Approximating the optimal rule with machine learning : some consumption-saving applications

John Dyer

September 2019, Paris-Dauphine University
Supervision: Marie Bessec and Pablo Winant

1 Introduction

Stochastic control is the main framework of modern intertemporal economic models. Most often it is addressed with stochastic dynamic programming. It allows to represent a sequential problem in which today's decision will impact tomorrow's choice. Therefore, the maximization of the intertemporal utility must take in consideration the impact of the choice of the control variable not only on the utility of the current period but on the upcoming state variables too. It is said to be stochastic when a stochastic process (most often a Markov chain) is also to be considered within the problem. The Bellman equation gives an explicit form for this problem, relating the optimal choice of today with the future optimal choices with the help of the value function.

Those problems are most commonly solved by state space discretization and iteration on the Bellman equation or backwards induction. Nevertheless, this can become computationnally heavy very quickly, if the size of the state grid is too big. For a low number of state variables (one or two) this can be overcome by narrowing down on the precision of the grid and iterating. But the computational burden gets too big for more state variables, forcing the computational economists to keep the number of state variable small to solve a stochastic dynamic problem. This is called the curse of dimensionality .

Here, we will study a specific solution concept which allows us, in a way, to avoid the state space discretization step. If we do not construct a neural network *per se* (sometimes called a "universal approximator"), we use the machine learning toolbox to approximate the optimal rule with great simplicity, with the help of tensors. We simply aim at maximizing the lifetime-reward, by repeatedly simulating lives and computing the gradient of the lifetime utilities. This is the gradient descent technique.

The tensor objects are specfic objects of dedicated libraries such as *Pytorch*, which we will use, and *Tensorflow*. They allow us to keep track of the variable and to compute its gradient at once and automatically. This is a huge gain of time and the key point explaining the rapidity of the machine learning techniques. With the use of the gradient, we modify the parameters we want to approximate grossly towards optimality. Plus, the array objects from the *Python* library *numpy* allow us to compute several trajectories at once thanks to parallelism on vector instructions.

The studied problem is the canonical and stylized consumption-saving problem. In this framework, a consumer tries to maximize the utility of her intertemporal flow of consumption, deciding each period what proportion of her cash to consume and what proportion to save for the next period, while earning a random flow of income. This problem, introduced in its modern form in the 1950s by Milton Friedman, seems quite simple, and yet only in the late 1980s were computer powerful enough to give a precise approximation of the solution with plausible preferences and risk aversion. The optimal consumption rule is indeed non-linear and its finding computationally heavy.

In our paper, we focus on the work of Allen and Carroll, 2001 [1]. Since the optimal rule seems to be very complex, they address the problem of its learning by the consumers, and propose a much more simple approximated time-invariant rule, which associates a level of consumption for a given level of cash-on-hand. The rule, furthermore economically readable, is defined by only two parameters. They highlight a very low utility loss following the best approximated rule rather than the optimal rule. Since their research is about the learning of the rule, they explicit the number of simulations needed to find a good rule. They conclude that the number of periods needed to find a good rule is too big for the consumer to learn it on her own. This will give us a target. Indeed, if their solution concept is naive, we study a problem in every point similar, and the number of simulations we do can be compared with their work as much as later work. We find the best approximated rule no more by grid points search as Allen and Carroll, but by machine learning techniques.

However, our primarily focus is on the efficiency of machine learning approximations, not on the learning process. Thus, we move apart from the infinite-horizon model used by Allen and Carroll to the life-cycle model, where the income stream of the consumer change during her life, notably at retirement. We will then no more try to approximate two parameters but four, in order to show that the number of parameters to approximate is not a fatality, as it is in more bruteforce techniques.

We reproduce the results from Allen and Carroll, 2001 [1] with an extremely low computational cost and a great modelling simplicity. We take time to analyze the specificity of machine learning. On a second time, we complexify the model to increase the number of parameters to be approximated. Here again, the rapidity of the solution concept allows us to multiply the computations which in turn allows us to determine an approximated optimal rule.

The first part of the essay is a literature survey, which will be divided in three. We will first try to cover the rich research on the consumption-saving problem, starting from the permanent income hypothesis of Milton Friedman to the buffer stock theory of Deaton and Carroll. The underlying issue is the motive for saving, that is whether the precautionary motive or the life-cycle motive is dominant for saving - do we save to be protected against bad luck or for retirement. After focusing on the finding of the solution and how it influenced the debate for years, we will in the second part present Allen and Carroll, 2001 [1] and the concept of rule of thumb, to which we will refer a lot. The last part of the literature survey introduces machine learning and how it was used in the economical literature, be it as to mimic the search of the agents, to express the expectation term or to solve whole models, which is our preoccupation here.

In the second part, we will present our results. After a presentation of the solution concept, the lifetime-reward maximization and the idea of gradient descent, we will present in more details the model of Allen and Carroll, 2001 [1] which is our reference and our first algorithm. We will then discuss the results as well as the speed of the algorithm, compared to Allen and Carroll. Then, we will switch to a finite horizon and adapt the model to make life-cycle traits appear, such as evolving income and retirement. We will need comparison point, so we solve the model with dynamic programming and backward induction. This new model modifies the consumption rule and more parameters are now to be approximated, such that a grid points bruteforce search would no longer be possible. On the contrary, it does not affect our solution concept that much, which is interesting to note. This could prove to be an interesting way to tackle the "curse of dimensionality" and we will discuss this in the conclusion.

2 Literature survey

2.1 The consumption-saving problem

The 1992 book *Understanding Consumption*[9] by Angus Deaton presents the consumption-saving problem in detail. It aims at summarizing the work that has been done on that subject for the past 50 years and introducing some new ideas. The problem is to find a theory that describes the intertemporal choice of the agent given the trade-off between present and future consumption.

In the first pages of the book, Deaton presents the model in its most simple form. The agent lives two periods and decides on the first period how much of her wealth she wants to consume and how much she wants to save for the other period, so that her intertemporal utility is maximized ; solving this problem is straightforward. Adding more periods complicates the problem of course, and there are then two possible horizons : finite or infinite. A finite horizon model will most often be a life-cycle model and will aim at resemble the age profile of consumption and its evolution upon the life of the agent. On the other hand, an infinite-horizon model, if it cannot have any life-cycle interpretation, can be easier to solve and can highlight some key dynamics of the problem.

In 1957, in his book *A Theory of the consumption function*[13], Milton Friedman introduces the permanent income hypothesis (from now on PIH). It states that the agent consumes each year the annual proportion of her currently lifetime estimated financial and human wealth. It means the agent will perfectly smooth her consumption and save so that she can consume the permanent income every year. Friedman provides statistical evidences to justify his theory, which is major at the time.

The limitations of computational power were such that it was not possible to find a numerical solution to the consumption-saving problem with both risk aversion and uncertainty until the late 1980s. Until then, the problem was largely discussed without credible risk aversion in the utility functions or no uncertainty in income. As a result, the certainty equivalent model was dominant in the literature even though it has many empirical failures, as stated in Dornbusch and Fischer, 1998[10]. Those were notably a low sensitivity of consumption to current income and over-spending of the elderly, when comparing the model to the data. The usual conclusion was that those failures may be caused by the limited rationality of the agents or liquidity constraints (see Hayashi, 1987 [15]).

The 1989 article "Optimal Consumption with Stochastic Income: Deviations from Certainty Equivalence" by Zeldes[25] addresses this issue. Taking advantage of the progress of computers, it approximates the optimal solution of the problem with stochastic income and CRRA utility function, which had never been done until then. When income is stochastic, the permanent income hypothesis holds only if the utility function is quadratic and the consumption can be negative. Then, the optimal level of consumption with respect to the level of wealth is an upward sloping straight line. Here the consumption function is concave. Another motive of savings than the life-cycle one (saving for retirement) is highlighted : precautionary savings. It was defined by Leland, 1968[17], solving a two period model, as the difference between the consumption under certain income and the consumption under uncertain income. For a small horizon, Zeldes finds two consumption functions depending on different income processes, namely one or two sources of uncertainty. If they differ on level and on slope, they have similar properties, so the simpler one can be used to extend the length of the optimization up to 90 periods, while the other could not for computational reasons. The model gives results that differ a lot from the routinely used PIH models and solves most of puzzles about consumption without using bounded rationality nor exogenous liquidity constraints. This was a surprising come back to Friedman's rational expectations, as noted in Allen and Carroll, 2001[1].

Starting from this point, the buffer stock theory develops. The idea of a buffer stock is that consumers will have a non negative stock to smooth their consumption in case of lower income : it is precautionary saving. What will differ from the work of Zeldes is the assumption that consumers are greatly impatient. The buffer stock can be seen as a financial shield. It is the result of a tension between impatience (the consumer values today's consumption sensibly more) and prudence (a bad income draw would have a great impact on her utility). It is characterized by a target level, below which the consumer will save in order to be better protected, and above which the consumer will dissave to satisfy her impatience.

To make a buffer stock behavior appear without imposing liquidity constraints, it is necessary to allow the possibility of very low income, as in Carroll, Hall and Zeldes, 1992 [6]. There, the income distribution allows for near zero income, even though the probability is very low (approximately 0.5 percent). This is sufficient to create a buffer stock, given that the form of the utility function implies an extremely low utility near zero (tending to minus infinity at the zero bound). Consumers will therefore try to keep a buffer to avoid such a disastrous event in the last period, and recursively, in all previous periods. The question is then whether these near zero income events are credible. The paper shows it is measured in the data, but a measurement error cannot be ruled out. The possibility of this event is nevertheless crucial for their model to exhibit a buffer stock behavior.

On the other hand, Deaton, 1991[7], doesn't account for near zero income events by assuming an income floor. In order to generate a buffer stock behavior, he rather imposes liquidity constraints which make borrowing impossible. That way, there is no other option than precautionary saving to react to a bad income draw. It also allows to study a infinite horizon model, which is more convenient. The liquidity constraints model is not only relevant to understand the microeconomic behavior of liquidity constrained agents, but it also displays interesting characteristics of the saving behavior by creating a buffer stock behavior.

Those two papers highlighting buffer stock behavior both study infinite-horizon models. If Zeldes, 1989[25], studies a finite-horizon model, he does not consider enough impatience to create buffer stock behavior. Thus, Carroll, 1997[3], aims at taking the buffer-stock model to the data and studies how and when it shows in the life-cycle. As expected, the results differ from the traditional life-cycle models. By calibrating income to the data, Carroll compare different models with his : the Keynesian model with a constant propensity to consume, the standard life-cycle model and a mixed version of those two models by Campbell and Mankiw. He shows that only his buffer-stock version of the life-cycle model is consistent with different stylized facts and solves the empirical puzzles. The backward resolution of the finite-horizon models shows that consumers will have a buffer-stock behavior with consumption closely tracking income until roughly 50, at what point they will switch to a permanent income hypothesis behavior and save for retirement.

Gourinchas and Parker, 2002[14], goes in the same direction. After a thorough estimation of all the parameters of their model with the Method of Simulated Moments, they study the motive for saving and its evolution. They also find an evolution in the behavior of the consumers, who would switch from target savers to certainty equivalent consumers at the age of approximately 45. They show that a low discount rate or a low expected growth of income induce a permanent income hypothesis behavior, whereas a high discount rate or a high growth of income induce a buffer stock behavior. So, they argue than it is the expected growth of income more than the consumer's impatience that explain the buffer stock mechanism.

If we will not enter this issue in the present paper, we will rely upon and test the result that the behavior of consumers can be characterized as buffer stock before age 45, and as permanent income behavior after that age. This will be the second part of the thesis as we will take our approximation concept to a life-cycle model.

2.2 Rules of thumb and bounded rationality

But first, we will replicate the results from Allen and Carroll, 2001[1]. Reflecting on the consumption-saving debate and on the considerable findings that were made in the 1990s thanks to the development of computing, they ask how come consumers have been behaving very closely to the only recent approximation of the optimum. This would tend to confirm Friedman's learning hypothesis, that agents can learn any optimal solution by trial and error. Nevertheless, optimal behavior and perfect rationality of the agents is often questioned empirically and by experimentation. The idea of a rule of thumb is therefore attractive, because if it is good enough, it would explain that the observed behavior is similar to the optimal one, while embracing bounded rationality in the same time. The paper tests whether consumers can learn the optimal parameters of a simple rule of thumb and how much time it would take them to do so.

