SOCSOL-I

A MATLAB PACKAGE FOR APPROXIMATING THE SOLUTION TO A CONTINUOUS-TIME STOCHASTIC OPTIMAL CONTROL PROBLEM

ALISTAIR WINDSOR AND JACEK B. KRAWCZYK

Abstract. Computing the solution to a stochastic optimal control problem is a difficult problem. A method of approximating a solution to a given stochastic optimal control problem using Markov chains was developed in [1]. This paper describes a suite of Matlab functions implementing this method of approximating a solution to a given continuous stochastic optimal control problem.

Working Paper
SOCSol-I/1a

JEL Classification: C63 (Computational Techniques), C87 (Economic Software).
AMS Categories: 93E25 (Computational methods in stochastic optimal control).
Authors' Keywords: Computational economics Approximating Markov decision chains.

Research supported by VUW GSBGM and IGC.

Correspondence should be addressed to: Jacek B. Krawczyk.

About the authors: Alistair Windsor. Honours Student, Mathematics, Victoria University of Wellington. Email: `Alistair.Windsorvuw.ac.nz`
Jacek B. Krawczyk. Commerce & Administration Faculty, Victoria University of Wellington, PO Box 600, Wellington, New Zealand; fax +64-4-4712200. Email: `Jacek.Krawczykvuw.ac.nz`;
http://www.vuw.ac.nz/~jacek

Contents

SOCSol-I
a Matlab Package for Approximating the Solution to a Continuous-Time Stochastic Optimal Control Problem

1. INTRODUCTION

Computing the solution to a stochastic optimal control problem is a difficult problem. A method of approximating a solution to a given stochastic optimal control (SOC) problem using Markov chains was developed in [1]. This paper describes a suite of Matlab[1] functions implementing this method of approximating a solution to a given continuous SOC problem. This should be read and used in conjunction with [1].

2. THE STOCHASTIC OPTIMAL CONTROL PROBLEM

The method deals with finite horizon, free terminal state stochastic optimal control problems. Let the SOC problem be written in the following form

$$\min J(\mathbf{u}, \mathbf{x}_0) = \mathbb{E}\Big\{ \int_0^T f\big(\mathbf{x}(t), \mathbf{u}(t), t\big) \mathrm{d}t + h\big(\mathbf{x}(T)\big) \Big| \mathbf{x}(0) = \mathbf{x}_0 \Big\} \tag{1}$$

subject to

$$\mathrm{d}\mathbf{x} = g\big(\mathbf{x}(t), \mathbf{u}(t), t\big)\mathrm{d}t + b\big(\mathbf{x}(t), \mathbf{u}(t), t\big)\mathrm{d}\mathbf{W} \tag{2}$$

where $\mathbf{W}$ is a standard Wiener process. In the optimisation method, we also allow for constraints on control and state (local and mixed).

Throughout the paper, we shall denote the dimension of the state space by d, the dimension of the control by c, the length of the horizon by T, and the number of variables which are affected by noise by N.

To solve (1) subject to (2) and local constraints, we developed a package of Matlab programmes under the name SOCSol-I. The package is composed of the four main modules:

1. `GenSolve`
2. `GenSim`
3. `ContRule`
4. `ValGraph`

---

[1] See [2] for the introduction to Matlab.

3. GENSOLVE

3.1. **Purpose.** `GenSolve` takes the given S.O.C problem, approximates it with a Markov decision chain, and then solves the Markov decision chain. The result of this process is a discrete-time, discrete-space control rule. `GenSolve` does not perform the interpolation to convert the discrete-time, discrete-space control rule to a continuous-time, continuous-state control rule.

3.2. **Syntax.**

```
GenSolve( DeltaFunctionFile, InstantaneousCostFunctionFile,
     TerminalStateFunctionFile, Minimum, Maximum, StateStep,
     TimeStep, ProblemFile,Options)
```

`DeltaFunctionFile:`

A string containing the name (no `.m` extension) of a file containing a Matlab function representing the equations of motion.

If the problem is stochastic then the function returns a vector of $2d$: the first d are $g\big(\mathbf{x}(t), \mathbf{u}(t), t\big)$ the second d elements are $b\big(\mathbf{x}(t), \mathbf{u}(t), t\big)$. If the problem is stochastic but some of the variables are undisturbed by noise, *i.e.* $N < d$, then the variables for which the diffusion term is constantly 0 must come after the variables which are disturbed by noise.

