

# S<sup>2</sup>YSCODE: A Software Laboratory for Stochastic Systems Control and Decision Algorithms, FORTRAN & MATLAB Versions

Jongsup Choi<sup>1</sup>, Emmanuel Fernández-Gaucherand<sup>2</sup>, Dan Gerhart<sup>3</sup>

Department of Systems and Industrial Engineering  
The University of Arizona  
Tucson, Arizona 85721  
U.S.A.

<sup>1</sup>choi@sie.arizona.edu, <sup>2</sup>emmanuel@sie.arizona.edu, <sup>3</sup>gerhart@bigdog.engr.arizona.edu

## Abstract

This paper presents an overview of the architecture and main features of S<sup>2</sup>YSCODE, a software laboratory for Stochastic SYSTEMS CONTROL and DECISION algorithms. There are two versions currently available, one in FORTRAN 77 and the other a fully transportable MATLAB version. Our principal aims have been to develop a user-friendly, interactive and transportable software laboratory. The current state of development makes S<sup>2</sup>YSCODE specially well suited for pedagogical purposes. The current focus is towards the implementation of algorithms better suited for large scale problems.

**Keywords:** Stochastic Systems; Control and Decision; Dynamic Programming Algorithms, Software Laboratory, MATLAB.

## 1 Introduction

Stochastic control/decision systems (SC/DS) find important applications in many areas, such as: routing and flow control in communication networks, scheduling problems in production and manufacturing, capital investment and management, etc.; see [2], [10], [11], [12]. Dynamic Programming (DP) techniques, and its basic value iteration (VI) and policy iteration (PI) computational algorithms, are the main analytical tools for this kind of problems [1], [2], [5], [11], [12]. The major limitation in the use of these techniques for practical engineering design is the explosion of the computational burden in the number of states and actions.

Moreover, widely applicable and user-friendly algorithm software implementations and/or software tool boxes are conspicuously scarce. For example, "The Extended List of Control Software (ELCS) 5.0" [3] has no listings at all of software especially developed for SC/DS. The only competitive alternative the authors are aware of is the PC-DOS-based software package MDPS [4], which although useful, has a more limited scope since it is aimed mostly at operations research applications, and is not fully transportable.

The development environment for S<sup>2</sup>YSCODE has been the cluster of Sun SPARC stations in the Laboratory for Algorithmic Research, in the Systems & Industrial Engineering (SIE) Department at the University of Arizona.

The organization of this report is as follows. In section 2 we summarize the SC/DS amenable to computational study. Section 3 summarizes the optimal control/decision algorithms. The architecture and commands implemented by S<sup>2</sup>YSCODE to date are discussed in section 4.

## 2 Stochastic Control/Decision Problems

### 2.1 Sequential Decision Problems

We consider discrete-time stochastic systems modeled as a controlled Markov process [1], [2], [5], [10], [11], [12]. The dynamics are described by a system equation as follows:

$$x_{k+1} = f(x_k, u_k, w_k), \quad k = 0, 1, 2, \dots,$$

where given state (S), control (C), and disturbance (D) spaces, we have that,

- $x_k \in \mathbf{S}$  : the state variable at time (stage)  $k$ ,
- $u_k \in \mathbf{C}$  : control action,
- $w_k \in \mathbf{D}$  : disturbance, having a specified probability, distribution  $P(\cdot | x_k, u_k)$ ,
- $\mu_k : \mathbf{S} \rightarrow \mathbf{C}$ ; a control policy at time  $k$  with ,  
 $\mu_k(x_k) \in U(x_k) \subseteq \mathbf{C}$ , where  $U(x)$  is a given action constraint set.

Equivalently, and specially useful for the case of finitely many possible states, the controlled dynamics of the system can be given in terms of a collection of state transition matrices  $\{P(u) = [p_{ij}(u)]; u \in \mathbf{C}\}$ . Here  $p_{ij}(u)$  is the probability of making a transition from state  $x_k = i$  to  $x_{k+1} = j$ , under the action  $u_k = u$  [1], [2], [5], [10], [11], [12]. We refer the reader to the above references for further details on the model.

