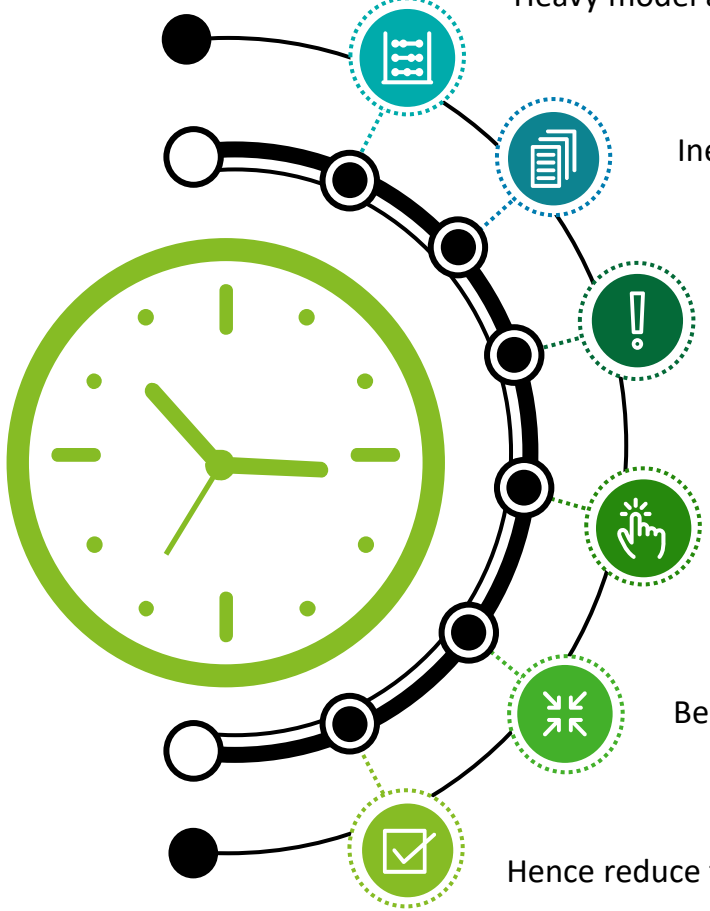
Deloitte.

Choose the Right Ones: Efficient Grouping of Policies by Means of Linear Programming Combined with Machine Learning

September 2022
Zoran Nikolić

MAKING AN IMPACT THAT MATTERS since 1845

Introduction
Companies struggle with long runtime of cash flow models



Heavy model and the large number of insurance contracts

Inefficiency and runtime challenge

Traditional method: high expenses incurred by these runs

Modern method: linear programming + machine learning approaches

Be able to reduce number of contracts that need to be projected

Hence reduce the model runtime

1. Challenge and Problem
2. A Traditional Method
3. A Leap Forward
4. Conclusion
5. Appendix

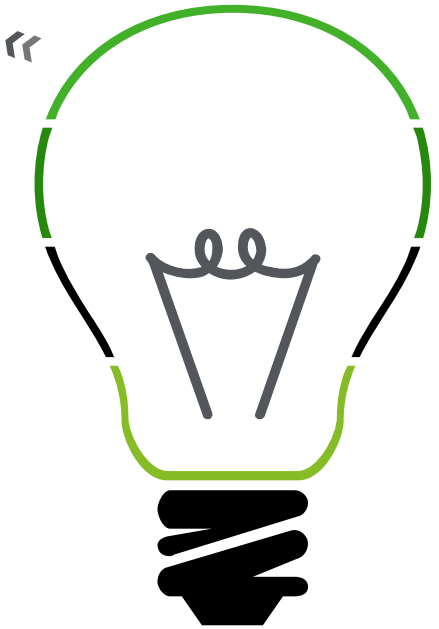
Deloitte.

Do you face runtime issues with your actuarial models?

