LPAssumptions: User Manual

Roger Colbeck* and V. Vilasini†
Department of Mathematics, University of York, Heslington, York YO10 5DD.
(Dated: May 20, 2020)

LPAssumptions (https://github.com/rogercolbeck/LPAssumptions) is a Mathematica package for solving linear programming problems where the objective function can contain unknown variables other than those being optimized over. This manual explains what the package does in more detail, and how to use it.

A generic linear programming problem can be specified by giving a vector c, a matrix M and a vector b such that the problem corresponds to

$$\begin{aligned} &\text{minimize } c.x \\ &\text{subject to } Mx \geq b \\ &x \geq 0 \end{aligned}$$

Any linear programming problem can be written in this form and this is the same form as used by Mathematica's LinearProgramming function.

If c, M and b are completely specified (consist of numerical values), LinearProgramming can already be used to solve the problem. The aim of this package is to also be able to cope with the case where there are unspecified constants in vector c representing the objective function. For example, we might wish to optimize $x_1 + ax_2$, where a is an unspecified constant in the range $0 \leq a \leq 1$, returning the solution for all values of a in the range.

Important: At the moment LPAssumptions requires all the unspecified constants to be in the objective function vector, c. Note that if they are instead all in the constraint vector b, they can be moved to c by considering the dual linear program.

To use the package, open the file LPAssumptions.m in Mathematica and press "run all code". Then open a new notebook and the command LPAssumptions should be available for use.

The main function of the package, is LPAssumptions[c, M, b, assump, (Options)]. It takes as input c (a vector corresponds to the objective function), M (the constraint matrix), b (the constraint vector) and assump (any initial assumptions on the unspecified variables involved). The argument assump should be given as a list of assumptions, e.g., assump=$\{0 \leq a \leq 1, t > 3\}$. If assump is omitted, then it is taken to be empty ($\{\}$). With no assumptions and no unspecified variables, provided the problem is feasible and bounded, the answer given by LPAssumptions should match that of LinearProgramming (up to the slightly different structure of the output).

The output takes the form $\{\{\text{constr}_1, \text{vec}_1\}, \{\text{constr}_2, \text{vec}_2\}, \ldots\}$, where constr_i are the set of constraints under which the optimum is achieved by the vector vec_i. The values of the objective function at the optima can then be computed using $c.\text{vec}_i$ (see also the option OutputValues→True below).

Additional options can be specified:

- Iterations (default: 50,000): The maximum number of iterations (each iteration corresponds to performing one pivot) that the function will perform before giving an error that an optimum was not found by then.
- PrintTemp (default: False): Using PrintTemp→True, it prints temporary messages while running.
- OutputValues (default: False): Using OutputValues→True, LPAssumptions outputs a list $\{\{\text{constr}_1, \text{val}_1\}, \{\text{constr}_2, \text{val}_2\}, \ldots\}$, where constr_i are the set of constraints under which the optimal value is val_i. By default, this option is set to False and the function instead outputs for each constraint, the point (as a vector) at which the optimal value is attained in that case, and the optimal value can be obtained by simply taking the dot product of this vector with the objective vector c.

For example running LPAssumptions[c, M, b, assump, {Iterations→1000, PrintTemp→True}] will only run at most 1000 iterations and give temporary messages during the evaluation.

Note also that if the constraints of the problem are not in the specified form the code can also cope with this. If the i^{th} constraint has of the form $M_i x \leq b_i$, then this can be specified by using $\{b_i, -1\}$ as the corresponding

*Electronic address: roger.colbeck@york.ac.uk
†Electronic address: vv577@york.ac.uk

entry of b; if some of the constraints are of the form $M_i x = b_i$, then this can be specified by using $b_i, 0$ and for $M_i x \geq b_i$ we use $\{b_i, 1\}$. For instance, to specify the problem with constraints $x_1 + 2x_2 = 2$ and $2x_1 - x_2 \leq 4$ we have $M = \{\{1, 2\}, \{2, -1\}\}$ and $b = \{\{2, 0\}, \{4, -1\}\}$. Note also that a maximization can be converted to a minimization by replacing c with $-c$.

