AMS 553 Final Project

Inventory Control Problem: Different Algorithms

Caleb Erubiel Andrade Sernas
Yujian Liu

Tsz Ching Ng
Caiwei Li

Yue Wang
Francisco Hawas Vargas

Fall 2015

Project Summary

This project focuses on the well known inventory control problem of two products with correlated demands. Although in this case the (s, S) policy will not be optimal, we use want to find this suboptimal policy because it is easy to implement in a real application.

The questions that we want to answer are the following:

1. What is the effect of correlation in the optimization problem?
2. Which of the algorithms presented below generates the best policy?
3. What are the difference in terms of running time between the algorithms?

In order to answer these questions we generated correlated the demands using the Gaussian Copula and have implemented three algorithms:

- Finite Difference Stochastic Approximation (FDSA).
- Simultaneous Perturbation Stochastic Approximation (SPSA).
- Simulated Annealing (SARS)

In addition, Ranking and Selection procedure is used in every comparison of two different policies during the optimizations.

To answer the first question we separately computed the optimal policies for each product, then we preformed the optimizations by considering the products jointly. The results indicate that the correlation has no effect. We think that this is because the cost functions are linear and that makes the problem separable.

To assess the second and third questions we run the three algorithms with different numbers of iterations (50, 100, 150, 200). We found that the algorithm that consistently generated the best policy is the SPSA. Not only that, this algorithm also generates the most consistent policy for different number of iterations. In terms of running time it is not clear which of the algorithms is faster.

Summary of Results

To sum up, if we need an algorithm for this level of iterations we should choose SPSA. But it is unclear if once we run more iterations the other algorithms will improve their accuracy keeping the running time low, and therefore change our conclusion. For details of the results, please refer to section 7.

Report Outline

The report is outlined as follows: In section 1 and 2 we will give an introduction of the optimization problem that we are interested in. In section 3 and 4, we will provide some important details in our works. The pseudo-code of the optimization algorithms and a list of programs implemented will be given in section 5 and 6 respectively. Finally we will provide some results and draw conclusions in section 7 and 8 respectively. For sake of clarity, we list all the figures at the end of the report.

Team members

Caleb Erubiel Andrade Sernas, Tsz Ching Ng, Yue Wang, Yujian Liu, Caiwei Li, Francisco Hawas Vargas

Contents

1 Introduction

"Sorry, we are out of that item. You can heard this during shopping many times." In such situation, what you have encountered is that the stores do not have a good inventory management system. They cannot place the order and replenish their inventories in a right time to avoid the shortage.

There are variety of different types of inventory. Raw material, Natural gas, coal and Luxury goods are inventories to many businesses. The total value of all inventories, including finished goods, partially finished goods, and raw materials, is more than a trillion dollars in the United States. This is more than US $\$4,000$ each for every man, woman, and child in the country.

A tough problem of inventory control is to formulate the problem correctly. There are a number of parameters that need to be concerned about and different parameters can be relevant for some purposes while being inadequate or dangerous for others.

For mathematical convenience, we will simplify the model with the following assumptions:

- Demand distribution is known.
- Demand distribution is i.i.d. over time.
- Shortages are allowed.
- Lead time for the receipt of orders is constant.
- The order quantity is received every period.

Based on the assumption above, we build our cost function and start our comparison of different policies.

2 Problem formulation and scope

The inventory control problem is to find the optimal order size and order time given a cost function for the inventory and demand distribution for the products. We limit our scope to a 2-inventory control problem, and each inventory j is control by a (s_j, S_j) policy.

In particular, we are going to use the following one-period cost function at time t:

$$g_t(s_1, s_2, S_1, S_2) = \sum_{j=1,2} 1_{\{I_{jt} < s_j\}} \left(k_j + c_j\left(S_j - I_{jt}\right)\right) + h_j I_{jt}^{+} + b_j I_{jt}^{-}$$

where k_j is the setup cost, c_j is the per unit ordering cost for product j, h_j is the holding cost, b_j is the backlog cost, and I_{jt} is the inventor level of product j at time t.

And we define the average cost as a time-average of the g_t's. That is:

$$g(s_1, s_2, S_1, S_2) = \frac{1}{120} \sum_{t=1}^{120} g_t(s_1, s_2, S_1, S_2)$$

This formulation implies the following assumptions:

- There are two products in this setting.
- All cost functions are linear.
- The horizon to be considered is 120(4 months).
- The demands are independent for different times but correlated between the different products for the same period.

We also assume that the demands for each product are exponentially distributed.

One important point is that for the assumptions of this problem the (s, S) type of policy may be suboptimal. However, for such type of policies, it is easy to implement the optimizations in a real setting because of its simplicity.

The questions that we want to answer are the following:

1. What is the effect of correlation in the optimization problem?
2. Which of the algorithms presented here generates the best policy?
3. What are the difference in terms of running time between the algorithms?

To answer the first question we solve this problem in two ways. After we generate the demands for the products we solve two different optimization problems for each product and one with the objective function shown above. The difference between the optimal policies for this two cases will give us an indication of the importance of correlation in the current environment.

The second and third question will be assessed by comparing the performance of the different algorithms in terms of the optimal policies given (objective function values) and the running times.

3 Generation of correlated random variables

One important issue of our problem is to generate correlated demands, with correlation ρ. Our approach is the Gaussian Copula method, that is composed of the following steps:

1. Generate two independent normal random variables.
$$N = \begin{bmatrix} N_1 \\ N_2 \end{bmatrix}$$
2. For the matrix:
$$\Sigma = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$$
Compute the Cholesky decomposition, this is, $\Sigma = LL^T$ where L is lower triangular.
3. The output is the random variable U_i for $i = 1, 2$:
$$U_i = F((LN)_i)$$
For $i = 1, 2$, where $F(\cdot)$ is the standard Normal cumulative distribution function and $()_i$ is the i-th element of the vector.
4. Finally, to get the desire marginal distribution:
$$Y_i = F_d^{-1}((U)_i)$$
Where F_d^{-1} is the inverse cumulative distribution of the desire distribution.

One of the reasons we use the exponential distribution for the marginal of the demands is that its c.d.f makes it very easy to apply the inverse transform inside of the Gaussian Copula algorithm.