1. Yes, we actuaries always want to calculate more than is possible.

2. No, I have perfectly optimized all my actuarial models.

3. I do not use actuarial models.



Stochastic Projection Model

The „Heavy“ Model

INPUTS

Policyholder data and **assets** on aggregated level

Economic scenarios

Assumptions

Parameters

STOCHASTIC MODEL

Assets and liabilities

Reflects the insurance company's types of products

Allows for all relevant cash flows and any interactions

Dynamic policyholder behavior and management actions

Discounting to present value

Monte Carlo stochastic modeling: projection performed thousands of times

OUTPUTS

Distribution for each measure

Best estimate

Confidence intervals

- **Necessary for products which include options and guarantees**
- **Stochastic modelling builds volatility and variability into the simulation**

Runtime Challenge

Economic importance of policy grouping

	Pure Liabilities Model	Stochastic Model (100 cores)
# Simulations	1	5'000
# Model Points	**Runtime**	**Runtime**
100'000	30 min	60 h
1'000	18 s	100 min

Grouping of insurance contracts has massive impact on the runtime of the model

Particular importance for Solvency II **stochastic calculations**

Formal Definition of the Problem (1/2)

Represent the entire portfolio with few policies

Setting: Portfolio with *M* policies, Projection of *N* variables, Projection period *L years*

- $A = [a_{m,n,l}], 1 \leq m \leq M, 1 \leq n \leq N, 1 \leq l \leq L$
  contains the future cash flows for each policy
- Now define

$$b_{n,l} = \sum_{m=1}^{M} a_{m,n,l}$$

  as the sum of cash flows of all policies for variable n in projection year l
- Array $B = [b_{n,l}]$ contains the aggregated cash flows
- B represents the entire portfolio

Target: Find the array of weights $X = [x_m]$ fulfilling

$$B = A * X$$

Note: A linear combination of policies should represent the entire portfolio.

Formal Definition of the Problem (2/2)

We want to reduce the number of policies in the model

Trivial solution : There is obviously the trivial solution (for $\mathbb{I} = [1]$):

$$B = A * \mathbb{I}$$

Target, reloaded: $A * X$ should be "close to" B and at the same time X should be "sparse".

- **"close to"** means for an array ϵ containing (small) allowed deviations

$$B - \epsilon * |B| \leq A * X \leq B + \epsilon * |B|$$

component-wise (replicate well each cash flow!)

- **"sparse"** means X must have as many zero-entries as possible (one additional constraint: $x_m \geq 0$)

The non-zero entries of X give us the weights for the **grouped portfolio**.

1. Challenge and Problem

2. A Traditional Method

3. A Leap Forward

4. Conclusion

5. Appendix

A Traditional Method (1/4)

Non-negative Least Squares (NNLS)

Recap: Recall our formal definition of the problem:

- Let A be the $(M * N * L)$-array containing the cash flows of individual insurance policies and B be the $(N * L)$-array containing aggregated cash flows.
- As above we require $x_m \geq 0$ for the weights and we want as many x_m to be zero as possible.

Task, reloaded:

- In the NNLS context the task becomes:

$$\arg\min_{X} \|A * X - B\|^2 \text{ subject to } X \geq 0 \text{ and } X \text{ sparse}$$

- It can be equivalently formulated as a quadratic programming problem

$$\arg\min_{X} \left(\frac{1}{2} X^T * Q * X + c^T X \right)$$

where $Q = A^T * A$ and $c = -A^T * B$.

A Traditional Method (2/4)
NNLS Algorithm (*)

Step 1

Choose the expected number of policies

Step 2

Iterate until the expected number of policies is reached:

Calculate the square deviation of a portfolio which contains one additional policy

Choose the policy which minimizes the deviation

Add this policy to the optimized portfolio

-> Matrix inversion is the expensive part!

(*) For more details about NNLS Algo please see appendix.

