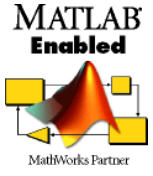
RR 2.0 Premium User's Guide

The Rapid Recursive® Toolbox for MATLAB® provides tools that compose, check and solve sequential decision problems, and report the results in a clear manner. Rapid Recursive technology can natively incorporate critical aspects of human decisions, including uncertainty, risk aversion, the existence of real options, and limited information. This technology closely matches how humans often think about decisions that affect their current situation, and their likely future situation.

These tools have been tailored for use in a broad range of decision problems in business management, investment, valuation, control systems, agriculture, risk evaluation, and choice of therapies.



I. How to contact Supported Intelligence

Inquiry	Email
Contact sales staff or request pricing	sales@SupportedIntelligence.com
Request technical support	help@SupportedIntelligence.com
Report a bug	bugs@SupportedIntelligence.com
Report an error or offer a suggestion about the User's Guide	doc@SupportedIntelligence.com
Provide general feedback	feedback@SupportedIntelligence.com

About Supported Intelligence

Supported Intelligence, LLC was founded in 2012 by Patrick L. Anderson to provide state-of-the-art investment, valuation and decision support software products along with customization services. Supported Intelligence is a partner in The MathWorks Connections Program.

Address: 7189 Gettysburg Dr,
Hudsonville, Michigan 49426 USA

Phone: (973) 800-2024

Website: http://www.SupportedIntelligence.com

License Agreement

The software described in this document is furnished under a license agreement, which protects the intellectual property of the creators of the software and contains other important provisions. The software may be used only under the terms of the license agreement, which is reprinted in Appendix A.

Trademarks

"Supported Intelligence" and "Rapid Recursive" are trademarks of Supported Intelligence, LLC.

Patents

The Rapid Recursive® Toolbox is protected by US patents 9,798,700; 10,460,249; and 10,546,248; Japanese patent 6284472; Korean patent 10-2082522; Supported Intelligence, LLC asserts patent protection in the United States, all Patent Cooperation Treaty countries, and worldwide.

Acknowledgement of Uncertainty and Risks Inherent in Use

The Rapid Recursive® Toolbox is intended to assist licensed users in evaluating future opportunities that involve judgments, predictions, and assessments of future economic, market, and other conditions.
Neither the user nor the licensor of the software can know with certainty the future or predict with confidence all future conditions that may affect the user. Furthermore, economic, market, and other conditions will change in the future.

If any person wishes to use the software to assist in evaluating such opportunities, that person agrees to exercise his or her best judgment when making any related decision, and acknowledges that the responsibility for such decision rests solely with that person. Furthermore, that person is advised to retain competent assistance from business, legal, accounting, economic, actuarial, or other professionals as necessary when making important financial decisions.

This Release

Rapid Recursive® Toolbox for MATLAB version 2.0.0; November 13, 2025

Release Acknowledgements

This 2.0.0 version of the Rapid Recursive® Toolbox relies on the diligent efforts of Dan Lipsy (COO leading product development), Luis Gómez (Senior Software Engineer) and Mohammad Shafiqul Islam (Senior Software Engineer)

Contents

I. Table of Acronyms

COGS	Cost of goods sold
DCF	Discounted cash flow
GUI	Graphical user interface
MDP	Markov decision problem
NPV	Net present value
PI	Policy iteration
RR	Rapid Recursive®
SDP	Sequential decision problem
VFI	Value function iteration

II. Getting Started

Product Overview

The Rapid Recursive® Toolbox for MATLAB® provides tools that compose, check, and solve sequential decision problems, and report the results in a clear manner. Rapid Recursive technology can natively incorporate critical aspects of human decisions, including uncertainty, risk aversion, the existence of real options, and limited information. This technology closely matches how humans often think about decisions that affect their current situation, and their likely future situation.

These tools have been tailored for use in a broad range of decision problems in business management, investment, valuation, control systems, agriculture, risk evaluation, and choice of therapies.

The Rapid Recursive® Toolbox is an approved product of The MathWorks Partner Connections Program.

Key features

Greater Power to Model Uncertainty and Choices

- The ability to compose and solve decision problems involving real options, interactions between decisions and market conditions, and asymmetric risks.
- The native capability to analyze hundreds, or even thousands, of possible scenarios.
- Patented capability to compose, error-check, solve, and report the results of sequential decision problems.

Robust Solution Algorithms

- Three widely-used algorithms for solving discrete-time sequential decision problems: value function iteration, policy iteration, and backward induction.
- The industrial-strength calculation, data handling, and visualization capabilities of MATLAB®.
- The ability to extend the power of the Rapid Recursive® Toolbox by using other toolboxes and functionality within MATLAB®.

Graphical User Interface

- Compose Tool: an interactive graphical user interface (GUI) that allows users to compose and solve a sequential decision problem without needing to program code.
- GUIs for selected solutions templates.

Error Checking

- Extensive error checking tools that provide highly customized error messages for input errors.

Reporting Tools

- Templates that publish a report summarizing a sequential decision problem—including its description, key inputs and solution—in .pdf (Adobe Acrobat), HTML (web page), XML, .doc (Microsoft Word), and .ppt (Microsoft PowerPoint) formats with a single click.
- Tools that report results in customized tables and graphs.

Solution Templates

- A set of included Solution Templates demonstrating how the Rapid Recursive® Toolbox can be used to compose and solve valuation and decision support problems, each of which is provided in open-code format that can be adapted by the user.

Data Import and Export

- Straightforward ability to import from and export to a variety of formats (including .xls, .csv, XML, CDF, & HDF).
- Ability to use various MATLAB® data structures to share data with other MATLAB® users and to export or import these data to users of other statistical, financial, scientific, and technical software in a number of possible formats.

User's Guide

- Information on the theory behind sequential decision problems and recursive models, as well as practical guidance on how to use the Rapid Recursive® Toolbox to model them.
- A guide to the Solution Templates.

Recognition of Your Intellectual Property Along with Ours

- A license agreement that explicitly recognizes the intellectual property rights of both creators of the software, and the users of the software.
- Thousands of lines of unencrypted code that you can use, adapt, revise, and share with other licensed users in a manner consistent with the license agreement.
- The Rapid Recursive® Toolbox license agreement reserves to the licensed user the rights to use and adapt a Solution Template, and report the results to others. Of course, it also prohibits the redistribution or reverse-engineering of the source code, as well as other violations of the intellectual property and other rights. The full text of the license agreement can be found in Appendix A.

Applications of the Rapid Recursive® Toolbox

The uses of sequential decision problems are very broad, so users of the Rapid Recursive® Toolbox come from a wide range of backgrounds.

Here are just a few examples of *who* could use the Rapid Recursive® Toolbox and *how* they might use it:

Academics and Research

- Researchers can use the Rapid Recursive® Toolbox to perform research in fields including, but not limited to: economics, finance, engineering, mathematics, operations research, environmental economics, agriculture, medical decision making, biology and ecology.
- Researchers, professors and graduate students can use the Rapid Recursive® Toolbox to learn about sequential decision problems, Markov decision problems, dynamic programming, control theory, decision processes, and related topics.
- Teaching staff can integrate the Rapid Recursive® Toolbox into coursework.
- Students can use the Rapid Recursive® Toolbox to solve homework problems and complete research assignments.

Investment, Valuation and Decision Support

- Analysts may use the Rapid Recursive® Toolbox for valuation. For example, financial analysts could use the Rapid Recursive® Toolbox to evaluate risky investments such as those involving real estate, pharmaceutical, start-up and technology companies.
- Managers can use the Rapid Recursive® Toolbox to assist in developing business strategy. For example, business managers can use the Rapid Recursive® Toolbox to evaluate oil and gas leases, natural resource rights, patent claims, and other intangible assets that may provide large payoffs—or none at all—in the future.
- Investors can use the Rapid Recursive® Toolbox to evaluate potential investments where real options (such as the option to wait, expand or shut down) are present, where asymmetric risks are evident, or where portfolios are being re-optimized periodically. For example, the Rapid Recursive® Toolbox can be used when analyzing dynamic asset allocation strategies.

Medical Decision Making, Risk Evaluation, and other Uses

- Doctors, patients, and health care institutions can use the Rapid Recursive® Toolbox to model difficult medical decisions involving the interaction of changing conditions, large risks, and significant costs.
- Policy makers, insurers, and business owners can evaluate asymmetric risks, including the risks of significant losses from natural disasters or man-caused events, and the most effective risk mitigation or risk avoidance strategies.

User's Background

Users are expected to have basic knowledge of MATLAB® or experience with another programming language (which would allow the user to quickly gain proficiency in MATLAB®). Only limited knowledge of MATLAB® is required before advanced use of the Rapid Recursive® Toolbox is possible.

Knowledge of other topics will also be beneficial to the user, but is not required. Among the areas of knowledge that would be helpful are:

- Exposure to different types of traditional decision models, such as discounted cash flow and net present value techniques, decision trees, and Monte Carlo analysis
- A background in a quantitative discipline such as economics, finance, engineering, computer science or mathematics
- Any prior knowledge of dynamic programming or Markov decision problems

Although some users of the Rapid Recursive® Toolbox will be proficient with mathematics, this guide assumes a minimal knowledge of mathematics and explains mathematical concepts using words and intuition while maintaining mathematical rigor.

Only limited MATLAB® knowledge required

Prior experience with MATLAB® is not essential to successfully use the Rapid Recursive® Toolbox.

The minimum amount of knowledge of MATLAB® that is recommended to Rapid Recursive® Toolbox users is listed below. As you can see, it is a short list. Even advanced use of the Rapid Recursive® Toolbox is possible with limited knowledge of MATLAB®.

Useful things to know in MATLAB® include:

- How to create 2 and 3 dimensional matrices
- How to perform arithmetic operations, e.g. adding and subtracting
- How to create a string (also known as a character array)
- How to create a cell array
- How to get data from a cell array
- How to create a structure array
- How to get data from a structure array
- How to clear your workspace
- How to run a script
- How to create a loop
- How to write a logical decision statement

Tip: MATLAB® has a number of easy-to-follow introductory videos that explain the basic MATLAB® commands that are useful to know when using the Rapid Recursive® Toolbox.

System requirements

Check that you have the following system requirements (technical specifications):

- 100 MB of disk space
- 1024 MB of RAM (although 2048MB or more is recommended)
- One of the following:
  - Windows: any Intel or AMD x86 processor supporting SSE2 instruction set
  - Mac: any Intel-based Macs with an Intel Core 2 or later
- Any one of the following versions of MATLAB® (32- or 64-bit) installed on your computer (student versions are fine):
  - 8.2.0 (R2013b) or later
- One of the following operating systems:
  - Windows 10 (64-bit)
  - Windows 8 (64-bit)
  - Windows 7 (64-bit)
  - Mac OS X 10.12 (Sierra)
  - Mac OS X 10.11 (El Capitan)

Warning: It is expected that the majority of the features of the Rapid Recursive® Toolbox may work on other operating systems and other versions of MATLAB® but only the ones listed above have been tested.

Compatibility Pledge

During the development and testing process, Supported Intelligence will ensure that all future versions of the Rapid Recursive® Toolbox, beginning with version 2.0.0, are compatible with the current and at least two prior MATLAB® releases, determined at the time of release of the Rapid Recursive® Toolbox. Supported Intelligence will also ensure that the current version of the Rapid Recursive® Toolbox is fully compatible with the current MATLAB® release at all times, which will be accomplished through incremental releases of the Rapid Recursive® Toolbox as necessary.

It is expected that majority of the Rapid Recursive® Toolbox features may work on other versions of MATLAB®, but only those described above will be tested and guaranteed.

Download and Installation

To install the Rapid Recursive® Toolbox, follow these steps:

1. Purchase a license of the Rapid Recursive® Toolbox.
2. Once you have received a license key, you will be able to log onto the Portal using that license key. That is located at: https://supportedintelligence.com/docs/api
3. There, you can choose to download and save the version of the Rapid Recursive® Toolbox that matches your computer system, **Mac or Windows**. This will work with the newest Mac or Windows 10/11 operating systems, with multiple versions built so you can access the version that is appropriate for your operating system and platform. Furthermore, all

versions are signed for your computer to recognize as a legitimate software installer.

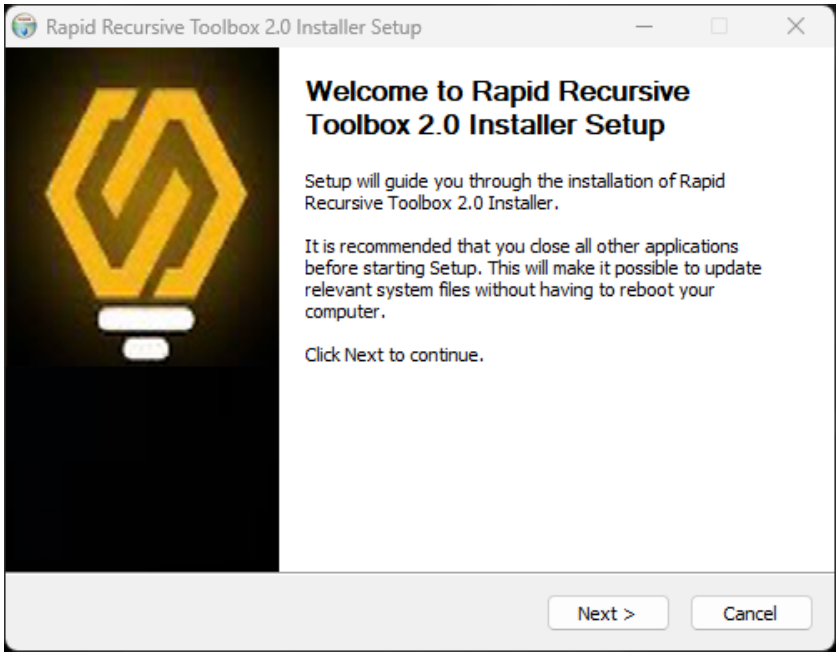
- If you are on Microsoft Windows and are unsure which version to use, use the Microsoft Store link and it will load the version for your system/platform.
- If you are on a Mac and are unsure which version to use, use the Signed Mac (Universal) installer, which will work for either Silicon or Intel Macs.

4. You can choose one of the following direct download options based on your need:
   - Microsoft Store - Windows Store Application
   - Signed Windows Installer (x86) - Version {current_version_number}
   - Signed Windows (x64) - Version {current_version_number}
   - Signed Windows (ARM 64-bit) - Version {current_version_number}
   - Signed Mac (Apple Silicon) - Version {current_version_number}
   - Signed Mac (Intel) - Version {current_version_number}
   - Signed Mac (Universal) - Version {current_version_number}
5. You will then follow the steps provided by the installer. We require that the Rapid Recursive® Toolbox is installed in a directory where the user has write-permissions. Here are some screen you will encounter in the install process:

Phase 1: Running the Installer Setup

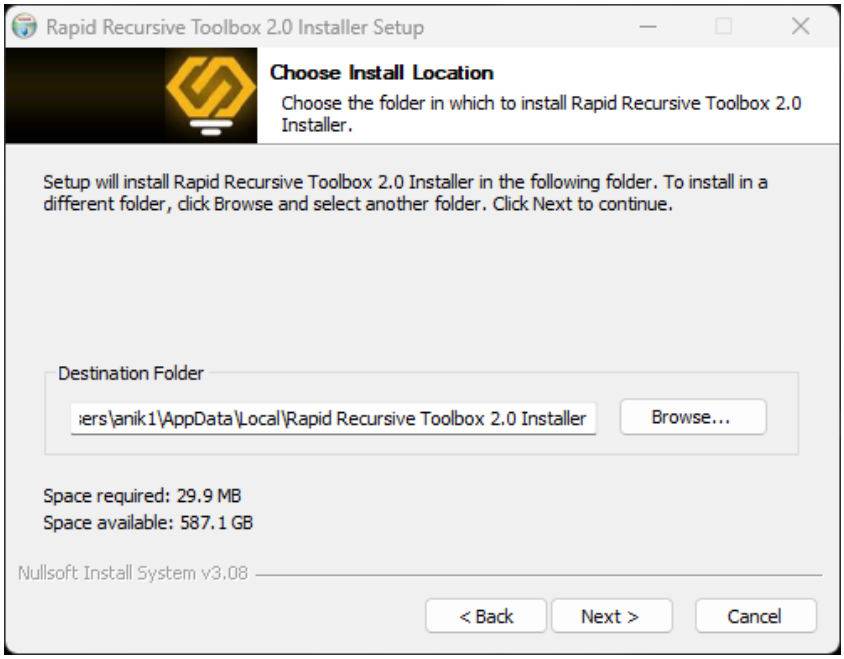
Step 1: Welcome Screen

The initial screen titled "Welcome to Rapid Recursive Toolbox 2.0 Installer Setup". Close other applications and click **Next.**



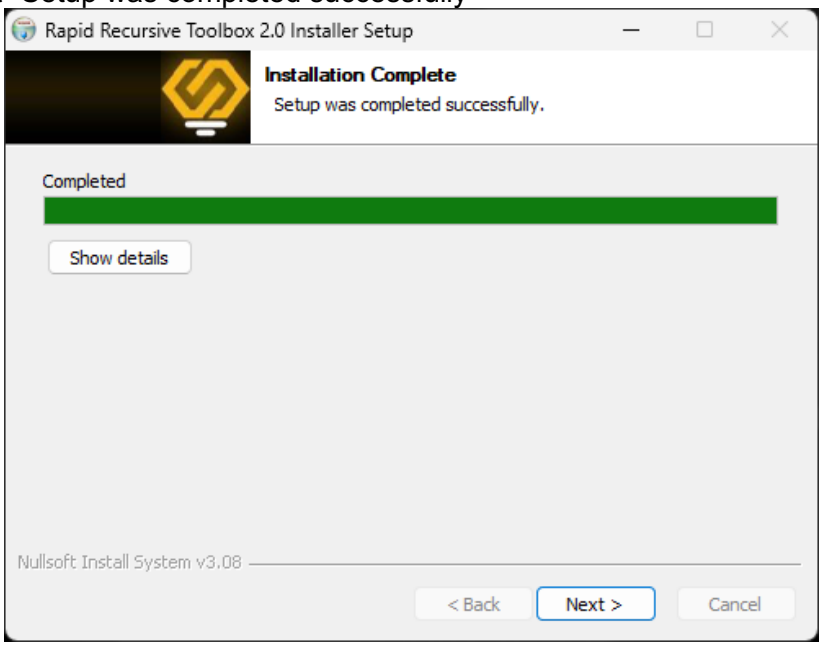
Step 2: Choose Install Location

Choose installer location and click Next.



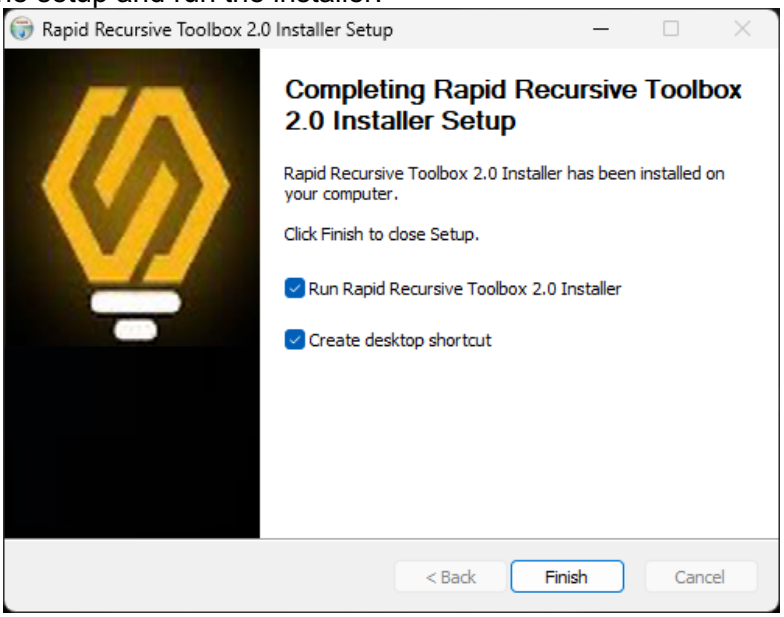
Step 3: Installation Complete

A progress bar shows "Completed," and the screen displays "Installation Complete" confirming that "Setup was completed successfully



Step 4: Completing Setup

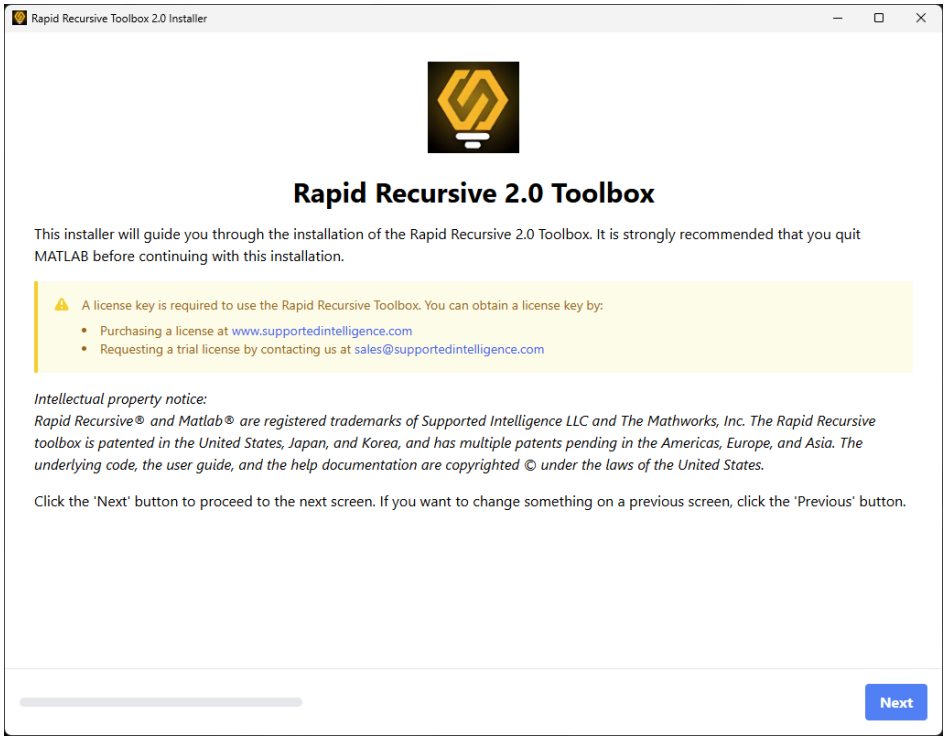
Complete the setup and run the installer.



