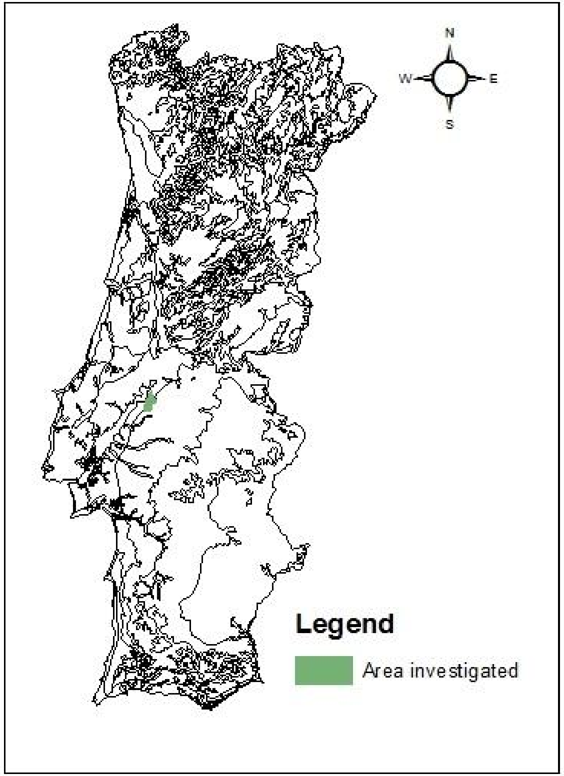
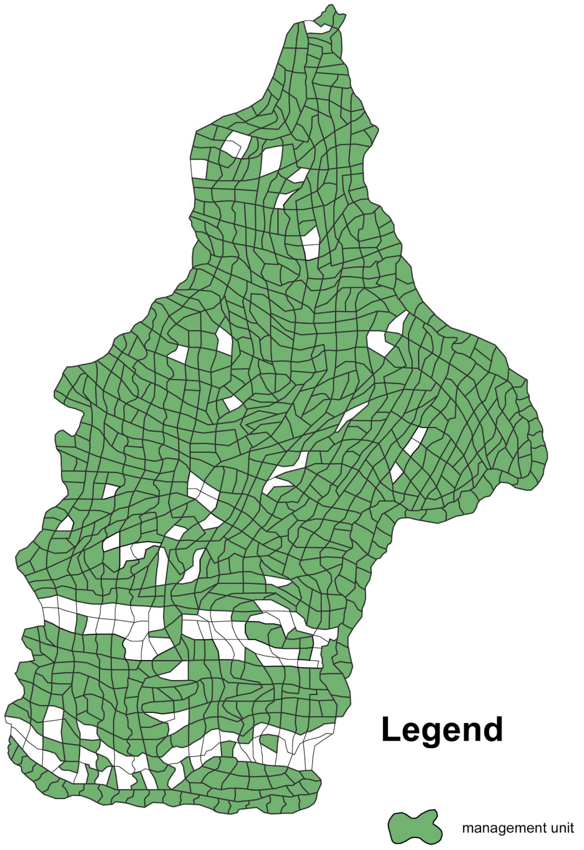
Project 2

1.

Introduction

Chouto Parreira





Chouto Parreira

- Predominantly dominated by eucalyptus
- This tree is prevalent in Portugal’s landscape, which has drawbacks
- It is a key resource used in paper and pulp manufacturing

2.

Problem Identification

Cost function

$$\max \quad Z = \sum_{i=1}^{N} \sum_{j=1}^{T} NPV_{ij} \cdot x_{ij}$$

- **We wish to maximize Z**
  - **NPV -** Net Present Value
  - **N** - Number of MUs (889)
  - **T** - Number of years (15)

Our Constraints

$$\text{s.t.} \quad \sum_{j=1}^{T} x_{ij} = 1, \quad \forall i \in N$$

$$VH_t = \sum_{i=1}^{N} \sum_{j=1}^{T} vh_{ij} \cdot x_{ij}$$

$$0.9 \cdot VH_{t-1} \leqslant VH_t \leqslant 1.1 \cdot VH_{t-1}, \quad t = 2, \ldots, T$$

$$x_{ij} + x_{i'j'} \leqslant 1, \quad \forall (i, i') \in I, \quad \forall (j, j') \in J$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in N, \quad \forall j \in T$$