A Traditional Method (3/4)

NNLS Examples

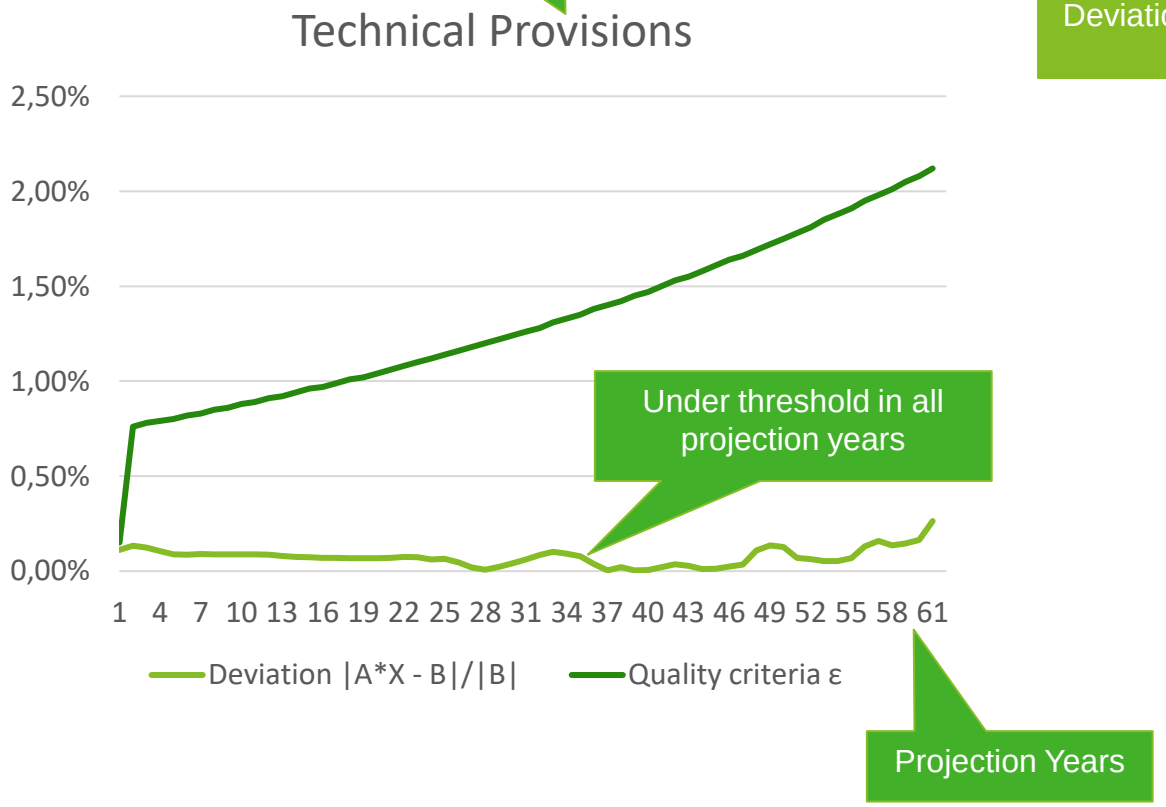
For a variable with large values quite good results

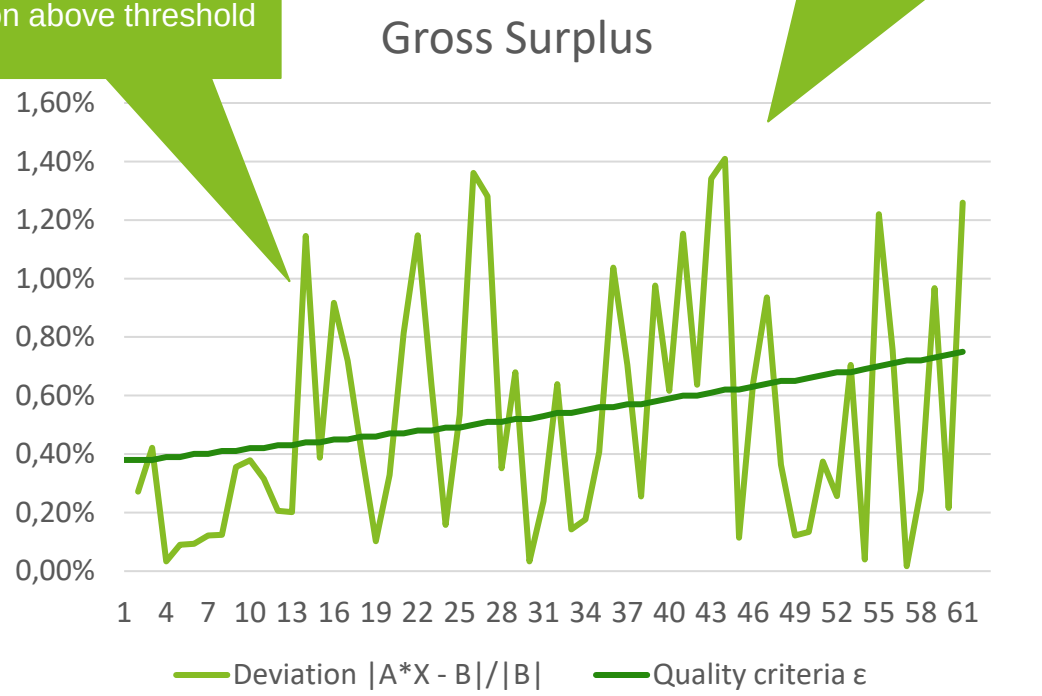
For gross surplus (much smaller values than for technical provisions) the quality criteria often not met