Phase 2: Running the Rapid Recursive 2.0 Toolbox Installer

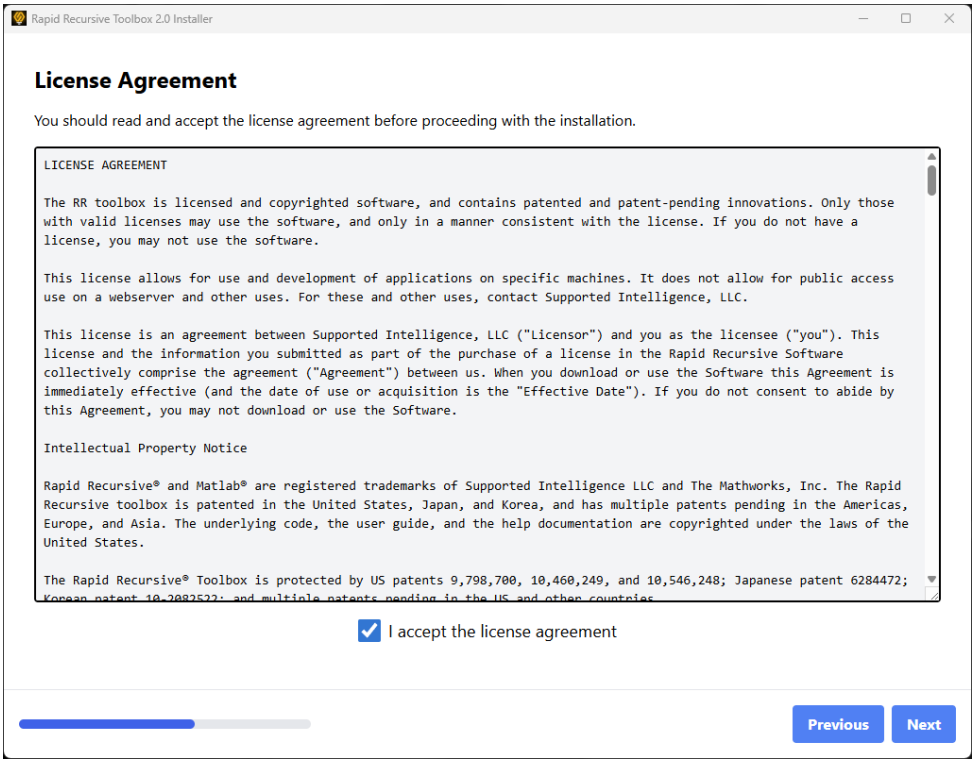
Step 5: Intro Screen

Welcome screen with a warning that a license key is required the setup and run the installer.



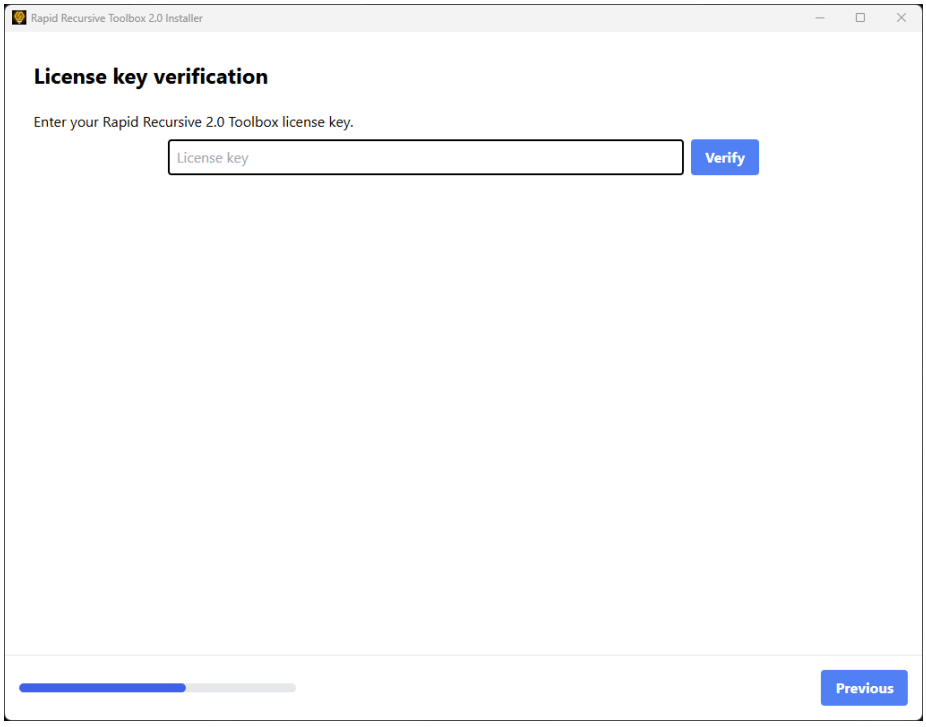
Step 6: License Agreement Screen

Accept license terms to continue.



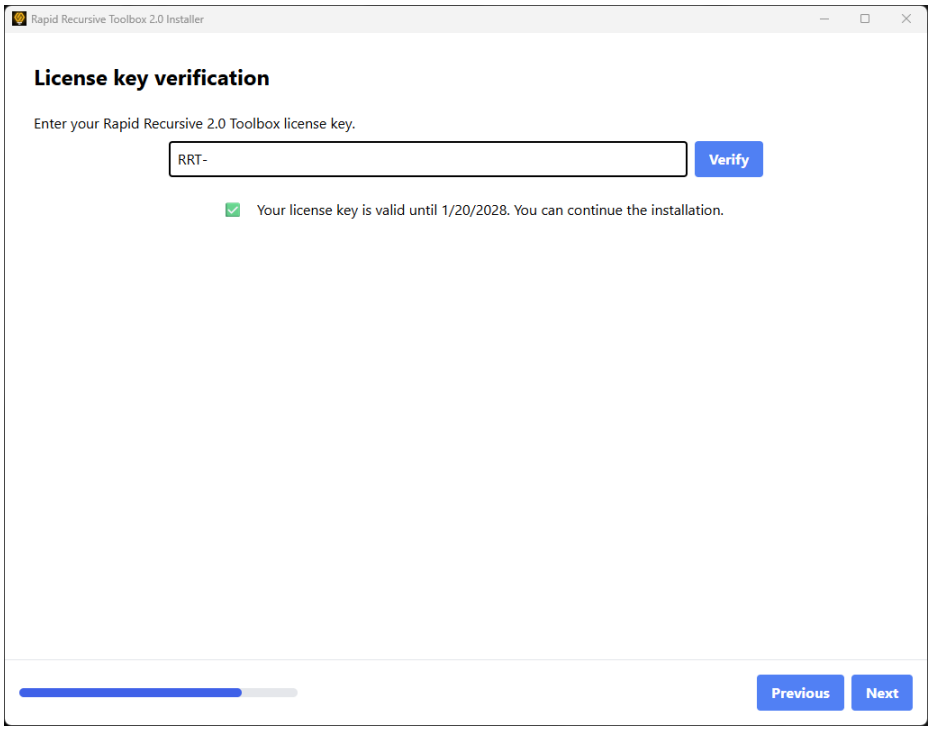
Step 7: License Key Verification Input

Enter license key and verify.



Step 8: License Key Verification Success

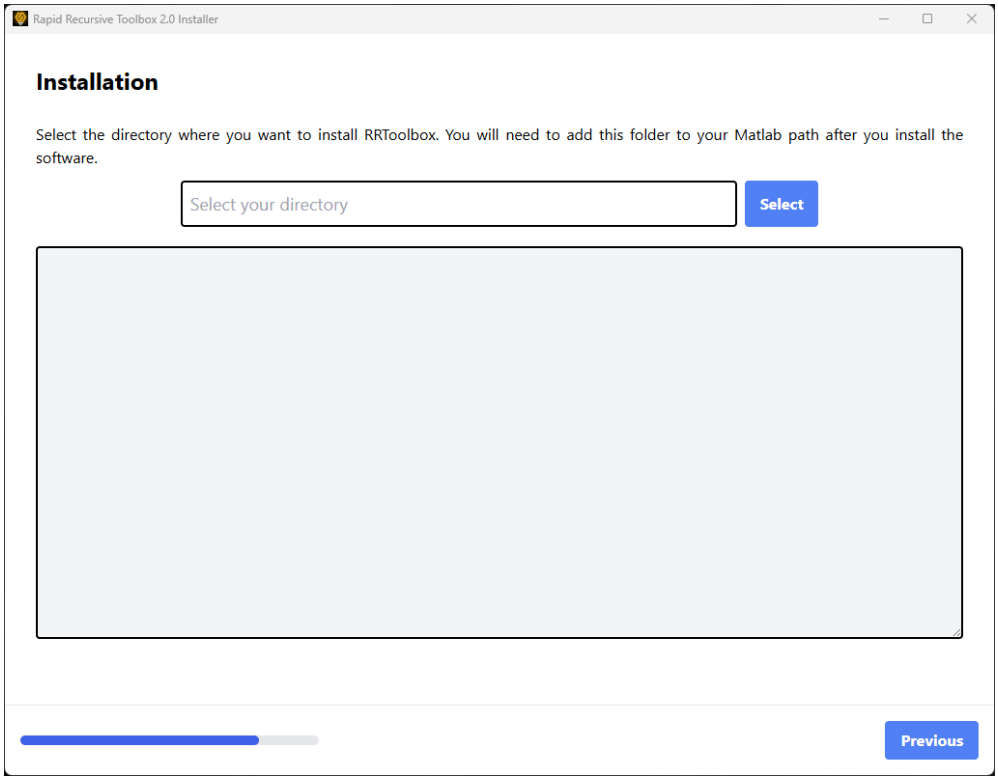
License key has been verified.



Step 9: Installation Folder Selection

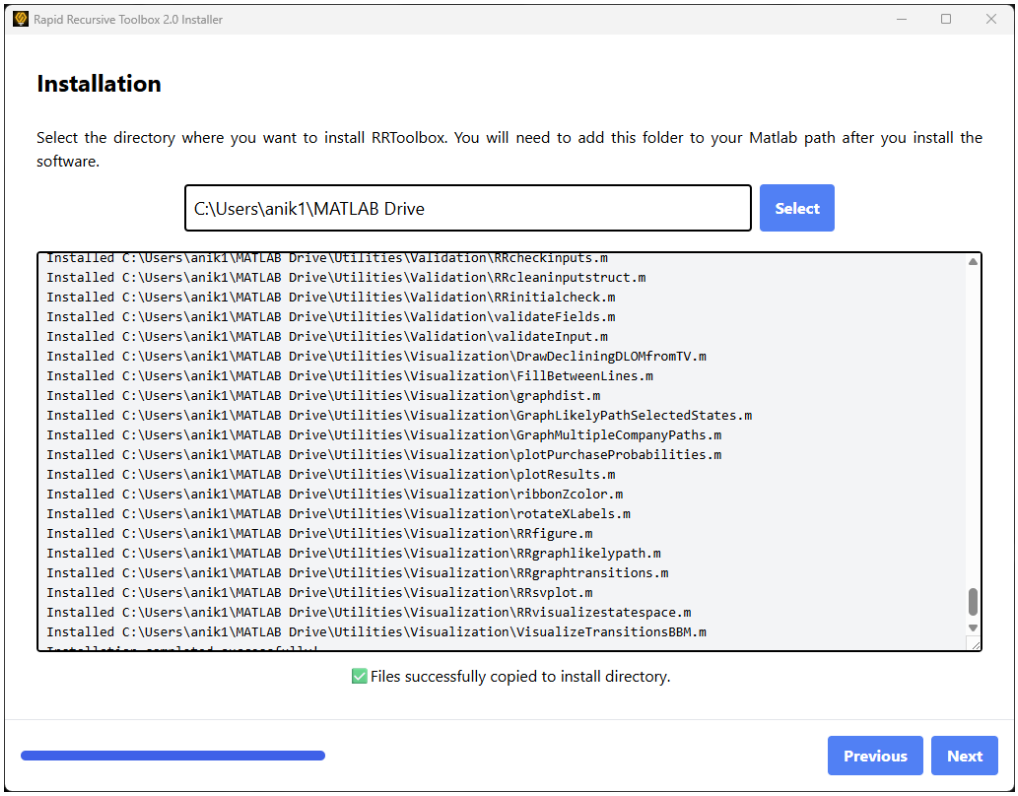
Select or confirm install folder. Recommended defaults:

- Windows:
  - %PROGRAMDATA% or %USERPROFILE%\Documents are some potential places you could install this
  - The MATLAB_DRIVE install folder would be some potential places you could install this, particularly if you want to run the toolbox online using MATLAB® Online
- Mac:
  - ~/Documents or ~/Downloads may be locations you would use
  - ~/MATLAB_DRIVE/ would be some potential places you could install this, particularly if you want to run the toolbox online using MATLAB® Online



Step 10: Installation Confirmation

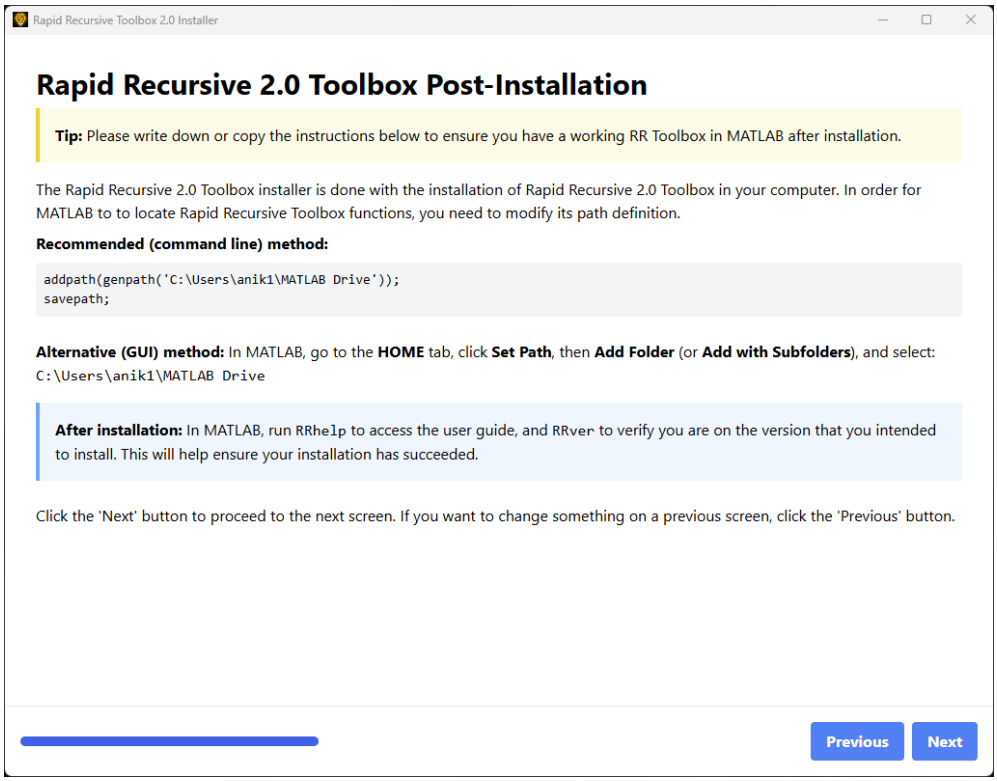
Completes installation with message.



Step 11: Final Instructions Screen

Copy MATLAB path command below and paste into MATLAB Command Window:

```
addpath(genpath('{install_folder_premium_path}'))
savepath
```



Step 12: Installation Confirmation

Open MATLAB® and run the following command from the command line: RRver. If a message indicates the MATLAB version in the command window, your installation is complete.

```
>> RRver
Rapid Recursive® toolbox version 2.0.52
Release date: November 7, 2025

ans =

    '2.0.52'

>>
```

Update

To update the Rapid Recursive® Toolbox, follow these steps depending on the way you installed the prior version:

1 . Direct download users can update via RR Toolbox update portal:

- URL to load: https://supportedintelligence.com/docs/api
- Note: you will need to have an active RR Toolbox license key (that has not expired) to download an update or install/use the RR Toolbox. If you have an issue, please contact support@supportedintelligence.com
- This portal provides the latest update version of the RR Toolbox, with support for the following direct download options:
  - Microsoft Store - Windows Store Application
  - Signed Windows Installer (x86) - Version {current_version_number}
  - Signed Windows (x64) - Version {current_version_number}
  - Signed Windows (ARM 64-bit) - Version {current_version_number}
  - Signed Mac (Apple Silicon) - Version {current_version_number}
  - Signed Mac (Intel) - Version {current_version_number}
  - Signed Mac (Universal) - Version {current_version_number}

2. Microsoft Store

- Microsoft Store will provide you the latest update automatically that will work with your setup

Uninstall

To uninstall the Rapid Recursive® Toolbox, follow these steps:

1. MATLAB®: Go into MATLAB and navigate to the Home tab of the ribbon and click "Set Path" and shift-click to select all RR Toolbox folders and right-click (or shift-click on Mac) and click "Remove".
2. Windows: Use "Add or Remove Programs."
3. Mac: Drag the Rapid Recursive Toolbox {version} Installer.app to Trash.

Quick Start

After installation is successfully completed, to quickly start using the Rapid Recursive® Toolbox choose one of the following tools:

- RRvalueiteration
- See an example of a valuation model

RRvalueiteration

(Recommended for advanced users).

RRvalueiterationis a value function iteration algorithm, which is the most widely used algorithm for solving sequential decision problems. A quick guide to using RRvalueiterationis available in the Function List, here.

Property valuation example

See an example of how to create a valuation model using the Rapid Recursive® Toolbox.

1. Type edit PropertyValuation in the MATLAB® command window and click enter.
2. A file will open up in the MATLAB® editor.
3. Run the file.
4. Read the contents of the file, especially the introductory sections. This step may require knowledge of a small number of MATLAB® commands, but most of the steps are explained in plain English.
5. Run the file again. A table with three columns should appear. Observe the results in this table.

III. Introduction to Sequential Decision Problems and Their Use in Valuation and Decision Support Models

Shortcomings of Discounted Cash Flow

The current ubiquitous way of valuing businesses is the discounted cash flow or net present value method.[1] This method involves forecasting a single scenario into the future, estimating a single stream of profits based on this scenario, and discounting this profit stream to the present using a discount rate.

The discounted cash flow method is considered static and two-dimensional because it does not allow for the possibility that the forecasted scenario changes in the future. This static nature of the discounted cash flow method leads to a number of well-known shortcomings of using discounted cash flow to value businesses. The discounted cash flow method:

- Does not take into account the value of real options that a company may have, such as the option to wait to make an investment, the option to expand capacity if demand increases, the option to sell if the company has a high valuation, and the option to move operations to a more favorable tax environment if tax rates change.
- Usually assumes future states are determined exogenously and does not take into account that future states can be affected by actions today. For example, it does not take into account how greater effort today is likely to lead to better future outcomes.
- Does not take into account strategies or policies that businesses have in place for future contingencies. For example a business might have a contingency plan for the entrance of a new competitor to its market.
- Produces results that are not strictly adhered to by business managers and analysts. For example, it is common for businesses to not strictly follow the rule to invest in a project if its net present value is greater than zero and not to invest otherwise.

Improved Methodology

Using the Rapid Recursive® Toolbox to frame valuation problems as sequential decision problems alleviates the major shortcomings of discounted cash flow analysis. The Rapid Recursive® Toolbox turns the static discounted cash flow model into a *dynamic* model, where future conditions facing businesses may change, and businesses can change their course of action accordingly.

The Rapid Recursive® approach:

- Values real options that businesses have.
- Takes into account the fact that businesses can change course in the future.
- Allows a business's actions today to affect the future conditions facing the business, e.g. expanding capacity can lead to increased sales in the future (or it can lead to increased costs).

[1] Other prevalent methods incude the market and asset methods.

- Provides strategic advice about the best course of action for a given set of conditions facing a business.
- Takes into account a myriad of future possibilities.

Using sequential decision problems to value businesses is discussed in greater detail in the book *The Economics of Business Valuation* by Patrick L. Anderson, published by Stanford University Press.

Sequential Decision Problems: Preface

What follows is an overview of the type of sequential decision problems handled by the Rapid Recursive® Toolbox. This overview is only intended to provide a high level and informal outline of key sequential decision problem topics that are useful for the users of the Rapid Recursive® Toolbox to know.

Reading this overview may be useful for users who have studied sequential decisions problems previously but require a refresher. This overview may also be useful for users who are being introduced to sequential decision problems for the first time and are looking for a "bare bones" guide of what they need to know in order to use the Rapid Recursive® Toolbox. Important topics are discussed that users may be interested in reading about further in other resources.

For more in-depth and formal treatments on sequential decision problems see books such as Puterman (2005) and Powell (2007). Stokey and Lucas (1989) and Ljungqvist and Sargent (2004) also provide excellent treatments of dynamic programming and its applications in macroeconomics.

Sequential Decision Problems: Introduction

A wide variety of real life problems involve "sequential decision problems." These problems involve a decision maker who is making a series of decisions or actions under uncertainty about the future conditions they will face. At each point in time, the action the decision maker chooses not only affects their immediate payoff, but also affects the future conditions the decision maker is likely to face. Typically, the decision maker's objective in a sequential decision problem is to make a series of decisions that will maximize their expected discounted stream of payoffs.

