10, with pure strategies at (s_2, t_1) and (s_2, t_3). This means that Player 1 should always play s_2, while Player 2 should always play t_1 or t_3.

Example 9.3.1b [2]:

```
pureCheck([[1,3],[4,2]])
```

$$[2, 3]$$

Indicates no pure strategies exist, and we must use mixed strategies to solve this game.

dominance(p)

Uses dominant strategies to reduce a payoff matrix to dimensions, stopping when the matrix is no longer reducible or when the dimensions are [2,2]. Accepts 1 argument:

- **p**: a payoff matrix for a two-person zero sum game.

Returns list of [p', [indices of deleted rows], [indices of deleted columns]]. If no dominant strategies exist, a p' will be unmodified and the lists will be empty.

Example:

```
dominance([[0,-2,-1,0],[3,5,6,-1],[5,-1,-3,-2]])
```

$$[\begin{bmatrix} -2 & 0 \\ 5 & -1 \end{bmatrix}, [3], [1, 3]]$$

Indicating row 3 and columns 1 and 3 have been removed by dominant strategies.

twobytwo(p)

Solves a two-person zero-sum game (with dimensions of [2,2]) by 2x2 shortcut method. Assumes no pure strategies. Accepts 1 argument:

- **p**: a payoff matrix (with dimensions of [2,2]) for a two-person zero-sum game.

Example:

```
twobytwo([[1,3],[4,0]])
```

$$[2, \begin{bmatrix} \frac{2}{3} \\ \frac{1}{3} \end{bmatrix}, \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}]$$

simplex_game(p)

Solves a two-person zero sum game by Simplex Algorithm. Assumes no possible pure strategies (because only 1 strategy is returned per player when solving by simplex). Accepts 1 argument:

- **p**: a payoff matrix for a two-person zero sum game.

Returns $[v, x, y]$. v is value of the game. x_i and y_j is the probability Player 1 and Player 2 should play s_i and t_j, respectively.

Example: see **solveGame() Example 9.6.1**

verifySecurityLevels(p, X, Y)

Computes bounds for the maximum security level of Player 1 (v_1) when given X, and the minimum security level of Player 2 (v_2) when Y. Accepts 3 arguments:

- p: a payoff matrix for a two-person zero sum game.
- X: a matrix where columns (X_i) are potential mixed strategies for Player 1.
- Y: a matrix where columns (Y_i) are potential mixed strategies for Player 2.

Returns $[v_1, v_2]$, where v_1 will be an empty list if X is an empty list, or v_2 will be an empty list if Y is an empty list.

Note: For Player 1, the security level represents the minimum average amount they can expect to gain by playing strategy X_i. For Player 2, it is the maximum average amount they should expect to lose when playing strategy Y_j. Player 1 wishes to maximize their security level, while Player 2 wishes to minimize theirs. If Y is an empty list, computes only bounds for v_1. If X is an empty list, computes only bounds for v_2. When provided both X and Y, computes both v_1 and v_2. X and Y can each be given multiple strategies (columns), and the strongest bound for the security level of each player will be returned (maximum for Player 1 and minimum for Player 2). If $v_1 = v_2$, that is the value of the game.

Example 9.4.1 [2]:

```
verifySecurityLevels([[1,3],[4,0]],[[1/2],[1/2]],[])
```

$$[\frac{3}{2},[]]$$

Therefore, on average, Player 1 can secure a payoff of at least $\frac{3}{2}$ per game by using the mixed strategy $[\frac{1}{2}, \frac{1}{2}]^T$.

Example (Problem Set 9.4, #1 [2]):

```
a:=[[1,2,3,4],[6,5,2,1],[7,0,1,8]];
x:=[[1/3,2/3],[1/3,1/3],[1/3,0]];
y:=[[1/6,0],[0,1/3],[5/6,1/2],[0,1/6]];
verifySecurityLevels(a,x,[])
```

$$[\frac{8}{3},[]]$$

We conclude $v_1 \geq \frac{8}{3}$ (when Player 1 plays X_2).

```
verifySecurityLevels(a,[],y)
```

$$[[],\frac{8}{3}]$$

We conclude $v_2 \leq \frac{8}{3}$ (when Player 2 plays Y_1). Since $v_1 = v_2$, the value of the game is $\frac{8}{3}$.

4 Acknowledgments

Thanks to Albert Chan for helping investigate bugs, as well as suggesting fixes and improvements, and to Bernard Parisse for creating and maintaining Giac/Xcas, as well as maintaining the HP Prime's CAS.

References

[1] Giac/Xcas, Bernard Parisse and Renée De Graeve, version 1.9.0 (2024), https://www-fourier.univ-grenoble-alpes.fr/~parisse/giac.html

[2] An Introduction to Linear Programming and Game Theory 3rd Edition, Paul R. Thie and Gerard E. Keough (ISBN: 978-0470232866)