Rule of thumbs are simple decision rules that map any state with a decision. In the consumption-saving problem, the rule of thumb will give a level of consumption for any level of cash-on-hand, that is the sum of today's income and assets according to Deaton, or the wealth immediately at the consumer's disposition. Rodepeter and Winter, 1999[22],x addresses the issue of the performance of different rules of thumbs by comparing five different ones to the dynamic programming numerical approximation. By simulating a given life-cycle both following a rule of thumb and the optimal behavior, they compute the needed increase in income for the rule of thumb to yield the same utility as the optimal behavior. They find the results to be very different, depending on the followed rule of thumb, the preference parameters and the certainty of income and income growth. The important idea is that a well chosen rule of thumb given the problem can yield a close utility to the the dynamic programming one.

Indeed, Allen and Carroll, 2001[1], do compute the utility loss implied by their rule of thumb. They calculate the average value that has to be withdrawn to the cash on hand in the value function following the optimal rule for it to be equal to the value function following the rule of thumb. This is the sacrifice value, which we will compute too in both our applications. Their sacrifice value varies a lot depending on the parameters of the rule of thumb : it is of 0.57 if the consumer spends all her wealth and of only 0.003 at its minimum, following a rule close to the best approximation rule they find (knowing the average income is 1). Furthermore, the form of the rule of thumb they use – which is the one we will use in the following section – is frugal and easily readable. It is therefore very useful as it is both analytically convenient and economically satisfying.

The fact the rule of thumb approximates the optimal rule this close might not be surprising as it is constructed as a linear approximation of the exact consumption rule. Indeed,a similar rule of thumb is introduced in Deaton, 1992[8]. He explicitly states that he chooses it to mimic the optimal consumption function. His goal was to check if a rule of thumb could be close to the optimum, not how and if it could be learned. He noticed two principle elements : a consumer will spend all of her cash on hand if it is below a critical level, a level above which she will save a proportion of her cash on hand. Carroll, 1996[2] shows that if an impatience condition is verified, depending on the interest rate and the time preference factor – as well as the amplitude of shocks on permanent income and the growth rate of income which we omit here, there is another critical level above which consumers will save and below which they will dissave.This allows to approximate the optimal rule by a simple form, depending only on two parameters and on the liquidity constraint. One of them is that second critical level which is the target wealth the consumer is trying to reach (the ideal buffer stock), and the other is a proportion coefficient that indicates the rapidity at which the consumer wants to achieve that target buffer stock. We will get in more detail and express this consumption rule in the next section.

Because it is intuitive and simple (only two parameters are to be found), it is credible that consumers can learn the rule during their lifetime, thus explaining their close-to-optimal behavior. Allen and Carroll, 2001[1], tests this hypothesis by measuring the time needed to obtain a good enough approximation. For a relatively gross grid point search (400 points of possible parameters), they find a search time of 4 million periods, a number that cannot reasonably go below 1 million - a period is a year according to the discount factor. This result would tend to reject the learning hypothesis, but it is very dependant on the solution concept. In Carroll, 2006[5], the method of endogenous gridpoints is introduced to solve dynamic stochastic optimization. The grid is no more static but endogenously constructed, so that there is no more useless calculations : the search is considerably reduced.

Nevertheless, the best candidate to reduce the time of search is social learning, as it seems to be empirically pertinent and still consistent with bounded rationality. Palmer, 2012[20], studies the same model as Allen and Carroll but allows for near perfect information sharing by agents. The results are similar to Allen and Carroll, except the number of periods is divided by the number of agents looking and sharing information, and thus much smaller. But the perfection of information sharing goes against bounded rationality, and more specifically bounded sociability , as stated by Salle and Seppecher, 2013[23]. Based on the results of social experiments, they develop multiple evolutionary algorithms mimicking learning with social interaction. When they allow agents to decide when they want to look for a better rule and give them a social memory, they find a convergence rate that can go below a hundred period. This contradicts Yildizoglu et al.[24], 2012, in which they compare different alternative to Allen and Carroll's individual learning and reject social learning in favor of an adaptive expectation model in which agents use an artificial neural network.

2.3 Machine learning as an approximation tool

Indeed, the authors test other bounded rationality learning mechanisms in this paper. Also focusing on the Allen and Carroll rule of thumb, they find unsatisfying results for both adaptive individual learning and social learning. Nevertheless, they introduce a new representation of the learning of the agents by allowing them to look forward, which converge towards a good enough rule in less than 1000 periods. In this mechanism, the agents have a personal mental model which represents the environment and which they use to form expectations on the utility stream of the different rules. To do so, they use a feed forward artificial neural network (ANN). Through its hidden layers, an ANN relates a given number of inputs to a given number of outputs. Here, the inputs are cash-on-hand, income and the two strategy parameters, and the output is the sum of utilities at a given horizon. Every time the horizon is reached, the record of past experiences is used to update the parameters of the ANN by computing the residual error between the effective outcome of a rule and the output of the network (its prediction). Then, the agent uses her model to test every possible rule of thumb and chooses the one that is supposed to give her the highest utility. Once the agent observes the effective utility stream, she uses this new observation point to train her ANN again, and so on.

Neural networks are an important aspect of machine learning and are seen as universal approximators for their capacity of approximating complex and non-linear functions. We will focus on a more parsimonious method that does not require the use of a neural network, since our preoccupation is not the learning of the rule. Anyway, we do train our consumption rule by Monte Carlo simulations and gradient descent (see next section).

Earlier literature used reinforcement learning to find a good rule of thumb. Lettau and Uhlig, 1999[18], used a simple classifier system to discriminate between a pre-constructed batch of consumption rules without success. Howitt and Özak, 2009[16], also use the reinforcement learning framework (learning as you go) to stylize the behavior of the agents. In this paper, the agents use the euler equation to compute a consumption error every period. They then use this consumption error to update the parameters of their consumption rule, through an OLS regression on the Euler equation.

Neural networks were also used to approximate and solve stochastic growth model as early as 2001. Duffy and McNelis, 2001[11], compare the utilisation of a neural network to the polynomial expansion in the parameterized expectation algorithm. Here, the neural network takes the states as input and gives an approximation of an intertemporal expectation as output. They find the neural network overperform the polynomial expansion, which introduces neural networks as a promising tool for economists.

Neural networks are indeed a very convenient way to approximate policy functions directly. By their form, they can relate state variables to an outcome (a policy) and is thus a perfect candidate to be a policy function approximator, as stated in Powell, 2011[21].

For a great number of hidden layers, we qualify the neural network as a deep neural network. Recent advances in the machine learning field highlight the great but not yet fully understood results of the deep neural networks in various applications. Han and E, 2016[12] apply a deep neural network as a substitute to dynamic programming to solve stochastic control problems directly. They do not have to compute the value function and only aim at minimizing the objective function by mapping the states into actions. This way, they avoid space discretization and can approximate the solution to a high-dimensional stochastic control problem. They write that deep neural networks may circumvent the curse of dimensionality in some stochastic control problems.

If we do not introduce a deep neural network, we also aim at minimizing an objective function directly, without specifying the value function nor the Euler equation. Indeed, inspired by the Maliar, Maliar and Winant, 2019[19], we use a very simple and parsimonious algorithm : the lifetime-reward maximization. This technique chooses a policy function and simulates a lifetime using it, to compute the discounted life-time utility in t. Thanks to machine learning tools and libraries, it updates the policy function towards optimality parameters and simulate a lifetime again, until we reach a local minimum. One of the examples used in this paper is the consumption-saving problem with liquidity constraints. We will replicate the model and use lifetime-reward maximization to find the best rule of thumb, before studying finite horizon and modifying the model. This method is only one of the three the paper studies, since it aims at giving an overall view of the question of the use of artificial intelligence to solve economical problems. The main appeal of the techniques they use is that they do not require model-specific code and can approximate many models. Nevertheless, if they can solve high-dimensional problems, they doubt machine learning can solve the curse of dimensionality because of the unavoidable trade-off between satisfying results and a satisfying accuracy criterion.

3 Models and results

3.1 The algorithm : lifetime-reward maximization

As in Maliar, Maliar and Winant, 2019[19], we first compute the mean of estimated lifetime rewards produced by series of random shocks and following a given decision rule, characterized by the parameters we want to find. Then, we compute the gradient of this rule with respect to the decision rule parameters, and update the latter with a chosen learning rate. We repeat this process until a criterion is reached, namely until the mean distance of the gradient to zero is arbitrarily small.

3.1.1 Lifetime-reward

Let x_t be the endogenous state in t, with an endogenous component depending on the choice of the precedent period a_{t-1} and an exogenous component z_t, that can be purely stochastic in the first model or have a deterministic trend and a stochastic component in the second model. We note the stochastic component ϵ_t.

Let T be the chosen horizon, be it arbitrary in the infinite horizon model or model-determined in the finite horizon model.

Let s be a vector constituted of an initial state and a serie of random variable so that $s = (x_0, \epsilon_1, \epsilon_2, \dots, \epsilon_T)$

For a given s, the lifetime-reward writes :

$$LR_s(\theta) = \sum_{t=0}^{T} \beta^t r(x_t, \phi(x_t, \theta))$$

With ϕ the policy function characterized by the estimated parameters vector θ, so that $\phi(x_t, \theta) = a_t$, and r the reward function.

3.1.2 The training step with batch gradient descent

The training step aims at modifying the parameter vector θ until a satisfying form of the policy function is found. For this matter, we use the gradient descent technique, that is we compute the gradient of θ with respect to the lifetime reward and update θ by withdrawing a fraction of the gradient, determined by a learning rate. If this technique dates back from Newton, the computation of the gradient can be very demanding for complex problems. The development of artificial intelligence and of dedicated libraries allows to compute a great number of derivatives in a single step thanks to the tensor object.

The precise technique we will use is batch gradient descent. At each step, we draw N number of vector s and simulate N lives. We compute the mean empirical lifetime-reward :

$$MLR(\theta) = \frac{\sum_{t=1}^{N} LR(s_t))}{N}$$

We then update the parameter vector θ by the mean gradient. This allows to minimize the effect of extreme draws and to converge more effectively towards the solution.

So at each step (called an epoch) k, we do :

$$\theta_k = \theta_{k-1} - \lambda_k * \frac{\sum_{i=1}^{N} \Delta_{\theta_{k-1}} LR_{s_i}(\theta)}{N}$$

Where $\Delta_{\theta_{k-1}} LR_{s_i}(\theta)$ is the gradient of θ_k with respect to the lifetime reward, and λ_k is the learning rate at step k.

3.1.3 Hyperparameters and regularization

A crucial aspect of machine learning is the determination of the hyperparameters.

Here, the hyperparameters are :

- The size of the batch N. There is a trade-off between choosing a large batch, which will be more precise but will slow the algorithm down and a smaller batch. We will different size of batches. Note that is $N = 1$, we talk of stochastic gradient descent.
- The learning rate λ and its dynamics, that is whether it is constant during the training, and if not how it updates. Since our state space of initial condition is large, we choose a large at first ($\lambda = 0.5$ or $\lambda = 0.1$) but rapidly reducing learning rate (diminishing of one percent at each step).
- The convergence criterion c, below which the batch gradient descent stops, and the regularization parameters. We will choose different convergence criterions depending on the problem. We discuss below our regularization technique.
- A maximum number of epochs $e = 300$ to avoid infinite search in far from optimal regions.

In the machine learning field, regularization is checking that the parameters found after the gradient descent are not dependant on the sample on which they were found. In supervised learning, where we train the algorithm to find a known result with the data, it means splitting the training sample in two, so that the result can be tested on fresh data. Here, we compute the mean gradient of the parameters vector on a great number of simulations and check whether it is below the regularization parameter. This allows to disregard solution that performed below a convergence criterion by luck. The regularization parameters r are therefore the number of checking simulations to perform and a convergence criterion that can be different, often a bit higher. We choose the number of estimation $r = 1000$ and a regularization parameter $= 2c$.