Only a small set of sequential decision problems have analytic or closed form solutions. This means that practical applications of sequential decision problems require numerical methods. The primary numerical method for solving sequential decision problems is dynamic programming.

Further, due to a problem called the "curse of dimensionality," most practical applications consider a subset of sequential decision problems called "Markov decision problems." Markov decision problems are sequential decision problems where the future conditions faced by the decision maker are affected only by the current conditions (and not past conditions) and the decision maker's current action. Markov decision problems are also referred to as dynamic programs, stochastic control problems, or recursive problems.

The Rapid Recursive® Toolbox handles types of Markov decision problems known as "discrete-time infinite-horizon Markov decision problems where the decision maker has the objective of maximizing their discounted stream of rewards." (The meaning of this will be discussed further below).

For ease of exposition, this guide often refers to "sequential decision problems" to mean "discrete-time infinite-horizon Markov decision problems," where the decision maker has the objective of maximizing their discounted stream of rewards. When it is important, the discrete-time or infinite-horizon aspects are emphasized and a distinction is made between *sequential* decision problems generally and *Markov* decision problems specifically.

Sequential Decision Problems: More Formally

The Rapid Recursive® Toolbox handles sequential decision problems that consist of:

1. A series of decision epochs at discrete points in time indexed by tt (time is discrete and divided into periods starting at $tt = 0$ and ending at $tt = TT - 1$, where TT may be finite or infinite).
2. A discrete set of possible states SS in which the decision maker could be.
3. A set of available actions AA for the decision maker.
4. A state transition function, which represents the probability that the state will be $ss' \in SS$ at time $tt + 1$ after the decision maker has taken action aa in state ss at time tt.
5. A reward function, which describes the reward received by the decision maker for taking action a in state s.
6. An objective for the decision maker to maximize their expected discounted reward.

At each decision epoch, the decision maker is in a state $ss \in SS$, which the decision maker observes, and the decision maker can choose an action $aa \in AA_{ss}$ (the ss subscript signifies that the set of actions that the decision maker can choose from may depend on the current state, s).

Immediately after taking an action $aa \in AA_{ss}$ in state ss at time tt, the decision maker receives a reward. The reward the decision maker receives is determined by the reward function, $RR(ss, aa)$, which tells the decision maker the reward they will receive when action aa is taken in state ss. Sometimes, although less commonly, the reward function is $RR(ss, ss', aa)$, which means that the reward at time tt depends on both the state ss at decision epoch tt and the state ss' at decision epoch $tt + 1$.

After making an action and receiving a reward at time tt, the decision maker transitions to a state at time $tt + 1$ that is randomly determined by the state transition function $PP(ss_{tt+1} = ss' | ss_{tt} = ss, aa_{tt} = aa)$. The state transition function represents the probability that the state will be ss' at time $tt + 1$ after the decision maker has taken action aa in state ss at time tt. Note an important aspect of the state transition function that may be easily passed over: **the action performed by the decision maker influences the state the decision maker will be in next period.**

Both the reward function and state transition function are "stationary" or "time homogeneous", which means that $RR(ss, aa)$ and $PP(ss_{tt+1} | ss_{tt}, aa_{tt})$ do not change over time. For example, if the decision maker is in state ss' and takes action aa' at time tt, they will receive the same reward as if they were in time $tt + 1$,

$tt + 2$, and so on. This assumption does not indicate that the decision maker will necessarily be able to reach state ss' at time $tt + 1$, $tt + 2$, etc.

The decision maker knows how their reward will be affected by their action at each possible state (i.e. the decision maker knows the reward function). The decision maker also knows how their actions will influence the likelihood of moving to specific future states (i.e. the decision maker knows the state transition function). Knowing all this information, the decision maker's goal is to find a "policy", which is an action to take in each state at each point in time, that will maximize the expected value of the discounted rewards the decision maker will receive at each decision epoch.

In mathematical notation, the decision maker's problem is:

$$VV(ss_0) = \max_{\{aa_{tt}\}_{tt=0}^{TT-1}} EE\left[\sum_{tt=0}^{TT-1} \beta\beta^{tt} RR(ss_{tt}, aa_{tt})\right]$$

subject to $aa_{tt} \in AA_{ss_{tt}}$ and where $\beta\beta \in [0,1)$ is the discount factor and ss_0 is given.

The solution to this problem consists of an optimal policy $\{aa_{tt}\}_{tt=0}^{TT-1}$ (where T may be ∞)and a value function $VV(ss_0)$. The optimal policy is a complete contingent plan that specifies the best action for the decision maker to take at any state at any time. The value function gives the expected discounted value of rewards assuming the decision maker follows this optimal policy.

This representation is also called the "sequence problem" representation. Most people, especially people who are new to this theory, will find that the sequence problem representation is an intuitive way of thinking about sequential decision problems. Sequential decision problems can be represented in a different form, a "functional equation" form, which is useful because there are existence theorems and methods for solving functional equations.

The functional equation representation of the above sequential decision problem is:

$$VV(ss) = \max_{aa \in AA_{ss}} \{RR(ss, aa) + \beta\beta EEVV(ss', aa)\}$$

where $ss' \in SS$ represents the state in the next epoch. This equation is also referred to as a "Bellman equation." The optimal policy to this problem is denoted *a*(s)*.

Example: Valuing shares with the option to wait to sell

In financial markets, some investors buy shares with the hope that the price of their shares will rise so they can sell the shares for a higher price. However, the prices of shares rise and fall every day, so investors are faced with the difficult decision of deciding the price at which to sell their shares.

The investor's decision can be modeled (and solved) as a sequential decision problem. The following example is a highly simplified way to model the problem. The example is simplified for illustrative purposes, but can be extended to real world problems.

The example is as follows. Consider an investor who owns shares in a non-dividend-paying public company. Assume that the price of the shares changes once per day, and right after the change in the price of the shares the investor decides whether to keep or sell their shares (assume that no partial sales of the shares are possible). If the investor sells, they receive the price of the shares multiplied by the number of shares minus a transaction cost. If the investor does not sell, they receive nothing. Assume that the movement of the share price is determined using a conditional probability density function where the share price tomorrow is dependent on the share price today e.g. Brownian motion (also known as a Wiener process).

In the language introduced in the previous section, decision epochs occur once a day; the actions available to the decision maker are "sell" or "do not sell", but after the decision maker sells he only has access to the "do not sell" action; the states are the prices of the shares and whether or not the decision maker has sold; the state transition function is based on Brownian motion; the reward function is the number of shares multiplied by the price of the shares minus transaction costs if the shares are sold and zero otherwise; and the investor's objective is to maximize the expected value of the discounted rewards.

When this model is solved (using techniques described later in this guide), it turns out that the optimal policy of the investor is to keep the shares when the price is below a certain amount, called a *reservation price*, and to sell when it is above the reservation price. The reservation price depends on the investor's discount factor as well as the distribution governing the movement of the share price, the state transition function.

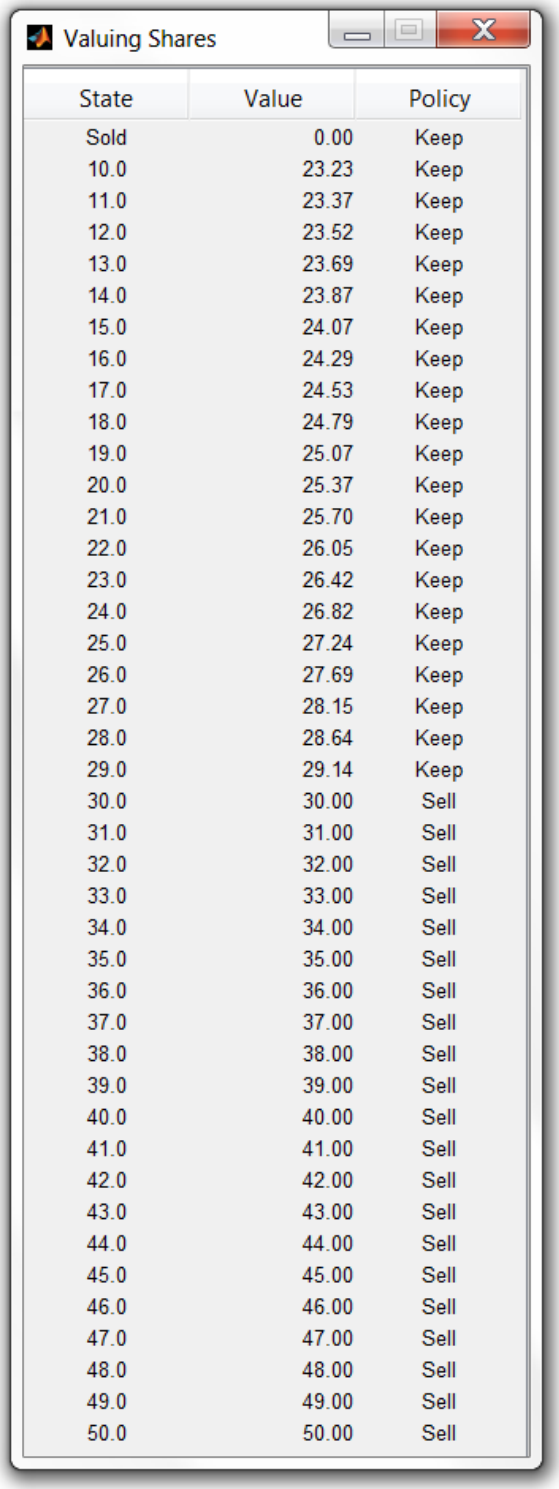
The value function, the second part of the solution to the model, estimates the value of the shares for every possible realization of the share price today.

See Figure 1 for an example of a solution to this example.

While the market valuation of the shares at any point in time is equal to the share price multiplied by the number of shares the investors owns minus a transaction cost, the sequential model provides a different valuation. If the share's market price is below the *reservation price*, the value of the shares as estimated by the sequential model is actually higher than the market price—this difference is the value of the opportunity for the investor to hold on to their shares and sell them when the share price is higher. This opportunity is called the investor's "real option" to wait to sell their shares.

This example can be extended in a number of ways to be made more realistic. For example, the investor could be given the ability to sell part of his shares, buy more shares, or to buy or sell shares of another company.

Figure 1: Valuing Shares Example



State	Value	Policy
Sold	0.00	Keep
10.0	23.23	Keep
11.0	23.37	Keep
12.0	23.52	Keep
13.0	23.69	Keep
14.0	23.87	Keep
15.0	24.07	Keep
16.0	24.29	Keep
17.0	24.53	Keep
18.0	24.79	Keep
19.0	25.07	Keep
20.0	25.37	Keep
21.0	25.70	Keep
22.0	26.05	Keep
23.0	26.42	Keep
24.0	26.82	Keep
25.0	27.24	Keep
26.0	27.69	Keep
27.0	28.15	Keep
28.0	28.64	Keep
29.0	29.14	Keep
30.0	30.00	Sell
31.0	31.00	Sell
32.0	32.00	Sell
33.0	33.00	Sell
34.0	34.00	Sell
35.0	35.00	Sell
36.0	36.00	Sell
37.0	37.00	Sell
38.0	38.00	Sell
39.0	39.00	Sell
40.0	40.00	Sell
41.0	41.00	Sell
42.0	42.00	Sell
43.0	43.00	Sell
44.0	44.00	Sell
45.0	45.00	Sell
46.0	46.00	Sell
47.0	47.00	Sell
48.0	48.00	Sell
49.0	49.00	Sell
50.0	50.00	Sell

Solving Sequential Decision Problems

Now that a sequential decision problem has been defined, the next question is "do solutions to sequential decision problems exist, and if they do exist, how does one find the solution?" Fortunately, there are a number of theorems that state the conditions under which solutions to a sequential decision problem exist and methods that can be used to find a solution if it exists.

A brief discussion of Blackwell's sufficient conditions is included in Appendix B.

Two algorithms for solving sequential decision problems, value function iteration and policy iteration, are also described in Appendix B.

Solving a Sequential Decision Problem Using Rapid Recursive®

The Rapid Recursive® Toolbox includes widely used algorithms for solving discrete-time Markov decision problems where the decision maker's objective is to maximize their expected discounted stream of rewards:

- RRvalueiteration(value function iteration, for solving infinite-horizon problems)
- RRpolicyiteration(policy iteration, for solving infinite-horizon problems)
- RRbackwardinduction(backward induction, for solving finite-horizon problems)

These solution algorithms can be run in MATLAB® with only a few end-user-supplied inputs: a transition matrix, ***P***; a reward matrix, ***R***; and a discount factor, ***beta***. The backward induction algorithm also requires the number of periods your problem lasts for, ***T***. A simple guide on how to use these algorithms in MATLAB® can be found in the Function List under: RRvalueiterationand RRpolicyiteration, and RRbackwardinduction.

IV. Rapid Recursive® Toolbox

"Input" and "Out" structures

The Rapid Recursive® Toolbox includes innovative ways for users to share their models and to easily pass inputs into functions in the Rapid Recursive® Toolbox. Users of the Rapid Recursive® Toolbox are able to share their models and the results of their models by sharing "***Input***" structures and "***Out***" structures.
Using ***Input*** structures and ***Out*** structures makes it easier to run functions from the Rapid Recursive® Toolbox as many of these functions accept "***Input***" and/or "***Out***" structures as inputs.

Although users are not required to use ***Input*** and/or ***Out*** structures when using the Rapid Recursive® Toolbox, their use is highly recommended.

Input structure

An "***Input***" structure is a structure array that contains input parameters that define and describe a sequential decision problem and how it will be solved. The default fields that are contained in an ***Input*** structure are listed in Table 1.

To create an ***Input*** structure, it is recommended that RRcreateinputstructis first used to create an ***Input*** structure containing empty values for its fields. Doing this will ensure that the order of the fields in the ***Input*** structure remains consistent. After this step, the empty fields can be filled in.

The values in ***Input*** can be retrieved the same way values can be retrieved from any structure array in MATLAB®. For example, the syntax:

- Input.P provides the transition matrix
- Input.R provides the reward matrix

Out structure

An "***Out***" structure is a structure array containing fields for results of a sequential decision problem. It is recommended that users obtain their ***Out*** structures from the first output argument of a run of RRvalueiteration, RRpolicyiteration, or RRbackwardinduction rather than creating ***Out*** structures on their own. Following this recommendation will save effort because most of the fields of the ***Out*** structure are filled in by RRvalueiteration, RRpolicyiteration, or RRbackwardinduction.You will also prevent errors using other Rapid Recursive® Toolbox functions later. More information about the fields that are contained in ***Out*** structures is listed in Table 2.

The values in ***Out*** can be retrieved the same way values can be retrieved from any structure array in MATLAB®. For example, the syntax:

- Out.V will provide the value function
- Out.policy will provide the policy function

Input *and* ***Out*** *structures can be saved as .mat files. Doing this allows these structures to be stored for future use or shared with other users. To save a variable as a .mat file in MATLAB®, click on the variable in your workspace, right click and then click "****Save As…****". To load the variable back into your workspace, simply double click on the file that you saved.*

Table 1: Input structure fields

Field	Description
desc	Title of the problem (a string). The default value of this field is 'Untitled sequential decision problem'.
stateinfo	A string or cell array containing relevant information about the state space for a particular problem.
S	The number of states.
states	1 x S vector of state values.
statelabels	1 x S cell array containing labels for the states in the sequential decision problem, where S is the number of states in the sequential decision problem. Each cell statelabels{j}should contain a string that represents the name of the j-th state in the sequential decision problem.
S1	The number of states in the first state dimension.
S1labels	A 1 x S1 cell array containing labels for the states along the first dimension of the state space for multidimensional problems.
S1note	A string or cell array containing relevant information about the first dimension of the state space.
S2	The number of states in the second state dimension.
S2labels	A 1 x S2 cell array containing labels for the states along the second dimension of the state space for multidimensional problems.
S2note	A string or cell array containing relevant information about the second dimension of the state space.
actioninfo	A string or cell array containing relevant information about the action set for a particular problem.

A	The number of actions.
actions	A 1 x A vector of action values.
actionlabels	1 x A cell array containing labels for the actions in the sequential decision problem, where A is the number of states in the sequential decision problem. Each cell actionlabels{j}should contain a string that represents the name of the j-th action in the sequential decision problem.
actioncosts	A 1 x A vector of the costs associated with taking each possible action.
transitioninfo	A string or cell array containing relevant information about the transition probabilities for a particular problem.
P	(Required) Transition matrix. For a detailed description on how to create a transition matrix see the documentation for RRvalueiterationor RRpolicyiteration.
P1	The S1 x S1 x A transition matrix for states in the first state dimension.
P2	The S2 x S2 x A transition matrix for states in the second state dimension.
rewardinfo	A string or cell array containing relevant information about the rewards for a particular problem.
R	(Required) Reward matrix. For a detailed description on how to create a reward matrix see the documentation for RRvalueiterationor RRpolicyiteration.
beta	A discount factor that must be strictly greater than 0 and no more than 1.
d	Discount rate.
g	Growth rate.
epsilon	(Used in value function iteration only). ***epsilon*** is the threshold for the maximum difference between the value function found by this algorithm and the true value function.
policy0	(Used in policy iteration only). ***policy0*** (S x 1) is the initial value of the

	policy that is iterated on during policy iteration.
maxiter	The maximum number of iterations that can occur before the solution algorithm stops.
V0	(Used in value function iteration only). ***V0*** is an S x 1 vector that serves as the starting point for value function iteration.
policyevalmethod	(Used in policy iteration only). Entering ***policyevalmethod*** = 0 specifies that the "policy evaluation" step in the policy iteration algorithm is performed using Gaussian elimination with partial pivoting. Entering ***policyevalmethod*** = 1 specifies that the "policy evaluation" step is performed using the Jacobi method. By default, RRpolicyiteration uses the Jacobi method.
verbose	Whether output from each iteration is displayed to the command window. Setting ***verbose*** to false will increase the speed of the algorithm.
periodicity	Frequency of periods, e.g. yearly, monthly, weekly.
T	Number of periods (can be passed as the number 'inf').
Tstates	A table listing all of the states for the current problem.
Tactions	A table listing all of the actions for the current problem.
Tactioncosts	A table listing all of the action costs for the current problem.
Treward	A table listing the rewards for the current problem.
Tparams	A table listing the key parameters for the current problem.
Tgd	A table displaying the discount factor and discount and growth rates for the current problem.
Ts1	A table listing the states along the first state dimension of the current problem.
Ts2	A table listing the states along the second state dimension of the current problem.
Table1	Miscellaneous table.
Table2	Miscellaneous table.

Table3	Miscellaneous table.
parameter1	Miscellaneous parameter.
parameter2	Miscellaneous parameter.
parameter3	Miscellaneous parameter.
note1	Miscellaneous note.
note2	Miscellaneous note.
date	Time and date. This field is usually filled in automatically.

Table 2: Out structure fields

Field	Data filled by	Description
V	RRvalueiteration, RRpolicyiteration, or RRbackwardinduction	S x 1 vector representing the value function, which along with the optimal policy, forms the solution to a sequential decision problem. The element in the s-th row of ***V*** represents the maximum value of the decision maker's objective function given the s-th state is the first state the decision maker is in.
V_sa	RRvalueiteration or RRpolicyiteration	S x A matrix representing the value of being in a specific state, represented by the row, and taking the action represented by the column, assuming that the optimal policy is followed in all future time periods.
policy	RRvalueiteration, RRpolicyiteration, or RRbackwardinduction	S x 1 vector representing the optimal policy, which along with the value function, forms the solution to a sequential decision problem. The s-th element of ***policy*** represents the optimal action for the decision maker when they are in the s-th state.
iterations	RRvalueiteration, RRpolicyiteration, or RRbackwardinduction	The number of iterations of the selected algorithm that occurred before the solution was found.
calculationtime	RRvalueiteration, RRpolicyiteration,	The number of seconds needed to find the

	or RRbackwardinduction	solution.
Input	RRvalueiteration, RRpolicyiteration, or RRbackwardinduction	An ***Input*** structure. There is only a value for this field if the user used an ***Input*** structure in RRvalueiteration or RRpolicyiteration.
desc	User	The title of the model (a string). If Out is created via an Input structure, Input.desc is carried over to Out.desc.
resultstable	RRvalueiteration, RRpolicyiteration, or RRbackwardinduction	A table of results using MATLAB's new table function (compatable with MATLAB version R2013b or later only).
algorithm	RRvalueiteration, RRpolicyiteration, or RRbackwardinduction	The type of algorithm used to find the solution (a string, either 'value function iteration' or 'policy iteration')
date	RRvalueiteration, RRpolicyiteration, or RRbackwardinduction	The time and date the solution algorithm was run (a date string).
note1	User	Empty. Can be filled in by user with notes.
note2	User	Empty. Can be filled in by user with notes.

Input Validation and Error Checking

Anyone with experience programming knows how easy it is to make simple coding mistakes. To help users create error-free inputs for RRvalueiteration, RRpolicyiteration, or RRbackwardinduction, the Rapid Recursive® Toolbox provides a tool that checks the inputs you plan to use in RRvalueiteration, RRpolicyiteration,or RRbackwardinduction.

RRcheckinputsprovides a detailed message if any errors are found, provides advice on how the error might have come about, and may also suggest how to fix the error. A simple guide on how to use RRcheckinputs can be found in the Function List: RRcheckinputs.

Displaying Results

After finding the solution to a sequential decision problem, the Rapid Recursive® Toolbox offers a number of ways to display results.

Displaying results to the screen

Results can be displayed to the screen using the dispcommand from MATLAB® and a results table that is stored in ***Out***, the first output argument of RRvalueiteration,RRpolicyiteration,and RRbackwardinduction. This results table can be found in the ***resultstable*** field of the ***Out*** structure.