Also no large outliers, since the least-squares algorithm punishes very large deviations

Deviation above threshold

Under threshold in all projection years





Projection Years

A Traditional Method (4/4)

Discussion of NNLS

Advantages

- Consideration of **cash flows in the projection**
- **Easy to implement** (least-squares regression)
- **Often** yields **satisfying results**
- **Widely used in the industry** which ensures acceptance by all stakeholders

Disadvantages

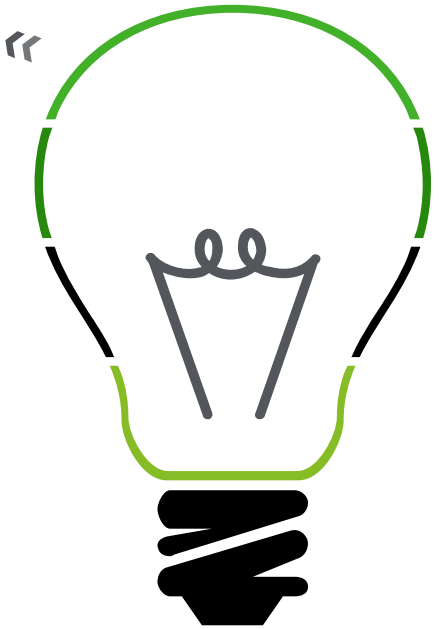
- In order to fulfil quality criteria **numerous iterations** with manual interventions in between needed
- Full integration in a workflow not possible
- No **direct control** that important variables like Gross Surplus are replicated sufficiently well
- **Exploding runtime** with increasing number of model points (non-zeros in X)

1. Challenge and Problem
2. A Traditional Method
3. A Leap Forward
4. Conclusion
5. Appendix

Deloitte.

Have you already worked with mixed Traditional/Machine Learning methods?

1. Yes
2. No
3. I don’t really know (what this means)
4. What is the difference between the two methods?



A Leap Forward: Linear Programming (1/6)

Back to the Basics

Recap: Let us recall again the problem of policy grouping:

- A is the array containing the cash flows of individual insurance policies
- B is the array containing aggregated cash flows
- for an array ϵ containing (small) allowed deviations

$$B - \epsilon * |B| \leq A * X \leq B + \epsilon * |B|$$

component-wise (replicate well each cash flow!)

- We require $x_m \geq 0$ for the weights and we want as many x_m to be zero as possible.

Recently, a student cleverly noticed that the conditions above can be formulated in terms of Linear Programming.