4 CRN-Ranking and Selection

In all the algorithms we need to compare the output from two different policies. In the case of one inventory problem (when we consider each product as separate entities) we will test the output of the simulation for (s_{i_1}, S_{i_1}) and (s_{i_2}, S_{i_2}). When we consider the two product inventory problem we will compare the output of $(s_{A,i_1}, S_{A,i_1}, s_{B,i_1}, S_{B,i_1})$ and $(s_{A,i_2}, S_{A,i_2}, s_{B,i_2}, S_{B,i_2})$.

Since the output of this simulations is a random variable for each of the policies tested, we need to consider a robust way to compare the results. Our approach is to use common random numbers (CRN) whenever possible and then use the ranking and selection algorithm discussed by Mohamed *et al.*[1], which will include the following steps:

- Compute $\bar{X}(n_0)$, $S^2(n_0)$, h (increasing function of β).
- Compute additional number of simulations,

$$n_i = \max\left(n_0 + 1, \left\lceil \frac{S^2(n_0)\beta^2}{\delta^2} \right\rceil\right)$$

- Compute weights $w_i = f(n_0, n_i, S^2(n_0), h, \delta)$ where $i = 1, 2$
- Compare the systems using with the following averages:

$$\tilde{X}_i(n_i) = w_i\bar{X}_i(n_0) + (1 - w_i)\bar{X}_i(n_i - n_0)$$

- Pick the smallest value between $\tilde{X}_i(n_i)$ $i = 1, 2$.

A complete explanation of this procedure can be found in Law's book[4].

5 Algorithms description

In this section we provide the pseudo-code of our implementation of each algorithm.

5.1 Finite Difference Stochastic Approximation (FDSA)

Algorithm FDSA

```
1: Initialization:
2:     Set initial policy $P_0$
3:     Set iterations as stopping criteria
4:     Set $k = 0$
5:     Set $P_{best} = P_{current} = P_0$
6: while $k <$ iterations do
7:     while $P_{next} \in \{(s \pm 1, S \pm 1)\}$ do
8:         Construct a candidate of better policy:
9:             Calculate $Cost(P_{next})$
10:        Compare the candidate to the current policy:
11:            if $Cost(P_{next}) \leq Cost(P_{current})$ then
12:                Set $P_{current} = P_{next}$
13:            end if
14:    end while
15:    if $Cost(P_{current}) < Cost(P_{best})$ then
16:        Set $P_{best} = P_{current}$
17:    end if
18:    Set $k \leftarrow k + 1$
19: end while
20: return $P_{best}$
```

Where $(s, S) = P_{current}$

In our simulation we set the parameters as follows:

- $Iterations = 50, 100, 150, 200$
- $P_0 = (50, 100)$ for both inventories.

5.2 Simultaneous Perturbation Stochastic Approximation (SPSA)

Algorithm SPSA

```
1: Initialization:
2:     Set initial policy $P_0 \in \mathbb{R}^{2n}$
3:     Set iterations as stopping criteria
4:     Set $k = 0$
5:     Set $P_c = P_0$
6: while $k <$ iterations do
7:     Construct a candidate of better policy:
8:         Uniformly select $\vec{\Delta} \in \{-1, 0, 1\}^{2n} \setminus \{\vec{0}\}$
9:         Generate a stream of demands $\omega$
10:        Set $\hat{g} \leftarrow \frac{\mathsf{Cost}(P_c + c_k\vec{\Delta}, \omega) - \mathsf{Cost}(P_c - c_k\vec{\Delta}, \omega)}{2c_k\|\vec{\Delta}\|}$
11:        Set $P_n \leftarrow P_c - a_k\hat{g}\vec{\Delta}$
12:    Compare the candidate to the current policy:
13:        if $\mathsf{Cost}(P_n, \omega) \leq \mathsf{Cost}(P_c, , \omega)$ then
14:            Set $P_c \leftarrow P_n$
15:        end if
16:    Set $k \leftarrow k + 1$
17: end while
18: return $P_c$
```

For stability and to handle discrete policies, we set the parameters in our simulation as follows (see section 6 for further explanations):

- $a_k = \min(1000(k+1)^{-1}, 10)$
- $c_k = \max(10(k+1)^{-0.49}, 0.5)$

5.3 Simulated Annealing

Algorithm Simulated Annealing

```
1: Step 1:
2:     Set initial policy $P_0$
3:     Set initial temperature $T_0$
4:     Set iterations as stopping criteria
5:     Set $k = 0$, count $= 0$
6:     Set $P_b = P_c = P_0$        ▷ Where $P_b = P_{best}$, $P_c = P_{current}$, $P_n = P_{neighbor}$
7:     Set $L$ = Markov chain length
8: Step 2:
9:     while $k <$ iterations do
10:        while count $< L$ do
11:            Choose $P_n \in N(P_c)$ with probability $\frac{1}{N(P_c)}$
12:            if $\mathsf{Cost}(P_n) \leq \mathsf{Cost}(P_c)$ then
13:                $P_c \leftarrow P_n$
14:                if $\mathsf{Cost}(P_c) < \mathsf{Cost}(P_b)$ then
15:                    Set $P_b \leftarrow P_c$
16:                end if
17:            else
18:                Set $P_c \leftarrow P_n$ with probability $\frac{1}{\exp\left(\frac{\mathsf{Cost}(P_n) - \mathsf{Cost}(P_c)}{T_k}\right)}$
19:            end if
20:            count++
21:            $k$++
22:            Set $T_k \leftarrow \left(\frac{1}{2}\right)^{\lfloor \frac{k}{L} \rfloor} T_0$
23:        end while
24:        Set count $\leftarrow 0$
```

```
    end while
return $P_b$
```

In our simulation we set the parameters as follows:

- $T_0 = 10$
- $L = 50$
- $Cooling\ factor = \frac{1}{2}$
- $Iterations = 150$
- $P_0 = (50, 100)$ for both inventories.

Some important parameters in this algorithms are the temperature and the markov chain length. The temperature will control the probability to look into policies that are worst than the current one. This is an important feature of this algorithm since allow us to escape from local optimum. The length of the markov chain measures the number of iterations where the temperature is held constant.