Here is sample code that displays the results table to the screen (this example assumes that the user has already defined the variables ***P***, ***R*** and ***beta***):

```
Out = RRvalueiteration(P,R,beta); Out.resultstable
```

The command window displays:

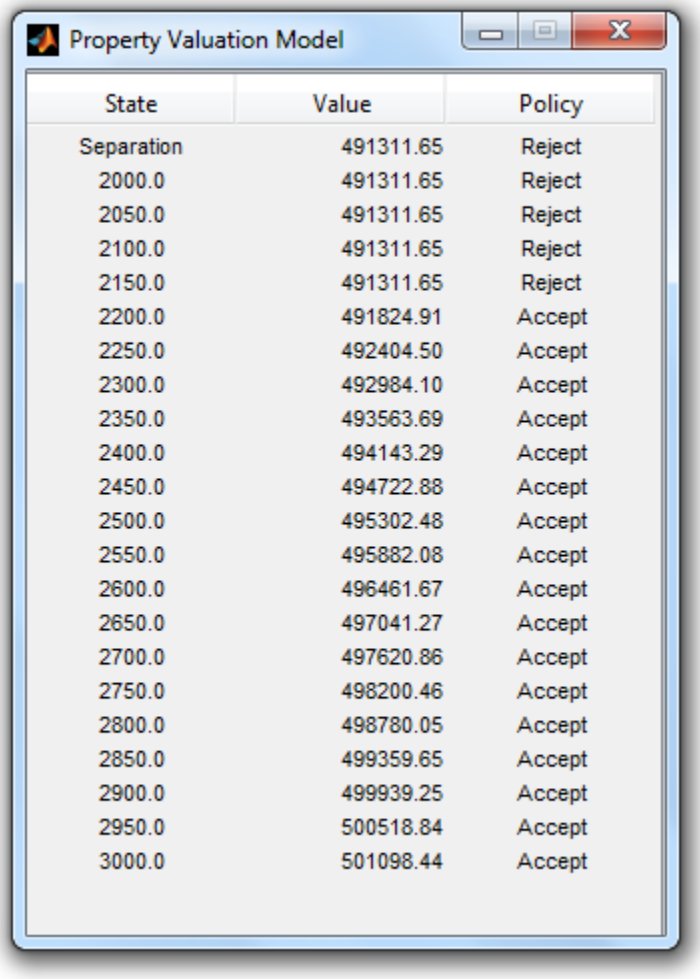
State	Value	Policy
1	2060	3
2	2284	3
3	2518	3
4	2779	2

Creating a uitable

An alternative to displaying results on the screen is to create a MATLAB® uitable using the Rapid Recursive® Toolbox's RRtable,which is specially designed to report results to sequential decision problems. RRtable creates a formatted uitable with a title, and can also include labeling of the states and actions if this information is provided.

You can see a screen shot of a uitable created by RRtablein Figure 2. A simple guide to using RRtable can be found in the Function List.

Figure 2: Screen shot of formatted uitable



State	Value	Policy
Separation	491311.65	Reject
2000.0	491311.65	Reject
2050.0	491311.65	Reject
2100.0	491311.65	Reject
2150.0	491311.65	Reject
2200.0	491824.91	Accept
2250.0	492404.50	Accept
2300.0	492984.10	Accept
2350.0	493563.69	Accept
2400.0	494143.29	Accept
2450.0	494722.88	Accept
2500.0	495302.48	Accept
2550.0	495882.08	Accept
2600.0	496461.67	Accept
2650.0	497041.27	Accept
2700.0	497620.86	Accept
2750.0	498200.46	Accept
2800.0	498780.05	Accept
2850.0	499359.65	Accept
2900.0	499939.25	Accept
2950.0	500518.84	Accept
3000.0	501098.44	Accept

displist

displist displays a list to the screen. This can be useful for displaying the list of states or list of actions to the screen, for example:

```
displist(100:100:700)

                --List-- 100
                   200
                   300
                   400
                   500
                   600
                   700
               ------------
```

A simple guide to using displistcan be found in the function List: displist.

dispreward

dispreward creates a table that contains labels for the states and actions next to the reward matrix. This can be helpful when creating a reward matrix.

For example, you can display the following table in the command window with:

```
R = [-50 50; -100 100];
actionlabels = {'buy','sell'}; statelabels = {'low', 'high'};
table = dispreward(R,statelabels,actionlabels); disp(table)
```

```
'Reward Matrix'    'buy'     'sell'
'low'              [ -50]    [  50]
'high'             [-100]    [ 100]
```

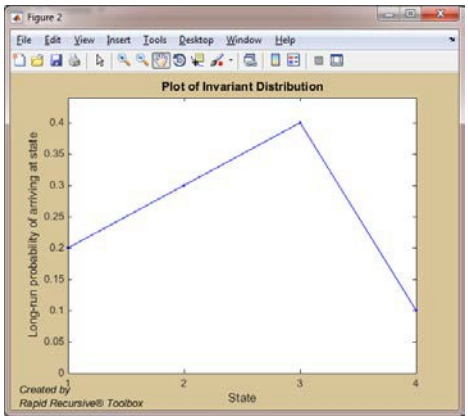
A simple guide to using dispreward can be found in the function List: **dispreward**.

graphdist

graphdist graphs a probability mass function. This can be useful for graphing the stationary distribution of a Markov chain, for example.

Example:

```
dist = [0.2 0.3 0.4 0.1];
graphdist(dist);
```



A simple guide to using graphdist can be found in the Function List: **graphdist**.

Comparing Results With DCF

The Rapid Recursive® Toolbox includes a function called capitalizedvalue that calculates the present value of a perpetual stream of income using the Gordon growth formula. capitalizedvalue can be used to compare the valuation of a business using dynamic programming versus the discounted

cash flow method. A simple guide on using capitalizedvaluecan be found in the Function List: capitalizedvalue.

Tip: If you use capitalizedvalue*to make this comparison, make sure that the timing of cash flows in your recursive problem is the same as the timing of cash flows in the Gordon Growth formula.*

Hint: A recursive formulation of a problem usually assumes that the first cash flow occurs immediately (at t = 0) while the Gordon Growth formula assumes that the first cash flow is received one period from the present (at t = 1).

Solution Templates

The Rapid Recursive® Toolbox includes several examples of using dynamic programming for valuation and decision support. These examples are written in a template that publishes a report to HTML, Microsoft Word or PDF. These templates can be re-used to report the set up and solution to your own sequential decision problem.

The following Solution Templates have been included in the Rapid Recursive® Toolbox:

- ForestManagement
- MachineReplacement
- PropertyValuation
- RentalPropertyManagement
- AutoMarketShare
- BeverageWholesaler
- ClassicReinvestmentProblem
- FirmInvestment
- GradSchoolTuitionPricing
- AutoDriveOrSell
- HouseholdSaving
- JobSearchWithFiring
- BasicBlackSwan
- BasicSolutionLegacyTemplate
- HiringWithPolicyRisk
- OperateOrAbandon
- AutomotivePromotionStrategyModel
- BasicBusinessDecisionModelTemplate

The *Guide to Recursive Models* describes some of these Solution Template in depth.

You can access the examples in the Rapid Recursive® Toolbox from the **Main Menu GUI** (graphic user interface):

1. Type **RRmainmenu**in the MATLAB® command window and press enter. The Main Menu window will

open.

2. Click on **Run an Example**. Another window will open.
3. **Select an example** from the drop down list and then click **Open**. A MATLAB® file (.m file) will open in MATLAB®.
4. **Save** the file under a new name. DO NOT SAVE OVER THE EXISTING FILE.
5. Click on the **publish** button in the shortcuts menu in the MATLAB® editor. If you do not have this shortcut, click on **File** and then **Publish SolutionTemplate.m**, where SolutionTemplate is the

name of the file. This will publish the template to your default output file format, which is usually HTML.

A common error that occurs when attempting to publish is not having "write-permissions" in the directory where MATLAB® is attempting to publish a document. To solve this error on a Windows machine, close MATLAB®, re-open MATLAB® by right clicking and selecting "Run as administrator", then attempt to publish again.

For more information about how to write code in MATLAB® that will publish to HTML, PDF or Microsoft Word, see Publishing MATLAB® Code (this is a MATLAB® help document).

V. Troubleshooting

If the numeric answers from the value function iteration or policy iteration algorithms do not look right, you may need to tweak the algorithms. Assuming you have correctly set up your model, here are a few tips:

- Increase the maximum number of iterations. If the maximum number of iterations is being reached, convergence may not have occurred.
- Decrease epsilon (only applies to value function iteration). We only know that the estimated value function is within epsilon of the true value function, so make sure epsilon is small enough.
- Change ***policyevalmethod*** from 0 to 1 or vice versa (only applies to policy iteration). ***policyevalmethod*** determines the way a matrix is inverted in the "policy evaluation" step of the algorithm. The options provided in the Rapid Recursive® Toolbox are Gaussian elimination with partial pivot and the Jacobi method. Depending on the parameters of your sequential decision model, one of the matrix inversion methods can create small rounding errors. If you think this might be happening, change your ***policyevalmethod***.

VI. Function List

The following functions are included in the Rapid Recursive® Toolbox; the formal documentation for each function follows.

Tip: Function names with the "RR" prefix, e.g. RRvalueiteration, are considered "core" functions in the Rapid Recursive® Toolbox.

Composing

createlabels	A multipurpose label creator. createlabels will combine any number of groups of string labels via a Cartesian or Kronecker Product. createlabels will also convert numeric vectors to a group of string labels.
CreateBigState	The BigState matrix captures multi-dimensional state information in a fashion that allows it to be used in other calculations (such as rewards) or in data visualizations. It is not required for composing and solving an RR decision problem, but is often helpful.
GetColor	A function to return standard color values using "parula" scheme and custom-selected colors. These allow you to use color variables such as "blue" for a pleasing blue shade. You can also customize these colors to your preference.
newST	Creates a new solution template, with tips and hints to guide you through creating your own recursive model.
RRpoissoncdf	Calculates the value of the cumulative density function (CDF) for the poisson distribution with expected value lambda, at the point x. The third argument, lowbound, is optional. It enables the calculation of the probability for events between lowbound and x.
RRcleaninputstruct	Removes groups of fields that are empty.
RRcreateinputstruct	Creates a structure array for use as an input argument to RRvalueiterationor

	RRpolicyiteration.
RRcreateoutstruct	Creates a structure array that contains fields useful for recording output from RRvalueiterationor RRpolicyiteration.
RRcreatetransition	Creates one frame of a transition matrix informed by the specified probability distribution or process.
RRcombinerewards	Creates a reward matrix for problems involving multi-dimensional state spaces from a set of reward vectors, one for each state dimension.
RRcombinetransitions	Creates a transition matrix for a two-dimensional problem by combining two statistically independent transition matrices.
RRcomposetool	Opens an interactive graphical user interface that allows users to compose and solve a sequential decision problem without needing to program any code.
RRinvesttransition	Creates a transition matrix that can be used in a reinvestment problem.
RRrewardmatrix	Calculates a reward matrix (reward function) from a set of matrices containing the states, applied actions, and received rewards for a population of observations taken over time.
RRshortincomestatement	Calculates and displays an income statement for a company whose business is described by a small set of key variables.
RRspecialkron	An altered form of the kronecker tensor product acting on one 2D (A) and one 3D matrix (B). The result is a large matrix formed by taking all possible products between the elements in each column of A and the elements in each frame of B.
RRtimetransition	Creates one frame of a transition matrix for

	time periods, where time moves forward to the next period with certainty.
RRtransitionmatrix	Calculates a transition matrix from a set of matrices containing the states and applied actions for a population of observations taken over time.
convertdiscount	Converts the discount factor from an annual basis to the time index specified by the user.
ItoProject	Generates an Ito projection given a method, drift, sigma, initial state vector, and number of periods.
TrendProject	Generates a projection of revenues given a base revenue, growth rate, and number of periods for which to project.
ExtractStateDimension	Breaks down a multi-dimensional Input structure into subInput structures, each one corresponding to a different state dimension.

Error checking

RRcheckinputs	Checks the inputs for RRvalueiteration and RRpolicyiterationfor errors.

Solving

capitalizedvalue	Calculates Gordon Growth formula.
RRbackwardinduction	Solves finite time sequential decision problem using backward induction.
RRfindinvariantdist	Finds the invariant distribution of a Markov chain.
RRoptimalpolicymc	Finds a Markov chain that represents how a sequential decision problem transitions from state to state if the decision maker follows the prescribed policy.
RRpolicyiteration	Solves infinite time sequential decision

	problem using policy iteration.
RRvalueiteration	Solves infinite time sequential decision problem using value function iteration.
RRlikelypath	Projects forward a likely path assuming that the most likely random events occur and the subject person follows the optimal policy every time.

Reporting

generalchart	Creates and displays a customizable bar chart.
dispcurrency	Converts a number to a string formatted as a currency value, according to the conventions of the specified currency.
displist	Displays a list on the command window.
dispreward	Displays a reward matrix with labels for the states and actions.
dispwelcome	Displays the traditional Rapid Recursive welcome message.
graphdist	Graphs a probability mass function.
RRcompareoptima	This function compares the profit maximizing solution of a sequential decision problem to the value maximizing solution of the same problem.
RRfigure	Creates a new figure with the default background color for the Rapid Recursive® Toolbox and a note indicating that the figure was created by the Rapid Recursive® Toolbox.
RRgraphtransitions	Creates a directed graph representation of the transition matrix for a specific action, for a properly composed RR decision model that includes states, actions, and a transition matrix. Here, the states are the nodes, and the edges between the nodes

	represent probabilistic transitions among the states. Such a representation can be made for a selected action.
RRgraphlikelypath	Illustrates the likely path calculated for a solution of a Rapid Recursive® model, showing the likely state evolution, the optimal actions (policies), and expected rewards at each time period going forward.
RRresultsreport	Displays the results of a solved sequential decision problem in an interactive format.
RRsimulateresults	Performs a Monte Carlo simulation of a solved sequential decision problem following the optimal policy.
RRslicestates	Creates a subset of an Output structure's results table based on specific states.
RRsvplot	Creates and displays a bar chart of the value in each state for a sequential decision problem.
RRtable	Creates a formatted uitable with results from a sequential decision problem.
RRviewreward	Displays a ribbon chart visualization of the reward matrix, where each row in the map is a state, and each column an action. Color differences indicate differences in the magnitude of the reward.
RRviewtransition	Displays one frame of the transition matrix for a recursive problem as a heat map of the transition probabilities.

Utilities

installcheck	Checks installation was completed correctly.
superclear	Closes all windows, clears the command window and clears the MATLAB® memory.

ind2sub_mat	Convert linear index of matrix to multiple subscript indices. ind2sub_mat is used to determine the equivalent subscript values corresponding to a given single index into an array.
sub2ind_mat	Convert multiple subscript indices to linear index. Sub2ind_mat is used to determine the equivalent single index corresponding to a given set of subscript values.
RRclearworkspace	Clears the workspace according to conventions at the end of a Rapid Recursive® solution template, leaving only the variables that have names beginning with '***Input***' or '***Out***'.
RRnormcdf	Calculates the value of the cumulative density function (CDF) for the normal distribution with mean mu and standard deviation sigma at the value x.
RRver	Version information for the Rapid Recursive® toolbox.
RRguide	Accesses the Guide to Recursive Models, which is intended to help users understand the general power, use, and features of recursive models and provides instructive examples.
RRhelp	Accesses the Rapid Recursive Toolbox User's Guide, which serves as a reference for command syntax, usage, troubleshooting, licensing, and installation.

Licensing

RRdeactivate	Opens a window that allows you to deactivate your Product Key.

Tip: If you are unsure what a function does, its documentation can be displayed in MATLAB by calling "help function_name" in the command line. This works for all functions in MATLAB.

createlabels

A multipurpose label creator. createlabels will combine any number of groups of string labels via a Cartesian or Kronecker Product. createlabels will also convert numeric vectors to a group of string labels.

Syntax

labels = createlabels(labels)
labels = createlabels(labels, delim)
labels = createlabels(labels, delim, format)

Description

labels = createlabels(labels)
will combine each element of the first group of string labels stored in labels with each element of the following group of string labels. The output will be all combinations of hybrid labels combined via a Kronecker Product (All elements of the first group taken with the first element of the second group, then all elements of the first group taken with the second element of the second group, etc.).

labels = createlabels(labels, delim)
will insert the string stored in delim between each each label from each group.

labels = createlabels(labels, delim, format)
when format is specified as 'cart', the output will be all combinations of hybrid labels combined via a Cartesian Product (The first element of the first group taken with all elements of the second group, then the second element of the first group taken with all elements of the second group, etc.).

Input Arguments

labels	(Required) A cell array of groups of string labels (often each corresponding to a state dimension) or numeric vectors. Groups of string labels must be specified as a cell array of strings.
delim	(Optional) The delimiter to insert between string labels from different groups. Default delimiter is '-'. Delim must be entered as a string. Use '' to specify no delimiter.
format	(Optional) 'cart' to combine labels via a Cartesian Product or 'kron' to combine labels via a Kronecker Product. Default format is 'kron' N.B. 'kron' format is consistent with other RR toolbox functions for combining multiple state dimensions

Output Arguments

labels	A cell array of combined string labels.

Examples

labels = createlabels({{'North','South'},{'East','West'},[0,1],{'Go','Stop'}})

labels =

'North-East-0-Go'

'South-East-0-Go'

'North-West-0-Go'

'South-West-0-Go'

'North-East-1-Go'

'South-East-1-Go' 'North-

West-1-Go' 'South-West-1-

Go' 'North-East-0-Stop'

'South-East-0-Stop' 'North-

West-0-Stop' 'South-West-

0-Stop' 'North-East-1-Stop'

'South-East-1-Stop' 'North-

West-1-Stop' 'South-West-

1-Stop'

labels = createlabels({{'North','South','East','West'},{'Red','Green','Blue'}},'-
>','cart') labels:

'North->Red'

'South->Red'

'East->Red'

'West->Red' 'North-

>Green' 'South-

>Green' 'East-

>Green' 'West-

>Green' 'North-

>Blue' 'South-

>Blue' 'East->Blue'

'West->Blue'

labels = createlabels({{'Step'},[1:4]}) labels =

'Step-1'

'Step-2'

'Step-3'

'Step-4'

CreateBigState

The BigState matrix captures multi-dimensional state information in a fashion that allows it to be used in other calculations (such as rewards) or in data visualizations. It is not required for composing and solving an RR decision problem, but is often helpful.

Syntax

(BigState, TBigstate) = CreateBigState(Input)

Description

(BigState, TbigState) = CreateBigState(Input)
pulls together a set of up to four state dimensions in order to run calculations on up to four dimensional state situations.

Input Arguments

Input	An 'Input' structure with the following fields: • States('***S_***') • State values('***states'***) • Description of states('***S_note'***) • Calculation of total state spaces('***S***') • A description of combined states('***statelabels***')
Optional Inputs	A shortened description of the states('***S_short***').

Output Arguments

Output	An S x S BigState matrix with each row representing a state, and each column a different dimension of the state.

Examples

```
exInput.S1 = 2;
exInput.S2 = 3;

exInput.state1 = [2,3];
exInput.state2 = [2,3,4];

exInput.S1note = 'Dim_1'; exInput.S2note = 'Dim_2';

exInput.S = exInput.S1*exInput.S2;

exInput.statelabels = cell(1,exInput.S); for i = 1:exInput.S
    exInput.statelabels{i} = ['State',num2str(i)];
```

```
end
exCase.Input = exInput;

CreateBigState(exInput)


ans =


     2     2

     2     3

     2     4

     3     2

     3     3

     3     4
```

GetColor

A function to return standard color values using "parula" scheme and custom-selected colors. These allow you to use color variables such as "blue" for a pleasing blue shade. You can also customize these colors to your preference.

Syntax

C = GetColor

Description

C = GetColor returns the colors referenced in the output arguments. You can access colors as `C.blue`, `C.red`, to use them in another place.

Output Arguments

C	RGB codes for: • ***C.blue*** • ***C.red*** • ***C.yellow*** • ***C.purple*** • ***C.green*** • ***C.cyan*** • ***C.maroon*** • ***C.beige*** • ***C.lightblue*** • ***C.SIblue*** • ***C.burgundy*** • ***C.lightred*** • ***C.Dodgerblue*** • ***C.cream*** • ***C.khaki*** • ***C.olivegreen*** • ***C.gold*** • ***C.champagne*** • ***C.papayawhip*** • ***C.lightgreen*** • ***C.contents*** • ***C.darkgreen*** • ***C.darkblue*** • ***C.sand*** • ***C.deepburgundy*** • ***C.lightgray*** • ***C.gray*** • ***C.parulablue*** • ***C.teal*** • ***C.middleblue*** • ***C.resolutionblue***

	• *C.catalinablue* • *C.firebrick* • *C.hotpink* • *C.carmine* • *C.cardinal* • *C.UAWred* • *C.darkgreen* • *C.limegreen* • *C.seagreen* • *C.UAWgreen* • *C.fern* • *C.brown* • *C.sienna* • *C.goldenrod* • *C.sandybrown* • *C.lightsalmon* • *C.wheat* • *C.mustard* • *C.orange* • *C.selectiveyellow* • *C.UAWyellow* • *C.silvergray* • *C.darkgray* • *C.slategray* • *C.cardinal* • *C.selectiveyel* • *C.GMblue*

Examples

C = GetColor has some of the following colors set:

```
C.blue        = [0.0000, 0.4471, 0.7412];      % MATLAB default blue
C.red         = [0.8500, 0.3250, 0.0980];      % MATLAB default red
C.yellow      = [0.9290, 0.6940, 0.1250];      % MATLAB default yellow
C.purple      = [0.4940, 0.1840, 0.5560];      % MATLAB default purple
C.green       = [0.4660, 0.6740, 0.1880];      % MATLAB default green
C.cyan        = [0.3010, 0.7450, 0.9330];      % MATLAB default cyan
C.maroon      = [0.6353, 0.0784, 0.1843];      % Custom maroon
C.beige       = [245, 245, 220]/255;           % Beige
C.lightblue   = [0.678, 0.847, 0.902];         % Light blue
C.SIblue      = [0.2734, 0.3516, 0.4688];      % Supported Intelligence blue
C.burgundy    = [0.6353, 0.2824, 0.3412];      % Burgundy
C.lightred    = [230, 159, 159]/256;           % Light red
C.Dodgerblue  = [30, 144, 255]/256;            % Dodger blue
```

newST

Creates a new solution template, with tips and hints to guide you through creating your own recursive model.