Algorithm 1 Algorithm. Liftetime-reward maximization

```
0: procedure Initialization
  Define the policy rule $\phi(x_t, \theta)$.
  Define the reward function $r$.
  Define the mean lifetime-reward $MLR$.
  Initialize the economical parameters.
  Initialize the hyperparameters.
  Initialize randomly the parameter vector $\theta$.
  Define the distribution of the random variable $s$. We simulate $U$ periods
  following the starting rule to get a relevant random starting wealth $x_0$.
  procedure Training
  while Regularization parameter $\rho$ is not reached do
    while The convergence criterion $c$ is not reached do
      Simulate $N$ draws of $s$ and $N$ lives in which the agent follows $\phi_(x_t, \theta)$.
      Compute the mean lifetime-reward $MLR$.
      Compute the gradient of $\theta$ with respect to $MLR$.
      if the maximum number of epochs e is reached. then
        break
      else
        Update $\theta$ and the learning rate.
      end if
    end while
    Simulate $r$ lives with a fixed optimized policy rule and compute the gradient
    of $\theta$ with respect to $MLR$;
  end while
  =0
```

3.2 The model in infinite horizon - Allen and Carroll

3.2.1 The model

We consider a consumption-saving problem with infinite horizon and borrowing constraint *à la* Allen and Carroll. The utility is therefore CRRA and of the form :

$$U(C) = \frac{C^{1-\alpha}}{1-\alpha}$$

The problem of the agent is to maximize her intertemporal utility flow. It writes, normalizing the current period to time 0 :

$$\max_{\{C_0\}_0^\infty} \sum_{t=0}^{\infty} \beta^t U(C_t)$$

s.t.

$$X_{t+1} = X_t - C_t + Y_{t+1}$$
$$C_t \leq X_t, \forall t$$

Where X_t is the cash-on-hand, or money at the disposition of the agent in t.
Y_t is the income in t. Here, as in Allen and Carroll, it takes the value $Y_t = [0.7, 1, 1.3]$, with respective probability $p = [0.2, 0.6, 0.2]$.
The second constraint captures borrowing constraints. If a consumer can save all her income, she cannot consume more than she possesses.

Allen and Carroll approximate the optimal consumption rule with the rule of thumb :

$$C(X) = \begin{cases} 1 + \gamma(X - \bar{X}) & \text{if } 1 + \gamma(X - \bar{X}) \leq X \\ X & \text{if } 1 + \gamma(X - \bar{X}) > X \end{cases}$$

We implement this rule of thumb, with $\bar{X}$ the buffer stock target and γ the speed factor at which the consumer wishes to reach her target.

The idea is that we derive the level of consumption from the distance to the buffer-stock $\bar{X}$. If the level of cash-on-hand is higher than the buffer stock, the consumer dissaves and consumes more than the average income. Else, the consumer saves and consumes less than the average income. This rule is subject to the borrowing constraint : if the computed level of consumption is higher than the money available, the consumer simply consumes all of it.

We parameterize the model following Allen and Carroll, so that $\beta = 0.95$ and $\alpha = 3$. We fix arbitrarily $T = 100 < \infty$, considering the weight of the 101st period and beyond are negligible ($\beta^{101} \approx 0.006$).

3.2.2 The estimation

Here, the reward function is the utility function and the policy function is the rule of thumb. Therefore we can rewrite the lifetime reward :

$$LR_s(\theta) = \sum_{t=0}^{T} \beta^t U(C(x_t, \theta))$$

Where $\theta = \{\gamma, \bar{X}\}$ is the vector of the estimated parameters.

Since the algorithm does not find the unique solution of the problem but rather an approximation, we run it 50 times and compute some statistics. We then compare those results with Allen and Carroll and compute the sacrifice value, comparing the found approximation with the dynamic programming approximation.

3.2.3 Distribution of the random parameters

Following the search in Allen and Carroll, we define the distribution of the starting value of θ as uniform in the space they choose. So $\theta_0 = [0, 1]$ and $\theta_1 = [1, 3]$. We notice that this distribution causes some disturbance, as the behavior of the algorithm can be unpredictable for absurd values. Indeed, if γ and $\bar{X}$ are close to their upper bound, the gradient descent might go to regions far from the solution. We would not have this problem if we restricted the distribution space. Nevertheless, we stick to the Allen and Carroll paper and add a maximum number of epochs to solve this issue, with the help of the regularization.

If the vector of random income $\{Y_t\}_t^\infty$ is drawn following the previously stated income distribution rule, there is a question on the initial level of wealth at which the agent begins her optimization. Following Carroll, 2001,[4], we simulate 10 periods previous to the optimization algorithm beginning with an initial wealth of 0. The wealth of the agent after this period is the initial wealth x_0. If the agents follow the optimal rule in this paper, we rather make them follow the random rule they drew, before any optimization. This way, our algorithm remains naive and does not use any dynamic programming result. Then, we can construct our vector s, and begin the optimization.

3.2.4 The sacrifice value

To get an idea of the fitness of a rule, we compute the sacrifice value. It gives the amount of wealth a consumer would sacrifice to avoid being forced to switch from the optimal rule to a given rule of thumb. For this matter, we solve the problem with dynamic programming by state space discretization and backward induction and find an approximation of the optimal value function $V^*(X)$. We display our code in Appendix 2. We can then compute the value function following a rule of thumb $V_\theta(X)$, and look for the ϵ so that $V_\theta(X) = V^*(X - \epsilon)$. We approximate the ergodic distribution of cash-on-hand by simulating ten periods a thousand times with the found rule of thumb. Then, we compute the mean sacrifice value using every simulated point, which gives more weight to the more common values of cash-on-hand.

3.2.5 Results

Appendix 1 displays the code used to obtain the results.

We recapitulate the results of our estimation in table 1. We see that the mean value of our two parameters are close to the minimizing sacrifice value parameters found in Allen and Carroll. Since the parameter vector θ was found without computing the value function and backward solution, we consider its approximation a success. Furthermore, the standard deviation of the parameters are very low, and the maximum and minimum found for each parameters are both satisfying : we find no absurd approximation, as the minimum and maximum of the sacrifice values show.

	Mean value	St. d.	Maximum	Minimum	A and C
θ_0	0.2333	0.0066	0.2490	0.2205	0.233
θ_1	1.2434	0.0055	1.2550	1.2316	1.243
Sacr. val.	0.0221	0.0073	0.0239	0.0204	0.003

Table 1: Computations statistics

Figure 1 shows the rule of thumb compared to the optimal solution. We see that the distance of the estimated rule of thumb to the optimal rule is small for a cash on hand lower than 2, which is most of the time the case in this model. Figure 2 shows the fifty estimated rules to make the spread appear – we see that it is relatively negligible for a cash on hand lower than 2. Figure 3 shows a cycle of an agent that follows the mean value rule.

Since we want to study the process of lifetime reward maximization with machine learning, we display hereafter some convergence graphs, depending on the hyperparameters.We study the impact of three hyperparameters : the learning rate, the convergence criterion and the batch size. We do not perform regularization here, since we aim at studying the mechanism and not find the best rule. We study three initial rules :

$$\theta^1 = \{0.5, 1.5\}$$
$$\theta^2 = \{0.01, 2\}$$
$$\theta^3 = \{0.99, 1\}$$

Those initial states are chosen as standard for the first one and extreme cases

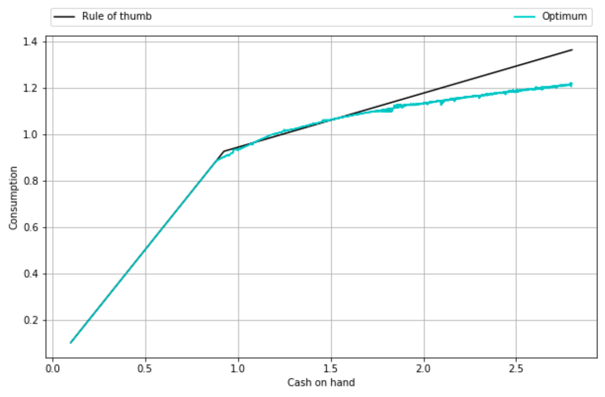


Figure 1: The estimated optimal function and the rule of thumb

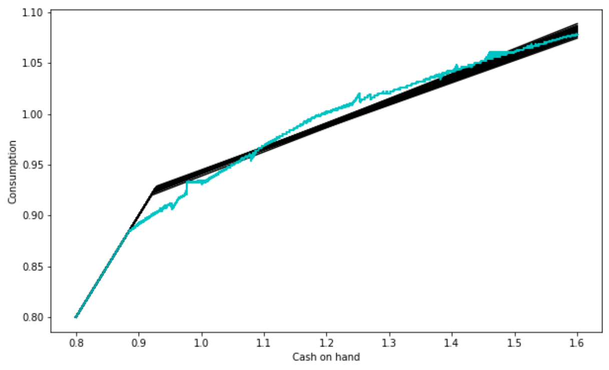


Figure 2: The estimated optimal function and all the estimated rules of thumbs

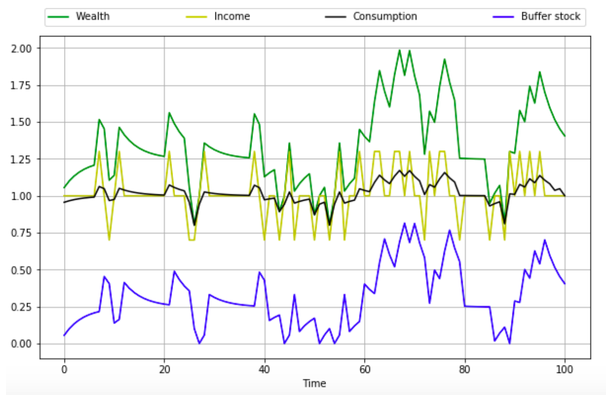


Figure 3: An example of 100 periods using the estimated rule of thumb

for the others. The second one can be seen as the rule in which the agent always tries to consume her mean income, and the third one as the rule in which she consumes all her wealth. Note that we cannot start at $\theta_0 = 0$ nor $\theta = 1, 1$, because the algorithm would not use the tensor θ in this case and the gradient would equal 0. More generally, if we compute an epoch with a θ implying that the agent only consumes all of her wealth during every period of her cycle, the gradient will be of zero, stopping the algorithm at a far from optimal point.

First, we set a large batch size of 10000. If this slows the algorithm down – the average duration of an epoch will be of 0.4 seconds, it allows to minimize the fluctuations. Therefore, the choice of the convergence criterion will be straightforward : it should be low. A high convergence criterion would make the approximation less precise, but the size of the batch allows to smooth across a large number of draws and we should therefore reach a precise convergence criterion when we are close to the best approximation. For this reason, the learning rate can be relatively high and rapidly decreasing. Figure 4 shows the behavior of θ depending on its starting point with a learning rate of 0.1 decreasing by one percent at each epoch, and a convergence criterion of 0.01. The numbers corresponding to the curves are the sacrifice value of each final rules.

We see that we always converge in less than thirty epochs. We even sometimes approximate a good rule in less than 5 epochs. We notice the convergence is stable for those starting points, except maybe for the second one. Indeed, the algorithm may diverge if $\theta_0 \leq 0$ because it would mean that one should save if above the buffer stock target and dissave if under. It is almost the case here. It would be the case for a higher θ_0, which explains why we choose this starting point. Note that a lower learning rate could keep us from diverging.

Nevertheless, if choosing a large batch size favors convergence, it is at the cost of the rapidity of the algorithm. A computation time of half a second for a single epoch is quite considerable. If it allows us to find a satisfying approximation in very few epochs, it cannot always be the case, especially if we try to approximate more parameters, as we will do in the next section. Therefore, we then study the case where the batch size is of one. The computation time of an epoch is then almost 10 times faster : it is approximately of one twentieth of a second (0.06 seconds). This technique is the stochastic gradient descent, which is unbiased because the average of the gradient at a given step converge to the average of the true gradient. However, each step is also very imprecise because it depends a lot on the random vector drawn. As a result, we must pick a low and stable learning rate. We choose a learning rate of 0.01 decreasing by 0.1 percent at each epoch.