6 Code description

The implementation of our project was done in Python, although we considered another two alternatives, namely, Matlab and C++, but given the power, flexibility and easiness of syntax of Python, as well as the abundance of libraries such as Numpy for statistical calculations, and matplotlib for plotting capabilities; and also because of the different levels of programming skills among those in our team, we considered Python as our best platform to develop our project.

The code is composed of different modules. A brief description is given here

- Inventory Class. An *Inventory* object was implemented to simulate the behavior of an inventory. It is initialized with the following input:

  $$Inventory\ (level,\ item-cost,\ backlog-cost,\ holding-cost,\ setup-cost,\ lead-time,\ ss-policy).$$

  This class incorporates methods such as to compute one-period cost, run a simulation given a stream of random demands, keep track of the level and cost history, and plot the level histogram.

- Sample Mean method. To compute a good estimate of the cost function of an m-period simulation for one inventory (or two correlated inventories), this function was created following the procedure described in section 4 using a first batch of $n_0 = 20$ replications. One difficulty was that whenever more runs were needed in the second batch of runs for one system, we just continued the stream of random variates for that particular system. In order to maintain synchronization, every run was initialized independently, but a copy of the streamed random variates was kept for the second system.

- Generate Correlated Demands method. This function was created to generate two streams of correlated exponential random variates. The correlation factor, and the two exponential rates are the basic parameters. This is explained in section 3.

- Simulated Annealing Ranking and Selection method. This function combines simulated annealing algorithm presented in section 5.3 and the ranking and selection procedure. It takes as argument either one inventory or two correlated inventories, and the basic parameters for the SA algorithm. At each iteration, the space of policies $\{(s_i, S_i, s_j, S_j)\}$ is explored randomly, given an initial policy (s_1, S_1, s_2, S_2). At each step, the ranking and selection method is used to determine whether the *next policy* is an improvement or not, and proceed accordingly to simulated annealing.

- Finite Difference Stochastic Approximation. The structure of the algorithm is a gradient-like method, with the iterates being generated as specified in section 5.1, where the gradient of the output of our simulation is approximated using finite differences. At each step, the ranking and selection method is used to determine whether the *next policy* is an improvement or not, finally after enough iterations (s_1, S_1, s_2, S_2) will convergence to best policy.

- Simultaneous Perturbation Stochastic Approximation. This function iteratively searches for better policies from the current policy P_c by estimating the gradient of the cost function. It also incorporates common random numbers (CRN) to reduce the variances of the gradient estimates. In each iteration, a random direction $\vec{\Delta}$ is selected and the directional derivative $\hat{g}$ of the cost function is approximated by central difference. Similar to gradient descent, the new policy P_n is set as a shift of the current policy P_c in the direction $-\hat{g}\vec{\Delta}$. Ranking and selection method is used to test whether P_n is actually an improvement from P_c, and accepts P_n if so.

  In every iteration k, both the step size c_k in gradient estimation and the step size a_k in gradient descent need to be chosen properly. According to Kleinman *et al.*[3], the SPSA algorithm achieves fastest convergence rate of $k^{-1/3}$ if the step sizes are of the forms $a_k = ak^{-1}$ and $c_k = ck^{-0.49}$. However, to handle the fact that the policies are discrete in $\mathbb{N}^{2n}$, c_k should be bounded below from 0.5 (otherwise $\hat{g} = 0$). On the other hand, although the SPSA algorithm is guaranteed to converge to a local minimum, the parameter a_k should be bounded from above by some reasonable thresholds (say 10). Otherwise, the algorithm will end up in a much worse local minimum, especially when the ranking and selection method is not used to test the credibility of the new policy. Such an example is shown in section 7.3.

7 Results

The parameters used in the simulations are the following:

Cost	Product A	Product B
Level	50	50
Item	3	3
Backlog	5	20
Holding	1	1
Setup	32	5

- The leading time for A is 0 and for B is 0.
- Both marginal distribution are exponential with parameter 0.12 and 0.10, and the correlation is -0.8. The initial policy for both products is $(50, 100)$.

7.1 Separate inventory decisions

Inventory control: Product A

This section presents the results considering that Product A is managed separately from Product B.

50 iterations	FDSA	SPSA	SARS
Running Time	37.7	65.3	15.3
Objective Function	93.5	63.0	79.8
(s,S)	(46, 90)	(11,40)	(20,98)

100 iterations	FDSA	SPSA	SARS
Running Time	70.1	192.3	70.4
Objective Function	86.9	62.6	87.0
(s,S)	(36,88)	(13,38)	(42,78)

150 iterations	FDSA	SPSA	SARS
Running Time	130.4	214.2	57.4
Objective Function	76.7	63.4	68.8
(s,S)	(17, 77)	(10,41)	(28,44)

200 iterations	FDSA	SPSA	SARS
Running Time	189.2	288.2	77.4
Objective Function	66.2	62.4	63.0
(s,S)	(13, 69)	(12,35)	(11,42)

The convergence of the objective function is presented in figures 1a,1b,1c,1d for SPSA algorithm, in figures 2a,2b,2c,2d for SARS algorithm and in figures 3a,3b,3c,3d. The inventory evolution for Product A considering 200 iterations is presented in figure 10c for FDSA, figure 10b SARS and figure 10a for SPSA.

Inventory control: Product B

This section presents the results considering that Product B is managed separately from Product A.

50 iterations	FDSA	SPSA	SARS
Running Time	62.2	148.5	27.9
Objective Function	88.8	68.4	87.7
(s,S)	(40, 88)	(36,43)	(33,91)

100 iterations	FDSA	SPSA	SARS
Running Time	114.2	360.7	160.4
Objective Function	83.9	68.0	68.4
(s,S)	(38, 80)	(33,41)	(28,54)

150 iterations	FDSA	SPSA	SARS
Running Time	229.3	536.5	302.8
Objective Function	79.5	68.7	68.2
(s,S)	(33, 71)	(33,46)	(37,59)

200 iterations	FDSA	SPSA	SARS
Running Time	423.5	715.9	206.6
Objective Function	75.2	68.1	71.2
(s,S)	(30, 72)	(32,43)	(33,47)

The convergence of the objective function is presented in figures 4a,4b,4c,4d for SPSA algorithm, in figures 5a,5b,5c,5d for SARS algorithm and in figures 6a,6b,6c,6d. The inventory evolution for Product B considering 200 iterations is presented in figure 11c for FDSA, figure 11b SARS and figure 11a for SPSA.