If the problem is deterministic then the function returns a vector of length d corresponding to the value of $g\big(\mathbf{x}(t), \mathbf{u}(t), t\big)$ only.

In any case the function should have a header of the form:

```
function Delta = DeltaFunction(Control , StateVars , t )
```

where `Control` is a vector of length c, `StateVars` is a vector of length d, and `t` is a scalar.

`InstantaneousCostFunctionFile:`

A string containing the name (no `.m` extension) of a file containing a Matlab function representing the instantaneous cost function, $f\big(\mathbf{x}(t), \mathbf{u}(t), t\big)$ and the (local and mixed) inequality constraints on control and state.

The function returns a vector of $I + 1$ values. The first value is the value of $f\big(\mathbf{x}(t), \mathbf{u}(t), t\big)$. The remaining I values are the values of the inequality constraints which should be set up in an appropriate way for the minimization routine `constr`. The function should have a header of the form:

```
function StageReturn = f( Control , StateVars , t )
```

where `Control` is a vector of length c, `StateVars` is a vector of length d, and `t` is a scalar.

`TerminalStateFunctionFile`:

A string containing the name (no `.m` extension) of a file containing a Matlab function representing the terminal state function, $h\big(\mathbf{x}(T)\big)$.

This function should return only the single real value given by $h\big(\mathbf{x}(T)\big)$ (even if the function is zero). The function should have a header of the form:

```
function TermReturn = h( StateVars )
```

where `StateVars` is a vector of length d.

`Minimum`, `Maximum`, **and** `StateStep`:

These functions determine the finite state grid for the Markov chain which we hope will approximate the SOC problem.

The value of `Minimum` is the least possible state.

The value of `Maximum` is the maximum possible state.

The value of `StateStep` determines the distance between states. It has to be chosen so that the difference between `Minimum` and `Maximum` is divisible by `StateStep`.

Each of these values can be given in two different ways:

1. as a vector of length d where the same set of numbers will apply to every decision stage of the Markov chain.
2. as a matrix of with $S+1$ rows and d columns, where S is the number of decision stages in the Markov Chain and is determined by `TimeStep` below. The i-th row is then used to determine the state grid in the i-th stage.

`TimeStep:`

This determines the number of decision stages their associated times.

`TimeStep` is a vector of step lengths that partition interval $[0, T]$; *i.e.*, their sum should be T. The number of elements of `TimeStep` is the number of decision stages in the Markov decision chain, which we denote S.

`ProblemFile:`

A string containing the name (with no extension) of the problem. This name is used to store the solution on disk. `GenSolve` produces two files with this name; one with the extension `.DPP`, which contains the parameters used to compute the Markov decision chain, and one with the extension `.DPS`, which contains the solution itself.

These files together produce the continuous-time, continuous-state control rule. Notice that the routines `GenSim` and `ValGraph` (explained below) also require that the function files `.DPP` and `.DPS` still exist and are unchanged.

`Options:`

This vector controls various internal setting which need only be adjusted infrequently. The `Options` vector is optional. If you do not pass an `Options` vector then the default values are used.

The first 18 entries are the same as those for the Matlab Optimization Toolbox routines and are just passed directly to the `constr` routine used within the algorithm. The remaining entries are

19. this contains the value c for your problem. The default is 1.
20. this should be set non-zero if your problem is stochastic. The default value is 0, *i.e.* the problem is assumed to be deterministic.
21. this should be set to the number N if $N < d$. The default value is 0 which `GenSolve` interprets as meaning all variables are noisy. (If the problem is deterministic *i.e.* `Option(20)`=0, then this option should be 0.)

To change some of the `Options` values you should first call the function `GenOpts` which returns the default `Options` vector, and then change the appropriate values.

4. GenSim

4.1. **Purpose.** `GenSim` takes the solution computed by `GenSolve`, derives a continuous-time, continuous-state control rule and simulates the continuous system using this rule. It returns graphs of the timepaths of the state and control variables and the associated value.

4.2. **Implementation.** The derivation of the continuous-time, continuous-state control rule from the solution computed by `GenSolve` requires some form of interpolation in both state and time. In the effort to keep the script simple the interpolation in state is linear. States which are outside the state grid simply move to the nearest state grid point. For times between Markov chain times, the control profile for the most recent Markov chain time is used.

The differential equation which governs the evolution of the system is approximated using the Euler-Muruyama approximation scheme. The integral for the performance criterion is approximated using the left-hand endpoint rectangle rule.

