Search for optimal λ's: a simple but efficient meta-heuristic

Bart JOURQUIN
Université catholique de Louvain, CORE, B-1348 Louvain-la-Neuve, Belgium
bart.jourquin@uclouvain.be

If a systematic test of all the values of λ for a Box-Cox transform of a single independent variable in the range [-2, 2] with a step of 0.1, i.e., 41 different values, is undoubtedly feasible, the exhaustive combinations of λ's in multivariate cases rapidly becomes a very (too) long computing task because of the combinational nature of the problem. This document proposes a simple, efficient and generic heuristic, useable for a combination of *N* λ's. It is tested and validated for cases with *N* =1, 2 and 3.

1 Description

The basic idea of the proposed heuristic is to explore the neighborhood of a given combination of λ's in *N* dimensions, starting from an initial "valid" combination (i.e., a solution with the expected sign for all the estimators of the independent variables and a minimal significance level decided by the modeler).

A specific "hill climbing" algorithm (Russell and Norvig, 2003) is developed. It is a mathematical optimization technique which belongs to the family of local searches. It is an iterative algorithm that starts from an arbitrary solution, and further tries to find better solutions by making incremental changes to the solution until no further improvement can be obtained. Since only convergence to a local maximum can be guaranteed, repeated alternative starting values are usually tested to locate the global maximum, and hence the maximum Log-Likelihood. This is usually referred to as a "shotgun" or "random-restart" hill climbing meta-heuristic (Christian and Griffiths, 2016).

Generally, once all the solutions around an initial solution explored, the strategy used in the hill-climbing algorithms is to pursue the exploration towards the solution that presents the highest improvement (steepest climb). Such an approach implicitly considers that there exists a continuum between all the "valid" solutions. However, this is not always the case for the problem discussed here (see the plots produced by the R scripts provided in this project). Therefore, the algorithm tests all the solutions in each direction and further explores all the solutions with the expected signs having a higher Log-Likelihood than the current solution, even if all the estimators are not highly ('***') significant. As multiple search paths are explored, the algorithm must "remember" the solutions to (re)start from. Consequently, the algorithm presented here belongs to the family of hill climbing with backtracking heuristics (Witten et al., 2011).

An important drawback of this strategy is that a same solution has a high probability to be encountered along several search paths. In order to avoid time consuming recomputing of already computed solutions, a hash table - a data structure that implements an associative array mapping keys (combinations of λ's) and values (solved model for these λ's) - is used to store all the already computed results. It is checked each time the result of a logit model is needed during the search. The computing of a λ's specific logit model is thus only performed when it is not yet present in the hash table.

Beside the definition and the initialization of some global variables (Pseudo-code 1), the heuristic has two major phases: the identification of an initial combination of λ's to start from and the systematic exploration of its neighborhood until no better solution is found.

```
N                         number of lambas
range                     range to search λ's in (from -range to +range)
step                      interval between 2 successive values of λ
nbDraws                   number of valid initial λ combinations to randomly draw
solutionsToExplore        list of solutions to explore around (empty)
bestSolution              best solution found so far (empty)
```

Pseudo-code 1: Definition of the global variables

To start with (Pseudo-code 2), a set of *nbDraws* combinations of λ's that produce "valid" solutions are randomly drawn, among which the solution with the highest Log-Likelihood is retained as starting solution for further exploration. Our experience shows that 1 random draw of an initial λ is enough when *N* = 1. For *N* = 2 and *N* =3 respectively, 5 and 15 draws appear to be sufficient to obtain efficient results (see the "Validation" section below).

```
n = 0
while (n < nbDraws) {
  lambdas = random draw of a combination of N lambdas in range;
  solution = retrieveOrComputeLogit(lambdas);
  if (isValid(solution)) {
    n = n + 1;
    if (solution$logLik > bestSolution$logLik) {
      bestSolution = solution;
    }
  }
}
initialize solutionsToExplore with bestSolution;
```

Pseudo-code 2: Identify a good initial combination of λ'

The exploration around a combination of λ's can be visualized as in Figure 1. The black dots represent the solution to start with. The other dots are one step backward or forward in each dimension.

