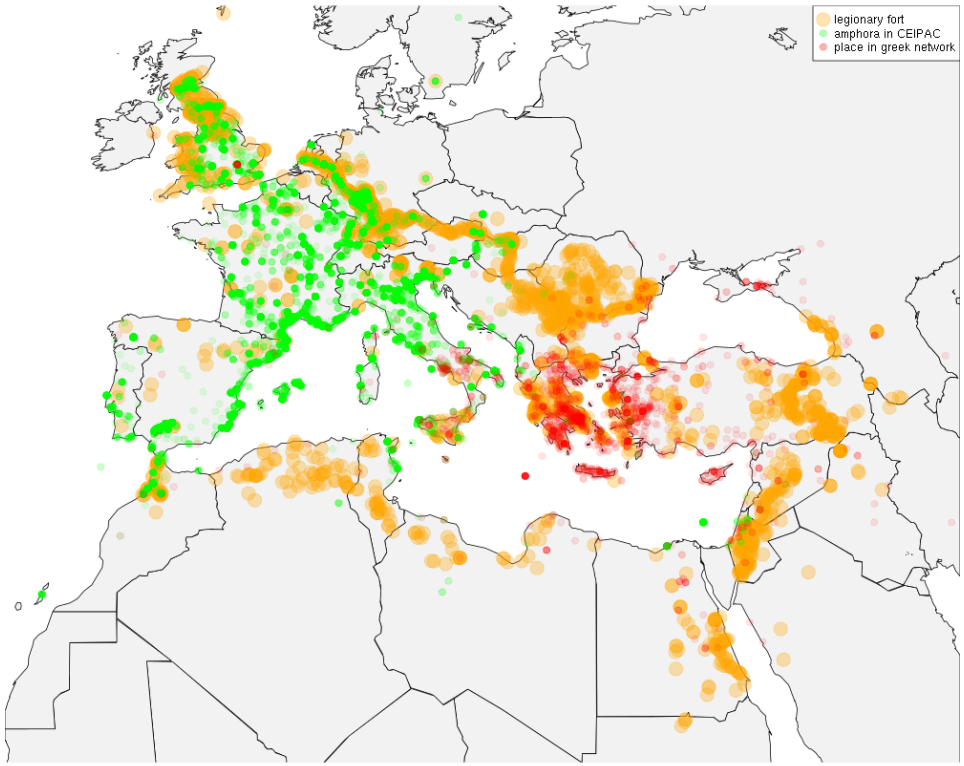
Model to “understand”



- Collection of data:
  - Why those data are distributed like that?

→ Model to understand what happened

What your model *will explain*:

- general law,
- detailled description,
- "de-idealization" : make less and less abstraction. More precision, less generalization :

  "Model the size of the universe."

Define precisely:

- Questions at sake,
- Hypotheses tested,
- Assumption made,
- Avalaible data.

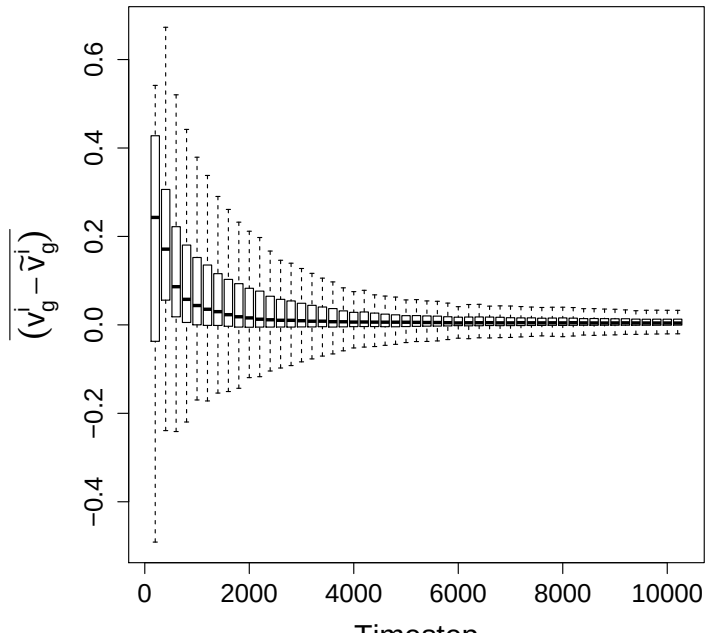
→ Close relations between model and people that *knows* about the system.

"To build a model helps to develop empathy & understanding across a community."
(it is even mandatory)

Underlying code

```
//Compute the score for each good
while(it!=allGood.end())
{
        std::string good=std::get<0>(*it);
        //in the case it is its production good
        if(good == std::get<0>(romanAgent.getProducedGood()))
                romanAgent.setQuantity(good,romanAgent.getPrice(good));

        //fit= |a-b|/euclideDist(a,b) my favorite one
        if(romanAgent.getQuantity(good)==(romanAgent.getNeed(good)))uti
        else utilityFunction+=std::abs((romanAgent.getQuantity(good))-(
)uantity(good))+(romanAgent.getNeed(good))*(romanAgent.getNeed(good))))
```



```
//Compute the score for each good
while(it!=allGood.end())
{
        std::string good=std::get<0>(*it);
        //in the case it is its production good
        if(good == std::get<0>(romanAgent.getProducedGood()))
                romanAgent.setQuantity(good,romanAgent.getPrice(good));

        //fit= |a-b|/euclideDist(a,b) my favorite one
        if(romanAgent.getQuantity(good)==(romanAgent.getNeed(good)))uti
        else utilityFunction+=std::abs((romanAgent.getQuantity(good))-(
)uantity(good))+(romanAgent.getNeed(good))*(romanAgent.getNeed(good))))
```

```
//Compute the score for each good
while(it!=allGood.end())
{
        std::string good=std::get<0>(*it);
        //in the case it is its production good
        if(good == std::get<0>(romanAgent.getProducedGood()))
                romanAgent.setQuantity(good,romanAgent.getNeed(good));

        //fit= |a-b|/euclideDist(a,b) my favorite one
        if(romanAgent.getQuantity(good)==(romanAgent.getNeed(good)))ut
        else utilityFunction+=std::abs((romanAgent.getQuantity(good))-
)uantity(good))+(romanAgent.getNeed(good))*(romanAgent.getNeed(good))))
```

Results

```
//Compute the score for each good
while(it!=allGood.end())
{
        std::string good=std::get<0>(*it);
        //in the case it is its production good
        if(good == std::get<0>(romanAgent.getProducedGood()))
                romanAgent.setQuantity(good,romanAgent.getNeed(good));

        //fit= |a-b|/euclideDist(a,b) my favorite one
        if(romanAgent.getQuantity(good)==(romanAgent.getNeed(good)))ut
        else utilityFunction+=std::abs((romanAgent.getQuantity(good))-
)uantity(good))+(romanAgent.getNeed(good))*(romanAgent.getNeed(good))))
```