A Leap Forward: Linear Programming (2/6)

Re-Formulation

Task, reloaded:

Minimize a norm of the weights vector $|X|$ subject to the following constraints:

$$A * X \leq B + \epsilon * |B|$$
$$-A * X \leq -B + \epsilon * |B|$$
$$0 \leq X$$

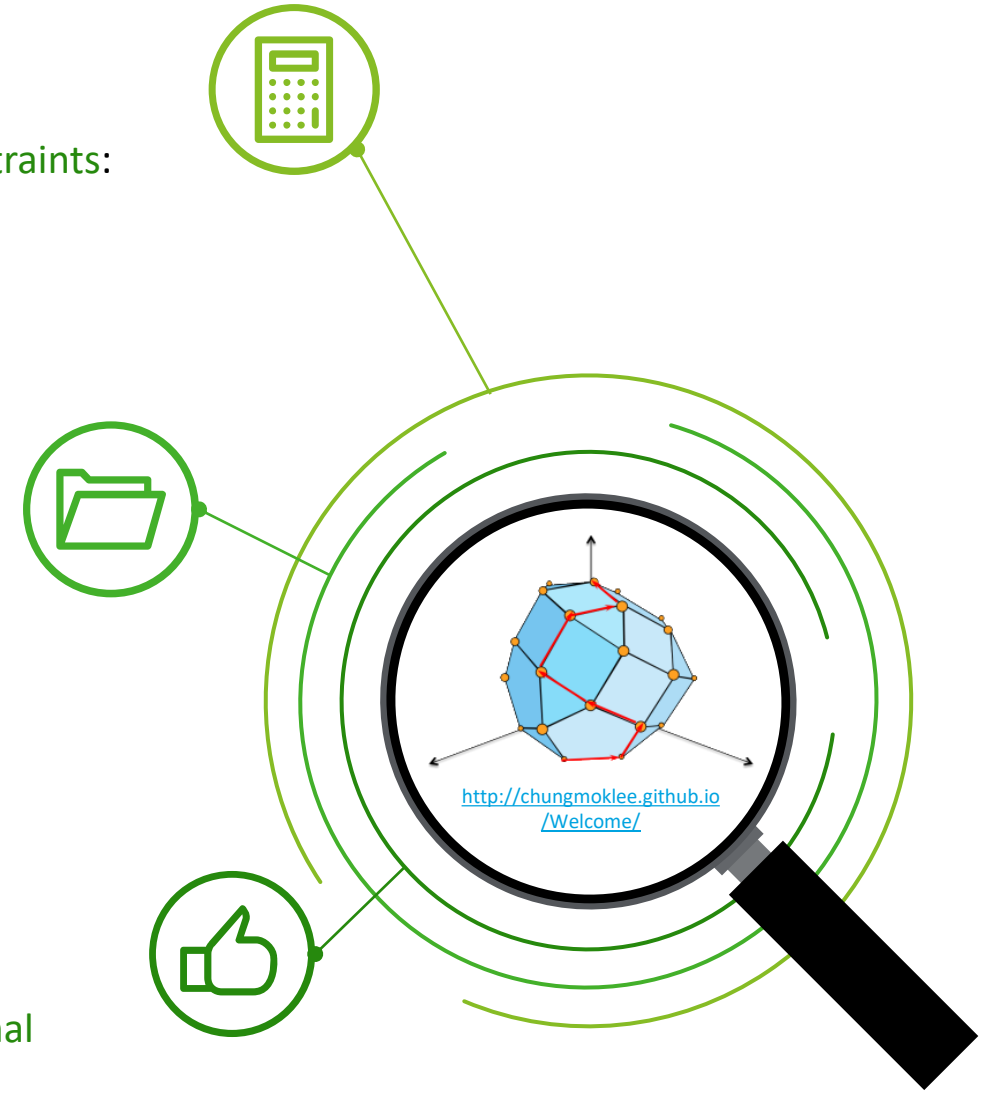
Resources:

- Many efficient solvers for this problem
- Excellent results with free GLOP tool from Google Operation Research team
- An explicit implementation: **DGO ML**

Advantages:

The given quality criteria are always met as they are embedded in the problem definition.