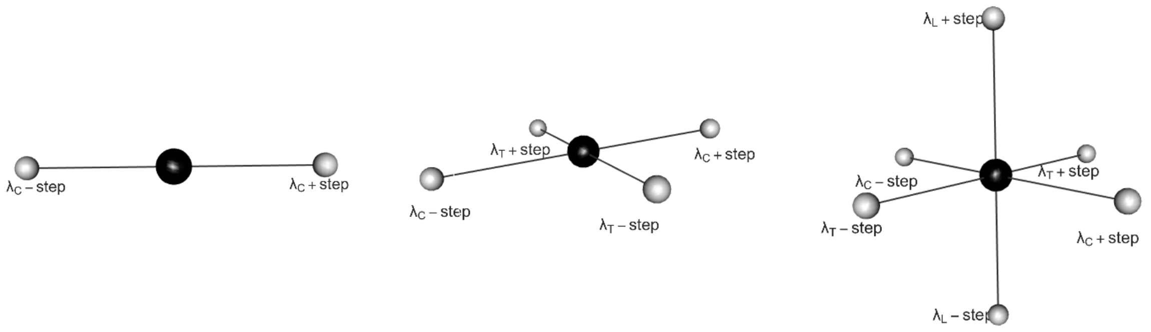
$\lambda_C - \text{step}$
$\lambda_C + \text{step}$
$\lambda_T + \text{step}$
$\lambda_C + \text{step}$
$\lambda_C - \text{step}$
$\lambda_T - \text{step}$
$\lambda_L + \text{step}$
$\lambda_C - \text{step}$
$\lambda_T + \text{step}$
$\lambda_T - \text{step}$
$\lambda_C + \text{step}$
$\lambda_L - \text{step}$

Figure 1: "Explore around" for N= 1, 2 and 3

During the exploration process, corresponding to the second phase of the heuristic, every gray dot corresponding to a solution with the expected signs and having a higher Log-

Likelihood than the solution corresponding to the black dot is added to the list of solutions to explore later. All the encountered “valid” solutions are compared to the best one found so far and replace it if better (Pseudo-code 3). Once all the solutions around the black dot are explored, it is removed from the list. Consequently, this list grows and shrinks dynamically, and the exploration ends when the list is empty, meaning that no better “valid” solution can be found (Pseudo-code 4). The particularity of the hill climbing algorithm presented here is that the exploration isn’t limited towards the steepest direction, but that all the directions in which the slope increases are tested, as long as the signs of the estimators of the solutions are expected.

Pseudo-code 3 can be implemented as a recursive function (called from itself several times, once for each “dimension” of the combination of λ’s).

```
exploreAround(solution, step) {
  for each lambdas around(step) of solution {
    newSolution = retrieveOrComputeLogit(lambdas);

    if (isValid(newSolution) and newSolution$logLik > bestSolution$logLik) {
      bestSolution = newSolution;
    }

    if (hasExpectedSigns(newSolution) and newSolution$logLik > solution$logLik) {
      add newSolution to solutionsToExplore;            }
    }
}
```

Pseudo-code 3: Details of the "exploreAround" function

```
repeat {
  solution = first element of solutionsToExplore;
  exploreAround(solution, step);
  remove solution from solutionsToExplore;
} until solutionsToExplore is empty
```

Pseudo-code 4: Explore the neighborhood until no better solution is found

This strategy can further be optimized (Pseudo-code 5) using successive values for *step*, starting from a coarse value and ending with the final granularity of 0.1. Values of 0.4, 0.2 and 0.1 were used for the case presented in this project. This strategy helps to rapidly converge towards the neighborhood of a good solution using large steps, then exploring the surrounding with gradually smaller steps.

```
for each currentStep in (0.4, 0.2, 0.1) {
  repeat {
    solution = first element of solutionsToExplore;
    exploreAround(solution, currentStep);
    remove solution from solutionsToExplore;
  } until solutionsToExplore is empty

  initialize solutionsToExplore with bestSolution;
}
```