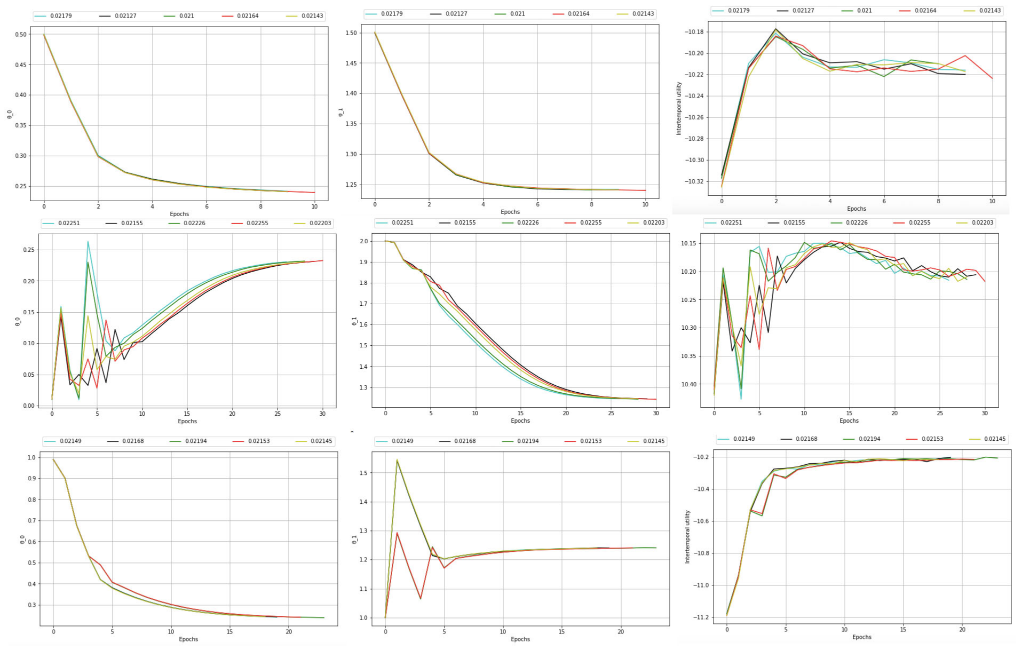


Figure 4: $N = 10000$; $\lambda = 0.1$

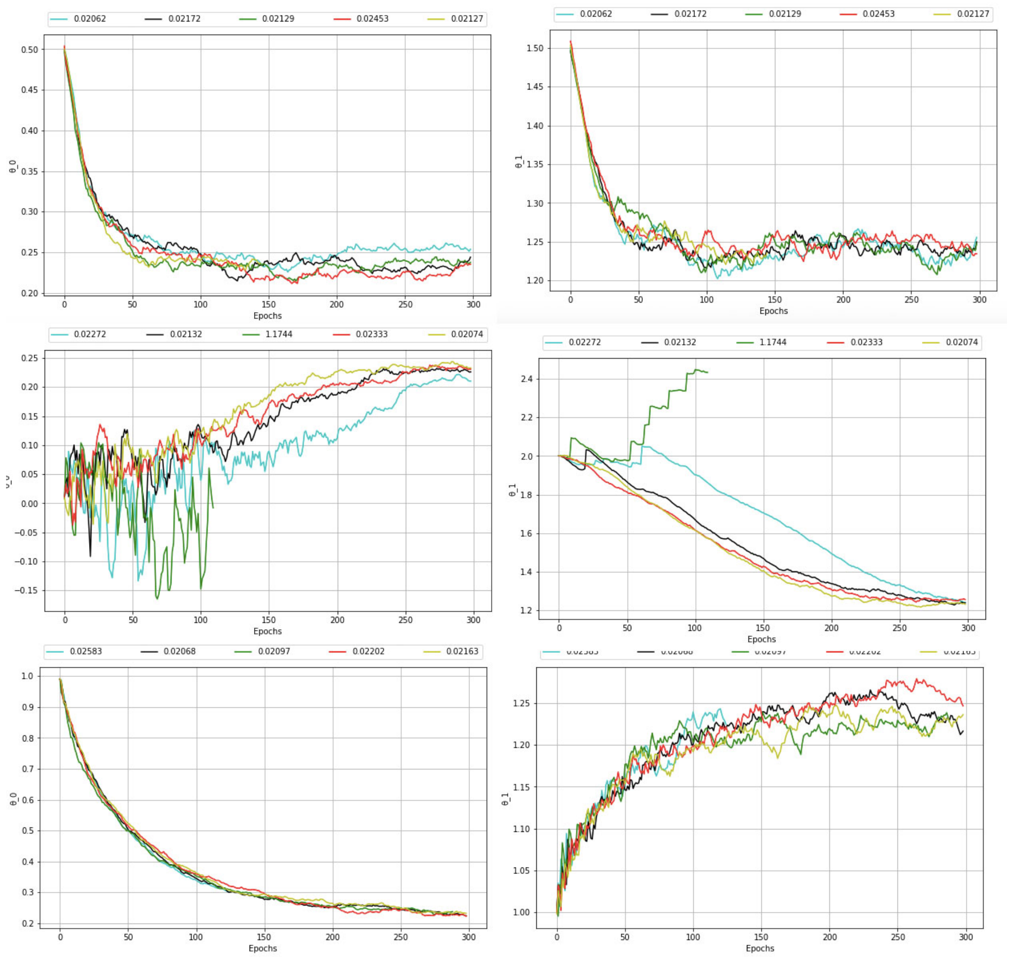


Figure 5: $N = 1$; $\lambda = 0.01$

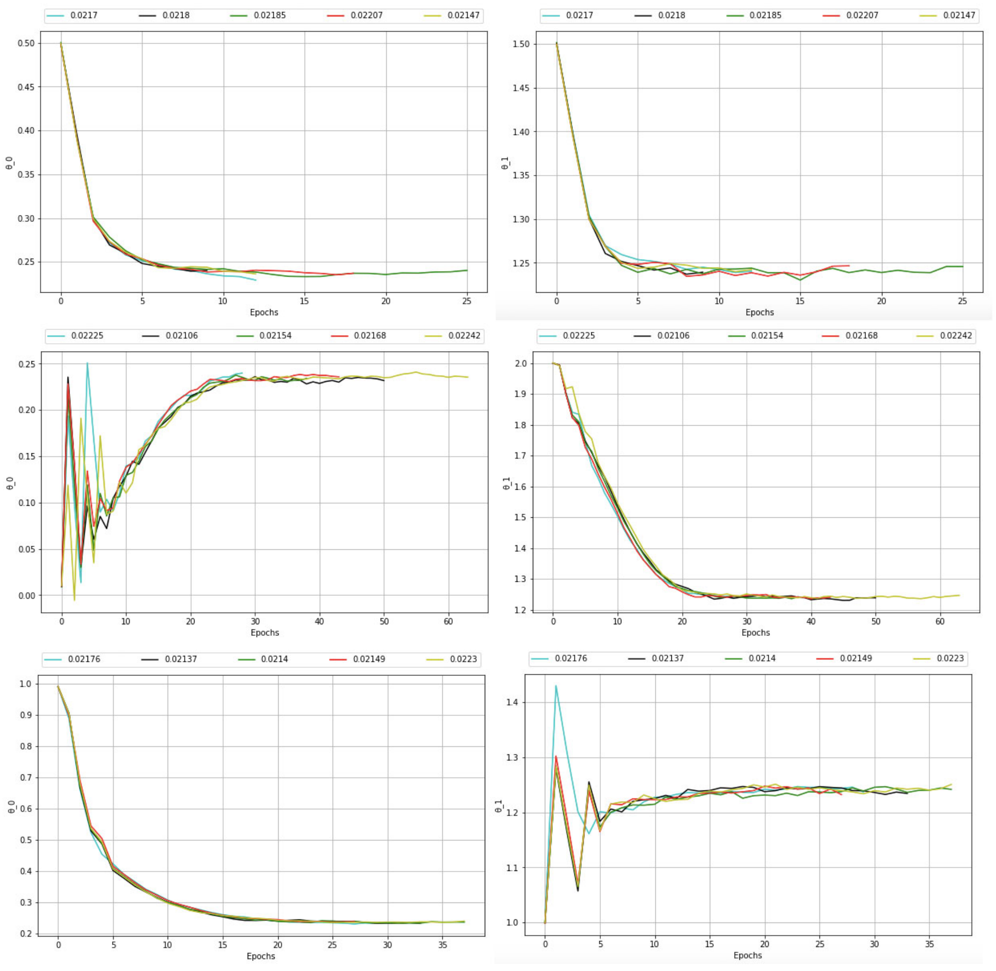


Figure 6: $N = 100$; $\lambda = 0.1$

Furthermore, it is unlikely that we find a gradient that fits a tight convergence criterion even in the best approximation zone since there is a high level of randomness at each step. On the other hand, if we choose a high convergence criterion, we might end the algorithm too soon and find an approximated rule too dependant on the initial rule. So we decide to keep a tight convergence criterion and end the algorithm after 300 epochs, because we see that by that time the approximation should be good enough.

Figure 5 shows the convergence graphs for the different initial rules. We do not plot all the utility graphs because we see that they does not make sense here, since the utility is mainly caused by the luck in the simulated cycles. As expected, we see that the sacrifice value of the rules estimated after 300 epochs and that did not reach the convergence criterion are still close to the best approximations. On the 15 estimations, only one did not converge, with an initial $\theta = \theta^2$. The unexpected convergence path of the stochastic gradient descent makes this initial point unstable since the initial θ_0 is almost of 0, and the behavior of the rule with $\theta_0 \leq 0$ is very unstable.

Finally, we run the algorithm with a batch size of 100. It allows both to have a steady convergence and a fast execution. Indeed, computing parallelization of the array object makes the computation time of a batch of 100 to be almost the same as a stochastic gradient descent (approximatively 0.07 seconds). We choose a learning rate of 0.1 decreasing at the rate of one percent per period, as in the large batch simulations.

The results are similar, but not only is the computation faster, we see that there are more fluctuations among the simulations. This is positive as it means the starting point is not as deterministic as it is for a bigger batch. This size of batch since therefore to be a good answer to the trade-off between a large and a small batch.

3.2.6 Learning

With their solution concept, Allen and Carroll find that the agent must live through 4 million periods to find a good enough rule (with a sacrifice value of less than 0.05) 80 percent of the time. If they imagine that a hill climbing routine could converge much faster towards a good rule, they state however that there is a minimum of 2500 periods for testing a given rule that cannot be lowered, so that it would still take 10000 periods even if only 4 rules had to be tested.

But the stochastic gradient descent allows to fluctuate without needing to evaluate the true gradient of a rule, so that we do not need 2500 periods to modify the rule but only one cycle. Stochastic gradient descent is a good candidate for individual learning, since the agent lives a cycle and then updates her consumption rule so that she would have had a higher utility in that cycle. We defined a cycle to be of 100 periods, but we can lower it to 50 without altering the optimum much. With such a cycle length, we compute twenty stochastic gradient descent for initial rules $\theta^1 = \{0.5, 1.5\}$ and $\theta^3 = \{0.99, 1\}$. We recapitulate the results in Table 2.

We see that if we start the algorithm at a reasonable point, only 25 epochs are enough to approximate a good rule almost every time with a sacrifice value lower than 0.05. That is a total of 1250 periods. Furthermore, if we lower the number of periods per epoch to 40, we move apart from the optimum but it allows to find a good rule in less than 1000 periods. Even if we start at an extreme point, that is the rule consume your wealth , we can approximate a good rule in 5000 periods 75 percent of the time – the failures are not caused by time but because we fluctuated to a degenerated point that stopped the algorithm.

Nb. of epochs	Acceptable rules	St. d.	Periods to a good rule
θ^1	25	95%	1250
θ^3	100	75%	5000

Table 2: Individual learning with machine learning

On the other hand, batch gradient descent could be a good candidate for social learning. Allen and Carroll propose this mechanism as a reasonable description of the learning of the agents. We can conceive social learning as a social group that keeps track of itself and in which agents use not only their own experience but the experience of the whole group to modify their rule. We compute the statistics of twenty optimizations with a batch size of 50 in Table 3. This behavior produces a very fast convergence that is of only 3 epochs, or 150 periods. We note that it is still higher than the agent life, since the economical parameters are fitted over a one year period. However, it is hard to find faster results in this context. In their social learning framework, Salle and Seppecher, 2013[23] settles for a satisfying learning of 100 periods.

Nb. of epochs	Acceptable rules	St. d.	Periods to a good rule
θ^1	3	100%	150
θ^3	8	90%	400

Table 3: Social learning with machine learning

One may argue that this framework is not a relevant candidate to simulate the learning of the agent because of the difficulty of the computation of the gradient. Even so, those figures tell a lot on the rapidity and power of the machine learning techniques.

3.3 The model in finite horizon : The life-cycle

3.3.1 The model

We consider a similar problem but with finite horizon T. The agent lives a finite number of periods and knows when she will die. Therefore, she sees her income increase during her life and stabilize at a lower level after retirement. She solves the problem :

$$\max_{\{C_0\}_0} \sum_{t=0}^{T} \beta^t U(C_t)$$

s.t.

$$X_{t+1} = X_t - C_t + Y_{t+1}$$

$$C_t \leq X_t, \forall t$$

With the utility function again of the form :

$$U(C) = \frac{C^{1-\alpha}}{1-\alpha}$$

We keep the same economical parameterization as the previous model.