Our Constraints

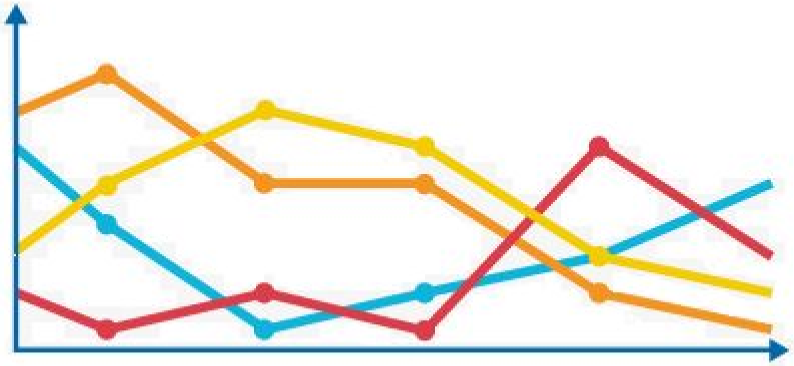
$$x_{ij} = 0, \quad \forall \; ZerosVar_{ij} = 1$$

3.

Solving the Problem with Matlab

Our Data

- **Our variables**
  - **N** - Number of MUs - set to 889
  - **T** - Number of years - set to 15 years
  - **NPV -** Net Present value - 889x15 matrix
  - **VOL** - MU volume - 889x15 matrix
  - **Area** - MU area - 889x1 matrix
  - **ZerosVar -** MU age - 889x15 matrix



Our Data

- Variables NPV and Vol are defined in m^2
- It's necessary to multiply tem by the Area of each MU

ANPV=Diag(Area)*NPV

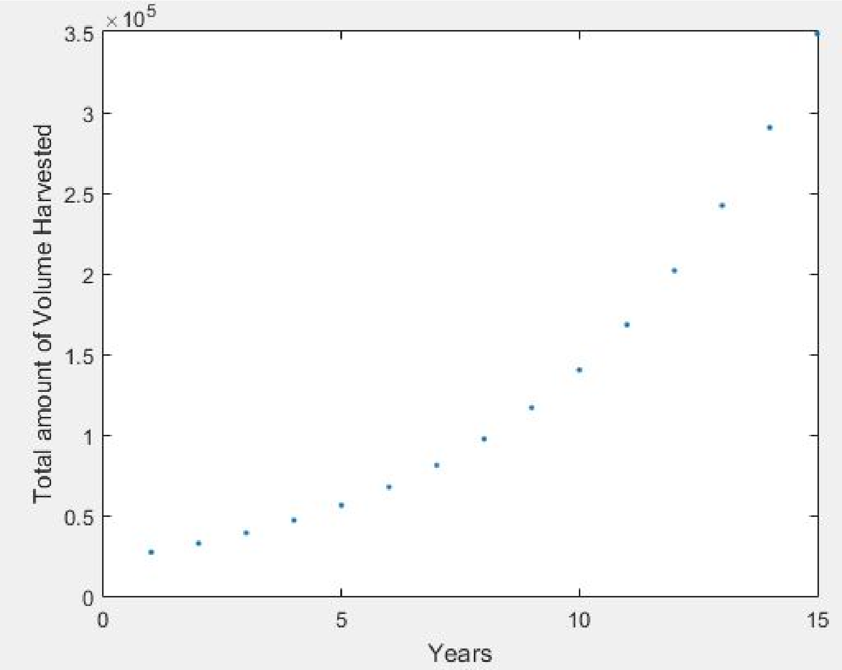
AVOL=Diag(Area)*NPV

Using CVX

- Modeling system for Convex optimization
- Fairly straightforward implementation
- Accurate results, non-binary output, takes 9-10 seconds to run