7.2 Joint inventory decisions

Inventory control: Products A and B

This section presents the results considering that Product A and B are managed jointly.

50 iterations	FDSA	SPSA	SARS
Running Time	124.5	60.8	28.8
Objective Function	188.1	146.7	192.1
(s,S)	(46, 96);(38, 88)	(8,51);(43,55)	(29,79);(75,94)

100 iterations	FDSA	SPSA	SARS
Running Time	279.1	151.7	449.7
Objective Function	182.5	146.9	147.9
(s,S)	(43,95);(40,80)	(12,37);(39,52)	(11,37);(43,57)

150 iterations	FDSA	SPSA	SARS
Running Time	448.5	223.9	354.5
Objective Function	173.3	144.6	147.1
(s,S)	(40, 90);(35, 75)	(11,38);(43,52)	(7,37);(45,63)

200 iterations	FDSA	SPSA	SARS
Running Time	492.5	342.9	654.9
Objective Function	178.1	143.6	146.1
(s,S)	(40, 90);(34, 70)	(12,37);(38,51)	(10,48);(39,54)

The convergence of the objective function is presented in figures 7a,7b,7c,7d for SPSA algorithm, in figures 8a,8b,8c,8d for SARS algorithm and in figures 9a,9b,9c,9d. The inventory evolution for Product A and B considering 200 iterations is presented in figure 12e and 12f for FDSA, figure 12c and 12d SARS and figure 12a and 12a for SPSA.

7.3 Ranking and Selection

Here we provide an example to explain the importance of the Ranking and Selection procedure in our algorithms. In this example, we consider only the optimization of product A using SPSA. We first compute the optimal (s, S) policy using the SPSA algorithm with ranking and selection described in section 5.2, but to introduce instability we remove the upper bound of the time step and use the following a_k's instead:

$$a_k = a(k+1)^{-1}$$

where $a = 100, 1000, 5000$ in 3 different cases, and we run for 100 iterations. We denote this unmodified algorithm by “SPSA-RS”.

Then we repeat the optimizations using SPSA but without ranking and selection. That is to remove line 12-15 in section 5.2 during the optimizations. We denote this algorithm by “SPSA-noRS”. Again we consider $a = 100, 1000, 5000$ in 3 different cases.

The results are the following:

$a_k = 100(k+1)^{-1}$	SPSA-RS	SPSA-noRS
Running Time	204.9	42.3
Objective Function	63.4	64.0
(s,S)	(11,41)	(12,51)

$a_k = 1000(k+1)^{-1}$	SPSA-RS	SPSA-noRS
Running Time	257.1	43.0
Objective Function	63.2	63.9
(s,S)	(13,37)	(11,36)

$a_k = 5000(k+1)^{-1}$	SPSA-RS	SPSA-noRS
Running Time	255.3	41.1
Objective Function	63.3	2293.2
(s,S)	(14,42)	(-4950,100)

The convergence of the objective function is presented in figure 13a,13b,13c for SPSA-RS algorithm, and in figure 14a,14b,14c for SPSA-noRS algorithm.

In terms of running time, we see that SPSA-noRS outperforms SPSA-RS. The major reason is that the variances near the optimal policy are much larger than those suboptimal policies (≈ 7 at the initial policy and ≈ 30 near the optimal policy). Hence the ranking and selection procedure requires a larger number of samples near the optimal solutions.

However, in terms of stability, we see from the graphs that in SARS-noRS, the objective values increase rapidly at the beginning, and eventually go down to optimal for $a = 100, 1000$. However, it is stuck at a much worse local minimum (-4950,100) for $a = 5000$. In contrast, SPSA-RS always converges monotonically to the optimal in all of the 3 cases.

8 Conclusion

8.1 Algorithm comparison

Objective function comparison

The algorithm that achieves the best objective function in almost all the instances is the SPSA algorithm (except for Product B when we use 150 iterations). Our implementation of FDSA is the one that always gives the worst objective value.

Another important fact about SPSA is that the best objective function is very stable when we increase the number of iterations. This indicates that we do not need to run a large numbers of iterations in order to find an optimal policy.

In the SARS case it shows convergence in all the cases but it seems to be fragile for low number of iterations, for example this erratic behavior happen in Product A for 100 runs and in Product B at the end. However in the case where we look for the optimum of the sum of cost functions we get fast convergence.

In the case of FDSA, shows convergence in the three cases but it does not reach the levels of the other two algorithms.

Another that we can see in the figures related to SPSA (1a,1b,1c,1d,4a,4b,4c,4d,7a,7b,7c,7d) is that the convergence is very smooth while the algorithm is running. The contrary applies for SARS and FDSA which is something that we can expect given the instability we observed in the final results. For this two algorithms is clear that we need more iterations.

Running time comparison

In all of the algorithms the time to run 200 iterations is more than 4 times the time to run 50 iterations.

In the case of Product A the fastest algorithm is SARS, for Product B it is not clear if SARS or FDSA is the fastest algorithm. Although it is clear that for Product A and B SPSA is slowest algorithm. When the objective function is the sum of the costs then SPSA is the fastest algorithm.

8.2 Correlation effect

We can not appreciate an effect of the correlation. When we compare the planning of both products separately (policies from Product A and B) and the combine objective function (policy from the sum) we can see, mostly in the stable SPSA, than the correlation has not effect. In both cases the policies for Product A and B are almost identical.

In SARS and FDSA, we see differences but are probably due to the instability of the solution.

We think that the reason for this lack of importance of correlation is that all the cost functions are linear and that makes the problem separable. Thus not inducing differences when we optimize each product separately or jointly.

9 Bibliography

[1] Ahmed Mohamed, Alkhamis Talal. Simulation based optimization using simulated annealing with ranking and selection.

[2] Feinberg Eugene, Lewis Mark. On optimality of (s,S) Policies.