We consider both finite and infinite horizon cost criteria for optimal control. In particular, the following cost criteria is used.

## 2.2 Finite Horizon Cost (FH)

$$J_N^\pi(x_0) := \mathbb{E}_{x_0, w_0, \dots, w_{N-1}}^\pi \left\{ \sum_{k=0}^{N-1} \alpha^k g[x_k, \mu_k(x_k), w_k] + \alpha^N g_N[x_N] \right\},$$

where we have that

- $x_0$  is the initial state,
- $\pi = \{\mu_0, \dots, \mu_{N-1}\}$  is a policy for the finite horizon,
- $0 < \alpha < 1$  is the discount factor,
- $g(x, u, w)$  is the cost per stage,
- $g_N(x)$  is the (final) cost at stage  $N$ .

## 2.3 Discounted Cost

$$J_\alpha^\pi(x_0) := \lim_{N \rightarrow \infty} \mathbb{E}_{x_0, w_0, \dots, w_{N-1}}^\pi \left\{ \sum_{k=0}^{N-1} \alpha^k g[x_k, \mu_k(x_k), w_k] \right\},$$

## 2.4 Average Cost

$$J^\pi(x_0) := \lim_{N \rightarrow \infty} \frac{1}{N} \mathbb{E}_{x_0, w_0, \dots, w_{N-1}}^\pi \left\{ \sum_{k=0}^{N-1} g[x_k, \mu_k(x_k), w_k] \right\}.$$

# 3 Computational Methods

For problems with FH, the DP algorithm, which is a backward induction method, (see [2], [5], and [12]) is implemented by **S<sup>2</sup>YSCODE**. The two main general-purpose algorithms for computing both optimal values and policies in problems with infinite horizon are policy iteration (PI) and value iteration (VI). Starting from arbitrary initial guesses, both algorithms iteratively seek to improve policies or values, respectively, using the optimality equations and optimality conditions of DP [2], [5], [10], [11], [12]. Both algorithms terminate in a finite number of iterations when the spaces involved are finite sets, as necessarily required in computations. However, when the number of states is large, these methods become impractical because of the large computational overhead per iteration. In this situation, the modified policy iteration (MPI) algorithm, a value-oriented successive approximation method, may be advantageously used. **S<sup>2</sup>YSCODE** implements all these algorithms. We briefly describe the MPI algorithm next.

## 3.1 Modified Policy Iteration

The evaluation step of the policy iteration algorithm is usually implemented by solving a linear system of equations using Gaussian elimination, which requires  $\frac{1}{3M^3}$  multiplication and divisions, where  $M :=$  cardinality of  $\mathbf{S}$ . When the number of states is large, computing an exact solution can be computationally prohibitive. An alternative is to use successive approximations to obtain an approximate solution. This is the basis of the MPI method. The number of successive approximations per MPI iteration,  $m$ , is specified by the user. MPI algorithm can be obtained both for

AC and DC problems. We present below the corresponding MPI algorithm for the DC case, and refer the reader to [10] for further details. The MPI algorithm of order  $m$  is as follows [10], [11]:

- Step 1 : Initialization.  
Select arbitrarily an initial value for the cost-to-go vector  $v^0 \in \mathbf{R}^M$ , specify an error bound  $\epsilon$ , and set  $n = 0$ .

- Step 2 : Policy Improvement.  
For each  $i \in \mathbf{S}$  and each  $u \in U(i)$ , compute

$$g(i, u) + \sum_{j \in \mathbf{S}} \alpha p(j | i, u) v^n(j)$$

Initialize  $U_{n,i}^*$  as the empty set. For each  $i$ , put  $u$  in  $U_{n,i}^*$  if  $u$  attains the maximum in the equation above.

- Step 3 : Policy Evaluation.  
For each  $i$  in  $\mathbf{S}$ , set  $d^n(i)$  equal to any  $u$  in  $U_{n,i}^*$ .