4.3. **Syntax.**

```
SimulatedValue = GenSim(ProblemFile, InitialCondition,
     SimulationTimeStep, NumberofSimulations, UserSuppliedNoise)
```

`ProblemFile:`

A string containing the name (with no extension) of the problem. This name is used to retrieve the solution produced by `GenSolve` from the disk. `GenSim` requires that the function files used by `GenSolve` still exist and are unchanged.

`InitialCondition:`

This is a vector of length d which contains the initial condition. The initial condition is the point from which the simulation starts.

`SimulationTimeStep:`

This is a vector of step lengths that partition interval $[0, T]$; *i.e.*, their sum should be T.

The more simulation steps the less error there is from approximating the equations of motion using the Euler-Muruyama scheme and from approximating the performance criterion using the left-hand endpoint rectangle method. In general the simulation step is much smaller that the time step used to compute the solution (*i.e.* 1the `TimeStep` argument to `GenSolve`).

If no `SimulationTimeStep` vector is given, or if the `SimulationTimeStep` given is empty, then the `TimeStep` vector used in computing the `ProblemFile` is used.

`NumberofSimulations:`

This is the number of simulations that should be performed. Making the number negative suppresses plotting of the timepaths and the number of simulations is $|\cdot|$.

Multiple simulations are normally performed when dealing with a stochastic system. Each simulation uses a randomly determined noise realization (unless this is suppressed by the `UserSuppliedNoise` argument).

If no `NumberofSimulations` is given then it defaults to 1.

`UserSuppliedNoise:`

This enables the user to override the random generation of noise realizations.

Passing 0 as this argument uses a constantly 0 noise realization.

This argument accepts a matrix with one row for each entry of `SimulationTimeStep` and N columns.

`NumberofSimulations` should be 1 if this argument is used. This argument is entirely optional, if no `UserSuppliedNoise` is given the noise realization is selected randomly for each simulation.

This argument has no effect for deterministic problems.[2]

`SimulatedValue:`

The output consists of a vector of the values of the performance criterion for each of the simulations performed.

If the problem is stochastic, and the noise realizations are random, then the average of the values from a large number of simulations is an approximation to the expected value of the continuous stochastic system (under the continuous-time, continuous-state control rule derived from the solution computed by `GenSolve`). However, this average is left for the user to compute.

---

[2] The current location of some of the noise code is inefficient for deterministic problems. The code should be moved within stochastic only block.

5. CONTRULE

5.1. **Purpose.**

The purpose of `ContRule` is to produce graphs of the continuous-time, continuous-state control rule derived from the solution computed by `GenSolve`. The control rule profile is produced for a given time and holding all but one state variable constant.

5.2. **Syntax.**

```
ContRule( ProblemFile, Time, InitialCondition, IndependentVariable);
```

`ProblemFile:`

A string containing the name (with no extension) of the problem. This name is used to retrieve the solution produced by `GenSolve` from the disk.

`Time:`

A scalar telling the routine for what time you want to compute the control profile. If `Time` is not a Markov chain time then the first Markov chain time before `Time` is used.

`InitialCondition:`

This vector determines the values of the fixed state variables. The entry for the `IndependentVariable` is not used.

`IndependentVariable:`

This scalar tells the routine which of the state variables to vary. *I.e.*, numbers like "1" or "2" *etc.* have to be entered following the state variables' ordering determined in the function `DeltaFunctionFile`. The control rule profile appears with this state variable along the x-axis.

6. VALGRAPH

6.1. **Purpose.** The purpose of `ValGraph` is to automate the process of computing the expected values for the continuous system (under the continuous-time, continuous-state control rule derived from the solution computed by `GenSolve`) as the initial conditions change. In a similar spirit to `ContRule` above, it deals with one state variable at a time (indicated in `IndependentVariable`), with the other state variables remaining fixed.

6.2. **Syntax.**

```
ValGraph(ProblemFile, InitialCondition, IndependentVariable,
     IndependentVariableVector, SimSteps, NumberOfSimulations, ScaleFactor)
```

`ProblemFile`:

A string containing the name (with no extension) of the problem. This name is used to retrieve the solution produced by `GenSolve` from the disk.

`InitialCondition`:

This vector determines the values of the fixed state variables. The entry for the `IndependentVariable` is not used.

`IndependentVariable`:

This scalar tells the routine which of the state variables to vary. The value graphs appear with this state variable along the x-axis.

`IndependentVariableVector`:

This vector contains the values of the `IndependentVariable` at which you wish to calculate the value of the system.