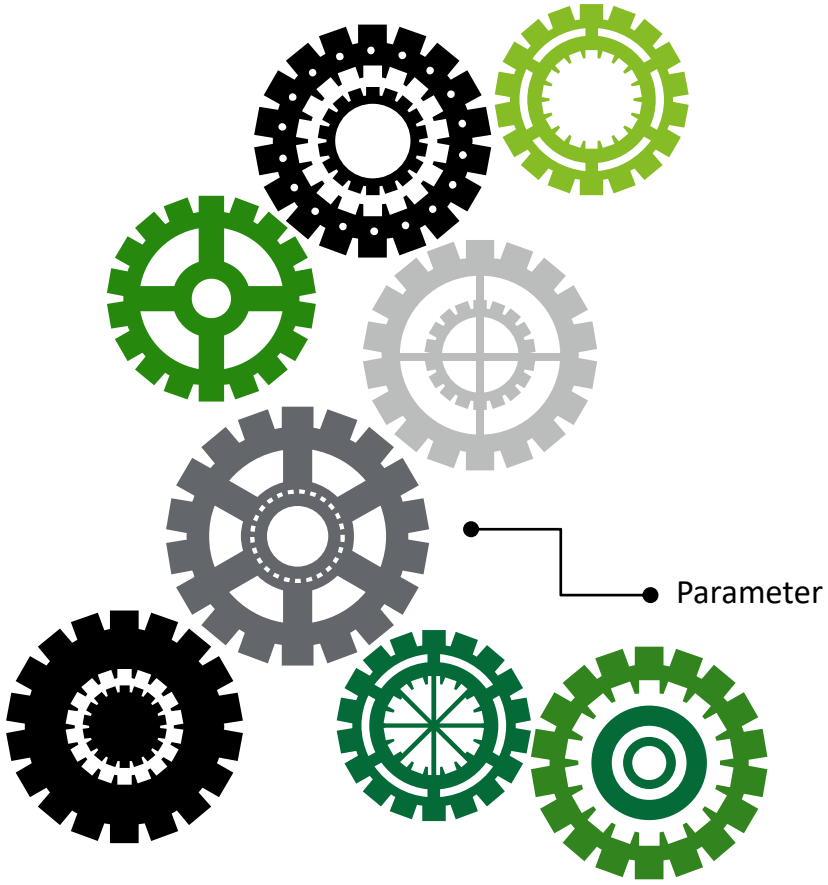
The solution can be found on one of the edges – enormous computational advantage!



A Leap Forward: Linear Programming (3/6)

Machine Learning Meets Linear Programming

- A paradigm in machine learning:
  model parameters get changed in a grid search or in a more sophisticated manner in order to initiate the training under different conditions.
- The linear optimization follows an algorithm which is sensitive to small changes of the input parameters.
- Use this feature of the model and vary pragmatically the parameters in order to obtain a set of results.
- In a “greedy” manner, the final model is the one with the lowest number of non-zero weights.