- 3.1  
Set  $k = 0$  and define  $u^0(i)$  by

$$u^0(i) = \max_{u \in U_i} \{g(i, u) + \sum_{j \in \mathbf{S}} \alpha p(j | i, u) v^n(j)\}$$

- 3.2  
If  $\|u^0 - v^0\| < \epsilon(1 - \alpha)/2\alpha$  go to 3.4, ( $\|\cdot\|$  denotes the supremum norm [10, p. 454]). Otherwise continue.

- 3.3  
Compute  $u^{k+1}$  by

$$u^{k+1}(i) = g(i, d^n(i)) + \sum_{j \in \mathbf{S}} \alpha p(j | i, d^n(i)) u^k(j)$$

- 3.4  
If  $k = m$ , go to 3.5. Otherwise increment  $k$  by 1 and return to 3.1.

- 3.5  
Set  $v^{n+1} = u^{m+1}$  increment  $n$  by 1 and go to Step 2.

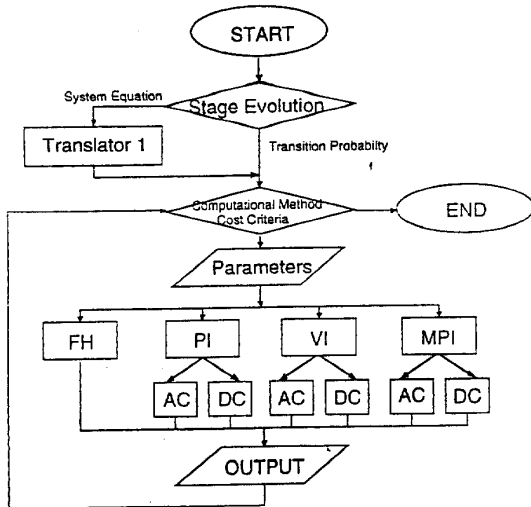
- Step 4 : Termination.  
For each  $i \in \mathbf{S}$ , set  $d^\epsilon(i) = d^n(i)$  and stop.

This algorithm combines features of both PI and VI. Like VI, it is an iterative algorithm that terminates with an  $\epsilon$ -optimal policy [10], [11]; however, VI avoids Step 3 above. Like PI, the algorithm contains an improvement step, Step 2, and an evaluation step, Step 3.

# 4 S<sup>2</sup>YSCODE Architecture and Examples

## 4.1 S<sup>2</sup>YSCODE Architecture

The following diagram shows the main modules in **S<sup>2</sup>YSCODE**, and their interconnections.



A FORTRAN subroutine interactive environment and a MATLAB toolbox, built using M-Files, were written to implement the FH, PI, VI and MPI algorithms. We have tested the FORTRAN programs with up to 700 states. In addition, these programs can generate input data during execution, which is an important feature given that the input data requires a large memory space allocation. We have tested our FORTRAN and MATLAB implementations on SPARC Sun workstations. We present next some computational experiments using benchmark problems found in [2] and [4], illustrating the input data file formats, the interactive capabilities, and output of S<sup>2</sup>YSCODE.

## 4.2 FORTRAN Version

The FORTRAN version of S<sup>2</sup>YSCODE consists of a main program, "dyn", and several subroutines. The user can execute the program interactively, and change options to execute different experiments. The output can be seen on the terminal screen, and it is saved in the output file specified by the user during the interactive process.

### 1. Subroutines.

- Trans: transforms a given system equation description to a transition probability matrix description for the system dynamics.
- SOLFin: computes the optimal policy and value for the finite horizon problem.
- SOLPDis: computes the optimal policy and value for the discounted cost using policy iteration.
- SOLPAve: computes the optimal policy and value for the average cost using policy iteration.
- SOLVDis: computes the optimal policy and value for the discounted cost using value iteration.

- SOLVAve: computes the optimal policy and value for the average cost using value iteration.
- SOLMDis: computes the optimal policy and value for the discounted cost using modified policy iteration.
- SOLMAve: computes the optimal policy and value for the average cost using modified policy iteration.