3.3.2 Income process

We calibrate the income process so that it resembles the operatives income profile computed in Carroll, 1997[3], based on the aggregate US data.

The income process writes :

$$Y_t = z_t + \epsilon_t$$

Where z_t is a deterministic component so that :

$$z_0 = 1$$

$$z_{t+1} = \begin{cases} z_t + 0.025 \text{ for } t < R \\ 1.4 \text{ for } t \geq R) \end{cases}$$

And ϵ_t is the stochastic component, and we have with probability p=[0.15,0.7,0.15] :

$$\epsilon_t = \begin{cases} [0.5, 1, 1.5] \text{ for } t < R \\ 0 \text{ for } t \geq R) \end{cases}$$

R is the age of retirement.

Note that we enlarged the spread of the stochastic component and reduced the probability of an extreme realization in the same time. If we do not, the increase of the deterministic component of the income would make the bad shocks acceptable because of the form of the utility function, and a buffer stock behavior would not appear.

3.3.3 The sacrifice values

In order to evaluate our results, we compute an approximation of the dynamic programming problem, obtained by state space discretization and backward induction. We find a consumption function in function of the cash on hand and a value function value for every state at each period. We display our code in Appendix D. This allows us to compute the sacrifice value of a given rule by comparing the optimal value function to the value function when following the rule of thumb, as in the previous section.

We fix $T = 55$ and $R = 40$. The agents therefore live 40 years working (from age 25 to 65) and then 15 years of retirement (from age 65 to 80) after which they die.

3.3.4 Rules of thumb and switching date

As stated in Carroll, 1997[3], and Gourinchas and Parker, 2002[14], there are two dominant saving motives in the life cycle of the agents : precautionary saving and retirement saving. The precautionary motive which produce a buffer stock behavior is dominant in the early life, while retirement saving become dominant at a certain age and the behavior of the agent resemble more a certainty equivalent model.

Thus, a fixed simple form for the rule of thumb could not capture this change in behavior. We choose that the agents will rather use two distinct rules depending on their age : they will switch from one rule to another at a given date D. The first rule should resemble the one found in the infinite horizon problem. Therefore, before D, the agent follows the consumption rule :

$$C^1(X) = \begin{cases} z_t + \gamma(X - (z_t + \bar{X})) & \text{if } z_t + \gamma(X - (z_t + \bar{X})) \leq X \\ X & \text{if } z_t + \gamma(X - (z_t + \bar{X})) > X \end{cases}$$

This rule is the same as in the previous section, except the mean value of the income is no more 1 but increases. This is what captures the deterministic trend in the income of the agent z_t in the rule.

After the switch, the new rule should be of the type "permanent income". There is no more buffer stock target. We define it as a proportional fluctuation from a mean level of consumption :

$$C^2(X) = \begin{cases} P + \gamma * X & \text{if } P + \gamma * X \leq X \\ X & \text{if } P + \gamma * X > X \end{cases}$$

Where P is a component capturing the level of income from which the agent wants to deviate. Thus, the consumer will slowly consume all of her savings after retirement. A high P and γ would indicate impatience from the consumer, especially if the rule begins before retirement, whereas low values would indicate measure and patience.

We therefore estimate the rule :

$C^{\theta}(X) =$ if $t < D$:

$$\begin{cases} z_t + \theta_0(X - (z_t + \theta_1)) & \text{if } z_t + \theta_0(X - (z_t + \theta_1)) \leq X \\ X & \text{if } z_t + \theta_0(X - (z_t + \theta_1)) > X \end{cases}$$

if $t \geq D$:

$$\begin{cases} \theta_2 + \theta_3 * X & \text{if } \theta_2 + \theta_3 * X \leq X \\ X & \text{if } \theta_2 + \theta_3 * X > X \end{cases}$$

It is obvious that the choice of D will be crucial, as we make the agents brutally change from one rule to another. Carroll, 1997[3], and Gourinchas and Parker, 2002[14], suggest that it is around 10 years before retirement. However, we test different switching dates D and compare the estimations with the help of the sacrifice values.

3.3.5 Results

We optimize the two rules in a single step. However, at date D, not only do we switch rules, we also reset the discount factor. This captures the reflection of the agent who chooses a rule at the beginning of her working life and another at D. It is as if the agent consciously decides to change her behavior at that age because she gets closer to retirement. Therefore, if we have two different rules, they both have an impact on the other : they are not independent. It is in fact a single non-linear rule.

Here, the sacrifice values will help us not only to compare different hyperparameters but also to discriminate for the best switching date D.

Based upon last section results, we fix $\lambda = 0.1$ with a discount rate of 0.99. We also choose a batch size of 100 and a low convergence criterion c = 0.003 since we rather find the best possible rule than a fast convergence. This low criterion will also allow us to escape local minima and to find the same rule from any starting estimation point. Indeed, we find similar rules for any reasonable starting points.

To find the best rule and the best switching date, we follow the following protocol :

- Select four switching dates $D = \{25, 30, 35, 40\}$ and compute the best rules for each of them by averaging over 10 optimizations.
- Select the switching date with lowest mean sacrifice value and compute the rules near it.
- Select the best switching date and the best rule.

We report in Figure 7 the sacrifice value of the best approximated functions at switching date 25, 30, 35 and 40. We see that $D = 35$ is the best candidate, so we then compute for the near dates. Figure 8 shows this computation.

By minimizing the mean sacrifice values, we find $D = 37$ to be the best candidate. This is only 3 years before retirement, so considerably less than in previous work. This might be caused by the model and notably the small length of life after retirement. However, the behavior is comparable. The mean sacrifice value is extremely low : 0.02682. This result is dependent on the values for which we computed the sacrifice, but we still consider it as very encouraging.

The corresponding θ is $\theta^* = \{0.2599, 0.4311, 1.2422, 0.1560\}$.

The first rule is therefore very similar than the one found in the infinite horizon model. The buffer stock target is only higher. It is of 0.43 above the mean income, when it was of 0.23 above in Allen and Carroll. It can be explained by the fact that the shocks are harder.

The second rule floor consumption is very low. At date D, it is lower than the mean income (1.925) by approximately 0.7. This allows a little drop in consumption to accumulate asset before retirement. The form of the rule makes it almost uncorrelated to income, as we can see in figure 9. The agent consumes her floor income plus a fixed percentage of her wealth. This second part is decreasing since wealth decreases $(1.24 + 0.156 * 1.4 > 1.4)$, so that the consumer consumes less and less until she has no more asset at death.

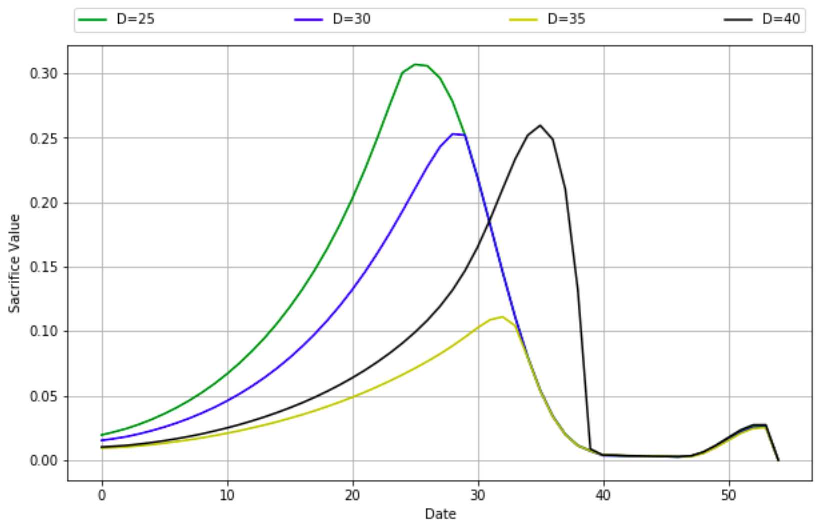


Figure 7: Sacrifice value of the approximated rules with different switching dates

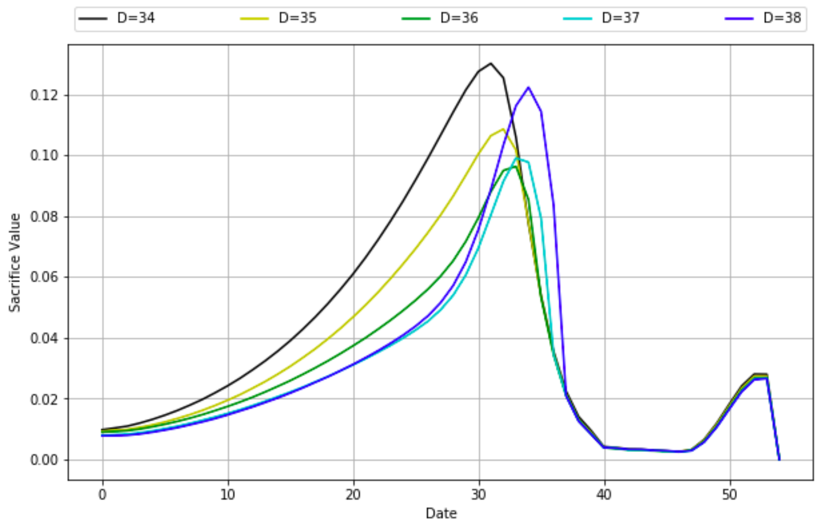


Figure 8: Sacrifice value of the approximated rules with different switching dates, a closer look

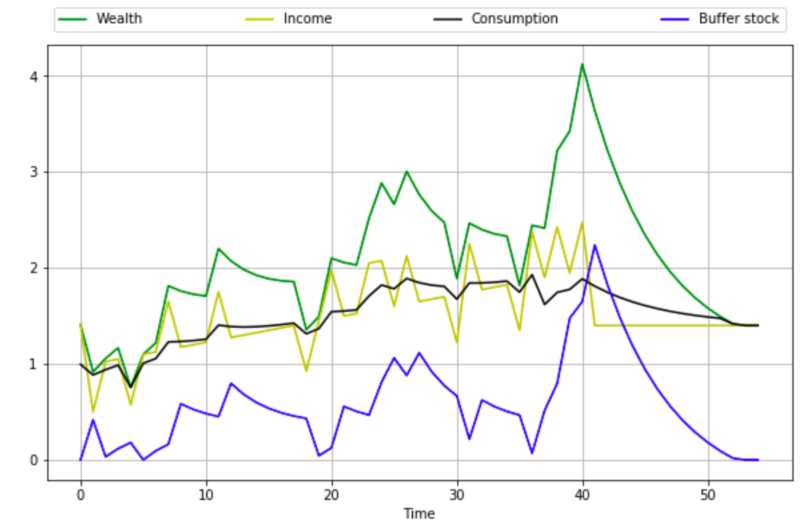


Figure 9: Simulation of a life-cycle following the estimated rule with $D = 37$

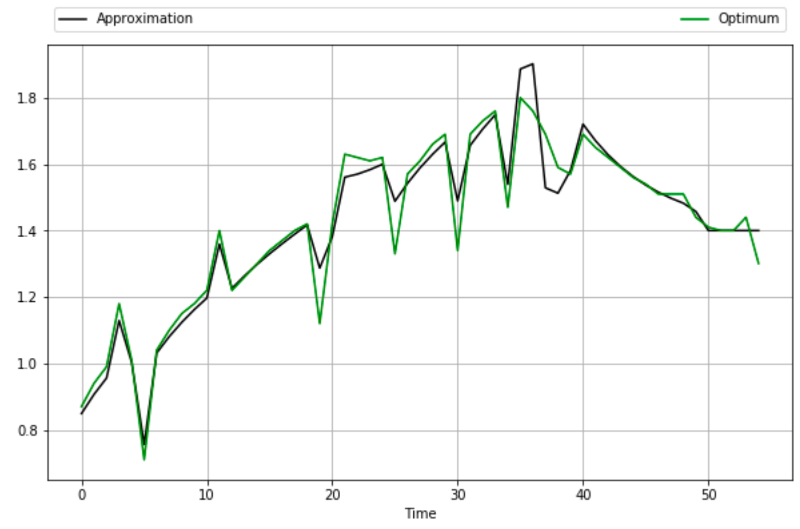


Figure 10: Comparison of the behavior of the optimal rule with the behavior of the estimated rule

4 Conclusion

The developments of machine learning in the past decades are enormous, thanks to new tools and increasing computation power. In economics however, if applications were made incorporating machine learning, we believe its full potential is yet to be exploited.