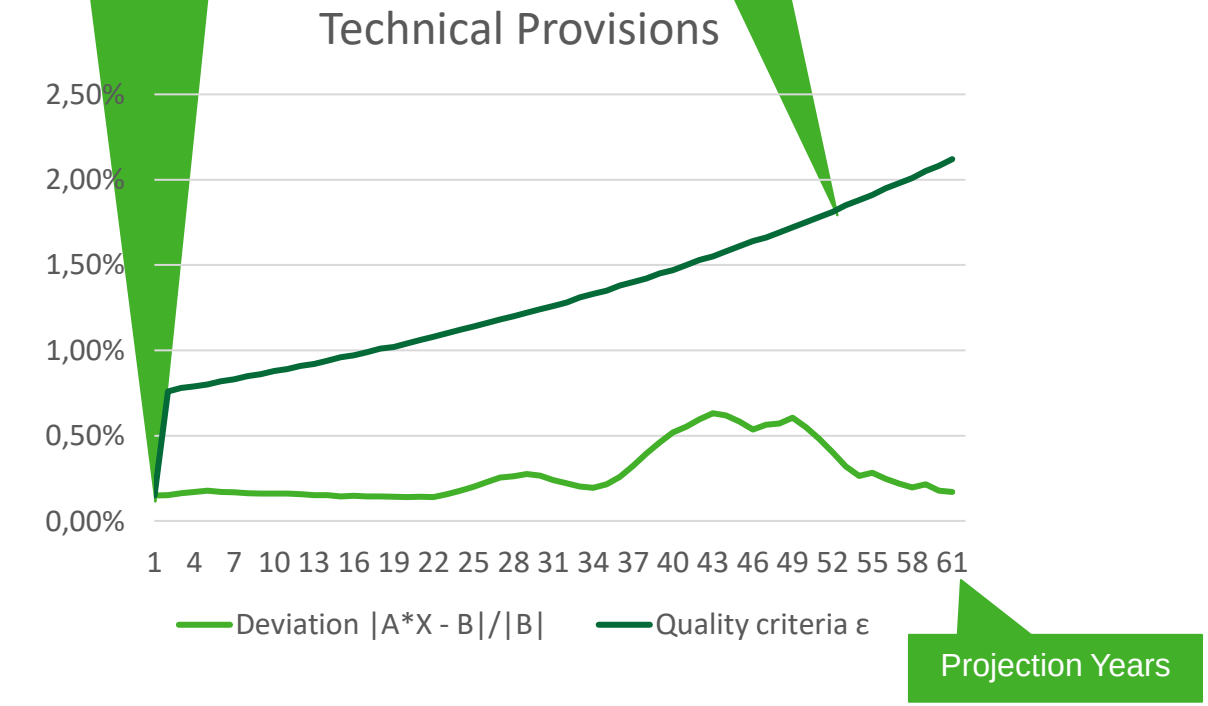


A Leap Forward: Linear Programming (4/6)

Linear Programming examples

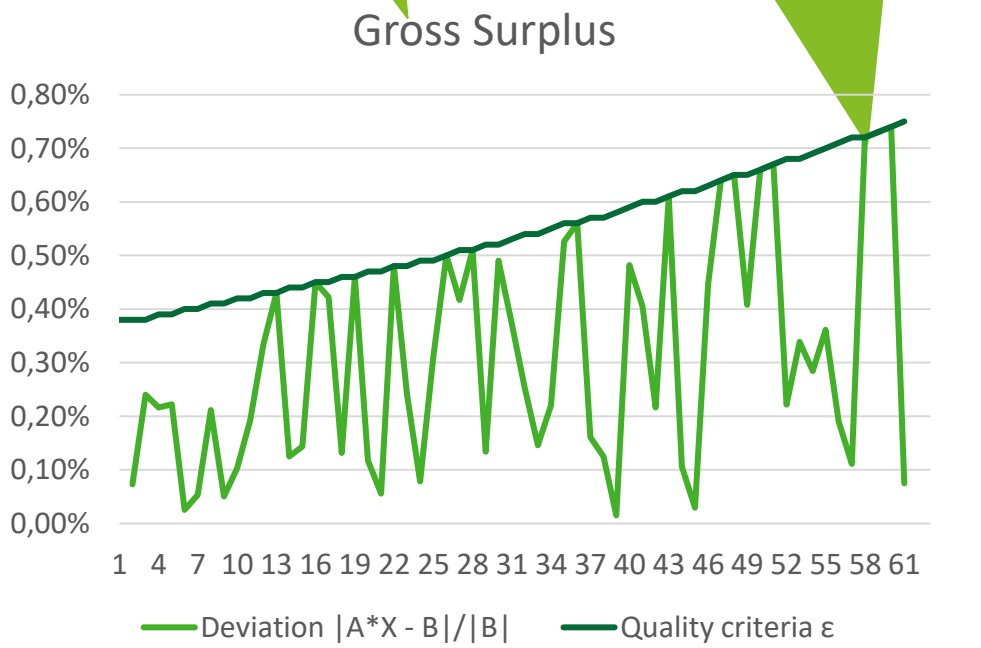
The light line is always below the dark one

The first year is still challenging, but doable



For gross surplus which has much smaller values the quality criteria are never breached

Often the light line „touches“ the dark one, but never moves beyond the dark line



A Leap Forward: Linear Programming (5/6)