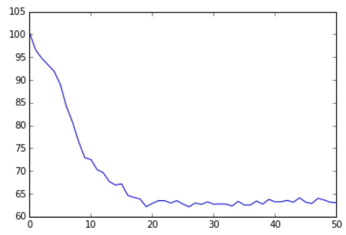
[3] Kleinman Nathan, Spall James, Naiman Daniel. Simulation Based Optimization with Stochastic Approximation using common random numbers.

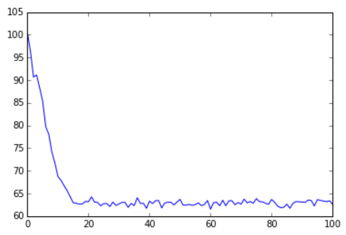
[4] Law Averill, Simulation Modeling and Analysis.

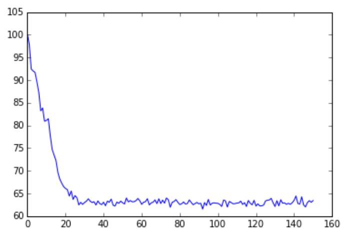
10 Figures

Separate inventory decisions

Inventory control: Product A


(a) 50 iterations


(b) 100 iterations.


(c) 150 iterations

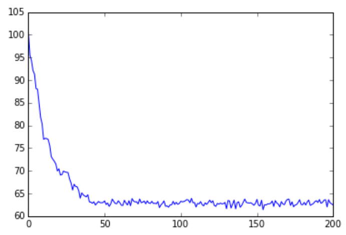
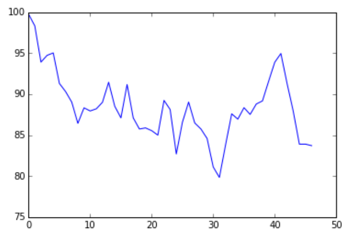
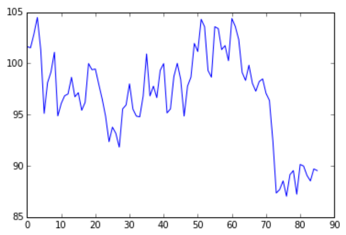
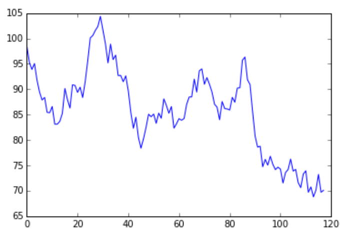

(d) 200 iterations

Figure 1: SPSA: Convergence of the objective function, Product A


(a) 50 iterations


(b) 100 iterations


(c) 150 iterations

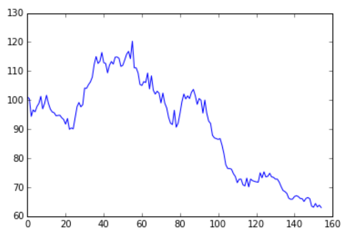
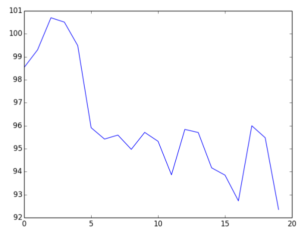
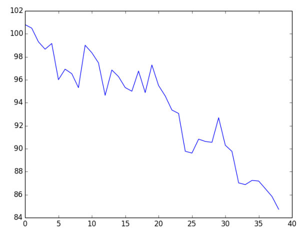
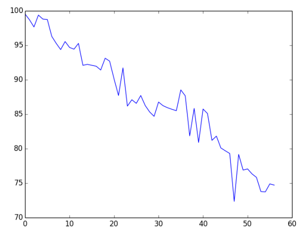

(d) 200 iterations

Figure 2: SARS: Convergence of the objective function, Product A


(a) 50 iterations


(b) 100 iterations


(c) 150 iterations

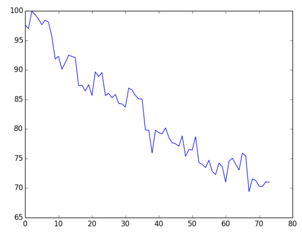
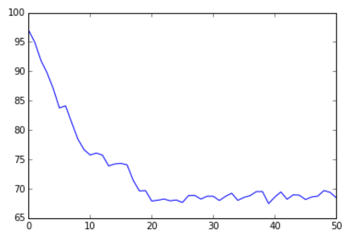
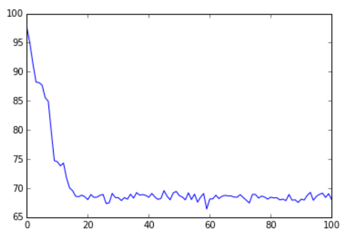
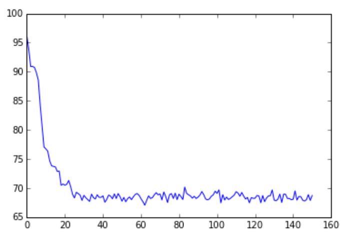

(d) 200 iterations

Figure 3: FDSA: Convergence of the objective function, Product A

Inventory control: Product B


(a) 50 iterations


(b) 100 iterations


(c) 150 iterations

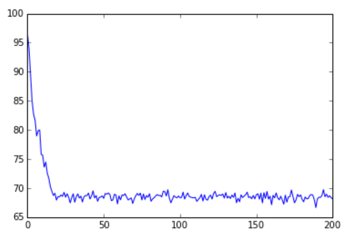
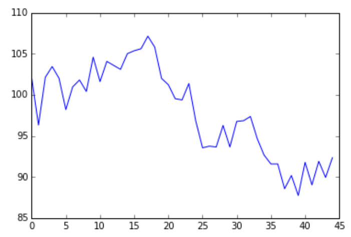
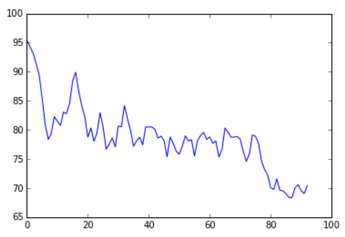
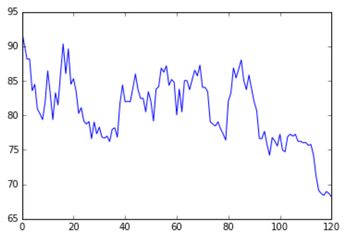