We proposed a very simple and tractable technique that allows us to solve economic models straightforwardly. By only the specification of the form of the decision rule, we can approximate the solution of complex models in a very parsimonious way.

We studied a standard framework, the consumption-saving problem. Plus, the concept of rule of thumb resembles the idea of approximate optimal rule, so that it gives us a form for the rule to approximate, as well as economic meaning. This made the paper by Allen and Carroll a perfect candidate to study. We did in fact achieve to approximate their optimal rule of thumb efficiently and with great speed. In a second time, we moved away from this model and their rule of thumb and tried to create one "from scratch" for a life-cycle model. Here again, we achieved to find a good approximation of the optimal rule and to give it economical meaning. However, we had to use the dynamic programming approximation to determine a switching date. A more complex form of the decision rule could have spared us this trouble.

Indeed, in our paper, we stuck to very simple decision rules. We showed that the number of parameters to approximate was not a fatality in machine learning. Therefore, we could use other decision rules such as deep neural networks. If they can approximate almost any functions, the difficulty to give meaning to the approximation function is a serious drawback. For further research, it would be interesting to develop a tractable neural network inside which we could look to find economic meaning. It being naive, it could shed a different light on standard economic problems.

References

[1] T. Allen and C. Carroll. Individual learning about consumption. *Macroeconomic Dynamics*, 5(2):255–271, 2001.

[2] C. Carroll.

[3] C. Carroll. Buffer-stock saving and the life cycle/permanent income hypothesis. *The Quarterly Journal of Economics*, 112(1):1–55, 1997.

[4] C. Carroll. Death to the log-linearized consumption euler equation! (and very poor health to the second-order approximation). *Advances in Macroeconomics*, 2001.

[5] C. Carroll. The method of endogenous gridpoints for solving dynamic stochastic optimization problems. *Economics Letters*, 91(3):312–320, 2006.

[6] Hall R.-Zeldes S. Carroll, C. The buffer-stock theory of saving: Some macroeconomic evidence. *Brookings Papers on Economic Activity*, 2:61–156, 1992.

[7] A. Deaton. Saving and liquidity constraints. *Econometrica*, 59:1221–1248, 1991.

[8] A. Deaton. Household saving in ldcs: credit markets, insurance and welfare. *Scandinavian Journal of Economics*, 94(2):253–273, 1992.

[9] A. Deaton. *Understanding consumption.* 1992.

[10] R. Dornbusch and S. Fischer. *Macroeconomics.* 1981.

[11] McNelis P. Duffy, J. Approximating and simulating the stochastic growth model: Parameterized expectations, neural networks and the genetic algorithm,. *Journal of Economic Dynamics and Control*, 25(9):1273–1303, 2001.

[12] Han J. E, W. Deep learning approximation for stochastic control problems. 2016.

[13] M. Friedman. *A theory of the consumption function.* 1957.

[14] Parker-J. Gourinchas, P.-O. Consumption over the life-cycle. *Econometrica*, 70:47–89, 2002.

[15] F. Hayashi. Tests for liquidity constraints: A critical survey. *Advances in Econometrics*, 1987.

[16] Özak Ö. Howitt, P. Adaptive consumer behavior. *Journal of Economic Dynamics and Control*, 39(C):37–61, 2009.

[17] H. Leland. Saving and uncertainty: The precautionary demand for saving. *The Quarterly Journal of Economics*, 82(3):465–473, 1968.

[18] Uhlig H. Lettau, M. Rules of thumb versus dynamic programming. *American Economic Review*, 99(1):148–174, 1999.

[19] Maliar S.-Winant P. Maliar, L. Will artificial intelligence replace computationnal economists any time soon ? *Working paper*, 2019.

[20] N. Palmer. Learning to consume : individual learning versus social learning. *Learning to consume : individual learning versus social learning*, 2012.

[21] W. Powell. A tour of the jungle of approximate dynamic programming. 2011.

[22] Schlafmann K. Rodepeter, R. and J. Winter. Rules fo thumb in life-cycle saving decisions. *The Economic Journal*, 122:479–501, 1999.

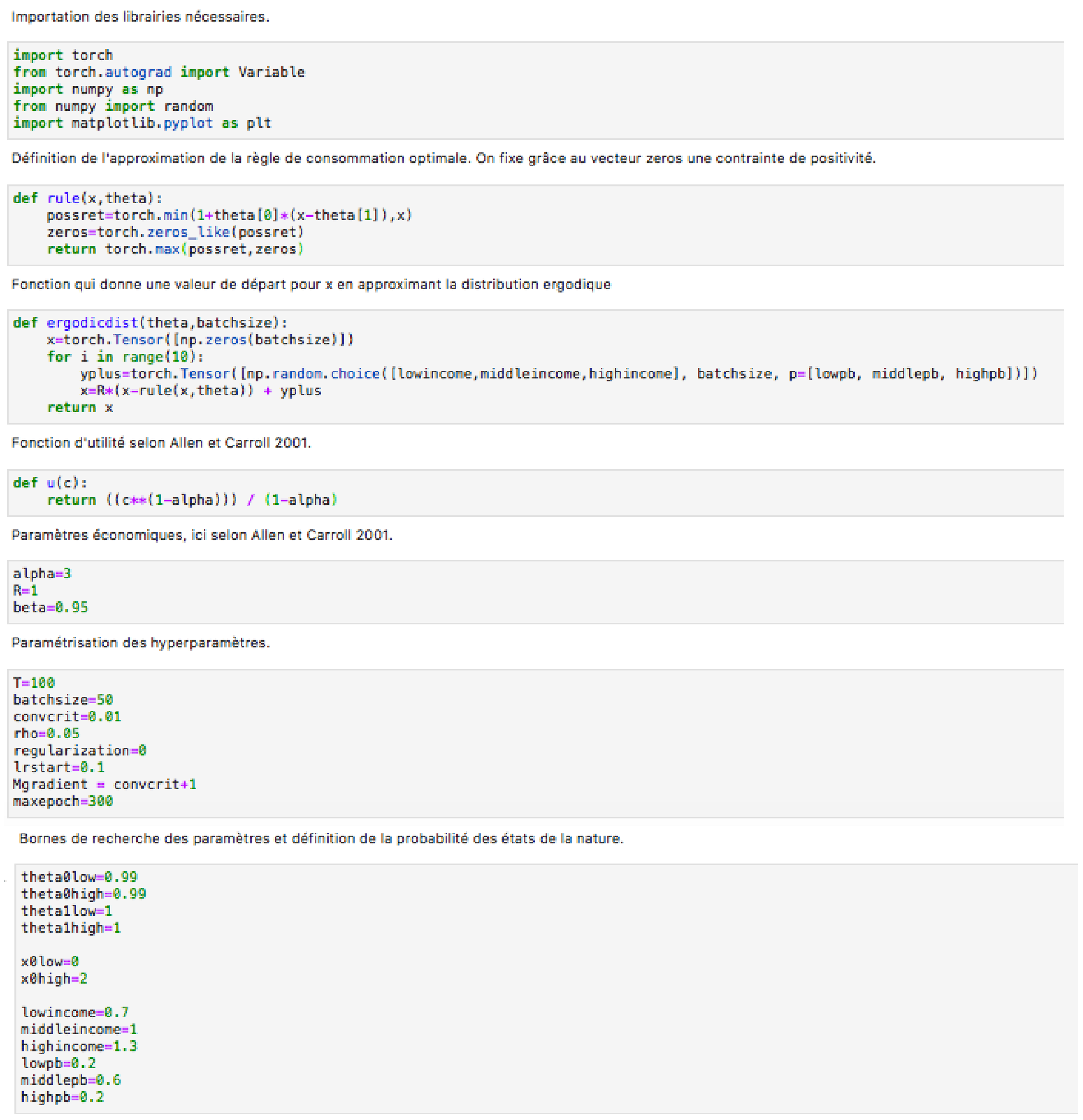
[23] I. Salle and P. Seppecher. Social learning about consumption. *Macroeconomic Dynamics*, 20(7):1785–1825, 2013.

[24] Senegas M.-A. Yildizoglu M. Salle, I. and M. Zumpe. Social learning about consumption. *Macroeconomic Dynamics*, 18(4):727–752, 2013.

[25] S. Zeldes. Optimal consumption with stochastic income: Deviations from certainty equivalence. *The Quarterly Journal of Economics*, 104(2):275–298, 1989.

Appendices

A Machine learning code to the infinite-horizon model



Importation des librairies nécessaires.

```
import torch
from torch.autograd import Variable
import numpy as np
from numpy import random
import matplotlib.pyplot as plt
```

Définition de l'approximation de la règle de consommation optimale. On fixe grâce au vecteur zeros une contrainte de positivité.

```
def rule(x,theta):
    possret=torch.min(1+theta[0]*(x-theta[1]),x)
    zeros=torch.zeros_like(possret)
    return torch.max(possret,zeros)
```

Fonction qui donne une valeur de départ pour x en approximant la distribution ergodique

```
def ergodicdist(theta,batchsize):
    x=torch.Tensor([np.zeros(batchsize)])
    for i in range(10):
        yplus=torch.Tensor([np.random.choice([lowincome,middleincome,highincome], batchsize, p=[lowpb, middlepb, highpb])])
        x=R*(x-rule(x,theta)) + yplus
    return x
```

Fonction d'utilité selon Allen et Carroll 2001.

```
def u(c):
    return ((c**(1-alpha))) / (1-alpha)
```

Paramètres économiques, ici selon Allen et Carroll 2001.

```
alpha=3
R=1
beta=0.95
```

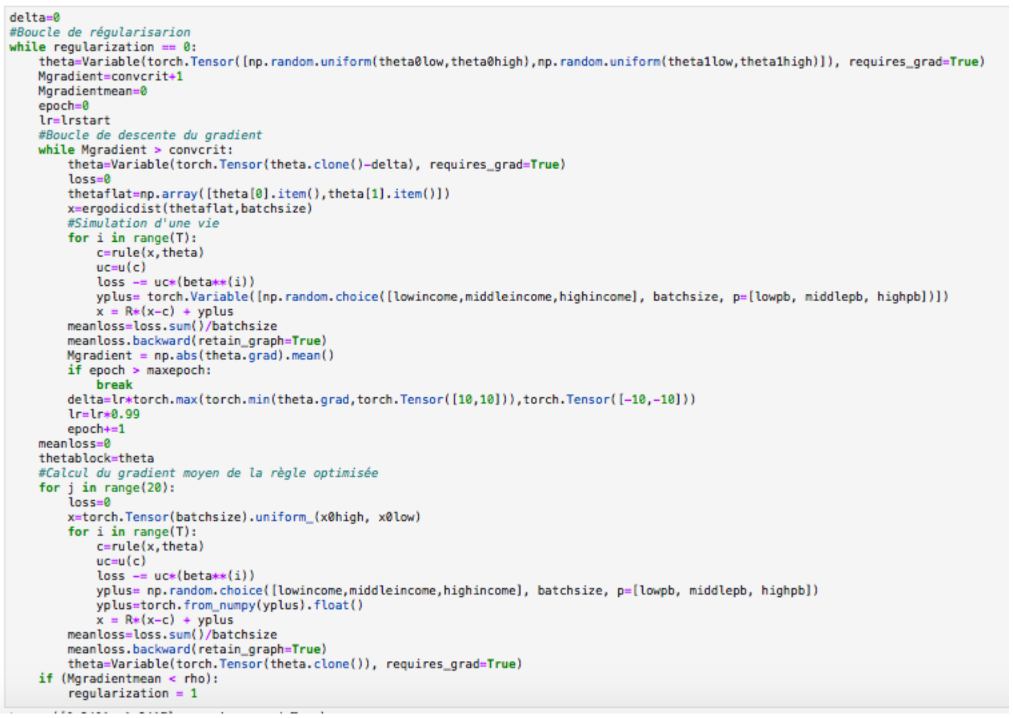
Paramétrisation des hyperparamètres.

```
T=100
batchsize=50
convcrit=0.01
rho=0.05
regularization=0
lrstart=0.1
Mgradient = convcrit+1
maxepoch=300
```

Bornes de recherche des paramètres et définition de la probabilité des états de la nature.