Pseudo-code 5: Explore the neighborhood until no better solution is found (optimized version)

The two phases are repeated several times (shotgun meta-heuristic with 3 attempts in the exercise presented in this document) in order to try to locate the global maximum. Again, the use of the hash table presented before helps to limit the number of logits to compute as, from run to run, already computed solutions can be directly fetched.

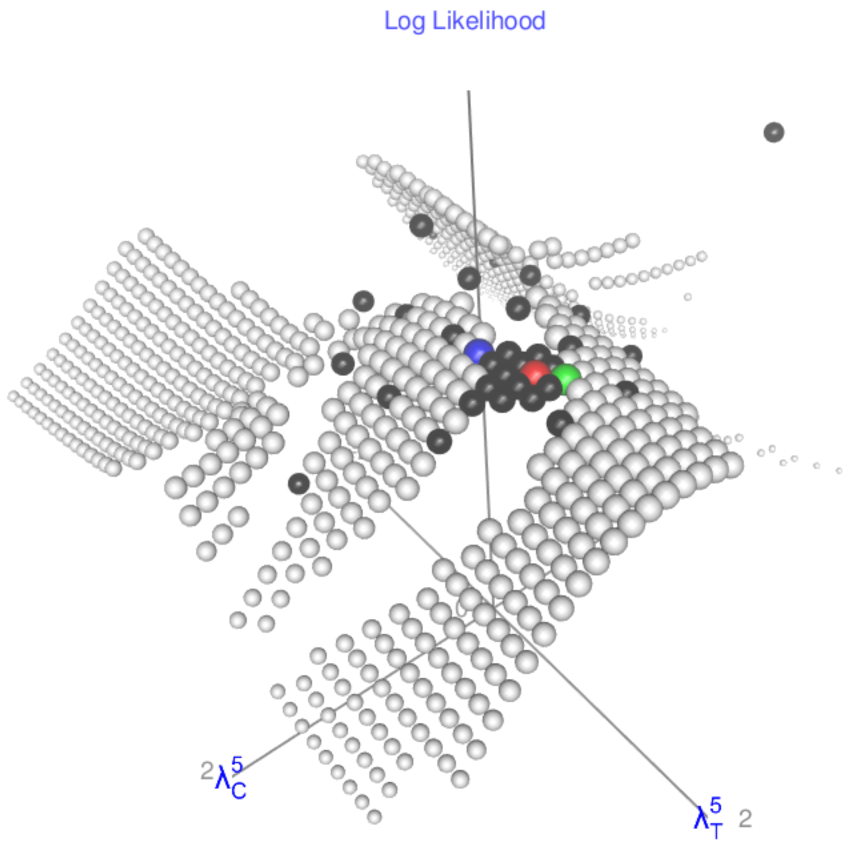

Figure 2: Set of solutions tested by the heuristic (bivariate case, NST-R 5)

Figure 2 gives an idea on how the heuristic finds its path to a solution. The example given here is a bivariate case, because those representing problems with three variables are too cluttered to be presented in this document. This example has been chosen because it doesn't illustrate a "hit" as the solution found by the heuristic (blue dot) doesn't correspond to the exact solution under constraint (green dot). All the black dots represent the solutions tested by the heuristic. Some seem to be in the middle of nowhere; they correspond to random draws of the first phase, that are not "valid" solutions. The density of black dots becomes higher in the neighborhood of the solution. This is a result of the "optimized" approach presented in Pseudo-code 5. In this example, only 34 logit computations were needed (instead of 1 641) to identify a solution (blue dot), which Log-Likelihood is very close to the best solution (green dot). The heuristic didn't converge to the green dot because the red dot (unconstrained max Log-Likelihood) and its surrounding solutions are not "valid" solutions.