Using CVX

- Despite having a continuous output, it was able to fulfill every constraint

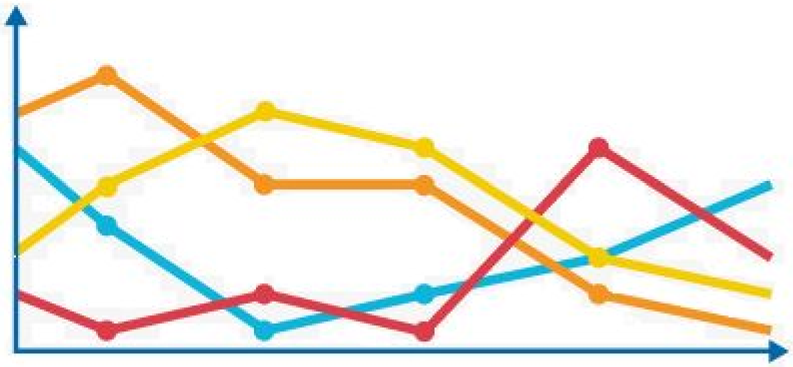


Using Linprog

- Mixed-integer linear programming solver

- Can only output a vector, so problem definition is trickier

- Can only minimize a function



Using Linprog

- Handling Variation constraints

- Variation constraints are of

Type A*x ≤ B

$$\begin{bmatrix} \cdots & \cdots & 13335 \\ \vdots & \ddots & \vdots \\ 28 & \cdots & \cdots \end{bmatrix} * \begin{bmatrix} x_{11} \\ x_{12} \\ \cdots \\ \cdots \\ x_{1T} \\ x_{21} \\ \cdots \\ x_{2T} \\ \cdots \\ \cdots \\ x_{N1} \\ \cdots \\ \cdots \\ x_{NT} \end{bmatrix} = [\ldots \quad \ldots \quad \ldots \quad 28]$$

Using Linprog

- Handling equality constraints

- Variation constraints are of

Type Aeq*x = Beq

$$\begin{bmatrix} 1 & 0 & \dots & 1 & \dots & (13335) \\ 0 & 1 & \dots & 0 & \cdots & \cdots \\ (889) & \dots & & \dots & & \\ \hline (900) & \cdots & \cdots & \cdots & & \end{bmatrix} * \begin{bmatrix} x_{11} \\ x_{12} \\ \dots \\ \dots \\ x_{1T} \\ x_{21} \\ \dots \\ x_{2T} \\ \dots \\ \dots \\ x_{N1} \\ \dots \\ \dots \\ x_{NT} \end{bmatrix} = [1 \;\; 1 \;\; 1 \;\; \dots \;\; 1 \,|\, 0]$$

Using Linprog

- Linprog takes less than a second
- Reaches a similar solution to CVX
- It Violates the variation constraint 6 times (out of 28)

Comparing both methods

- After rounding, CVX and Linprog are equal in 13329 out of 13335 cells

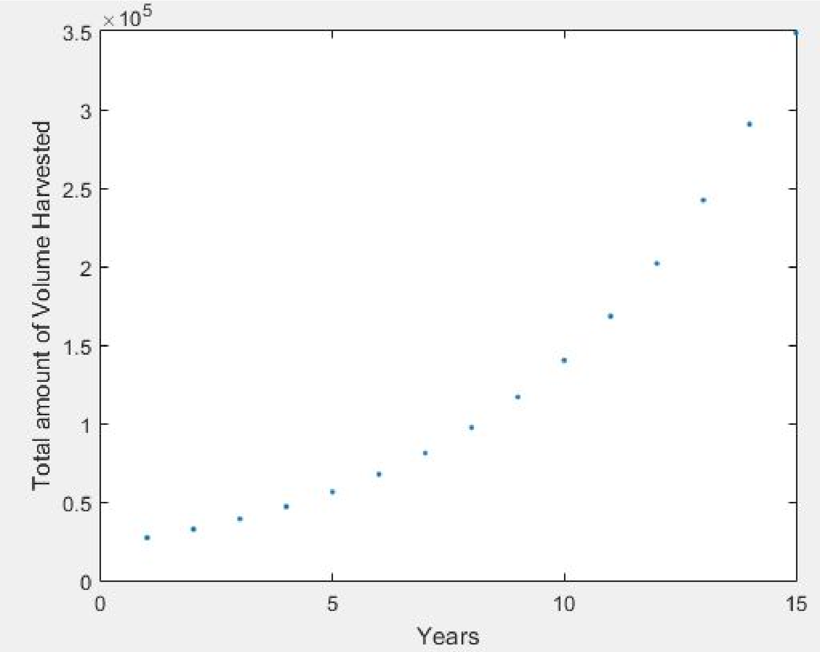
- CVX meets all constraints, Linprog fails in 6