The algorithm works by using the two phase simplex algorithm [1]. It relies on Mathematica's Simplify[expr,assump] command to decide on how to proceed through the computation, where expr is an inequality. If Mathematica is unable to decide whether the inequality is true or false, the algorithm splits into two cases, one in which it assumes expr is true, and the other in which it assumes it is false. It then carries on adding additional assumptions as necessary until termination (or the number of iterations is exceeded). Problems can arise if expr is true (or false) but Mathematica's Simplify is unable to determine this (see the examples notebook for such a case).

For more details, see the example notebook (`LPAssumptions_examples.nb`) available at http://www-users.york.ac.uk/~rc973/LPAssumptions.html. In the examples, we also show how to compute the local weight (see e.g. [2, 3] of a no-signalling probability distribution where the distribution has unspecified parameters (specifically we consider a noisy PR-box with inefficient detectors, see Section IV D F of [4]). This code was also used in [5].

Appendix: Internal commands run by LPAssumptions

This appendix should help anyone wishing to understand the algorithm in more detail, or to further develop the code. These commands are private within the package, so cannot be directly called. To use them directly, the relevant ones need to be uncommented at the start of the package before the package is run.

StartTab[c, M, b]: Generates the initial tableau for the linear programming problem of minimizing $c.x$ subject to $Mx = b$, $x \geq 0$, where each element of b is nonnegative (any linear programming problem can be brought into this form by adding slack variables to make inequalities equality, and by changing the sign of rows of M and the corresponding element of b if necessary). It outputs the matrix $\begin{pmatrix} -c^T & 0 \\ M & b \end{pmatrix}$ (adding one row and column to M).

SlackenMix[c, M, b]: Converts the problem of minimizing $c.x$ subject to $Mx \geq b$, $x \geq 0$ into the form required by StartTab by adding slack variables i.e., it outputs $\{c', M', b'\}$ corresponding to minimizing $c'.x$ subject to $M'x = b'$, $x \geq 0$ where b' is non-negative. If b is given as a list of pairs, the second entry in each pair indicates whether there is an equality (0), greater than (1) or less than (-1), as in Mathematica's LinearProgramming.

ProcessCols[A, assump, col]: Takes a tableau A and processes the columns corresponding to col to output a new tableau such that the chosen columns have only one non-zero element. The command makes assumptions assump about any variables, e.g., assump= $\{a < 2, t > 3\}$ where a and t are free elements in the first row of A. [It can be better to specify the assumptions as open intervals ($<$ or $>$) rather than closed ones ($\leq$ or $\geq$) and deal with the equalities separately if required.] If col is a number, this acts on the first col columns (this version hasn't been fully tested), while if col is a list of columns, this acts on those, e.g., col=$\{2, 4, 5\}$ processes columns 2, 4 and 5. If no argument corresponding to col is provided, ProcessCols[A, assump] processes a number of columns equal to the number of constraint rows in the tableau (the number of rows in A minus 1).

FixArtificialVariables[A, artificial, assump]: Takes a tableau A, a list of artificial variables, artificial and a set of assumption assump and outputs a new tableau in which the artificial variables are removed from the basis.

FindBasis[A, (Options)]: Takes a tableau A and outputs a list of the positions of the elements of the basis, i.e., the positions of the 1s in columns that are all 0s except for one 1. Options: IgnoreFirst→True doesn't worry about the first entry being zero when checking the column (by default, this is set to False); st→n starts checking on the n^{th} row (default, st→2 since the first row represents the objective function).

NewBasis[basis, row, col]: Given a basis, a row and a column, outputs {leaving, newbasis}, where leaving is the leaving variable and newbasis is the new basis assuming a pivot on that row and column, where the new basis is given in the form of the previous command.

ExtractValues[A, (Options)]: Takes a tableau A and outputs a list of pairs comprising the basis variables (the column representing it) and the values they take at the current feasible point. The only option available is st for specifying the starting row as in FindBasis for example and the default is again st→2.

LPSolve[A, assump, (Options)]: Solves the linear programming problem specified by tableau A under assumptions assump with starting row st (the default is st→ 2, otherwise set it with the option st→n). The default choice of st→ 2 meaning that the second row is the first constraint row to consider pivoting on. Taking st→ 3 can be useful where the second row is an objective function for a related problem, e.g., in phase I. The tableau must be in canonical form (i.e., with an subset of columns equal to the identity matrix or a permutation thereof and with b positive). By default, the command performs at most 50,000 pivots (this can be altered using the Iterations option), and gives an error if the optimum isn't found by then. It also prints any unresolvable expressions (e.g., due to not enough assumptions). For example, a step in the algorithm may depend on whether $a + c < 0$, but this may not be decidable from the constraints. The final output is the tableau after all pivoting is complete. The solution is the element in the first row and last column of this output. A further option is ArtificialVars. Setting ArtificialVars→ $\{10, 11, 12\}$ means that the variables in columns 10, 11 and 12 are artificial. When LPSolve has a choice it will try to move artificial variables out of the basis. Further, the maximum number of iterations to be performed can be set using the Iterations option, the default is 50,000 iterations.