1.1 Performances

This heuristic is implemented in an R script and its results are compared to the exact solutions computed using A brute force approach (that tests all the possible combinations of λ's).

As the heuristic starts from a randomly drawn solution, it was run 50 times in order to measure its average and worst-case performances.

The next two tables give some performance indicators for the bivariate and trivariate cases[1]. For each group of commodities, one finds:

- The average number of logit computations needed by the heuristic;
- The largest number of logit computations (worst case) encountered during the 50 runs;
- The number of 'hits' (when the solution of the heuristic corresponds to the exact one);
- The average difference (in %) between the Log-Likelihood of the solution found by the heuristic and the one of the exact solution identified by the brute force approach;
- The largest (worst) difference (in %) between the Log-Likelihood of the solution found by the heuristic and the one of the exact solution.

When the heuristic is applied to the bivariate case (Table 1), it converges after an average of 74 logit computations, i.e., about 4% of what is needed to obtain the exact solution with a brute force approach. Even if the final solution can differ from the exact one (cfr. the number of hits), its Log-Likelihood is always very close (a difference of less than 1%) to the best one, even for the worst cases.

Finally, Table 2 shows that, when applied to the trivariate case, the heuristic finds a solution after an average of 705 logit computations, which represents only 1% of the runs needed to find the exact solution with the brute force algorithm. The heuristic clearly breaks the combinational logic of the problem. Nevertheless, the Log-Likelihoods of the solutions are very close to the best ones, even for the worst cases.

	Logit computations (brute force = 1,681)		Hits	Log-Likelihood (%)	
NST-R	Average	Worst	(50)	Avg Δ with best	Worst case Δ with best
0	80	109	50	0%	0%
1	66	122	38	0%	-0.1%
2	84	127	49	0%	0%
3	72	97	50	0%	0%
4	63	85	11	-0.1%	-0.2%
5	69	96	49	0%	0%
6	67	97	50	0%	0%
7	80	115	35	-0.1%	-0.2%
8	78	132	47	-0.1%	-0.9%
9	77	111	50	0%	0%

Table 1: Performances indicators for the bivariate case (with 5 shots)

[1] While the heuristic can be applied to the univariate case, its usefulness is limited.

	Logit computations (brute force = 68,921)		Hits	Log-Likelihood (%)	
NST-R	Average	Worst	(50)	Avg Δ with best	Worst case Δ with best
0	704	821	50	0%	0%
1	482	575	50	0%	-1%
2	876	1082	1	-0.5%	-1.2%
3	977	1126	50	0%	0%
4	1391	1592	3	-0.3%	-0.8%
5	408	480	50	0%	0%
6	538	611	50	0%	0%
7	556	693	30	0%	0%
8	448	548	48	0%	-0.3%
9	669	795	32	0%	0%

Table 2: Performances indicators for the trivariate case (with 10 shots)

Knowing that on a decent recent computer the computing time of one logit model for this dataset takes about 1 second, 20 hours are needed to solve the brute force trivariate case for one group of commodities. The heuristic converges after an average of 11 minutes and 45 seconds.

2 References

Christian, B., Griffiths, T., 2016. Algorithms to live by: The computer science of human decisions, First U.S. Edition. ed. Henry Holt and Company, New York.

Russell, S.J., Norvig, P., 2003. Artificial intelligence: a modern approach, 2nd ed. ed, Prentice Hall series in artificial intelligence. Prentice Hall/Pearson Education, Upper Saddle River, N.J.

Witten, I.H., Frank, E., Hall, M.A., 2011. Data mining: practical machine learning tools and techniques, 3rd ed. ed, Morgan Kaufmann series in data management systems. Morgan Kaufmann, Burlington, MA.