(d) 200 iterations

Figure 4: SPSA: Convergence of the objective function, Product B


(a) 50 iterations


(b) 100 iterations


(c) 150 iterations

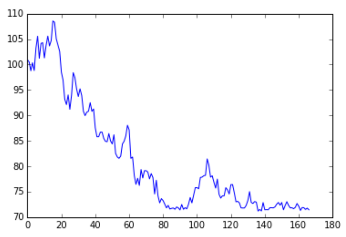
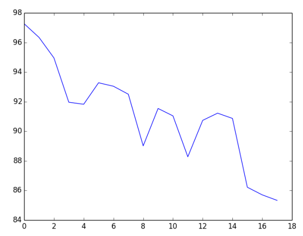
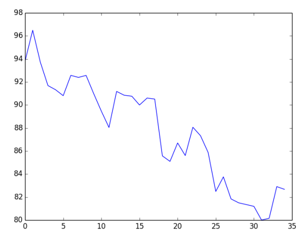
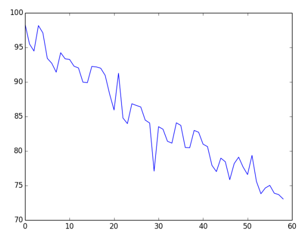

(d) 200 iterations

Figure 5: SARS: Convergence of the objective function, Product B


(a) 50 iterations


(b) 100 iterations


(c) 150 iterations

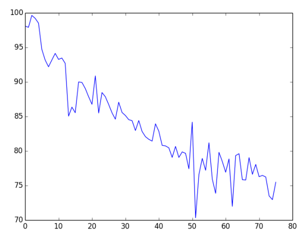
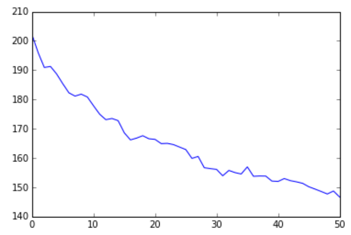
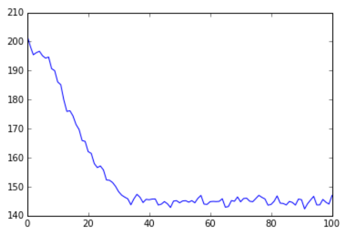

(d) 200 iterations

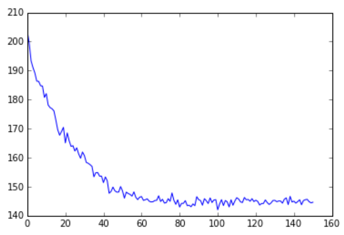
Figure 6: FDSA: Convergence of the objective function, Product B

Joint inventory decisions

Inventory control: Products A and B


(a) 50 iterations


(b) 100 iterations


(c) 150 iterations

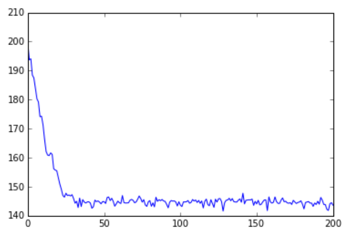
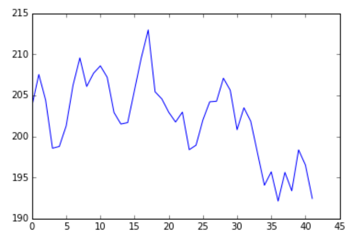
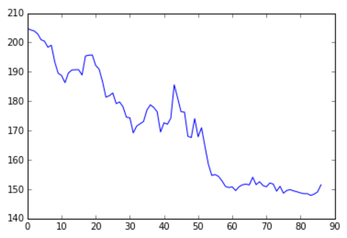
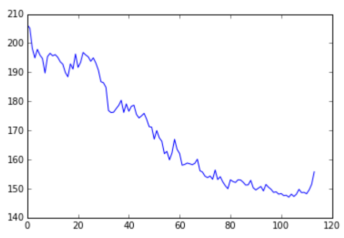

(d) 200 iterations

Figure 7: SPSA: Convergence of the objective function, Product A and B combined


(a) 50 iterations


(b) 100 iterations


(c) 150 iterations

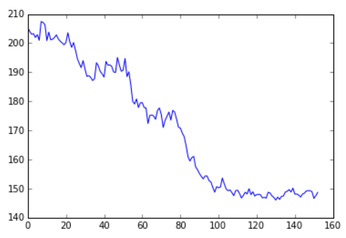
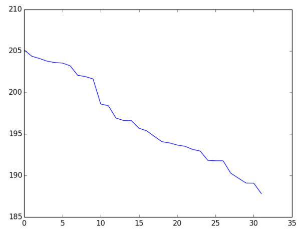
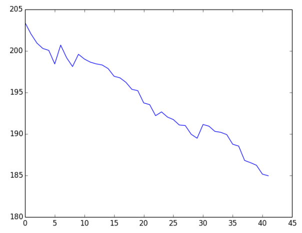
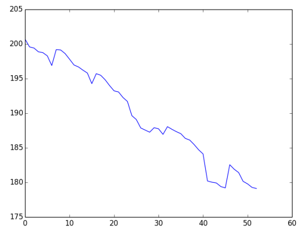