Linear Programming examples

Example 1

Fast and Reliable High Quality Optimization : Significant Model Points Reduction

Cluster	Original amount of MPs	Amount of MPs after optimization	DGO ML Run time (s)	Change in MPs
1	4.956	**72**	34	**-98,55%**
2	48.623	**156**	163	**-99,68%**
3	660.648	**148**	1.026	**-99,98%**
4	63.251	**95**	204	**-99,85%**
5	140.523	**271**	289	**-99,81%**
6	123.430	**166**	170	**-99,87%**
7	4.956	**68**	28	**-98,63%**
8	33.823	**100**	58	**-99,70%**
9	107.327	**248**	227	**-99,77%**
10	1.169	**23**	32	**-98,03%**
All	1.188.706	**1.347**	2.230	**-99,89%**

The solver significantly reduces the amount of Model Points within the constraints of the allowed deviations.

Example 2

Fast and Reliable High Quality Optimization: Different Capital Market Scenarios

Cluster	Original amount of MPs	Amount of MPs after optimization	Change in MPs
Cluster_1	8.450	446	**-94,72 %**
Cluster_2	5.707	447	**-92,17 %**
Cluster_3	2.190	385	**-82,42 %**
Cluster_4	7.515	465	**-93,81 %**
Cluster_5	7.276	461	**-93,66 %**
All	**31.138**	**2.204**	**-92,92 %**

Significant reduction of MPs also for dynamic hybrid clusters with five different capital market scenarios per cluster.

- Depending on the size of the cluster and type of the cash flow projection tool, the amount of scenarios used has reached 200 scenarios in practice
- Also possible to use even more market scenarios during the optimization

A Leap Forward: Linear Programming (6/6)

Discussion

Advantages

- Allows direct **consideration of the goodness-of-fit** in the problem definition
- **Easy to implement** and **intuitive to understand**
- Usually yields the **lowest number of model points for a given goodness-of-fit** of all models
- Can deal with **very large problems**
- Easy **integration in the business workflow**, since no manual intervention necessary
- Good runtime

Disadvantages

- **No control** for the **number of model points** in the resulting model (must be coded as an additional constraint)
- Usually **commercial solvers** have much **better performance** than plain vanilla free solvers (additional costs)

1. Challenge and Problem
2. A Traditional Method
3. A Leap Forward
4. Conclusion
5. Appendix

Conclusion

1. **Policy clustering** is still a very important topic because the reduction in runtime of the actuarial models leads to an immediate positive economic impact.

2. **Classical techniques** are intuitive and often yield satisfactory results. However, for certain lines of business these methods require a large number of model points.

3. The **modern techniques** bear a great potential since they apply state-of-the-art machine learning methods. With **linear programming** the number of policies can be significantly reduced and at the same time the validation becomes streamlined.

4. **Neural networks** yield particularly promising results for very small numbers of model points. However, there does not exist a business solution for neural network clustering.

Contact

Zoran Nikolic
Partner
Deloitte Germany
Cologne Office

Phone: +(49) 221 9732416
Mobile: +(49) 151 58077609
znikolic@deloitte.de

1. Challenge and Problem
2. A Traditional Method
3. A Leap Forward
4. Conclusion
5. Appendix

Appendix

NNLS algorithm (*):

STEP 1:

Choose the number of policies i the user expects in the resulting model (these are the non-zero entries in X), set

$$P = \emptyset, R = \{1, \dots, M\}, X = [0], Y = A^T * B$$

STEP 2: While $|P| < i$ (the expected number of policies not reached):

- Let *j* be the index with $\max(y_j)$ (which policy explains best the portfolio)
- Remove *j* from *R* and add it to *P*
- Let A^P be A restricted to the variables in P, set $s^R = 0$ and

$$s^P = \left((A^P)^T A^P\right)^{-1} (A^P)^T B$$

This is the expensive part!

- Concat S^R and S^P to a vector S (S^P are the non-zero entries)
- Set $X = S$ and $Y = A^T(B - A * X)$

(*) Based on Lawson, Hanson: Solving Least Squares Problems (1974)

Back to NNLS