`SimSteps`:

This is as for `GenSim` above, `ValGraph` calls `GenSim` for each of your initial conditions and then takes the mean of the resulting values.

`NumberOfSimulations:`

This is similar to `NumberOfSimulations` for `GenSim` above. If you specify only one simulation for a stochastic problem then a constantly zero noise realization is used. `ValGraph` automatically suppresses the output of timepaths.

This defaults to 1.

`ScaleFactor:`

This simply multiplies all the resulting values.

Maximization problems are treated by `GenSolve` by taking the negative of all the payoffs and then minimizing. The `ScaleFactor` can be set to -1 thus producing the payoffs that you expect.

This defaults to 1.

7. TECHNICAL INFORMATION

7.1. **Encoding the State Space.** Within the `GenSolve` routine each point of the state grid is assigned a unique number. This state number is then used as an index in all the result matrices.

For each stage a `CodingVector` is computed. As a precursor to this, we compute the number of states along each of the d state variable axes

```
States=round((Maximum-Minimum)./StateStep + 1);
```

where `Maximum`, `Minimum`, and `StateStep` are the appropriate vectors for the current stage. Taking the product of entries of `States` we get the `TotalStates` for the current stage. Now we define the `CdoingVector` as follows

```
c=cumprod(States);
CodingVector=[1,c(1:Dimension-1)];
```

where `Dimensions` is the internal name for d, the number of state variables. This vector is used to compute the state number for each point in the state grid for the current stage.

Suppose `x` is a point on the state grid. We convert `x` into a vector of numbers, `StateVect`, with $1 \leq$ `StateVect(i)` $\leq$ `States(i)`

```
StateVect=round((x-Min)./StateStep+1);
```

Finally the state number, `State`, for `x` is computed by taking the dot product of `StateVect-1` with the `CodingVector` and then adding 1

```
State=(StateVect-1)*CodingVector'+1;
```

This process assigns a unique `State` to each point of the state grid. Every number between 1 and `TotalStates` corresponds to a point of the state grid. By construction 1 corresponds to `Min` and `TotalStates` to `Max`.

Converting a state number back into a point of the state grid is equally simple. The first step is to convert the `State` into a `StateVect`, this is carried out by a function `sntosvec`

```
function StateVect=SNumToSVect(SNum,CVect,Dimension)

SNum=SNum-1;
StateVect=zeros(1,Dimension);
```

```
for i=Dimension:-1:1
     StateVect(i)=floor(SNum/CVect(i));
     SNum=SNum-StateVect(i)*CVect(i);
end
StateVect=StateVect+1;
```

Converting a `StateVect` to its associated `x` is as simple as

```
x=(StateVect-1).*StateStep+Min;
```

7.2. **The Solution Files.** `GenSolve` writes two files to disk. The file with the `.DPP` extension contains the parameters used to compute the approximating Markov chain. The file with the `.DPS` extension contains the solution matrices. These matrices are indexed using the state number encoding of the state space, see section 7.1.

The `.DPP` File:

The first three lines of this file hold, as strings, the variables `DeltaFunctionFile`, `InstantaneousCostFunctionFile`, and `TerminalStateFunctionFile`. These variables are retrieved using `fscanf` with a call to `fgets` after the third `fscanf` to remove the remaining newline.

The remaining portion of the file holds the matrices `Minimum`, `Maximum`, `StateStep`, and `TimeStep`, with space at the end for any additional information (currently `GenSolve` appends the computation time, in seconds, taken to compute the solution). All these variables (with the exception of the computation time) are stored and retrieved using the `matwrite` and `matread` functions. These simple functions store the matrices in a binary form which is more time and space efficient than a text representation.

The `.DPS` File:

This file contains only matrices, written and retrieved by the `matwrite` and `matread` functions. The file begins with the matrices representing the optimal control, there is one matrix for each dimension of the control space. This can be retrieved using a call of the form

```
% Read Optimal Decision Matrices for each control
% dimension.

for i=1:ControlDimension
     eval(['ODM',int2str(i),'=matread(fid);'])
fclose(fid);
```

This is followed by a single matrix holding the cost-to-go for each state of each stage of the Markov chain.

REFERENCES

[1] KRAWCZYK J.B. & A. WINDSOR, 1997, **An Approximated Solution to Continuous-Time Stochastic Optimal Control Problems Through Markov Decision Chains**, Working Paper *Version 8a.* School of Economics and Finance, Victoria University of Wellington.

[2] 1992, **Matlab. High-Performance Numeric Computation and Visualization Software**, *The MathWorks Inc.*