### 2. Input Parameters.

- ProbTyp : MAXimization or MINimization
- CostTyp : FH, Discounted or Average
- Method : FH, PI, VI, or MPI
- NS : the number of states
- MC : the number of actions (decisions)
- Nstage : the number of stages for FH
- Norder : the order for MPI
- Alpha : the discount rate
- eps : the error bound (epsilon) used in the stopping rule

### 3. Execution Procedures.

In order to execute the FORTRAN version, the user has to type "dyn". Then the user will be prompted for:

- Input Data File Name.  
This file includes all parameters, cost/reward value, transition probability matrix, and initial policies (PI)/values (VI,MPI)/final stage cost (FH).
- Output Data File Name.  
If the output file already exist, then the new output will append to it. Otherwise it will create a new output file.
- Method Type. Select one option among FH, PI, VI, and MPI. The number of stages (FH), error bound (PI, VI, MPI), and number of order (MPI) are required to input according to the previous selection.
- Cost Type. Select one option among FH, discounted cost, and average cost. The discount rate is required in the discounted case.

## 4.3 MATLAB Version

The MATLAB version provides some additional features, e.g., a menu-driven demo facility and a fully integrated architecture. The latter allows the sequential execution of different experiments within the same session. The demo facility informs the beginner user of S<sup>2</sup>YSCODE what the input file structure is, and how to execute the commands. The user can choose the desired command through the integrated menu-driven MATLAB program. The MATLAB version consists of four commands which are "fin", "val", "pol", and "mpi". Each one requires four parameters for execution.

- **fin(NS,NA,STG,Alpha)**: FINITE HORIZON command. Computes the optimal policy based on the given state transition matrices and the costs (or rewards). The parameters to be specified are:
  - NS : the number of states
  - NA : the number of actions
  - STG : the number of stages
  - Alpha : the discount rate
- **val(NS,NA,Alpha,Eps)**: VALUE ITERATION command. Computes the optimal policy based on the given state transition matrix and the costs (or rewards). The parameters to be specified are: NS, NA, Alpha, and Eps which denotes the user-specified error bound.
- **pol(NS,NA,Alpha,Eps)**: POLICY ITERATION command. Computes the optimal policy based on the given state transition matrices and the costs (or rewards). The parameters to be specified are: NS, NA, Alpha, and Eps.
- **mpi(NS,NA,Alpha,Eps)**: MODIFIED POLICY ITERATION command. Computes the optimal policy based on the given state transition matrices and the costs (or rewards). The parameters to be specified are: NS, NA, Alpha, and Eps.

When the functions are invoked with all four arguments the user will be prompted for:

1. Problem Title.
2. Problem Type:  
one word "max", "Max", "min", "Min" must be used to denote a maximization or minimization type problem.
3. Input File Format.  
If the input data, e.g., transition probability matrices, are in ASCII format choose 'a' or 'A'. If the input is of the MATLAB format choose 'm' or 'M'.
4. Cost (or reward) filename.  
This file MUST have a three character suffix (arbitrarily selected). It should contain a  $NS \times NA$  matrix where the  $(i, j)$ th element is the cost (or reward) of being in state  $i$  and implementing action  $j$ . If using MATLAB input, this matrix must be named 'G'.
5. Initial conditions filename.  
This file MUST have a three character suffix (arbitrarily selected). It should contain a row vector of length NS. The  $i$ th element denotes the initial action to use when in state  $i$ . This vector should contain integers only. If using MATLAB input this vector must be named 'U'.
6. Final stage cost filename ("fin" function only):  
This file MUST have a three character suffix (arbitrarily selected). It should contain a row vector of length NS. The  $i$ th element denotes the final cost of each state to achieve at final stage.