LPSolveAssumptions[A, assump, st, current, (Options)]: Solves the linear programming problem specified by tableau A under assumptions assump with starting row st. The default choice of st is 2 meaning that the second row is the first constraint row to consider pivoting on. Taking st equal to 3 can be useful where the second row is an objective function for a related problem, e.g., in phase I. The option current is used to specify the current extra assumption being resolved, this is needed internally since the function calls itself and can be set to "{}". The command performs at most 50,000 pivots, and gives an error if the optimum isn't found by then. In this case, any unresolvable expressions (e.g., due to not enough assumptions) are resolved into both options. The output is a list $\{\{\text{constr}_1, \text{vec}_1\}, \{\text{constr}_2, \text{vec}_2\}, \ldots\}$, where constr_1 are the set of constraints under which the solution vec_1 is valid etc. Options: Iterations, PrintTemp, OutputValues (all 3 the same as in LPAssumptions) and OutLength which keeps track of the number of variables (dimension of the objective vector c) in the original linear programming problem. This is needed because the algorithm involves adding slack and artificial variables to produce a new problem with the same solution (but more variables) and the information regarding the original number of variables is lost. By default, this is set to the value -1 which denotes that all variables are to be considered in the given tableau A.

PhaseI[A, assump]: Takes a tableau A corresponding to the problem of minimizing $c.x$ subject to $Mx = b$, $x \geq 0$ in the form A= $\begin{pmatrix} -c^T & 0 \\ M & b \end{pmatrix}$ and generates a suitable input to PhaseII which requires an identity matrix or permutation thereof amongst the columns and no artificial variables in the basis. This uses the AddId command that takes A to $\begin{pmatrix} 0 & \mathbf{1} & 0 \\ -c^T & 0 & 0 \\ M & I & b \end{pmatrix}$, where $\mathbf{1}$ is a vector of 1s. It also uses FixArtificialVariables when needed to move artificial variables out of the basis. When solving this, one does not pivot on the second row which is retained and adjusted so as to be used in PhaseII. The Iterations option is available as in LPSolve.

PhaseII[A, assump]: Takes the output tableau of PhaseI, drops the irrelevant rows and columns to generate a starting tableau for the original problem which it solves using LPSolve, outputting the final tableau. The Iterations option is available as in LPSolve.

PhaseIIAssumptions[A, assump]: Takes the output tableau of PhaseI, drops the irrelevant rows and columns to generate a starting tableau for the original problem which it solves using LPSolveAssumptions, outputting the list as for that command. The Iterations, PrintTemp, OutputValues and OutLength options are available as in LPSolveAssumptions.

[1] George B. Dantzig. Origins of the simplex method. In Stephen G. Nash, editor, A History of Scientific Computing, pages 141–151. ACM, New York, NY, USA (1990). https://dl.acm.org/citation.cfm?id=88081

[2] Marek Zukowski et. al. Strengthening the Bell theorem: Conditions to falsify local realism in an experiment (1999). https://arxiv.org/abs/quant-ph/9910058

[3] Thomas Cope and Roger Colbeck. Bell inequalities from no-signaling distributions. Physical Review A **100** 022114 (2019). https://doi.org/10.1103/PhysRevA.100.022114

[4] S. Pirandola, U. L. Andersen, L. Banchi, M. Berta, D. Bunandar, R. Colbeck, D. Englund, T. Gehring, C. Lupo, C. Ottaviani, J. Pereira, M. Razavi, J. S. Shaari, M. Tomamichel, V. C. Usenko, G. Vallone, P. Villoresi and P. Wallden. Advances in Quantum Cryptography (2019). https://arxiv.org/abs/1906.01645

[5] V. Vilasini and Roger Colbeck. Analysing causal structures using Tsallis entropies (2019). https://arxiv.org/abs/1907.02551