Syntax

newST('filename')

Description

newST('filename')

Creates a new .m file in the current folder, with the name specified by filename, that contains an outline for a new Rapid Recursive® solution template.

Input Argument

filename	A string specifying the desired name for the new solution template.

Example

newST('Test')opens an outline for a new solution template and saves the file as Test.m in the current folder.

RRpoissoncdf

Calculates the value of the cumulative density function (CDF) for the poisson distribution with expected value lambda, at the point x. The third argument, lowbound, is optional. It enables the calculation of the probability for events between lowbound and x.

Syntax

P = RRpoissoncdf(x, lambda, lowbound)

Description

P = RRpoissoncdf(x, lambda, lowbound)
calculates the cumulative distribution given an ***x*** and expected value ***lambda***. The user has the option of also manually entering a positive, non-zero, integer as the third input argument. If this is not entered, the function defaults to 0 for the lower bound.

Input Arguments

x	***x*** is the point at which the CDF will be evaluated. The variable x is expected to be an integer.
lambda	***lambda*** is the expected value for the applicable Poisson distribution. Must be greater than 0.
lowbound	***lowbound*** is the lower limit for cumulative distribution. This enables the determination of the probability that the value falls within the band ***lowbound***< value<=***x***. This cannot be less than 0 and is expected to be an integer. Defaults to 0.

Output Arguments

P	***P***: Value of the CDF of the Poisson distribution at ***x***, given ***lambda***.

Examples

P = RRpoissoncdf(1, 1) P =

.7358

P = RRpoissoncdf(1, 15, 1) P =

0

RRcleaninputstruct

Removes groups of fields that are empty.

Syntax

Input2 = RRcleaninputstruct(Input1)

Description

Input2 = RRcleaninputstruct(Input1)
removes fields from the structure array ***Input1*** when all fields within a group of optional fields are empty.

Input Argument

Input1	A structure array, usually with fields such as ***P***, ***R***, ***beta***.

Output Argument

Input2	A structure array based on Input1 with removed groups of empty fields.

Example

The following example uses RRcreateinputstructto create a structure array Input1 with default fields. Then RRcleaninputstruct is used to remove groups of optional fields that are empty.

```
Input1 = RRcreateinputstruct

Input2 = RRcleaninputstruct(Input1)
```

See Also

RRcreateinputstruct

RRcreateinputstruct

Creates a structure array for use as an input argument to RRvalueiterationor RRpolicyiteration.

Syntax

Input = RRcreateinputstruct('scheme')

Description

Input = RRcreateinputstruct('scheme')
creates a structure array ***Input*** for use as an input argument to RRvalueiterationor RRpolicyiteration. ***Input*** contains all fields that are required by RRvalueiterationor RRpolicyiterationas well as additional fields that may be useful. Even more fields are added depending on the value of 'scheme'. All field values are empty, except ***date*** and ***desc***. The empty fields can be filled in by the user before use with RRvalueiterationor RRpolicyiteration.

Input Argument

'scheme'	A string that indicates the type of fields to include in Input. Selecting 'scheme' as 'e' creates a structure array with default fields, and for backwards compatibility with older versions, use scheme 'd' Note : Beginning in version 1.5.0, scheme 'd' produces a structure with default fields. Schemes 'a', 'b', and 'c' are still supported for backwards compatibility, but 'd' is now the recommended input argument.

Output Argument

The output argument is ***Input***, which is a structure array containing the following fields. All field values are empty, except ***date*** and ***desc***.

Field	Description
desc	Title of the problem (a string). The default value of this field is 'Untitled sequential decision problem'.
stateinfo	A string or cell array containing relevant information about the state space for a particular problem.
S	The number of states.
states	A 1 x S vector of state values.
statelabels	1 x S cell array containing labels for the states in the sequential decision problem, where S is the number of states in the sequential

	decision problem. Each cell statelabels{j}should contain a string that represents the name of the j-th state in the sequential decision problem.
S1	The number of states in the first state dimension.
S1labels	A 1 x S1 cell array containing labels for the states along the first dimension of the state space for multidimensional problems.
S1note	A string or cell array containing relevant information about the first dimension of the state space.
S2	The number of states in the second state dimension.
S2labels	A 1 x S2 cell array containing labels for the states along the second dimension of the state space for multidimensional problems.
S2note	A string or cell array containing relevant information about the second dimension of the state space.
actioninfo	A string or cell array containing relevant information about the action set for a particular problem.
A	The number of actions.
actions	A 1 x A vector of action values.
actionlabels	1 x A cell array containing labels for the actions in the sequential decision problem, where A is the number of states in the sequential decision problem. Each cell actionlabels{j}should contain a string that represents the name of the j-th action in the sequential decision problem.
actioncosts	A 1 x A vector of the costs associated with taking each possible action.
transitioninfo	A string or cell array containing relevant information about the transition probabilities for a particular problem.
P	(Required) Transition matrix. For a detailed description on how to create a transition matrix see the documentation for RRvalueiterationor RRpolicyiteration.
P1	The S1 x S1 x A transition matrix for states in the first state dimension.

P2	The S2xS2xA transition matrix for states in the second state dimension.
rewardinfo	A string or cell array containing relevant information about the rewards for a particular problem.
R	(Required) Reward matrix. For a detailed description on how to create a reward matrix see the documentation for RRvalueiteration or RRpolicyiteration.
beta	A discount factor that must be strictly greater than 0 and no more than 1.
d	Discount rate.
g	Growth rate.
epsilon	(Used in value function iteration only). ***epsilon*** is the threshold for the maximum difference between the value function found by this algorithm and the true value function.
policy0	(Used in policy iteration only). ***policy0*** (S x 1) is the initial value of the policy that is iterated on during policy iteration.
maxiter	The maximum number of iterations that can occur before the solution algorithm stops.
V0	(Used in value function iteration only). ***V0*** is an S x 1 vector that serves as the starting point for value function iteration.
policyevalmethod	(Used in policy iteration only). Entering ***policyevalmethod*** = 0 specifies that the "policy evaluation" step in the policy iteration algorithm is performed using Gaussian elimination with partial pivoting. Entering ***policyevalmethod*** = 1 specifies that the "policy evaluation" step is performed using the Jacobi method. By default, RRpolicyiteration uses the Jacobi method.
verbose	Whether output from each iteration is displayed to the command window. Setting ***verbose*** to false will increase the speed of the algorithm.
periodicity	Frequency of periods, e.g. yearly, monthly, weekly.
T	Number of periods (can be 'inf').

Tstates	A table listing all of the states for the current problem.
Tactions	A table listing all of the actions for the current problem.
Tactioncosts	A table listing all of the action costs for the current problem.
Treward	A table listing the rewards for the current problem.
Tparams	A table listing the key parameters for the current problem.
Tgd	A table displaying the discount factor and discount and growth rates for the current problem.
Ts1	A table listing the states along the first state dimension of the current problem.
Ts2	A table listing the states along the second state dimension of the current problem.
Table1	Miscellaneous table.
Table2	Miscellaneous table.
Table3	Miscellaneous table.
note1	A string containing additional information about the model.
note2	Another string containing additional information about the model.
parameter1	Miscellaneous parameter.
parameter2	Miscellaneous parameter.
parameter3	Miscellaneous parameter.
note1	Miscellaneous note.
note2	Miscellaneous note.
date	Time and date. This field is usually filled in automatically.

Tip

RRcreateinputstruct('e')creates a structure array with default fields.

Example

```
% Create a structure array with default fields Input =
RRcreateinputstruct('e')

% Fill in required field values before using it in RRvalueiteration P(:,:,1) = [0.6 0.4; 0.6 0.4];
P(:,:,2) = [0.5 0.5; 0.5 0.5];
Input.P = P;
Input.R = [20 30; 35 25];
Input.beta = 0.9;

% Run value function iteration using Input Out =
RRvalueiteration(Input);
```

See Also

RRcleaninputstruct | RRvalueiteration | RRpolicyiteration

RRcreateoutstruct

Creates a structure array that contains fields useful for recording output from RRvalueiteration, RRpolicyiteration, or RRbackwardinduction.

Syntax

Out = RRcreateoutstruct('scheme')

Description

Out = RRcreateoutstruct('scheme')
creates a structure array ***Out*** that contains fields for ***desc***, ***Input***, ***V***, ***policy***, ***resultstable***, ***iterations***, ***calculationtime***, ***algorithm***, ***note1***, ***note2*** and ***date***. All field values are empty, except ***desc*** and ***date***. ***date*** contains the date and time of the creation of ***Out***. The empty fields of ***Out*** can be filled in by the user with output from a run of ***RRvalueiteration*** or ***RRpolicyiteration***, although this is generally not necessary as the first output argument of both ***RRvalueiteration*** and ***RRpolicyiteration*** is a structure array that contains values for ***V***, ***policy***, ***resultstable***, ***iterations*** etc.

Input Argument

'scheme'	Selecting 'scheme' as 'a' creates a structure array with default fields. Currently, 'a' is the only supported scheme.

Output Argument

Out	A structure array containing the following fields (all fields are empty except ***desc*** and ***date***) for all the output arguments described in this table as well as: • ***desc***: the title of the model (a string) • ***Input***: Input structure • ***V***: Value function • ***policy***: optimal policy • ***resultstable***: a table of the results (a cell array) • ***iterations***: number of iterations • ***calculationtime***: computation time in seconds • ***algorithm***: the type of algorithm used to find the solution (a string) • ***note1***: empty. Can be filled in later with notes. • ***note2***: empty. Can be filled in later with notes. • ***date***: the time and date the value function iteration was run (a date string).

Example

```
% Create a structure array with default output fields Out =
RRcreateoutstruct('a')

Out =

                desc: 'default RR output structure' Input: []
             Results: '---------- '
                   V: []
              policy: []
         resultstable: []
               Other: ' ---------- '
          iterations: []
     calculationtime: []
           algorithm: []
               Notes: ' ---------- '
               note1: []
               note2: []
                date: '24-Sep-2014 16:42:38'
```

See Also

RRcreateinputstruct | RRvalueiteration | RRpolicyiteration

RRcreatetransition

Creates one frame of a transition matrix informed by the specified probability distribution or process.

Syntax

function P = RRcreatetransition(states,dist,varargin)

Description

P = RRcreatetransition(states, dist)
creates one frame of the transition matrix with the distribution specified by distwith default values for mu (0), sigma (1), rho (0.9) in the state space specified by statesusing midpoint binning.

P = RRcreatetransition(states, dist, 'mu', mu)
sets the mean of the distribution specified by distto the value stored in mu.

P = RRcreatetransition(states, dist, 'sigma', sigma)
sets the standard deviation of the distribution specified by distto the value stored in sigma.

P = RRcreatetransition(states, dist, 'rho', rho)
sets the auto-regressive parameter to the value stored in rho.

P = RRcreatetransition(states, dist, 'bin', bin)
sets the binning method to the string stored in bin.

Input Arguments

<table>
<tr><td>states</td><td>An integer indicating the number of states in the problem, a vector containing the specific state values, or a cell containing state labels (strings) for numeric states (e.g. {'10' '20' '30'}). The states MUST be listed in increasing order.

If states is entered as a vector, three cumulative distribution binning options are available, the default being 'lower' (see below). If states is entered as a scalar, the transition matrix will be created such that the distribution is centered around the middle one or two states.</td></tr>
<tr><td>dist</td><td>The desired probability distribution or process which will inform the creation of the transition frame.

Available Options and Available Parameters

<table>
<tr><th>dist</th><th>Distribution or Process</th><th>Available Parameters</th></tr>
<tr><td>'ar1'</td><td>First Order Autoregressive</td><td>mu,sigma,bin,rho</td></tr>
<tr><td>'Normal','n'</td><td>Normal Distribution</td><td>mu,sigma,bin</td></tr>
<tr><td>'Brownian','b','Brownian Motion'</td><td>Brownian Motion</td><td>sigma,bin</td></tr>
<tr><td>'Uniform','u'</td><td>Uniform Distribution</td><td>bin</td></tr>
<tr><td>'Poisson', 'p'</td><td>Poisson Distribution</td><td>mu, bin</td></tr>
</table>
</td></tr>
</table>

Optional Parameters

mu	The mean of the distribution. The default value is 0.
sigma	The standard deviation of the distribution. The default value is 1.
rho	The autoregressive parameter. The default value is 0.9.

bin	The state value binning method for the cumulative distribution function. Use only if states is entered as a vector. The default value is 'midpoint'.
	
	Available options:
	
	• 'lower'
	  ○ Each state value bin contains the portion of the distribution between the current state value and the next lowest state value. The lowest state value bin contains the portion between the lowest state value bin and –Infinity (or, in the Poisson case, 0). The highest state value bin contains the portion between the second highest state value and +Infinity.
	• 'upper'
	  ○ Each state value bin contains the portion of the distribution between the current state value and the next highest state value. The lowest state value bin contains the portion between the second lowest state value and -Infinity (or, in the Poisson case, 0). The highest state value bin contains the portion between the highest state value and +Infinity.
	• 'midpoint'
	  ○ Each bin contains the portion of the distribution between the midpoint of the current state value and the next lowest state value to the midpoint between the current value and the next highest state value. If there is no lower or higher state value, then the lower/upper bound is set to -/+ infinity. (For the Poisson distribution, the lower bound is zero, not negative infinity.)

Output Arguments

P	One frame of a transition matrix.

Examples

```
rho      =    0.8;
mu       =    5;
sigma    =    2;
bin      =    'lower';
states   =    0:2:10;
```

```
P = RRcreatetransition(states,'ar1',... 'rho',rho,'mu',mu,'sigma',sigma,'bin',bin)

P =

    0.0062    0.0606    0.2417    0.3829    0.2417    0.0668
    0.0005    0.0102    0.0861    0.2853    0.3759    0.2420
    0.0000    0.0009    0.0169    0.1178    0.3245    0.5398
    0.0000    0.0000    0.0018    0.0269    0.1553    0.8159
    0.0000    0.0000    0.0001    0.0034    0.0411    0.9554
    0.0000    0.0000    0.0000    0.0002    0.0060    0.9938

P = RRcreatetransition(states,'n','mu',mu,'sigma',sigma)

 P =

    0.0228    0.1359    0.3413    0.3413    0.1359    0.0228
    0.0228    0.1359    0.3413    0.3413    0.1359    0.0228
    0.0228    0.1359    0.3413    0.3413    0.1359    0.0228
    0.0228    0.1359    0.3413    0.3413    0.1359    0.0228
    0.0228    0.1359    0.3413    0.3413    0.1359    0.0228
    0.0228    0.1359    0.3413    0.3413    0.1359    0.0228
P = RRcreatetransition(states,'b','bin',bin)

 P =

    0.5000    0.4772    0.0227    0.0000    0.0000    0.0000
    0.0228    0.4772    0.4772    0.0227    0.0000    0.0000
    0.0000    0.0227    0.4772    0.4772    0.0227    0.0000
    0.0000    0.0000    0.0227    0.4772    0.4772    0.0228
    0.0000    0.0000    0.0000    0.0227    0.4772    0.5000
    0.0000    0.0000    0.0000    0.0000    0.0227    0.9772

states = 4;

P = RRcreatetransition(states,'u')

P =

    0.2500    0.2500    0.2500    0.2500
    0.2500    0.2500    0.2500    0.2500
    0.2500    0.2500    0.2500    0.2500
    0.2500    0.2500    0.2500    0.2500
```

RRcombinerewards

Creates a reward matrix for problems involving multi-dimensional state spaces from a set of reward vectors, one for each state dimension.

A multi-dimensional state space occurs when two or more independent sets of market conditions interact, with the combination of these conditions affecting the subject. Examples of this include companies that are affected by both demand for their product and price for a raw material; a person seeking a job in which both the person's credentials and their location affect the likelihood of a job offer; a startup company or intellectual property affected by the state of technology, the ability to finance R&D expenditures, and current market conditions; as well as many others. The Rapid Recursive® Toolbox allows users to model these situations by explicitly describing the state and the reward for every combination of the two or more dimensions. For example, a company affected by three different market conditions and two different conditions summarizing their intellectual property faces 3x2=6 different states in each time period.

Syntax

R = RRcombinerewards(rewards, ActionCosts)

R = RRcombinerewards(rewards, ActionCosts, 'context', context) R =

RRcombinerewards(rewards, ActionCosts, @reward_function)

Description

RRcombinerewardscreates the reward matrix for a sequential decision problem where the state space contains two or more dimensions. The matrix is constructed using a reward function that utilizes inputs from each dimension of the state. The reward function can be one of three default functions included with this command, or the user can supply a custom function. This command returns a reward matrix, which is an S x A matrix, where S, the number of states, is equal to numel(rewards{1}) * numel(rewards{2})*...* numel(rewards{N}) where N is the number of state dimensions, and A, the number of actions, is equal to numel(ActionCosts). The states in the reward matrix include all possible combinations of all state dimensions. The order of the states in the combined state space follow the order of a Kronecker Product of the state dimensions.

R = RRcombinerewards(rewards, ActionCosts)
creates a reward matrix for the full state space from the reward vectors in rewards by multiplying the reward vector values from each dimension and subtracting the action cost.

R = RRcombinerewards(rewards, ActionCosts, 'context', context)
calculates reward values using the method specified by the string stored in context.

R = RRcombinerewards(rewards, ActionCosts, @reward_function)
calculates reward values using a custom reward function specified by @reward_function.

Input Validation

Before attempting to create the reward matrix, RRcombinerewardsvalidates its inputs. If an error is found, RRcombinerewardsis stopped and messages identifying the error(s) are displayed in the command window.

Input Arguments

Rewards	(Required) A cell array of N reward vectors. Each reward vector must be size 1 x S_n where S_n is the size of the Nth state dimension. Each element of reward vector contains the reward parameter associated with being in that subspace of the given state dimension.
ActionCosts	(Required) A 1 x A vector of parameters associated with taking each action

Context	(Optional) A string specifying a built-it reward function.
	
	Available options:
	• 'multiply' (default)
	  ○ R = R1 * R2 * … * RN – actioncost
	• 'add'
	  ○ R = R1 + R2 + … + RN – actioncost
	• 'propCost'
	  ○ R = R1 * R2 – (action * R2)

@reward_function	(Optional) An anonymous reward function with an input for each state dimension and one input for the action. The action is assumed to be the final input.

Output Arguments

R	An S x A reward matrix with a reward value calculated for each multi-dimensional state and action according to the specified reward function.

Two–Dimensional Examples

Example 1:

Assume a store owner is trying to determine the reward for various price-inventory combinations and possible advertising campaign costs. Assume the price could be 5, 10, or 20 per unit, and inventory is drawn from {20, 50, 75}. Further, assume that the owner is debating between three levels of advertising:

none, minimal, and maximal, with corresponding costs of 0, 100, and 250. We could construct the reward matrix as follows:

```
price=[5 10 20];
inventory=[20 50 75];
advertisingcost=[0 100 250];

R = RRcombinerewards({price,inventory},advertisingcost)

R =
          100.00            0         -150.00
          250.00          150.00            0
          375.00          275.00          125.00
          200.00          100.00          -50.00
          500.00          400.00          250.00
          750.00          650.00          500.00
          400.00          300.00          150.00
         1000.00          900.00          750.00
         1500.00         1400.00         1250.00
```

Example 2:

```
R = RRcombinerewards({price, inventory}, advertisingcost, 'context','add') R =

    25    -75   -225
    55    -45   -195
    80    -20   -170
    30    -70   -220
    60    -40   -190
    85    -15   -165
    40    -60   -210
    70    -30   -180
    95     -5   -155
```

Example 3:

Suppose we wished to implement a custom function for calculating the reward. In this case, perhaps we want to multiply the two reward elements and divide by the action cost. We could achieve this by writing the following function and saving it as newReward.m:

```
function r = newReward(d1, d2, act) r = (d1 *
d2)/act;
end
```

Then we could create the reward matrix as follows:

```
R = RRcombinerewards({price, inventory}, advertisingcost, @newReward) R =
```

Inf	1.0000	0.4000
Inf	2.5000	1.0000
Inf	3.7500	1.5000
Inf	2.0000	0.8000
Inf	5.0000	2.0000
Inf	7.5000	3.0000
Inf	4.0000	1.6000
Inf	10.0000	4.0000
Inf	15.0000	6.0000

Tips

The inclusion of two different dimensions in the state powerfully extends the analytical possibilities of a Rapid Recursive model. It is often possible to dramatically improve the analytical model by using just 2 to 3 elements in each dimension of the state vector. As noted above, even 3 elements in one dimension and 2 in the other allow for 6 different state combinations every time period. Across a handful of time periods and with even a small number of possible actions each time period, this results in hundreds or thousands of possible paths that are considered within the recursive model.

The use of a custom-programmed reward function also allows for very sophisticated models that take into account the particular mechanics of specific subject companies, persons, and investment opportunities. However, many situations can be modelled quite well using one of the supplied reward functions.