7. Transition probability matrix filename.  
This file MUST have a three character suffix (arbitrarily selected). It should contain as many matrices as there are actions ( $k$ ). Each matrix should be of size  $NS \times NS$  where the  $(i, j)$ th element denotes the probability of going from state  $i$  to state  $j$  while under action  $k$ . If using ASCII input, the matrices must be placed one on top of the other (spaces in between are ok). For example, the transition matrix for action 1 should be placed at the beginning with the matrix for action 2 below it and so forth. On the other hand, if MATLAB input is used, these matrices must be named 'p1', 'p2', 'p3'... 'p(k)' but the relative placement of the matrices is irrelevant.
8. Output filename.  
This can be any legal MATLAB filename. The function's output will be sent here once an optimal solution has been found. Existing filenames can be used since the output will be concatenated.
9. Discounted or Average cost analysis.  
Use "D" for Discounted cost analysis or "A" for average cost analysis.

#### 4.4 Translation Module

S<sup>2</sup>YSCODE has the capability of handling either a system equation (e.g., an inventory control problem) or a transition probability matrix (e.g., a queueing control problem) description for the dynamics of the stochastic controlled system of interest. The computational algorithms are naturally formulated in terms of vector values and policies, and transition probability matrices. Therefore, a system equation description has to be *translated* first to that form before algorithmic computations are actually carried-out. This is accomplished by S<sup>2</sup>YSCODE via a *Translation Module*. For example, in the FORTRAN version, the user modifies a FORTRAN function statement, thus defining the system equation. With this, and the (conditional) disturbance probability distribution, S<sup>2</sup>YSCODE obtains an equivalent probability transition matrix description, in an automated fashion, which is then inputted to the computational module. The procedure to perform this translation is straight forward. e.g., see [2]. Given the system equation

$$x_{t+1} = f(x_t, u_t, w_t),$$

with the disturbance distributed as

$$w_t \sim P(\cdot | x_t, u_t),$$

Define sets as follows

$$W(x_t, u_t, x_{t+1}) = \{w_t | x_{t+1} = f(x_t, u_t, w_t)\}.$$

Then, the equivalent transition probability matrix for the system is computed as;

$$P_{x_t, x_{t+1}}(u_t) = P(W(x_t, u_t, x_{t+1}) | x_t, u_t).$$

## 4.5 Examples

### 4.5.1 Queueing Control Problem.

The next simple example is taken from [2, pp. 5-7]. The problem is to optimally control the service rate (either "Fast" or "Slow") in a finite capacity queue (max. buffer capacity = 3). We used the following cost function.

C(i,u)	u = Fast	u = Slow
i = 0	16	1
i = 1	7	7
i = 2	8	8
i = 3	1	9

1. Input Data File: The MATLAB input consists of three separate files:

```

**Contents of cost/reward, filename 'queue.ggg'
16 1
7 7
8 8
1 9
**Contents of final stage cost, filename 'queue.nnn'
0 0 0 0
**Contents of transition probability, filename
'queue.ppp'
0.20 0.50 0.30 0.00
0.16 0.44 0.34 0.06
0.00 0.16 0.44 0.40
0.00 0.00 0.16 0.84

0.20 0.50 0.30 0.00
0.06 0.29 0.44 0.21
0.00 0.06 0.29 0.65
0.00 0.00 0.06 0.94

```

2. Sample Execution.

```

>> fin(4,2,10,1)
Problem Title >> Queueing Control Problem
Problem Type (Max or Min) >> min
(A)scii or (M)atlab input files >> a
Cost/Reward matrix filename >> queue.ggg
Final Stage Cost filename >> queue.nnn
Transition Probability matrix filename >>
queue.ppp
loading...

File where you would like the output to be
sent >> queue.out
.....
====> Output sent to: queue.out

Queueing Control Problem
*****
The Optimal Solution
*****
the problem type: minimization
# of states : 4 # of stages : 10

```

# of actions : 2 discount rate : 1.00

```

STATE CONTROL COST-T0-GO
--stage 0
1 2 31.899581
2 2 35.135471
3 2 31.289594
4 1 22.363492
--stage 1
1 2 29.456936
2 2 32.700203
3 2 28.860307
4 1 19.935527
.....
--stage 9
1 2 1.000000
2 1 7.000000
3 1 8.000000
4 1 1.000000

```

### 4.5.2 Inventory Control Problem.

The following example is taken from Bertsekas [2, pp. 18-22]. The problem is specified by an inventory balance equation. Our translation module automatically generates the transition matrices given below.