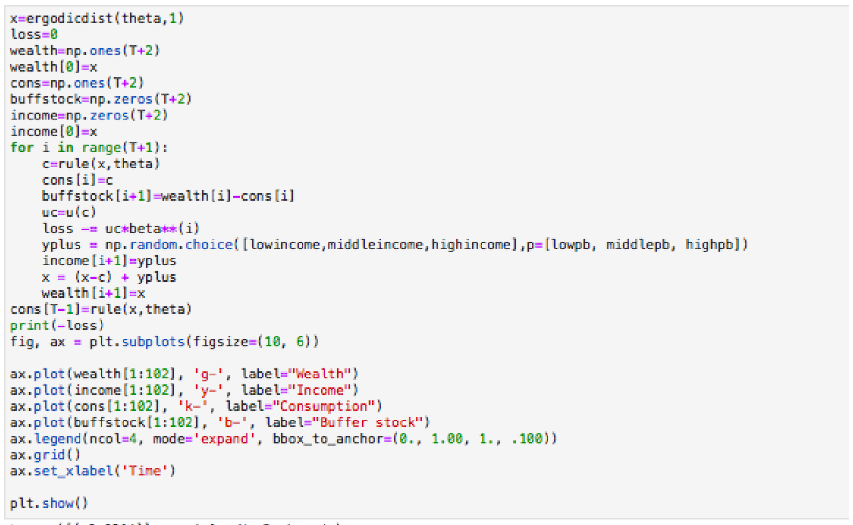
```
theta0low=0.99
theta0high=0.99
theta1low=1
theta1high=1

x0low=0
x0high=2

lowincome=0.7
middleincome=1
highincome=1.3
lowpb=0.2
middlepb=0.6
highpb=0.2
```

Algorithme principal



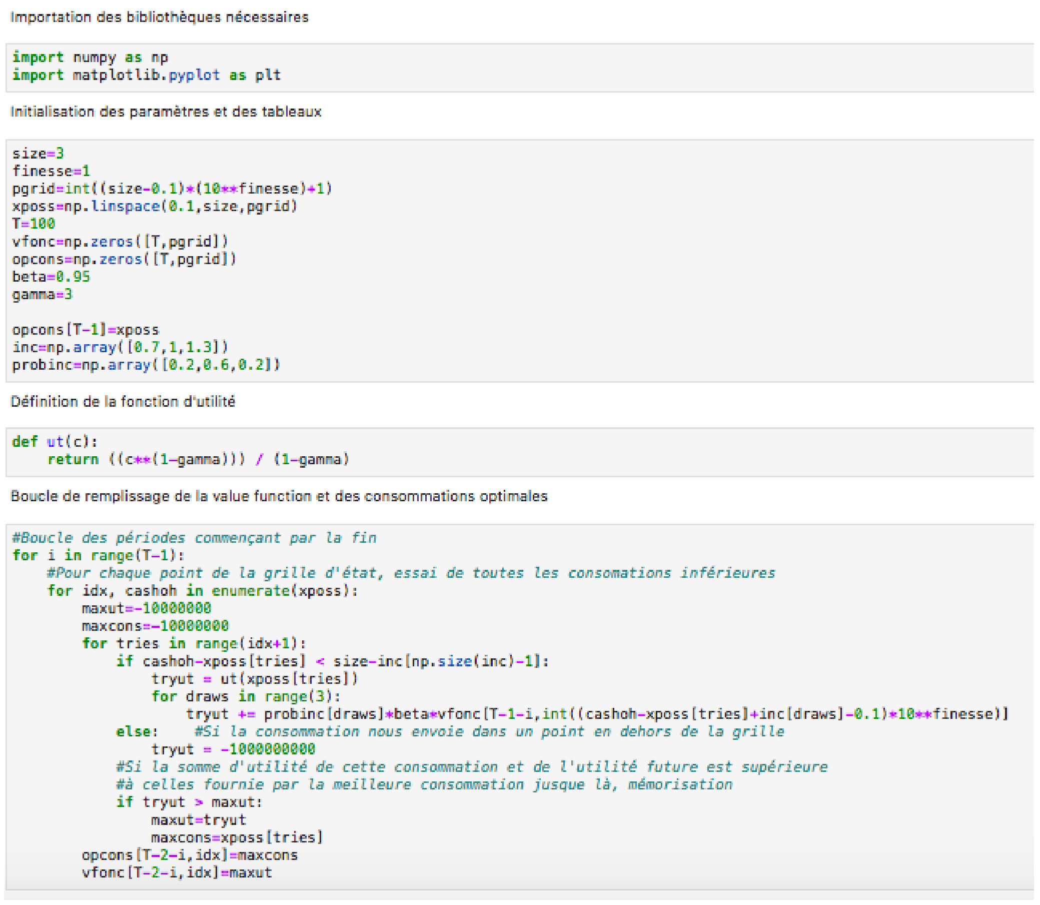
```python
delta=0
#Boucle de régularisarion
while regularization == 0:
    theta=Variable(torch.Tensor([np.random.uniform(theta0low,theta0high),np.random.uniform(theta1low,theta1high)]), requires_grad=True)
    Mgradient=convcrit+1
    Mgradientmean=0
    epoch=0
    lr=lrstart
    #Boucle de descente du gradient
    while Mgradient > convcrit:
        theta=Variable(torch.Tensor(theta.clone()-delta), requires_grad=True)
        loss=0
        thetaflat=np.array([theta[0].item(),theta[1].item()])
        x=ergodicdist(thetaflat,batchsize)
        #Simulation d'une vie
        for i in range(T):
            c=rule(x,theta)
            uc=u(c)
            loss -= uc*(beta**(i))
            yplus= torch.Variable([np.random.choice([lowincome,middleincome,highincome], batchsize, p=[lowpb, middlepb, highpb])])
            x = R*(x-c) + yplus
        meanloss=loss.sum()/batchsize
        meanloss.backward(retain_graph=True)
        Mgradient = np.abs(theta.grad).mean()
        if epoch > maxepoch:
            break
        delta=lr*torch.max(torch.min(theta.grad,torch.Tensor([10,10])),torch.Tensor([-10,-10]))
        lr=lr*0.99
        epoch+=1
    meanloss=0
    thetablock=theta
    #Calcul du gradient moyen de la règle optimisée
    for j in range(20):
        loss=0
        x=torch.Tensor(batchsize).uniform_(x0high, x0low)
        for i in range(T):
            c=rule(x,theta)
            uc=u(c)
            loss -= uc*(beta**(i))
            yplus= np.random.choice([lowincome,middleincome,highincome], batchsize, p=[lowpb, middlepb, highpb])
            yplus=torch.from_numpy(yplus).float()
            x = R*(x-c) + yplus
        meanloss=loss.sum()/batchsize
        meanloss.backward(retain_graph=True)
        theta=Variable(torch.Tensor(theta.clone()), requires_grad=True)
    if (Mgradientmean < rho):
        regularization = 1
```

Simulation d'une vie en utilisant la rule of thumb estimée



```python
x=ergodicdist(theta,1)
loss=0
wealth=np.ones(T+2)
wealth[0]=x
cons=np.ones(T+2)
buffstock=np.zeros(T+2)
income=np.zeros(T+2)
income[0]=x
for i in range(T+1):
    c=rule(x,theta)
    cons[i]=c
    buffstock[i+1]=wealth[i]-cons[i]
    uc=u(c)
    loss -= uc*beta**(i)
    yplus = np.random.choice([lowincome,middleincome,highincome],p=[lowpb, middlepb, highpb])
    income[i+1]=yplus
    x = (x-c) + yplus
    wealth[i+1]=x
cons[T-1]=rule(x,theta)
print(-loss)
fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(wealth[1:102], 'g-', label="Wealth")
ax.plot(income[1:102], 'y-', label="Income")
ax.plot(cons[1:102], 'k-', label="Consumption")
ax.plot(buffstock[1:102], 'b-', label="Buffer stock")
ax.legend(ncol=4, mode='expand', bbox_to_anchor=(0., 1.00, 1., .100))
ax.grid()
ax.set_xlabel('Time')

plt.show()
```

B Dynamic programming solution to the infinite-horizon model



Importation des bibliothèques nécessaires

```python
import numpy as np
import matplotlib.pyplot as plt
```

Initialisation des paramètres et des tableaux

```python
size=3
finesse=1
pgrid=int((size-0.1)*(10**finesse)+1)
xposs=np.linspace(0.1,size,pgrid)
T=100
vfonc=np.zeros([T,pgrid])
opcons=np.zeros([T,pgrid])
beta=0.95
gamma=3

opcons[T-1]=xposs
inc=np.array([0.7,1,1.3])
probinc=np.array([0.2,0.6,0.2])
```

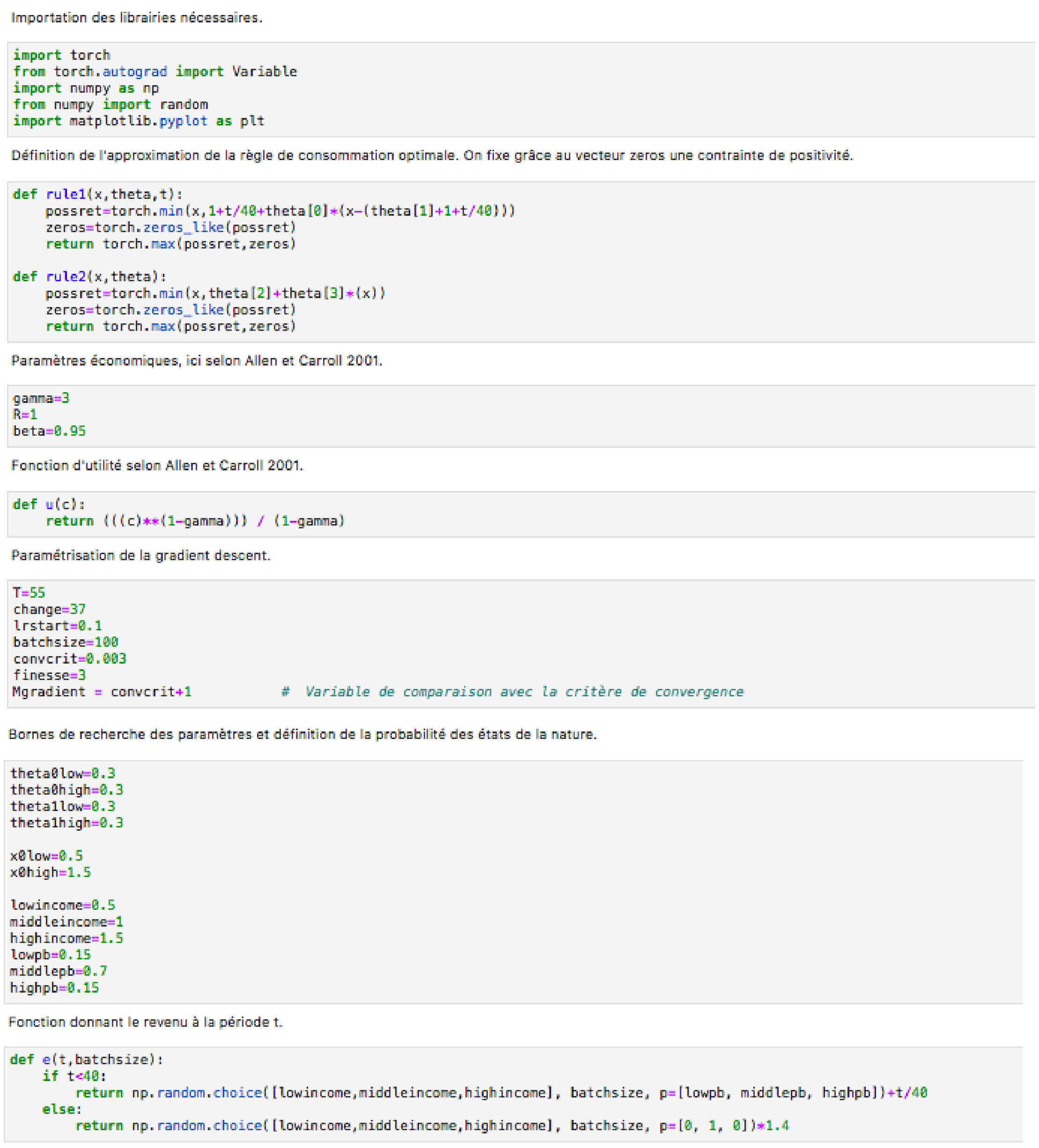
Définition de la fonction d'utilité

```python
def ut(c):
    return ((c**(1-gamma))) / (1-gamma)
```