In situations where an absorbing state also exists, such as when a real option can be exercised once and only once, an additional state can be included in the R matrix after the two dimensions of the state and the relevant actions are used to calculate it. For example, if there are 3 elements in both dimension 1 and dimension 2, and length(ActionCosts)=4, then an absorbing state with rewards of zero can be added as follows:

```
R(10,:)=zeros(1,4);
```

Or, more generally:

```
ShortS=length(D1rewards)*length(D2rewards); R(shortS+1,:)=zeros(1,length(ActionCosts));
```

Multi-Dimensional Example:

Example 4:

Suppose that, in addition to price and inventory (like Example 1), the reward to the store owner is also proportional to the exchange rate. We capture the effect of the exchange rate on the reward in "exchangerate".

```
exchangerate = [0.9,1,1.1];
```

RRcombinerewards will create rewards for all combinations of all three state dimensions when the following syntax is used:

```
R = RRcombinerewards({price,inventory,exchangerate},advertisingcost)
```

R =

90.00	-10.00	-160.00
180.00	80.00	-70.00
360.00	260.00	110.00
225.00	125.00	-25.00
450.00	350.00	200.00
900.00	800.00	650.00
337.50	237.50	87.50
675.00	575.00	425.00
1350.00	1250.00	1100.00
100.00	0	-150.00
200.00	100.00	-50.00
400.00	300.00	150.00
250.00	150.00	0
500.00	400.00	250.00
1000.00	900.00	750.00
375.00	275.00	125.00
750.00	650.00	500.00
1500.00	1400.00	1250.00
110.00	10.00	-140.00
220.00	120.00	-30.00
440.00	340.00	190.00
275.00	175.00	25.00
550.00	450.00	300.00
1100.00	1000.00	850.00
412.50	312.50	162.50
825.00	725.00	575.00
1650.00	1550.00	1400.00

See Also

RRcombinetransitions

RRcombinetransitions

Creates a transition matrix for a multi-dimensional problem by combining transition matrices from statistically independent state dimensions.

Syntax

P = RRcombinetransitions(P1, P2)

P = RRcombinetransitions(P1, P2, ..., PN)

Two-Dimensional Problems

P = RRcombinetransitions(P1, P2)

creates ***P***, a transition matrix for a two-dimensional problem, by combining two statistically independent transition matrices, ***P1*** and ***P2***, where ***P1*** (respectively, ***P2***) is a transition matrix representing how state variable 1 (state variable 2) evolves depending on the actions taken. The number of actions (frames) represented by ***P1*** must be the same as the number of actions represented by ***P2***.

The states in ***P*** are determined as follows. If the state space represented by ***P1*** is $\{xx_1, xx_2, \ldots, xx_{SS_1}\}$(i.e. there are SS_1 states in ***P1***), and the state space represented by ***P2*** is $\{yy_1, yy_2, \ldots, yy_{SS_2}\}$ (i.e. there are SS_2 states in ***P2***) then the state space represented by ***P*** contains all combinations of states from ***P1*** and ***P2*** in the following order: $\{(xx_1, yy_1), (xx_1, yy_2), \ldots, (xx_1, yy_{SS_2}), \ldots, (xx_2, yy_1), (xx_2, yy_2), \ldots, (xx_2, yy_{SS_2}), \ldots, (xx_{SS_1}, yy_{SS_2})\}$.

The (i,j,k) element of ***P*** is the probability that the decision maker moves from the i-th state at time t to the j-th state in time t+1 when the k-th action is taken by the decision maker at time t. For example, if $SS_1 \geq 2$ and $SS_2 = 3$, the (3,4,2) element of ***P*** represents the probability of moving from the third state (xx_1, yy_3) at time t to the fourth state (xx_2, yy_1) at time t+1 when the decision maker takes the second action at time t. An easy way to remember this is to remember that the rows of ***P*** represent the state at time t, the columns represent the state at time t+1, and the third dimension of ***P*** represents the actions. This is the standard format for transition matrices in the Rapid Recursive® Toolbox.

N-Dimensional Problems

P = RRcombinetransitions(P1,P2,...,PN)

creates P, a transition matrix for a N-dimensional problem, by recursively combining a set of statistically independent transition matrices. RRcombinetranstitions will combine the first two transition matrices using the method above, and then combine the third transition matrix with the resulting matrix. RRcombinetransitions repeats this process until all statistically independent transition matrices have been combined.

Note:

RRcombinetransitions(P1, P2, P3) = RRcombinetransitions((RRcombinetransitions(P1, P2), P3)

Inputs

P1	(Required) ***P1***, called a transition matrix, is the matrix representation of the state transition function. ***P1*** must be statistically independent from ***P2***. ***P1*** must be a S_1 x S_1 x A matrix. The (i,j,k) element of ***P1*** is the probability that the decision maker moves from the i-th state at time t to the j-th state in time t+1 when the k-th action is taken by the decision maker at time t. N.B. ***P1*** must have the same number of actions (frames) as ***P2***.
P2	(Required) ***P2*** must be a S_2 x S_2 x A matrix, entered in a similar fashion to ***P1***. Note that ***P1*** must have the same number of actions (frames) as ***P2***, and ***P1*** and ***P2*** must be statistically independent.
PN	(Required) ***PN*** must be a S_n x S_n x A matrix, entered in a similar fashion to ***P1***. Note that ***PN*** must have the same number of actions (frames) as ***P1***, and all transition matrices ***P1, P2, …. ,PN*** must be statistically independent.

Output

P	***P*** is a [(S_1xS_2x…xS_n) x (S_1xS_2x…xS_n) x A] transition matrix for a N-dimensional problem created by combining N statistically independent transition matrices ***P1, P2, …. ,PN*** The states in ***P*** are determined as follows. If the state space represented by ***P1*** is $\{xx_1, xx_2, \ldots, xx_{SS_1}\}$ (i.e. there are SS_1 states in ***P1***), and the state space represented by ***P2*** is $\{yy_1, yy_2, \ldots, yy_{SS_2}\}$ (i.e. there are SS_2 states in ***P2***) then the state space represented by ***P*** contains all combinations of states from ***P1*** and ***P2*** in the following order: $\{(xx_1, yy_1), (xx_1, yy_2), \ldots, (xx_1, yy_{SS_2}), \ldots, (xx_2, yy_1), (xx_2, yy_2), \ldots, (xx_2, yy_{SS_2}), \ldots, (xx_{SS_1}, yy_{SS_2})\}$. The (i,j,k) element of ***P*** is the probability that the decision maker moves from the i-th state at time t to the j-th state in time t+1 when the k-th action is taken by the decision maker at time t. RRcombinetranstitions will combine the first two transition matrices using the method above, and then combine the third transition matrix with the resulting matrix. RRcombinetransitions repeats this process until all statistically independent transition matrices have been combined.

Example

```
% P1 is 2x2x2 P1(:,:,1)=  [0.5
```

```
0.5;
```

```
                0.5       0.5];

P1(:,:,2)= [0.5           0.5;
            0.5           0.5];

% P2 is 3x3x2

P2(:,:,1)=  [0.9          0.1          0;
             0.8          0.1          0.1;
             0.7          0.2          0.1];

P2(:,:,2)= [0.8           0.1          0.1;
            0.1           0.8          0.1;
            0.1           0.1          0.8];

P = RRcombinetransitions(P1,P2) % Creates a 6x6x2 matrix
```

Displays to the MATLAB command window:

P(:,:,1) =

0.4500	0.0500	0	0.4500	0.0500	0
0.4000	0.0500	0.0500	0.4000	0.0500	0.0500
0.3500	0.1000	0.0500	0.3500	0.1000	0.0500
0.4500	0.0500	0	0.4500	0.0500	0
0.4000	0.0500	0.0500	0.4000	0.0500	0.0500
0.3500	0.1000	0.0500	0.3500	0.1000	0.0500

P(:,:,2) =

0.4000	0.0500	0.0500	0.4000	0.0500	0.0500
0.0500	0.4000	0.0500	0.0500	0.4000	0.0500
0.0500	0.0500	0.4000	0.0500	0.0500	0.4000
0.4000	0.0500	0.0500	0.4000	0.0500	0.0500
0.0500	0.4000	0.0500	0.0500	0.4000	0.0500
0.0500	0.0500	0.4000	0.0500	0.0500	0.4000

```
% P3 is 2x2x2
```

P3 = repmat(RRcreatetransition(2,'b'),1,1,2) P3(:,:,1) =

0.9744	0.0256
0.0256	0.9744

P3(:,:,2) =

0.9744 0.0256
0.0256 0.9744

P = RRcombinetransitions(P1,P2,P3) % Creates a 12x12x2 matrix P(:,:,1) =

Columns 1 through 6

0.4385	0.0115	0.0487	0.0013	0	0
0.0115	0.4385	0.0013	0.0487	0	0
0.3898	0.0102	0.0487	0.0013	0.0487	0.0013
0.0102	0.3898	0.0013	0.0487	0.0013	0.0487
0.3410	0.0090	0.0974	0.0026	0.0487	0.0013
0.0090	0.3410	0.0026	0.0974	0.0013	0.0487
0.4385	0.0115	0.0487	0.0013	0	0
0.0115	0.4385	0.0013	0.0487	0	0
0.3898	0.0102	0.0487	0.0013	0.0487	0.0013
0.0102	0.3898	0.0013	0.0487	0.0013	0.0487
0.3410	0.0090	0.0974	0.0026	0.0487	0.0013
0.0090	0.3410	0.0026	0.0974	0.0013	0.0487

Columns 7 through 12

0.4385	0.0115	0.0487	0.0013	0	0
0.0115	0.4385	0.0013	0.0487	0	0
0.3898	0.0102	0.0487	0.0013	0.0487	0.0013
0.0102	0.3898	0.0013	0.0487	0.0013	0.0487
0.3410	0.0090	0.0974	0.0026	0.0487	0.0013
0.0090	0.3410	0.0026	0.0974	0.0013	0.0487
0.4385	0.0115	0.0487	0.0013	0	0
0.0115	0.4385	0.0013	0.0487	0	0
0.3898	0.0102	0.0487	0.0013	0.0487	0.0013
0.0102	0.3898	0.0013	0.0487	0.0013	0.0487
0.3410	0.0090	0.0974	0.0026	0.0487	0.0013
0.0090	0.3410	0.0026	0.0974	0.0013	0.0487

P(:,:,2) =

Columns 1 through 6

0.3898	0.0102	0.0487	0.0013	0.0487	0.0013
0.0102	0.3898	0.0013	0.0487	0.0013	0.0487
0.0487	0.0013	0.3898	0.0102	0.0487	0.0013
0.0013	0.0487	0.0102	0.3898	0.0013	0.0487
0.0487	0.0013	0.0487	0.0013	0.3898	0.0102
0.0013	0.0487	0.0013	0.0487	0.0102	0.3898

```
    0.3898    0.0102    0.0487    0.0013    0.0487    0.0013
    0.0102    0.3898    0.0013    0.0487    0.0013    0.0487
    0.0487    0.0013    0.3898    0.0102    0.0487    0.0013
    0.0013    0.0487    0.0102    0.3898    0.0013    0.0487
    0.0487    0.0013    0.0487    0.0013    0.3898    0.0102
    0.0013    0.0487    0.0013    0.0487    0.0102    0.3898

  Columns 7 through 12

    0.3898    0.0102    0.0487    0.0013    0.0487    0.0013
    0.0102    0.3898    0.0013    0.0487    0.0013    0.0487
    0.0487    0.0013    0.3898    0.0102    0.0487    0.0013
    0.0013    0.0487    0.0102    0.3898    0.0013    0.0487
    0.0487    0.0013    0.0487    0.0013    0.3898    0.0102
    0.0013    0.0487    0.0013    0.0487    0.0102    0.3898
    0.3898    0.0102    0.0487    0.0013    0.0487    0.0013
    0.0102    0.3898    0.0013    0.0487    0.0013    0.0487
    0.0487    0.0013    0.3898    0.0102    0.0487    0.0013
    0.0013    0.0487    0.0102    0.3898    0.0013    0.0487
    0.0487    0.0013    0.0487    0.0013    0.3898    0.0102
    0.0013    0.0487    0.0013    0.0487    0.0102    0.3898
```

See Also

RRcombinerewards

RRcheckinputs

Checks the decision problem summarized in an Input structure for:

- conformance of the state vector, action vector, R matrix, and P matrix;
- conditions for convergence, given a specific solution algorithm selection; and
- required fields in the Input structure.

Syntax

[...] = RRcheckinputs(algtype, Input)

[...] = RRcheckinputs(algtype, P, R, beta, epsilon, maxiter, V0, verbose)
(syntax only valid when ***algtype*** = 'VFI')

[...] = RRcheckinputs(algtype, P, R, beta, policy0, maxiter, policyevalmethod, verbose) (syntax only valid when ***algtype*** = 'PI')

[isError optionalInputsErrorFlag errorMessageBank] = RRcheckinputs(...)

Description

In addition to output described below, any run of RRcheckinputswill display messages on the command window. These messages will be a combination of positive messages that state all checks on a certain input have been passed and error messages that describe an error with an input and advice on how to fix that error.

isError = RRcheckinputs(algtype, Input)
checks the structure array Input can be used in: RRvalueiterationif ***algtype*** = 'VFI' and RRpolicyiteration if ***algtype*** = 'PI'.

isError = RRcheckinputs(algtype, P, R, beta, epsilon, maxiter, V0, verbose) (only valid when ***algtype*** = 'VFI') checks the required inputs ***P***, ***R*** and ***beta***, and the optional inputs ***epsilon***, ***maxiter***, ***V0***, ***verbose*** and returns ***isError*** = true if the inputs cannot be used in RRvalueiterationand ***isError*** = false otherwise.

isError = RRcheckinputs(algtype, P, R, beta, policy0, maxiter,...
policyevalmethod, verbose)
(only valid when ***algtype*** = 'VFI') checks the required inputs ***P***, ***R*** and ***beta***, and the optional inputs ***policy0***, ***maxiter***, ***policyevalmethod***, ***verbose*** and returns ***isErr***the inputs cannot be used in RRpolicyiteration or and ***isError*** = false otherwise.

[isError optionalInputsErrorFlag errorMessageBank] = RRcheckinputs(...)
returns ***isError***, ***optionalInputsErrorFlag*** and ***errorMessageBank***, which are described below.

Tips

The first input argument to RRcheckinputsis always ***algtype***, the type of algorithm that you are checking inputs for, 'VFI', 'PI', or 'BI'. After the first input argument, the inputs are *exactly* the same as if

you were running the actual algorithm that you are checking inputs for, RRvalueiteration, RRpolicyiteration, or RRbackwardinudction.

Input Arguments

Except for ***algtype***, all inputs arguments are the same as for RRvalueiterationor RRpolicyiteration. Please see the documentation for those algorithms for more information about ***Input***, ***P***, ***R*** etc.

algtype	A string indicating the type of algorithm that the inputs are being checked for. The only valid values of ***algtype*** are 'VFI', 'PI', and 'BI' indicating value function iteration, policy iteration, and backward induction respectively. You may also control the amount of output echoed to the screen by entering ***algtype*** as a cell where the second argument is one of 'quiet', 'loud', or 'louder'. 'Quiet' will echo only the bare minimum of messages, while 'louder' will report the results of all checks performed. Error messages (if needed) will be reported regardless of the value of this argument. The default is 'quiet'.

Output Arguments

Up to three output arguments can be requested from RRcheckinputs. These variables are described below.

isError	True when at least one of the inputs has an error and cannot be used in the selected algorithm. False when the inputs can be used. There may be messages displayed to the command window even if ***isError*** = false; this occurs when there are potential errors that would not stop an algorithm from running, e.g. when there is only one state.
optionalInputsErrorFlag	6 x 1 vector that whose i-th entry is 1 if the i-th optional input in the following order ***epsilon***, ***maxiter***, ***V0***, ***verbose***, ***policy0***, ***policyevalmethod*** contained an error, and is 0 otherwise.
errorMessageBank	A cell array whose cells contain any error messages. Each cell contains a different error message. If there are no error messages, ***errorMessageBank*** is a 1 x 1 cell array containing an empty matrix.

Examples

Before using RRvalueiteration, the following inputs are run in RRcheckinputsto check for errors:

```
P(:,:,1)=  [0.9          0.1;
           0.9          0.1];
```

```
P(:,:,2)= [0.8          0.2;
           0.8          0.2];

P(:,:,3)= [0.7          0.3;
           0.7          0.3];

R =[10          8       4;
    50          25      15];

beta = 1.1;

iserror = RRcheckinputs({'VFI', 'louder'},P,R,beta)
```

The following is displayed to the command window, which explains the error is with ***beta***:

```
---------------
   RR Toolbox: The size of "P" and "R" conform.
---------------
   RR Toolbox: "P" passed all checks.
---------------
   RR Toolbox: "R" passed all checks.
---------------
   RR Toolbox: No value has been entered for the following optional inputs: "epsilon"
      "maxiter"
      "V0"
      "verbose"
   Default values will be used for these inputs.
---------------
   The RR Toolbox has checked the Input structure for conformance. convergence, and validation of key
inputs. At least one problem was
   discovered.
---------------
   RR Toolbox: There was an error with input(s) to RRvalueiteration. Please read the message(s) below for
more information.
---------------
   RR Toolbox: "beta" has been entered as a number strictly greater than 1.
Make sure "beta" is a number strictly greater than 0 and no greater than 1.
---------------

iserror =

     1


iserror = RRcheckinputs({'VFI', 'quiet'},P,R,beta)
---------------
   The RR Toolbox has checked the Input structure for conformance. convergence, and validation of key
inputs. At least one problem was
```

discovered.

RR Toolbox: There was an error with input(s) to RRvalueiteration. Please read the message(s) below for more information.

RR Toolbox: "beta" has been entered as a number strictly greater than 1.
Make sure "beta" is a number strictly greater than 0 and no greater than 1.

```
iserror =
     1
```

See Also

RRvalueiteration | RRpolicyiteration | RRbackwardinduction | RRtable

RRcompareoptima

This function compares the profit maximizing solution of a sequential decision problem to the value maximizing solution of the same problem.

Syntax

OptimaTable = RRcompareoptima(Out)

Description

OptimaTable = RRcompareoptima(Out)
takes the out structure from a sequential decision problem, and them compares the value optimizing and profit optimizing solutions.

Input Arguments

Out	(Required) An out structure from a sequential decision problem solved using the Rapid Recursive® Toolbox. Out must have the following fields: ***Input***, ***policy***, ***fields*** ***Out.Input*** must be a structure with the following fields: ***R,S,A,P, beta, statelabels, actionlabels***.

Output Arguments

OptimaTable	A table showing the value maximizing action and value, and profit maximizing action and value, for each state in the problem.

Examples

Out = RentalPropertyManagement OptimaTabel =

RRcompareoptima(Out) OptimaTable =

State	Value_Max_Action	Profit_Max_Action	Value_Max_V	Profit_Max_V
'Temp separation'	'Reject'	'Reject'	'$39,497.80'	'$31,088.05'
[1050.00]	'Reject'	'Accept'	'$39,497.80'	'$28,603.95'
[1150.00]	'Reject'	'Accept'	'$39,497.80'	'$29,709.53'

[	1250.00]	'Reject'	'Accept'	'$39,497.80'	'$30,815.12'
[	1350.00]	'Accept'	'Accept'	'$39,608.62'	'$31,920.70'
[	1450.00]	'Accept'	'Accept'	'$40,714.20'	'$33,026.29'
[	1550.00]	'Accept'	'Accept'	'$41,819.79'	'$34,131.87'
[	1650.00]	'Accept'	'Accept'	'$42,925.37'	'$35,237.45'
[	1750.00]	'Accept'	'Accept'	'$44,030.95'	'$36,343.04'

RRfigure

Creates a new figure with the default background color for the Rapid Recursive® Toolbox and a note indicating that the figure was created by the Rapid Recursive® Toolbox.

Syntax

[h, ax] = RRfigure

[h, ax] = RRfigure('PropertyName', 'PropertyValue')

Description

[h, ax] = RRfigure
creates a new figure with the Rapid Recursive® default background color and a note indicating that the figure was created by the Rapid Recursive® Toolbox.

[h, ax] = RRfigure ('PropertyName', 'PropertyValue')
creates a new figure with the default Rapid Recursive background color and the specified properties set to the provided values.

Input Arguments

There are no required inputs for this function, though any of the property/value pairs associated with standard MATLAB figures may be passed as optional inputs. Note, however, that the 'color' property for ***RRfigure*** will always be set to Supported Intelligence's standard beige.

Output Arguments

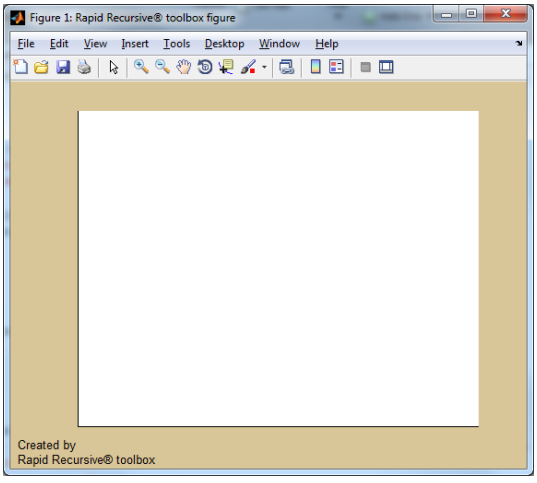
This function creates a new figure window.

hcontains the handle to this figure.

axcontains the handle to the axis object within the figure.

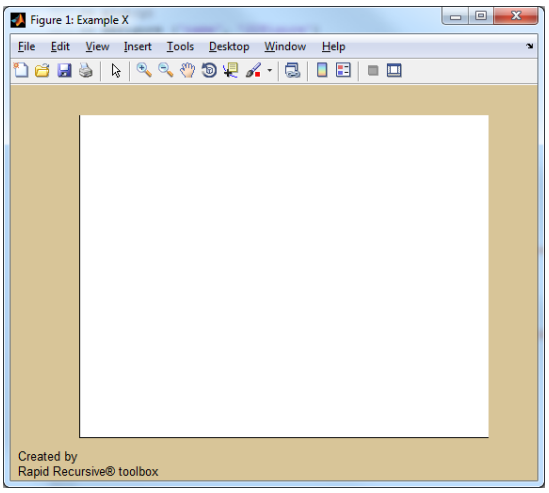
Example

[h, ax] = RRfigure



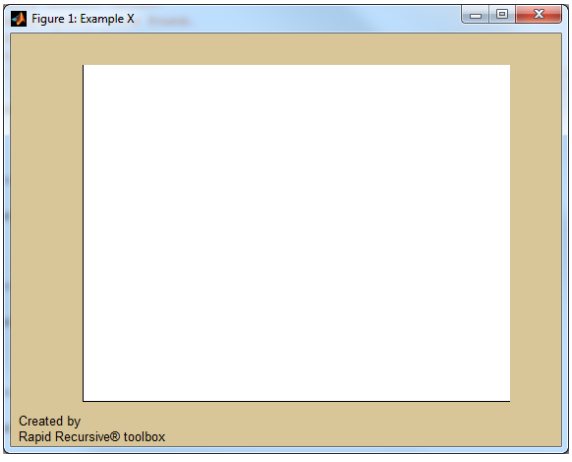
h = 1, ax = 123.022

[h, ax] = RRfigure('name', 'Example X')



h = 1, ax = 123.022

[h, ax] = RRfigure ('name', 'Example X', 'menu', 'none')



h = 1, ax = 123.022

See Also

figure (a MATLAB® function)

Note: For a comprehensive list of figure properties, run "help figure" in the MATLAB Command Window.