(d) 200 iterations

Figure 8: SARS: Convergence of the objective function, Product A and B combined


(a) 50 iterations


(b) 100 iterations


(c) 150 iterations

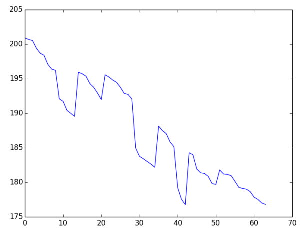
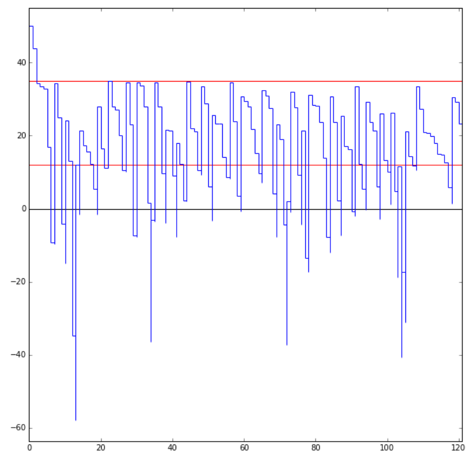
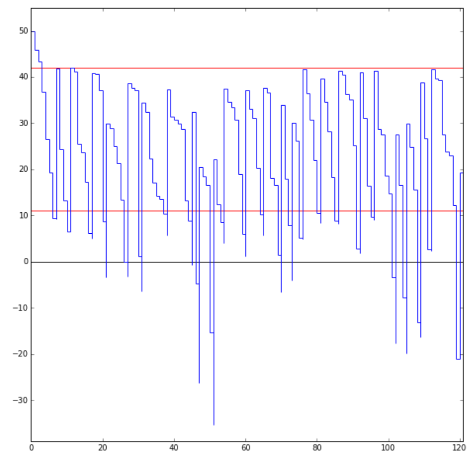

(d) 200 iterations

Figure 9: FDSA: Convergence of the objective function, Product A and B combined

Figures of Inventory


(a) SPSA


(b) SARS

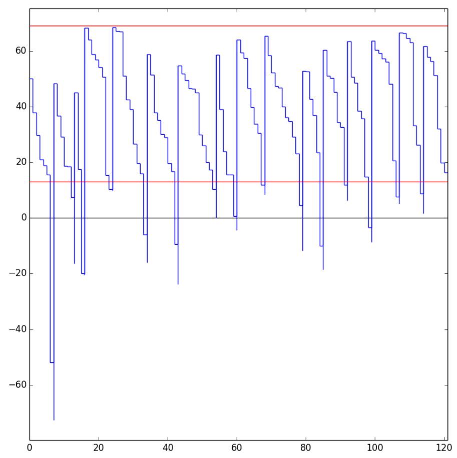
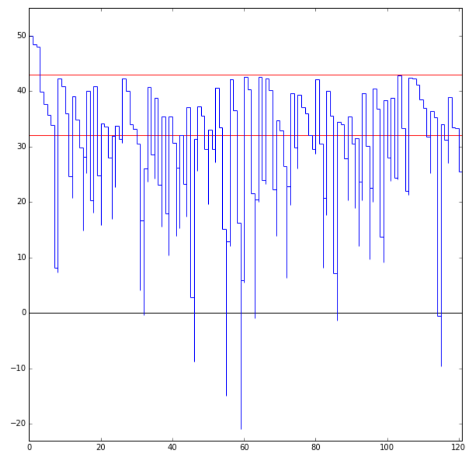
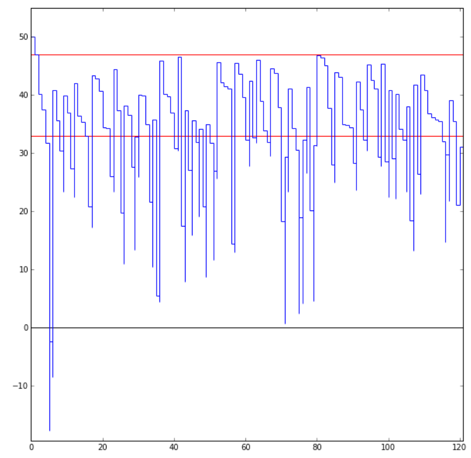

(c) FDSA

Figure 10: Evolution of inventory for 200 iterations, Product A


(a) SPSA


(b) SARS

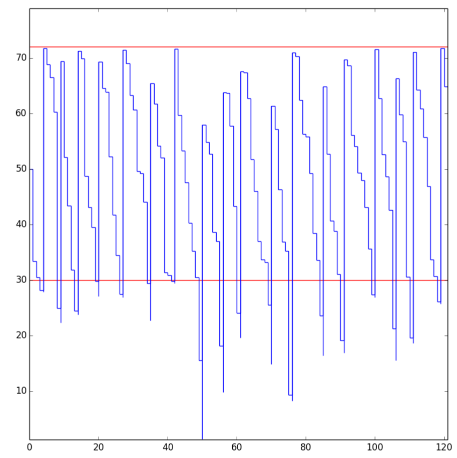
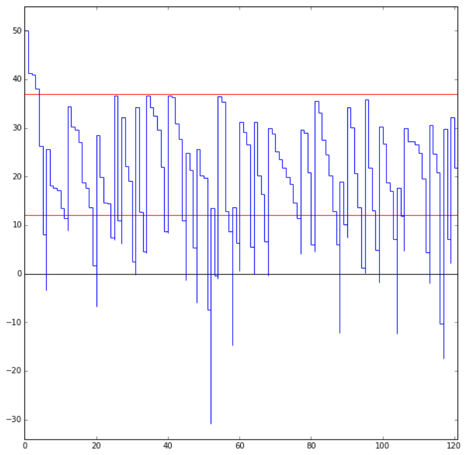
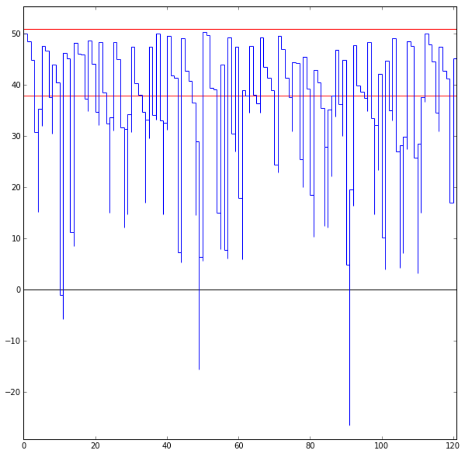

(c) FDSA

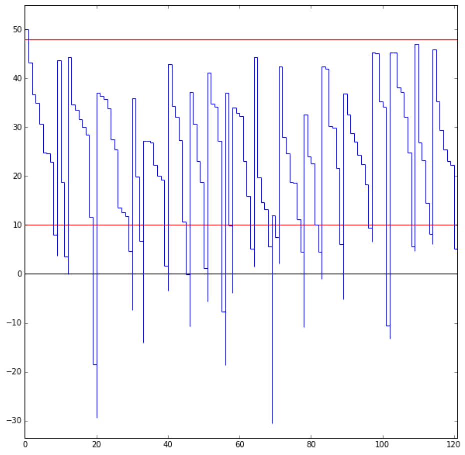
Figure 11: Evolution of inventory for 200 iterations, Product B