Boucle de remplissage de la value function et des consommations optimales

```python
#Boucle des périodes commençant par la fin
for i in range(T-1):
    #Pour chaque point de la grille d'état, essai de toutes les consomations inférieures
    for idx, cashoh in enumerate(xposs):
        maxut=-10000000
        maxcons=-10000000
        for tries in range(idx+1):
            if cashoh-xposs[tries] < size-inc[np.size(inc)-1]:
                tryut = ut(xposs[tries])
                for draws in range(3):
                    tryut += probinc[draws]*beta*vfonc[T-1-i,int((cashoh-xposs[tries]+inc[draws]-0.1)*10**finesse)]
            else:    #Si la consommation nous envoie dans un point en dehors de la grille
                tryut = -1000000000
            #Si la somme d'utilité de cette consommation et de l'utilité future est supérieure
            #à celles fournie par la meilleure consommation jusque là, mémorisation
            if tryut > maxut:
                maxut=tryut
                maxcons=xposs[tries]
        opcons[T-2-i,idx]=maxcons
        vfonc[T-2-i,idx]=maxut
```

C Machine learning code to the finite-horizon model



Importation des librairies nécessaires.

```
import torch
from torch.autograd import Variable
import numpy as np
from numpy import random
import matplotlib.pyplot as plt
```

Définition de l'approximation de la règle de consommation optimale. On fixe grâce au vecteur zeros une contrainte de positivité.

```
def rule1(x,theta,t):
    possret=torch.min(x,1+t/40+theta[0]*(x-(theta[1]+1+t/40)))
    zeros=torch.zeros_like(possret)
    return torch.max(possret,zeros)

def rule2(x,theta):
    possret=torch.min(x,theta[2]+theta[3]*(x))
    zeros=torch.zeros_like(possret)
    return torch.max(possret,zeros)
```

Paramètres économiques, ici selon Allen et Carroll 2001.

```
gamma=3
R=1
beta=0.95
```

Fonction d'utilité selon Allen et Carroll 2001.

```
def u(c):
    return (((c)**(1-gamma))) / (1-gamma)
```

Paramétrisation de la gradient descent.

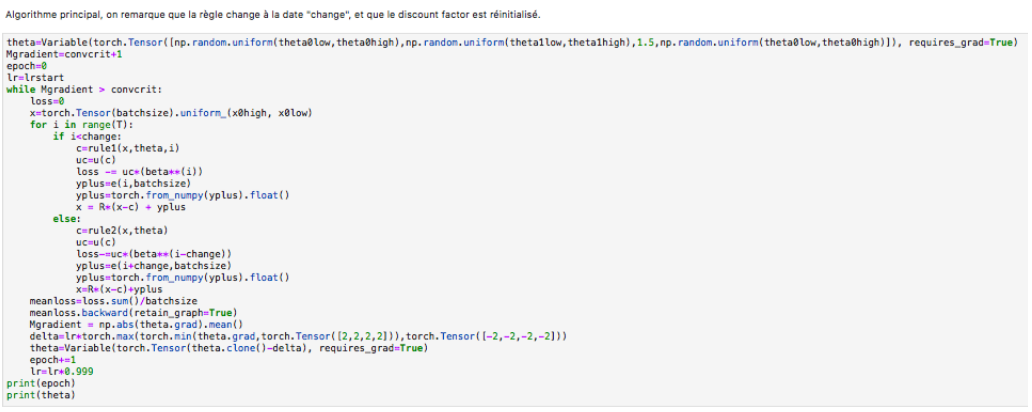
```
T=55
change=37
lrstart=0.1
batchsize=100
convcrit=0.003
finesse=3
Mgradient = convcrit+1          #  Variable de comparaison avec la critère de convergence
```

Bornes de recherche des paramètres et définition de la probabilité des états de la nature.

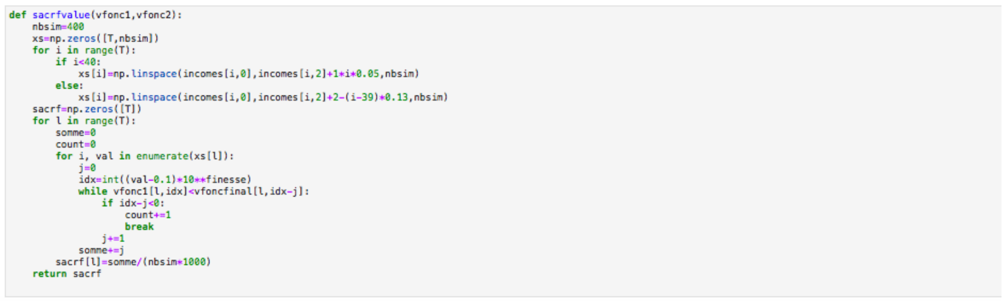
```
theta0low=0.3
theta0high=0.3
theta1low=0.3
theta1high=0.3

x0low=0.5
x0high=1.5

lowincome=0.5
middleincome=1
highincome=1.5
lowpb=0.15
middlepb=0.7
highpb=0.15
```

Fonction donnant le revenu à la période t.

```
def e(t,batchsize):
    if t<40:
        return np.random.choice([lowincome,middleincome,highincome], batchsize, p=[lowpb, middlepb, highpb])+t/40
    else:
        return np.random.choice([lowincome,middleincome,highincome], batchsize, p=[0, 1, 0])*1.4
```

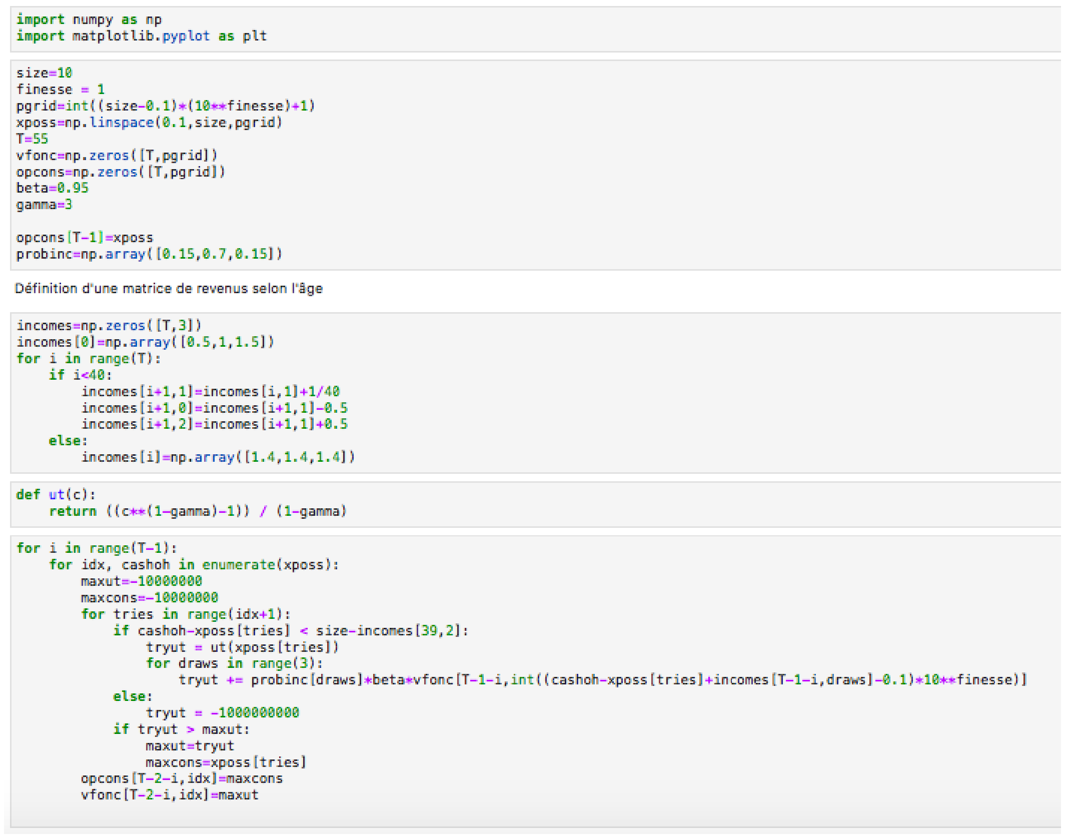

Algorithme principal, on remarque que la règle change à la date "change", et que le discount factor est réinitialisé.

```python
theta=Variable(torch.Tensor([np.random.uniform(theta0low,theta0high),np.random.uniform(theta1low,theta1high),1.5,np.random.uniform(theta0low,theta0high)]), requires_grad=True)
Mgradient=convcrit+1
epoch=0
lr=lrstart
while Mgradient > convcrit:
    loss=0
    x=torch.Tensor(batchsize).uniform_(x0high, x0low)
    for i in range(T):
        if i<change:
            c=rule1(x,theta,i)
            uc=u(c)
            loss -= uc*(beta**(i))
            yplus=e(i,batchsize)
            yplus=torch.from_numpy(yplus).float()
            x = R*(x-c) + yplus
        else:
            c=rule2(x,theta)
            uc=u(c)
            loss-=uc*(beta**(i-change))
            yplus=e(i+change,batchsize)
            yplus=torch.from_numpy(yplus).float()
            x=R*(x-c)+yplus
    meanloss=loss.sum()/batchsize
    meanloss.backward(retain_graph=True)
    Mgradient = np.abs(theta.grad).mean()
    delta=lr*torch.max(torch.min(theta.grad,torch.Tensor([2,2,2,2])),torch.Tensor([-2,-2,-2,-2]))
    theta=Variable(torch.Tensor(theta.clone()-delta), requires_grad=True)
    epoch+=1
    lr=lr*0.999
print(epoch)
print(theta)
```



```python
def sacrfvalue(vfonc1,vfonc2):
    nbsim=400
    xs=np.zeros([T,nbsim])
    for i in range(T):
        if i<40:
            xs[i]=np.linspace(incomes[i,0],incomes[i,2]+1*i*0.05,nbsim)
        else:
            xs[i]=np.linspace(incomes[i,0],incomes[i,2]+2-(i-39)*0.13,nbsim)
    sacrf=np.zeros([T])
    for l in range(T):
        somme=0
        count=0
        for i, val in enumerate(xs[l]):
            j=0
            idx=int((val-0.1)*10**finesse)
            while vfonc1[l,idx]<vfoncfinal[l,idx-j]:
                if idx-j<0:
                    count+=1
                    break
                j+=1
            somme+=j
        sacrf[l]=somme/(nbsim*1000)
    return sacrf
```

D Dynamic programming solution to the finite-horizon model



```
import numpy as np
import matplotlib.pyplot as plt
```

```
size=10
finesse = 1
pgrid=int((size-0.1)*(10**finesse)+1)
xposs=np.linspace(0.1,size,pgrid)
T=55
vfonc=np.zeros([T,pgrid])
opcons=np.zeros([T,pgrid])
beta=0.95
gamma=3

opcons[T-1]=xposs
probinc=np.array([0.15,0.7,0.15])
```

Définition d'une matrice de revenus selon l'âge

```
incomes=np.zeros([T,3])
incomes[0]=np.array([0.5,1,1.5])
for i in range(T):
    if i<40:
        incomes[i+1,1]=incomes[i,1]+1/40
        incomes[i+1,0]=incomes[i+1,1]-0.5
        incomes[i+1,2]=incomes[i+1,1]+0.5
    else:
        incomes[i]=np.array([1.4,1.4,1.4])
```

```
def ut(c):
    return ((c**(1-gamma)-1)) / (1-gamma)
```

```
for i in range(T-1):
    for idx, cashoh in enumerate(xposs):
        maxut=-10000000
        maxcons=-10000000
        for tries in range(idx+1):
            if cashoh-xposs[tries] < size-incomes[39,2]:
                tryut = ut(xposs[tries])
                for draws in range(3):
                    tryut += probinc[draws]*beta*vfonc[T-1-i,int((cashoh-xposs[tries]+incomes[T-1-i,draws]-0.1)*10**finesse)]
            else:
                tryut = -1000000000
            if tryut > maxut:
                maxut=tryut
                maxcons=xposs[tries]
        opcons[T-2-i,idx]=maxcons
        vfonc[T-2-i,idx]=maxut
```