RRviewreward

Displays a ribbon chart visualization of the reward matrix, where each ribbon corresponds to a different state in the reward matrix and shows how rewards change as actions vary.

Syntax

h = RRviewreward(Input, 'ParameterName', ParameterValue) h =

RRviewreward(R, 'ParameterName', ParameterValue)

Description

[h, ax] = RRviewreward(Input, 'ParameterName', ParameterValue)
displays a ribbon chart visualization of the reward matrix stored in ***Input.R***. If no values are provided for ***statelabels*** or ***actionlabels***, the function attempts to use the values stored in ***Input.statelabels*** or ***Input.actionlabels***.If those fields are not found, numeric labels are used. Returns the handle to the figure as ***h***; and the axes handle as ***ax***.

[h, ax] = RRviewreward(R,'ParameterName', ParameterValue)
displays a ribbon chart visualization of the reward matrix stored in ***R.*** If no values are provided for ***staelabels*** or ***actionlabels,*** numeric labels are used. Returns the handle to the figure as ***h***; and the axes handle as ***ax***.

Input Arguments

Note, only one of the following input arguments is required.

Input	An input structure for a Rapid Recursive problem. This must contain at least a field for ***R***, the S x A reward matrix. The function will also look for the fields "statelabels" and "actionlabels," unless these are explicitly provided as optional input parameters (see below).
R	S x A reward matrix. *Note: RRviewreward cannot currently display three dimensional reward matrices.*

Optional Parameters

statelabels	A cell array of strings containing one label for each state. *Note: The number of states equals the number of rows in the reward matrix.*
actionlabels	A cell array of strings containing one label for each action.

	Note: The number of actions equals the number of columns in the reward matrix.
colorshift	Toggles the brief colorshift. Permitted values are 'on' and 'off'. The default value is 'off'.
orbit	Toggles the camera panning feature. Permitted values are 'on' and 'off'. The default value is 'off'.

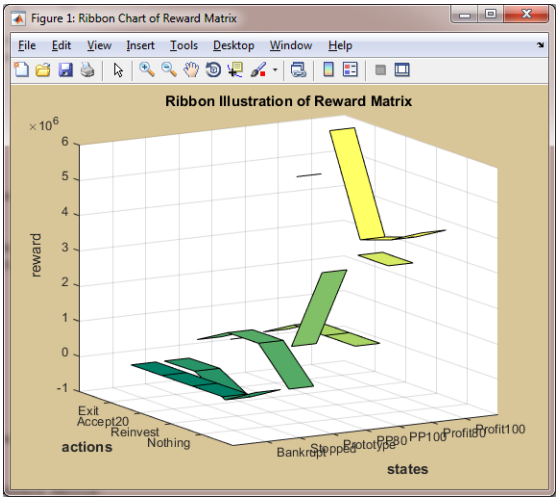
Output Arguments

h	The handle to the figure window containing the ribbon chart.
ax	The handle to the axes containing the ribbon chart.

Examples

In this example, the Input structure from the "StartupEntrepreneur" template was used to generate a ribbon chart visualization of the reward matrix.

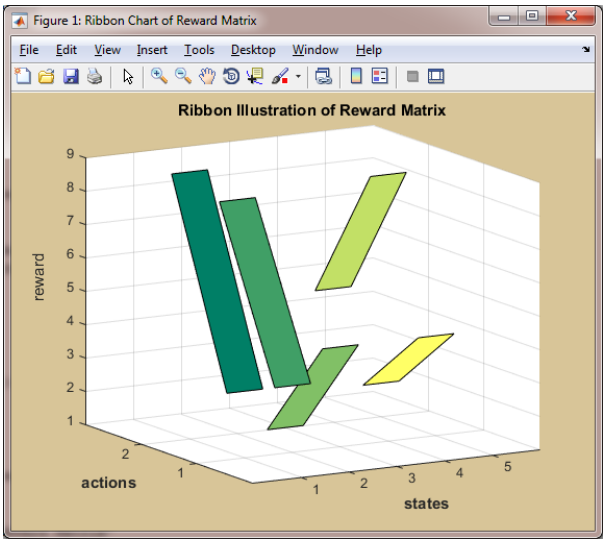
[h, ax] = RRviewreward(Input)



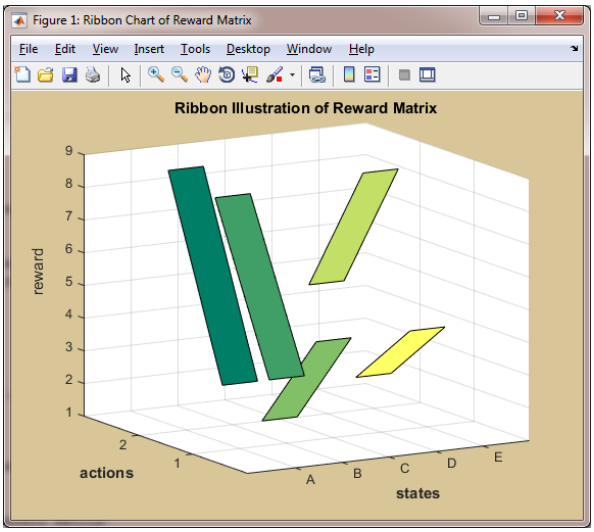
h = 1

ax = 173.0015

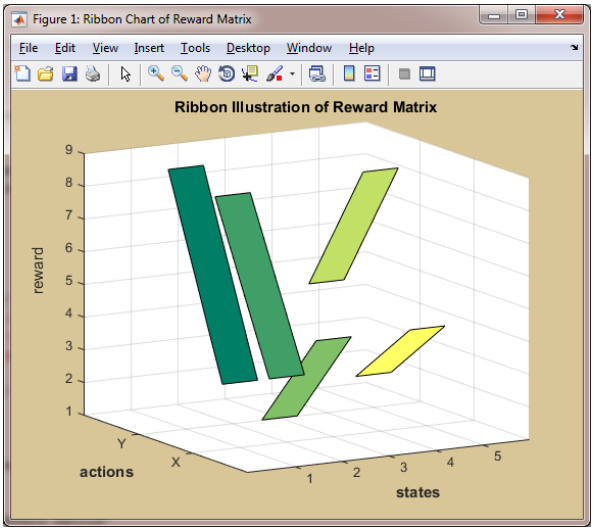
```
R = [3 9;
     3 8;
     4 1;
     9 5;
     4 2];

[h, ax] = RRviewreward(R)
```




```
h = 1

ax = 173.0022

RRviewreward(R, 'statelabels', {'A' 'B' 'C' 'D' 'E'});
```



RRviewreward(R, 'actionlabels', {'X' 'Y'});



RRviewtransition

Displays one frame of the transition matrix for a recursive problem as a heat map of the transition probabilities.

Syntax

h = RRviewtransition(Input, 'ParamName', ParamValue)

Description

h = RRviewtransition(Input, 'ParamName', ParamValue)
displays the transition matrix according to the parameter-value pairs given. By default, the function will display the frame for the first action in the problem, cycle through the remaining actions, and stop again on the first. The function returns the handle to the figure window as ***h***.

Inputs

Input	(Required) An input structure from a recursive problem
P	(Required) In place of Input, you may alternatively enter ***P***, an S x S x A transition matrix.

Optional Parameters

action	An integer that indicates the action to be shown.
scroll	An 'on' | 'off' argument that controls the scrolling behavior of the function. The default is 'off'.
handle	A handle object that indicates the Matlab figure in which the transition matrix should be displayed.
statelabels	S x 1 cell array containing labels for each state in the decision problem set to ***Input.statelabels*** by default.

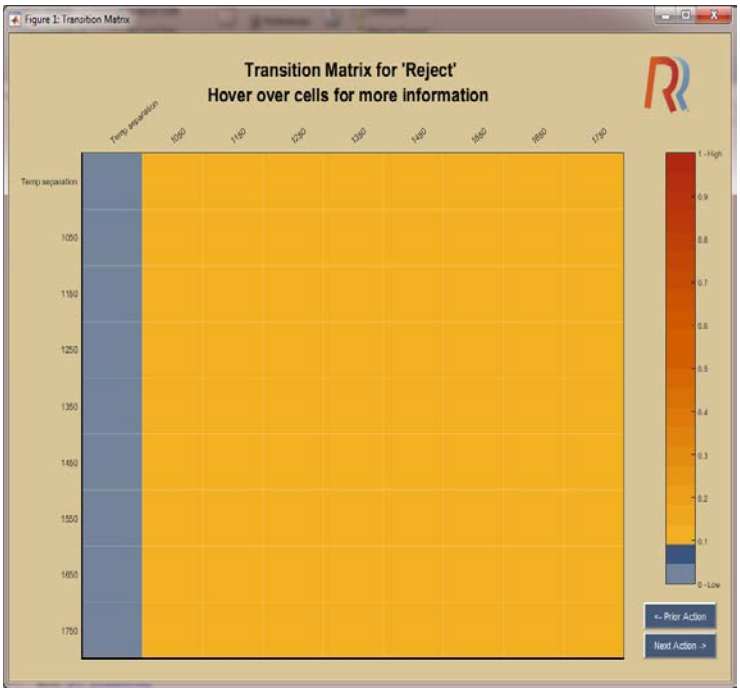
Output

h	The handle for the Matlab figure window in which the transition matrix is displayed.

Examples

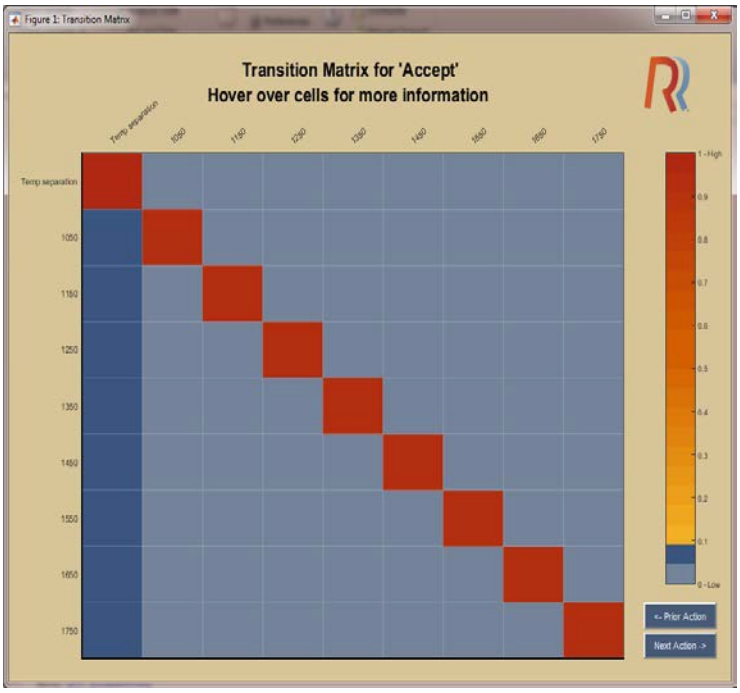
Run RentalPropertyManagement template:

h = RRviewtransition(Input)



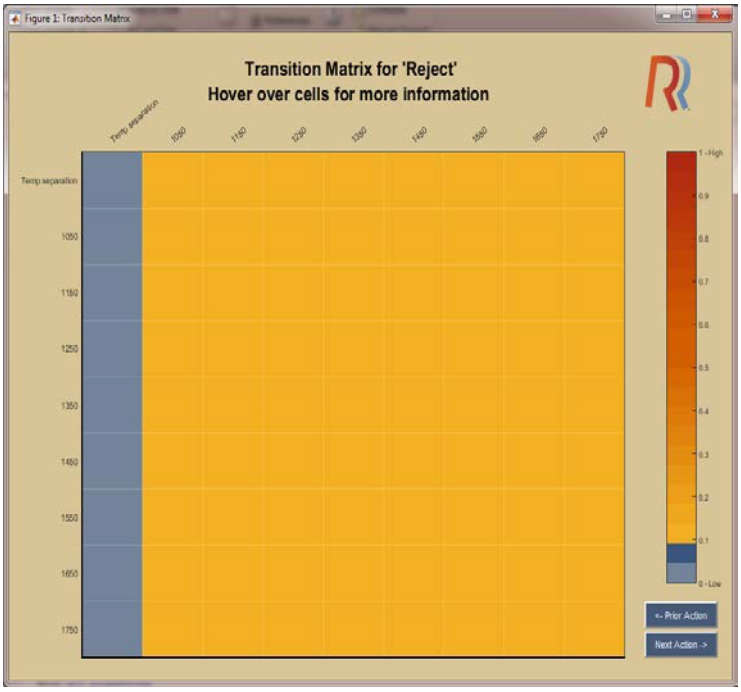
h = 1

h = RRviewtransition(Input, 'action', 2)

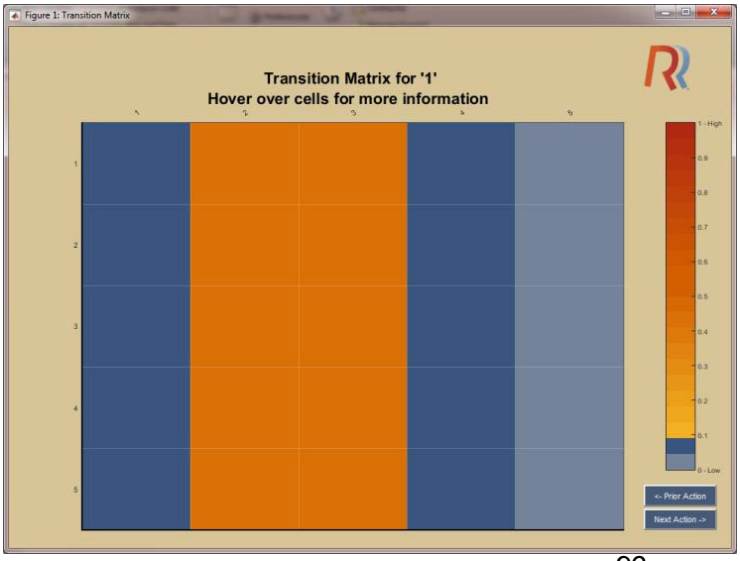


h = 2

```
g = figure;

h = RRviewtransition(Input,'scroll',false,'handle',g)
```




```
h = 1.00

P = RRcreatetransition(5,'n'); RRviewtransition(P)
```



ItoProject

Simulates certain Ito processes or stochastic integrals, which rely upon Brownian motion. Note, this function requires the Financial Toolbox in order to run.

Syntax

[data, h] = ItoProject(method, drift, sigma, X0, T, TimeIncrement,...
plotflag, seed)

Description

[data, h] = ItoProject(method, drift, sigma, X0, T, TimeIncrement,...
plotflag, seed)

A function to simulate certain Ito processes or stochastic integrals, which rely upon Brownian motion and can be described as Stochastic Differential Equations ("SDEs"). The SDEs that can be simulated with this function include simple Brownian motion, Brownian motion with drift, and Geometric Brownian motion.

Inputs

method	(Required) The method of Brownian motion to be used ("simple", "drift", or "geometric").
drift	(Required) The magnitiude of drift, entered as a number.
sigma	(Required) The diffusion parameter, entered as a number.
X0	(Required) The initial state value vector, which is required for non-scalar inputs of **drift** and **sigma**.

Optional Parameters

T	(Optional) The number of periods to project.
TimeIncrement	(Optional) Used for the Brownian motion (default = 1).
plotflag	(Optional) A flag to indicate whether to show/suppress the plot of motion (a value of 0 suppresses the default figure and plot).
seed	(Optional) A seed for random number generation.

Output

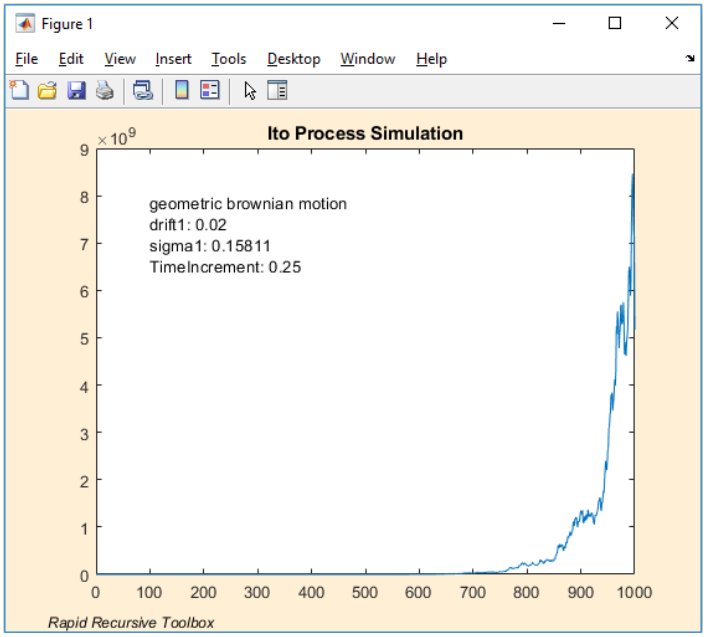
data	A structure with fields for method, X0, the projected data series x, and other data.

h	The handle for the Matlab figure window in which

	the transition matrix is displayed.

Example

```
[data, h] = ItoProject('geometric', 0.08/4, sqrt(0.1/4), 10*4, 1000, 1/4) data =

    struct with fields:

                    method: 'geometric' drift: 0.02

                     sigma: 0.16

                     x0: 40.00

                       T: 1000.00

             TimeIncrement: 0.25

                         x: [1000×1 double]

                         z: [1000×1 double]

                       eta: [1000×1 double]

                      note: 'Generated by ItoProject; Bus Econ toolbox' date: '7-Nov-2025

                      11:55:00'

                      seed: 737391.50


h =

    Figure (1) with properties: Number: 1

                      Name: ''

                     Color: [1 0.9373 0.8353]

                  Position: [680 558 560 420] Units: 'pixels'
```



TrendProject

A function which projects baseline, high, and low growth trends based on: a base revenue, a base growth rate, a deviation from the base, and a number periods for which to project.

Syntax

[projections, inputs] = TrendProject(base, rate, deviation, numperiods)

[projections] = TrendProject

Description

[projections, inputs] = TrendProject(base, rate, deviation, numperiods) TrendProject provides the projections of revenue given a baseline, rate of growth, a deviation, and a number of periods to observe. The function returns a three-column by numperiods-row matrix whose columns correspond with a high revenue projection, baseline revenue projection, and low revenue projection as varied by the deviation variable.

If the function is called as:

[projections] = TrendProject
a dialogue window will appear and prompt the user for inputs, and a figure of the projections will be produced.

Inputs

base	(Required) The base revenue at the beginning of the projection.
rate	(Required) The growth rate, expressed as a decimal.
deviation	(Required) The deviation from the growth rate, expressed as a decimal.
numperiods	(Required) The number of periods for which to project.

Output

projections	A three-column by numperiods-row matrix whose columns correspond with high, baseline, and low revenue projections.
inputs	A structure which contains the input variables passed to the function.

Example

[projections, inputs] = TrendProject(100, .10, .05, 10)

a =

100.00	100.00	100.00
110.50	110.00	109.50
122.10	121.00	119.90
134.92	133.10	131.29
149.09	146.41	143.77
164.74	161.05	157.42
182.04	177.16	172.38
201.16	194.87	188.76
222.28	214.36	206.69
245.62	235.79	226.32

b =

struct with fields:

base: 100.00

rate: 0.10

deviation: 0.05

numperiods: 10.00

ExtractStateDimension

A function which breaks down a multi-dimensional state-space Input structure into sub-Input structures, each corresponding to a space dimension.

Syntax

[subInputsCell] = ExtractStateDimension(Input)

Description

[subInputsCell] = ExtractStateDimension(Input)
returns a cell array with entries for each space dimension present in the Input structure parameter. The required fields in the Input structure are: statelabels, A, actionlabels, the state dimension parameters S1, S2, S3, etc., and the corresponding transition dimesion parameters P1, P2, P3, etc.

Inputs

Input	(Required) An Input structure with the fields: statelabels, A, actionlabels, the state dimensions S*x*, and corresponding transition dimensions P*x*.

Output

subInputsCell	A cell array with each of its entries being a sub-Input structure which corresponds to one of the state-dimensions of the original Input structure.

Example

AutoDriveOrSell;

subInputsCell = ExtractStateDimension(Input) subInputsCell =

1×3 cell array

{1×1 struct} {1×1 struct} {1×1 struct}

subInputsCell{1} ans =

struct with fields:

S: 6.00

```
          statelabels: {6×1 cell}
                    A: 3.00
       actionlabels: {'Drive'   'DriveLess'   'Sell'} P: [6×6×3 double]
                 name: 'InputS1' subInputsCell{2}
ans =
    struct with fields:
                    S: 3.00
          statelabels: {'MPG24'   'MPG26'   'MPG30'}
                    A: 3.00
       actionlabels: {'Drive'   'DriveLess'   'Sell'} P: [3×3×3 double]
                 name: 'InputS2'


subInputsCell{3} ans =
    struct with fields:
                    S:  4.00
          statelabels:  {4×1 cell}
                    A:  3.00
       actionlabels:  {'Drive'   'DriveLess'              'Sell'}
                    P:  [4×4×3 double]
                 name:  'InputS3'
```

RRslicestates

Returns a subset of a BigState table that correspond to particular states.