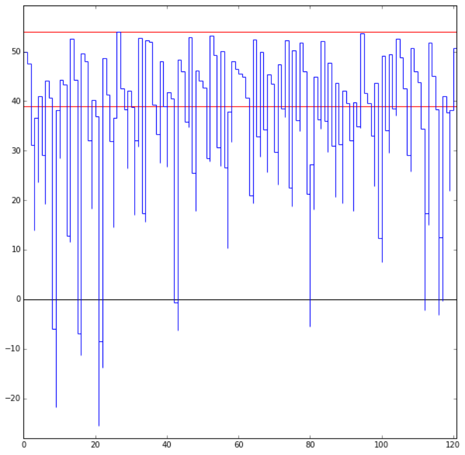
(a) SPSA: Product A



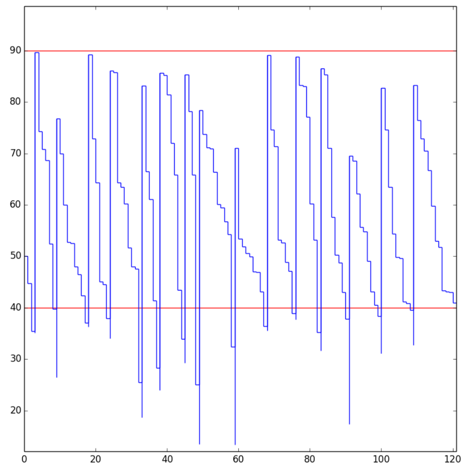
(b) SPSA: Product B



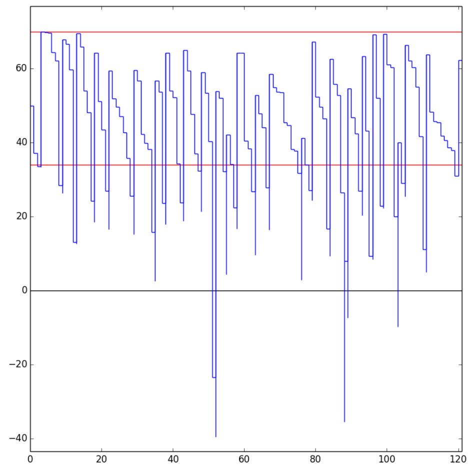
(c) SARS: Product A



(d) SARS: Product B



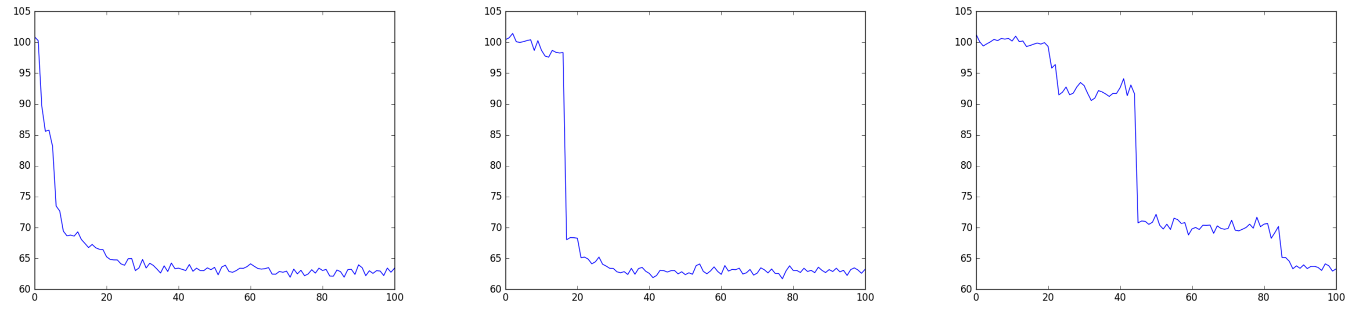
(e) FDSA: Product A



(f) FDSA: Product B

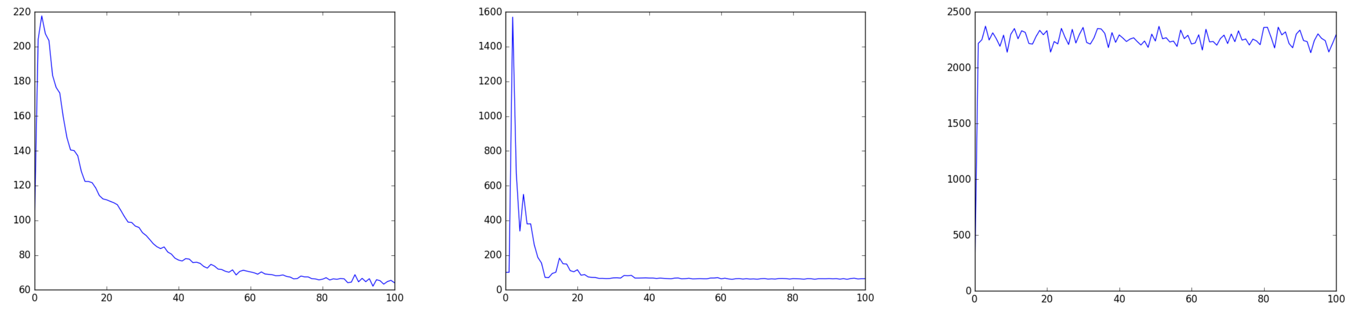
Figure 12: Evolution of inventory for 200 iterations, with joint objective function

Ranking and Selection



(a) $a_k = 100(k+1)^{-1}$ (b) $a_k = 1000(k+1)^{-1}$ (c) $a_k = 5000(k+1)^{-1}$

Figure 13: Convergence of the objective function, Product A, using SPSA-RS with different a_k's



(a) $a_k = 100(k+1)^{-1}$ (b) $a_k = 1000(k+1)^{-1}$ (c) $a_k = 5000(k+1)^{-1}$

Figure 14: Convergence of the objective function, Product A, using SPSA-noRS with different a_k's