Smoothing the Outputs

- In Order to turn the continuous outputs into Binaries, we try using “pick the highest” method.
- For each row, we pick the highest number, set it to one, and reduce the rest to zero
- After applying this method, CVX and Linprog mach 100%

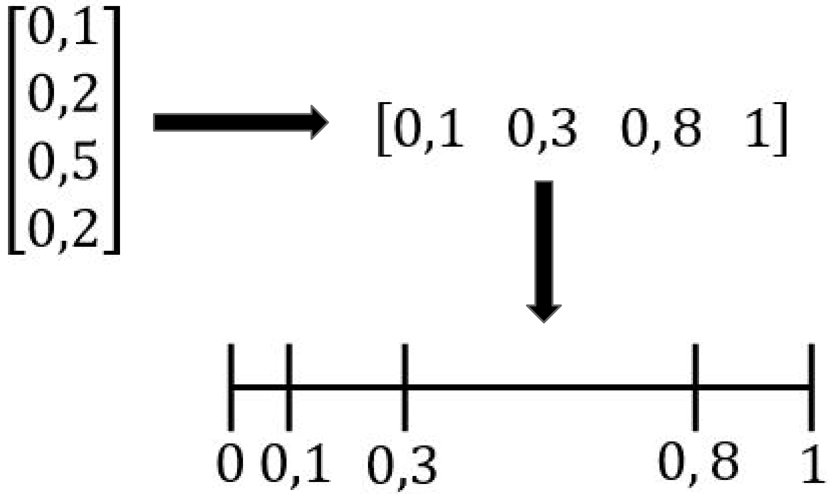
Smoothing the Outputs

- However, after using “pick the highest”, failure rate is 9 out of 28
- These violations aren’t very severe