1. Input Data File: The FORTRAN input contains all information in a file, which we called, inven.dat for this example.

```

the Inventory disturbance probability
Problem type
min
Ns, Mc
3 3
G(i,k)
3.3 1.7 2.9
0.7 2.9 1e+20
0.9 1e+20 1e+20
k = 1, row i * col j
1.0 0.0 0.0
0.9 0.1 0.0
0.2 0.7 0.1
k = 2, row i * col j
0.9 0.1 0.0
0.2 0.7 0.1
0.0 0.0 0.0
k = 3, row i * col j
0.2 0.7 0.1
0.0 0.0 0.0
0.0 0.0 0.0
initial U
1 1 1

```

2. Sample Execution: Dynamic programming part.
 

```

coyote ] dyn
Type the input data filename(10 chars) :
inven.dat
Type the output data filename(10 chars) :
inven.out

```

```
Type the method
- Policy Iteration : P
- Value Iteration : V
- Modified Method : M
- Finite DP       : F
- Quit           : Q   (1 chars) : f
Type the number of stage : 10
```

```
Type the cost type
- Discounted cost : D
- Average cost    : A
- Nodiscount Finite: N
- Quit           : Q   (1 chars) : n
===> You've got the optimal cost!!!
```

```
*****
The Optimal Solution
*****
the problem type : minimization
# of states : 3 # of stage : 10
# of actions : 3 discount rate : 1.00
```

STATE	CONTROL	COST-TO-GO
=== Stage 0 ===		
1	2	16.10000
2	1	15.10000
3	1	14.54444
=== Stage 1 ===		
1	2	14.50000
2	1	13.50000
3	1	12.94444
.....		
=== Stage 9 ===		
1	2	1.70000
2	1	0.70000
3	1	0.90000

## Acknowledgements

This research was partially supported by The Engineering Foundation under grant RI-A-93-10, and by CIMD under grants F93UR031 and S94US041.

## References

- [1] A. Arapostathis, V.S. Borkar, E. Fernández-Gaucherand, M.K. Ghosh, and S.I. Marcus, "Discrete-Time Controlled Markov Processes with the Average Cost Criterion: A Survey," *SIAM Journal of Control and Optimization*, vol. 31, 1993, pp. 282-344.
- [2] D.P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*, Prentice Hall, 1987.
- [3] D.K. Frederick, R. Kool, M. Rimvall, and A. van den Boom, *ELCS: The Extended List of Control Software 5.0*, monograph, 1992.

- [4] L. Horng and B.F. Lamond, "MDPS-Markov Process System, An Interactive Software Package for the IBM personal computer," *Working Paper no. 89-007*, University of Arizona, 1989.
- [5] P.R. Kumar and P. Varaiya, *Stochastic Systems: Estimation, Identification, and Adaptive Control*, Prentice-Hall, 1986.
- [6] J. MacQueen, Moler, *A Preview of MATLAB*, The MathWorks, Inc., 1992.
- [7] J. Little and C. Moler, "A Modified Dynamic Programming Method for Markovian Decision Problems," *Journal of Math. Anal. and Appl.*, Vol.14, pp. 38-43, 1966.
- [8] *The Student Edition of MATLAB*, The MathWorks, Inc., 1992.
- [9] E.L. Porteus and J.C. Totten, "Accelerated Computation of the Expected Discounted Return in a Markov Chain," *Operations Research*, Vol.26, No.2, pp. 350-358, 1978.
- [10] M.D. Puterman, "Dynamic Programming," *Encyclopedia of Physical Science and Technology*, Vol.4, Academic Press, 1987.
- [11] M.D. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley Interscience, 1994.
- [12] H.C. Tijms, *Stochastic Modelling and Analysis: A computational approach*, John Wiley and Sons, 1986.