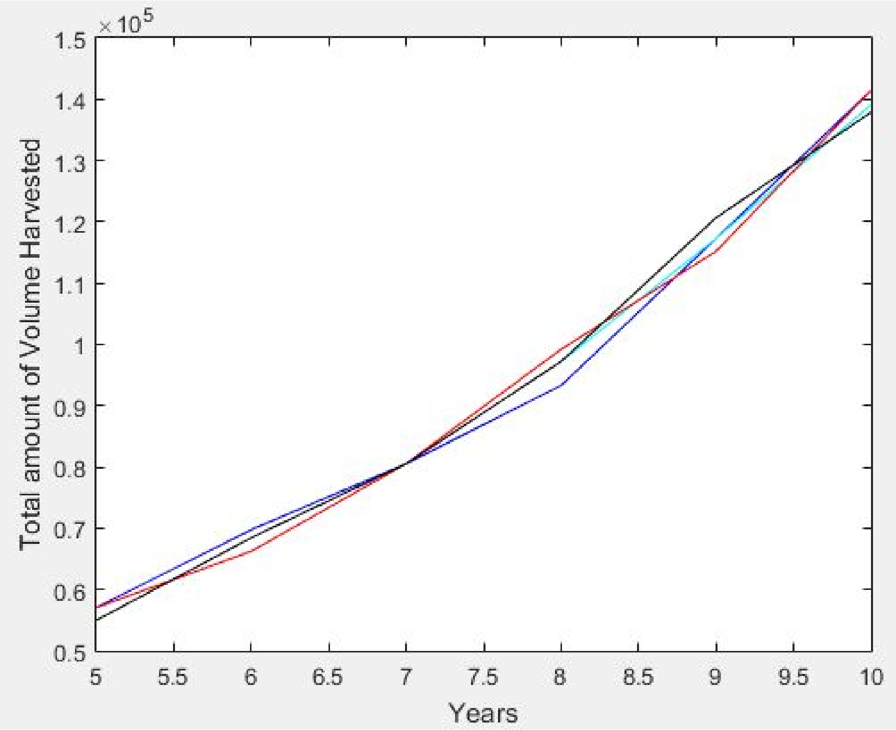
Smoothing the Outputs

- Another method used to smooth outputs is Monte Carlo



Smoothing the Outputs

- Running Monte Carlo multiple times run different results



Smoothing the Outputs

- By running Monte Carlo multiple times, we can get a failure rate of 6 out of 28

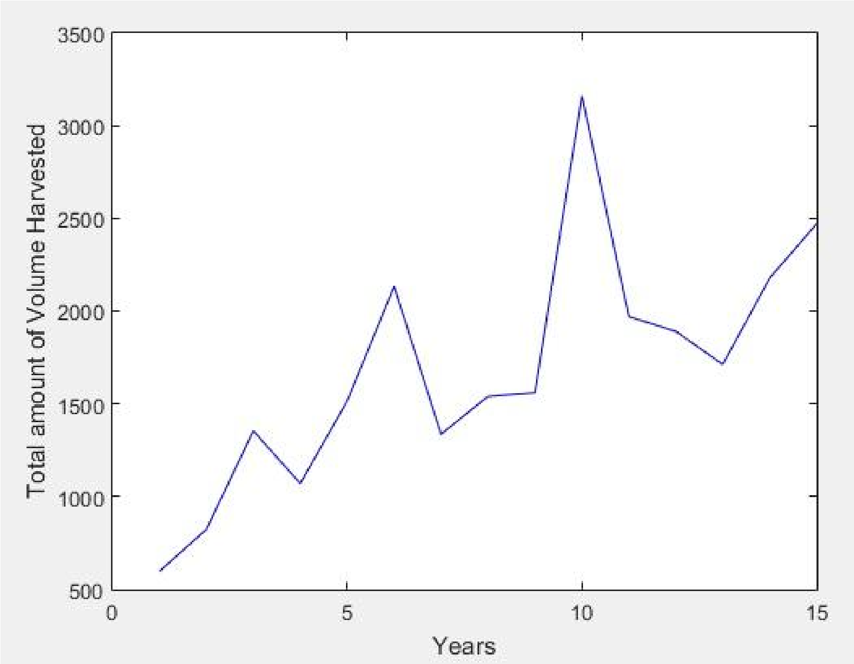
- That’s an equal failure rate to inprog, but in binary form

Intlinprog

- Mixed-integer linear programming solver
- Similar implementation to linprog
- Only outputs integer (In our case, binaries)

Intlinprog

- Can’t compute the original problem in useful time
- We need to reduce the problem to get a result



3.

Adjacency Constraints

Adjacency constraints

- Finally we revisit the adjacency constraint

$$x_{ij} + x_{i'j'} \leqslant 1, \qquad \forall (i,i') \in I, \quad \forall (j,j') \in J$$

- The variable adj is a 1803x2 matrix
- It contains which MUs are neighbours
- Neighbouring MUs can’t be harvested in the same year

Adjacency constraints

- After adding this constraint to CVX, it goes from 10s running time to 25 minutes
- Failure rate in this model is computed as the number of solutions near 0,5
- Failure rate is 1%

4.

Conclusion

Conclusions

- Binary programming is too computationally heavy
- Relaxing the output leads to better results, even if their not in the ideal form
- Those outputs can then be “smoth out” with “pick the highest” or “Monte Carlo” method

Conclusions

- Monte Carlo outputs are always different, since it's a random method
- It can however, randomly generate a solution that fits better than the one obtained with "pick the highest"
- For the optimal solution, several iterations of Monte Carlo should be run, and the one with the lowest failure rate should be picked

Conclusions

- Adjacency constraints complicate this problem exponentially

- It would be completely unfeasible to run this constraint with a binary method