Syntax

[slicedResultsTable] = RRslicestates(Out, stateSlice1, stateSlice2, ...)

Description

[slicedResultsTable] = RRslicestates(Out, stateSlice1, stateSlice2, ...) Returns a sliced subset of an Output structure's results table. This function requires that the CreateBigState function was employed when creating the Input's multi-dimensional state-space. The "state slices" are passed as arrays and indicate which state values should be selected from the results table. Note: empty array state slices indicate that all values of that state-dimension should be selected.

Inputs

Out	(Required) An Output structure which resulted from the solution of an Input structure via *RRpolicyiteration* or *RRvalueiteration*.
stateSliceX	(Required) Arrays indicating which values of each state-dimension should be selected from the rows of the BigState table.

Output

slicedResultsTable	A subset of the results table of the Output structure whose rows correspond with the requirements set by the *stateSlice* arrays.

Example

slicedResultsTable = RRslicestates(Output, [], [24, 26], [1:10])

This call to RRslicestates implies that Output has a three-dimensional state-space (given its three state slices), and will select all rows from the Output's resultstable that satisfy all of the following criteria:

First Dimension: All state1 values are allowed.

Second Dimension: Only state2 values of 24 and 26 are allowed. Third

Dimension: Only state3 values of 1 through 10 are allowed.

RRtrainreward

Trains and returns a reward matrix based on a data series of observed state-action combinations and their rewards to the decision maker.

Syntax

[R] = RRtrainreward(Input, Dataseries)

[R] = RRtrainreward(Input, Dataseries, 'method', method)

Description

[R] = RRtrainreward(Input, Dataseries)

Returns a trained reward matrix based on the state-action rewards observed from the data series. The dataseries must be a three-rowed cell array where the first row denotes the action taken by the decision maker, the second row denotes the state the decision maker was in, and the third row contains the reward obtained by the user for pursuing that state-action combination.

[R] = RRtrainreward(Input, Dataseries, 'method', method)

Allows the user to select which method they'd like to apply for calculating the final reward for each state-action combination, given a list of options.

Inputs

Input	(Required) An Input structure which contains the fields: *statelabels, actionlabels, S,* and *A*.
Dataseries	(Required) A three-rowed cell array where the first row contains the observed action label, the second row contains the observed state label, and the third row contains the reward obtained by user.
method	(Optional) The method to be applied to the observed state-action combination rewards to calculate the final R matrix. The options are: '*min*', '*max*', and '*mean*' (default).

Output

R	An S x A reward matrix, where S is the number of states found in the state history matrix and A is the number of actions represented in the action history matrix. The entries are the rewards for taking a given action (column) in a certain state (row).

Example

```
Input = RRcreateinputstruct('e'); Input.statelabels =
{'full', 'hungry'}; Input.actionlabels = {'eat', 'wait'};
Input.A = length(Input.actionlabels); Input.S =
length(Input.statelabels);
dataseries = {'eat', 'wait', 'eat', 'wait', 'eat';
                    'full', 'full', 'hungry', 'hungry', 'hungry' ;
                    -100, 0, 100, -100, 90};
R = RRtrainreward(Input, dataseries) R =
    -100          0
      95   -100
R = RRtrainreward(Input, dataseries, 'method', 'min') R =
    -100          0
      90   -100
```

RRtraintransition

Trains and returns a transition matrix based on a data series of observed state-action combinations and the resulting state to decision maker transitioned to.

Syntax

[P] = RRtraintransition(Input, Dataseries)

[P] = RRtraintransition (Input, Dataseries, 'method', method)

Description

[P] = RRtraintransition (Input, Dataseries)

Returns a trained reward matrix based on the state-action rewards observed from the data series. The dataseries must be a three-rowed cell array where the first row denotes the action taken by the decision maker, the second row denotes the state the decision maker was in, and the third row contains the reward obtained by the user for pursuing that state-action combination.

[P] = RRtraintransition (Input, Dataseries, 'method', method)

Allows the user to select which method they'd like to apply for calculating the final reward for each state-action combination, given a list of options.

Inputs

Input	(Required) An Input structure which contains the fields: *statelabels, actionlabels, S,* and *A*.
Dataseries	(Required) A three-rowed cell array where the first row contains the observed action label, the second row contains the observed state label, and the third row contains the state transitioned to by the user.
method	(Optional) The method to be applied to the observed state-action combination transitions to calculate the final R matrix. Currently, the only option available is '*mean*' (default).

Output

P	An S x S x A transition matrix, where S is the number of unique states found in the state history matrix and A is the number of unique actions represented in the action history matrix. The entries represent the likelihood of moving from one state (row) to another (column) given a specific action (frame).

Example

```
Input = RRcreateinputstruct('e'); Input.statelabels =
{'full', 'hungry'}; Input.actionlabels = {'eat', 'wait'};
Input.A = length(Input.actionlabels); Input.S =
length(Input.statelabels);
dataseries = {'eat', 'wait', 'eat', 'wait', 'eat';
                                    'full', 'full', 'hungry', 'hungry', 'hungry' ;
                'full', 'hungry', 'full', 'hungry', 'hungry'}; P =
RRtraintransition(Input, dataseries)

    P(:,:,1) =

    1.0000         0
    0.5000    0.5000

P(:,:,2) =

    0    1
    0    1
```

RRinvesttransition

Creates a transition matrix that can be used in a reinvestment problem.

Syntax

RRinvesttransition(S,theta)

Description

RRinvesttransition(S,theta)
creates an S x S x 3 transition matrix that can be used in a reinvestment problem. Sis the number of states and must be between 3 and 5. The transition matrix will have three actions, 1 to 3 (the actions are in order of increasing reinvestment).

thetais the probability that the decision maker moves up or down a state and must be between 0 and $\frac{1}{3}$. As an example to demonstrate how thetais used, this is the formula to calculate the second action of a transition matrix:

```
[1-theta,       theta,          0,          0;
   theta,  1-2*theta,      theta,          0;
       0,       theta, 1-2*theta,      theta;
       0,           0,      theta,  1-theta];
```

Example

```
S = 4;
theta = 0.1;
P = RRinvesttransition(S,theta)
```

Returns:

```
P(:,:,1) =

    1.0000         0         0         0
    0.9000    0.1000         0         0
    0.1000    0.8000    0.1000         0
         0    0.1000    0.8000    0.1000


P(:,:,2) =

    0.9000    0.1000         0         0
    0.1000    0.8000    0.1000         0
         0    0.1000    0.8000    0.1000
         0         0    0.1000    0.9000


P(:,:,3) =
```

0.1000	0.8000	0.1000	0
0	0.1000	0.8000	0.1000
0	0	0.1000	0.9000
0	0	0	1.0000

capitalizedvalue

Calculates present value according to the Gordon Growth formula.

Syntax

cv = capitalizedvalue(pi, d, g) cv = capitalizedvalue(pi, d)

Description

cv = capitalizedvalue(pi, d, g)
calculates the present value of a perpetual stream of income ***pi*** with discount rate ***d*** and growth rate of income ***g***. The first income is pi(1 + g) and is received one period forward from the present. Both ***d*** and ***g*** should be entered as decimals—not as percentages.

The formula used is: $cv = \frac{pi(1+g)}{d-g}$

Input Arguments

pi	***pi*** is the income that is received each period.
d	***d*** is the discount rate. Enter this as a decimal – not as a percentage e.g. for a 15 percent discount rate, enter 0.15. d must be at least 0 and strictly greater than ***g***.
g	(Optional) ***g*** is the growth rate. Enter this as a decimal – not as a percentage e.g. for a 2 percent growth rate, enter 0.02. This is an optional input. The default value of g is 0. If entered, ***g*** must be strictly less than d and strictly greater than -1.

Examples

cv = capitalizedvalue(1000, 0.15, 0.02) calculates the present value of receiving 1000 each period at a discount rate of 15 percent and a growth rate of 2 percent.

```
cv =
        7846.15
```

cv = capitalizedvalue(1000, 0.15) is the same as the first example, except with a zero growth rate.

```
cv =

        6666.67
```

See Also

RRvalueiteration | RRpolicyiteration

RRrewardmatrix

Calculates a reward matrix (reward function) from a set of matrices containing the states, applied actions, and received rewards for a population of observations taken over time.

Syntax

R = RRrewardmatrix(state_history, action_history, reward_history)

[R, states, actions] = RRrewardmatrix(state_history, action_history, reward_history)

Description

R = RRrewardmatrix(state_history, action_history, reward_history)
returns the reward matrix, R, implied by the state history, action history, and reward history provided as inputs. Each entry in Ris the average of the reward histories for being in a given state and taking a specific action.

[R, states, actions] = RRrewardmatrix(state_history, action_history, reward_history)
returns the reward matrix, R, as above, in addition to states, a vector of the state values, and actions, a vector of the action values found in the input matrices.

Input Arguments

state_history	(Required) An N x T matrix of integer values that catalog the discrete states occupied by each member of the population at each point in time. N is the number of entities and T is the size of the population.
action_history	(Required) An N x T (or T-1) matrix of integer values that catalog the discrete actions taken on each member of the population at each point in time. N is the size of the population and T is the number of time periods.
reward_history	(Required) An N x T matrix of values that represent the reward from each member of the population in each time period. N is the size of the population and T is the number of time periods.

Output Arguments

R	An S x A reward matrix, where S is the number of states found in the state history matrix and A is the number of actions represented in the action history matrix. The entries are the rewards for

	taking a given action (column) in a certain state (row).
states	An S x 1 vector of detected state values
actions	An A x 1 vector of detected action values

Examples

```
rng(8675309); % set seed for random number generator

s_hist = ceil(rand(100,5)*6); % create state history for 100 people in
                              % 6 states over 5 time periods.

a_hist = ceil(rand(100,5)*2); % create action history for 100 people
                              % with 2 actions over 5 time periods.

r_hist = rand(100,5)*1500; % create random reward history for 100
                           % people over 5 time periods.

[R, states, actions] = RRrewardmatrix(s_hist, a_hist, r_hist)

R =
   666.6478   755.8655
   744.8570   853.8416
   829.9362   783.8443
   836.3588   699.3868
   624.0394   779.4690
   794.0614   755.4698

states =
     1
     2
     3
     4
     5
     6

actions =
     1
     2
```

See Also

RRtransitionmatrix |RRcombinerewards

RRshortincomestatement

Calculates and displays to the MATLAB command window an income statement for a company whose business is described by a small set of key variables.

Syntax

r = RRshortincomestatement(revenue, COGS, opexp, otherExp)
r = RRshortincomestatement(revenue, COGS, opexp, otherExp, actioncost) r = RRshortincomestatement(revenue, COGS, opexp, otherExp, actioncost, dividend)

Description

RRshortincomestatement calculates and displays to the MATLAB command window an income statement whose business is described by the key variables captured in the inputs.

RRshortincomestatement calculates gross profit, net profit, and a dividend payout using the four required inputs of ***revenue***, ***COGS*** (cost of goods sold), ***opexp*** (operating expenditure), and ***otherExp***. The optional inputs are ***actioncost*** and ***dividend***.

The function calculates gross profit, net profit, and a dividend payout using the following equations:

gross profit = ***revenue*** – ***COGS***

operating profit = gross profit – ***actioncost*** – ***opexp***

net profit = operating profit - ***otherExp***

dividend payout = net profit * ***dividend***

The output argument, **r**, is equal to net profit if dividend equals zero, and is equal to the dividend payout otherwise.

Input Validation

Before attempting any calculations, RRshortincomestatementvalidates its inputs. If an error is found, RRshortincomestatementis stopped and messages identifying the error(s) are displayed to the command window.

Tips

actioncost and ***dividend*** are optional input arguments and can be entered only if all previous inputs have been entered. Either may be skipped by entering a value of 0 in its place.

Input Arguments

revenue	(Required) The firm's revenue for the most recent year.

COGS	(Required) Cost of goods sold in the most recent year. Can be entered either as a dollar value or a proportion of total revenue. If entered as a proportion, ***COGS*** must be a decimal between 0 and 1.

opexp	(Required) The firm's operating expenses over the most recent year, expressed either as a flat amount or as a proportion of total revenue. If entered as a proportion, ***opexp*** must be a decimal between 0 and 1.
otherExp	(Required) Other expenses captures the epenses that are incurred by the firm and not captured in the firm's operating expenses. These could include taxes, royalties or license fees, or unusual or miscellaneous expenses. This value may be entered either as a flat amount or as a proportion of total revenue. If entered as a proportion, ***otherExp*** must be a decimal between 0 and 1.
actioncost	(Optional) The cost of an action the firm has undertaken during the most recent year. Examples of this may be the cost of advertising, or the amount of an investment made by the firm's management.
dividend	(Optional) The proportion of net profit that is distributed in the form of dividend payouts. This value must be a decimal between 0 and 1.

Example

Consider a firm that earned $5 M in revenue over the last year, with $2.2 M in COGS, an operating expense ratio of 0.15, and other expenses equal to 10% of revenue. Their income statement could be generated as follows:

```
r = RRshortincomestatement(5000000,2200000,0.15,0.1)
                                ------------
    Revenue                          5.00  $M
    Less:COGS                        2.20  $M
                                ------------
    Gross Profit                     2.80  $M
    Less:Opexp                       0.75  $M
                                ------------
    Operating Profit                 2.05  $M
    Less:Other Expenses              0.50  $M
                                ------------
    Net Profit                       1.55  $M
                               ------------

ans =

    1550000
```

Now assume the same firm spent $500,000 on an advertising campaign and that they pay a dividend equal to 85% of net profits. Their income statement would now be:

```
r = RRshortincomestatement(5000000,2200000,0.15,0.1,500000,0.85)

                                 ------------
    Revenue                             5.00  $M
    Less:COGS                           2.20  $M
                                 -----------
    Gross Profit                        2.80  $M
    Less:Opexp                          0.75  $M
    Less:Action Cost                    0.50  $M
                                 -----------
    Operating Profit                    1.55  $M
    Less:Other Expenses                 0.50  $M
                                 -----------
    Net Profit                          1.05  $M
    Dividend                            0.89  $M
                                 ------------

ans =

      892500
```

Note

This is an abbreviated version of an accounting income statement. It contains significantly fewer elements than a standard accounting statement, and is intended to be used to capture changes in major categories. It is not intended as a complete statement of income for a company, nor a calculation of tax liability.

See Also

capitalizedvalue

RRspecialkron

An altered form of the kronecker tensor product acting on one 2D (A) and one 3D matrix (B). The result is a large matrix formed by taking all possible products between the elements in each column of A and the elements in each frame of B.

Syntax

K = RRspecialkron(A, B)

Description

K = RRspecialkron(A, B)
is a modification of the Kronecker product that allows for statistically depended matrices of differing dimensions to be combined.

Input Arguments

A	***A*** is an MA x NA matrix.
B	***B*** is an MB x NB x NA matrix, or a cell vector of length NA, where all elements are MB x NB matrices.

Output Arguments

Output	***K***: An (MB * MA) x (NB * NA) matrix. Elements in ***K*** follow the form: [A(1,1) * B(:,:,1),..., A(1, NA) * B(:,:,1); ... A(MA,1) * B(: ,:, NA), ..., A(MA,NA) * B(:,:,NA)]

Examples

A = [1 2; 3 4];

B = [[1 0; 0 1], [2 3; 4 5]];

RRspecialkron(A, B)

ans =

1	0	2	3	2	0	4	6
0	1	4	5	0	2	8	10
3	0	6	9	4	0	8	12

0 3 12 15 0 4 16 20

RRtimetransition

Creates one frame of a transition matrix for time periods, where time moves forward to the next period with certainty.

Syntax

timeP = RRtimetransition(nPeriods)

timeP = RRtimetransition(nPeriods, finalBehavior)

Description

timeP = RRtimetransition(nPeriods)
returns as timePa square matrix (with size nPeriods x nPeriods), where all of the entries immediately above the main diagonal are set to 1. The first entry of the final row is also set to 1, representing the case where time wraps back to the first period.

timeP = RRtimetransition(nPeriods, finalBehavior)
returns the same as above, though setting finalBehaviorto 1 will treat the final period as an absorbing state rather than a wrap-around state (i.e. timeP(nPeriods, nPeriods) = 1). Setting finalBehavior to 'wrap'treats the final period as a wrap-around state.

Input Arguments

nPeriods	(Required) The number of time periods in the model.
finalBehavior	(Optional) If 'absorb' (default), the final period is treated as an absorbing state. If 'wrap', the final period is treated as a wrap-around state (the final period transitions back to the first period).

Output Arguments

timeP	An ***nPeriods x nPeriods*** square matrix with 1's immediately above the main diagonal.

Examples

timeP = RRtimetransition(5) timeP =

```
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
     0     0     0     0     1
```

timeP = RRtimetransition(5, 'wrap') timeP =

```
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
     1     0     0     0     0
```

See Also

RRtransitionmatrix | RRcombinetransitions | RRcreatetransitions

RRtransitionmatrix

Calculates a transition matrix from a set of matrices containing the states and applied actions for a population of observations taken over time.

Syntax

P = RRtransitionmatrix(state_history, action_history)

[P, states, actions] = RRtransitionmatrix(state_history, action_history)

Description

P = RRtransitionmatrix(state_history, action_history)
returns the transition matrix, P, implied by the state and action histories supplied as input arguments.

[P, states, actions] = RRtransitionmatrix(state_history, action_history) returns the transition matrix, P, as above, as well as a list of the unique states, states, and unique actions, actions, found in the input matrices.

Input Arguments

state_history	(Required) An NxT matrix of integer values that catalog the discrete states occupied by each member of the population at each point in time. N is the size of the population and T is the number of time periods.
action_history	(Required) An NxT (or T-1) matrix of integer values that catalog the discrete actions taken on each member of the population at each point in time. N is the size of the population and T is the number of time periods.

Output Arguments

P	An S x S x A transition matrix, where S is the number of unique states found in the state history matrix and A is the number of unique actions represented in the action history matrix. The entries represent the likelihood of moving from one state (row) to another (column) given a specific action (frame).
states	An S x 1 vector of detected state values.
actions	An A x 1 vector of detected action values.

Examples

```
rng(10); % set seed for random number generator

s_hist = ceil(rand(100,5)*6); % create random state history for 100
                              % people in 6 states over 5 periods a_hist =
ceil(rand(100,5)*2); % create random action history for 100
                              % people with 2 actions over 5 periods [P, states,
actions] = RRtransitionmatrix(s_hist, a_hist)

    P(:,:,1) =
      0.2667    0.1333    0.0667    0.1333    0.2222    0.1778
      0.1579    0.1842    0.1842    0.0526    0.1316    0.2895
      0.1786    0.0714         0    0.4286    0.2143    0.1071
      0.1591    0.2273    0.1136    0.2273    0.2045    0.0682
      0.1724    0.2069    0.1724    0.1379    0.1724    0.1379
      0.0500    0.2500    0.1000    0.3500    0.0500    0.2000

    P(:,:,2) =
      0.2581    0.1290    0.2258    0.1613    0.1935    0.0323
      0.2400    0.2000    0.1600    0.1600    0.1600    0.0800
      0.1875    0.1875    0.0625    0.2813    0.1563    0.1250
      0.1765    0.1176    0.2353    0.2059    0.1176    0.1471
      0.2105    0.1316    0.1579    0.1053    0.1842    0.2105
      0.1111    0.0556    0.1389    0.1944    0.2222    0.2778

states =
     1
     2
     3
     4
     5
     6

actions =
     1
     2
```

See Also

RRrewardmatrix | RRcombinetransitions | RRcreatetransition | RRtimetransition

RRbackwardinduction

Solves a finite-horizon sequential decision problem using backward induction.

Syntax

Out = RRbackwardinduction(Input) *Recommendedsyntax [...]
= RRbackwardinduction(Input)
[...] = ... RRbackwardinduction(P, R, beta, T, h, verbose) [Out,V,policy,calculationtime] = RRbackwardinduction(...)

Description

RRbackwardinductionsolves, using backward induction, discrete-time finite-horizon sequential decision problems where the decision maker has the objective of maximizing their expected discounted reward.

[...] = RRbackwardinduction(Input)
solves a sequential decision problem defined by ***Input***, which must be a structure array with fields for at least ***P***, ***R***, ***beta*** and ***T***. Input can also contain fields for ***h*** and ***verbose***, but these are not required.

[...] = RRbackwardinduction(P, R, beta, T, h, verbose)
solves the sequential decision problem defined by ***P***, ***R***, ***beta*** and ***T*** with the optional parameters, ***h*** and ***verbose***.

[Out, V, policy, calculationtime] = RRbackwardinduction(...)
solves a sequential decision problem and returns: ***Out***, a structure array containing fields for ***V***, ***policy***, ***calculationtime***, and ***Input***; the value function, ***V***; optimal policy, ***policy***; and the total computational time in seconds, ***calculationtime***.

Tips

An optional input can be entered only if all previous optional inputs have been entered. Optional inputs can be skipped by entering [] in place of the optional input, e.g. users can skip entering a value for ***h*** by using the following syntax:

[...] = RRbackwardinduction(P, R, beta, T, [], verbose)

If only a subset of the output arguments is required, the arguments you do not require can be skipped by entering ~ instead. For example, the following syntax can be used to skip ***V***, ***iterations*** and ***calculationtime***, but still request Out and ***policy***:

[Out,~,policy,~,~] = RRbackwardinduction(P,R,beta,T)

Input Arguments

Let S be the number of states and A be the number of actions.

Input	A structure array that contains fields for at least ***P***, ***R*** and ***beta*** and their values. Input can also contain optional fields for ***policy0***, ***maxiter***, ***policyevalmethod*** and ***verbose*** and their values.
P	(Required) ***P***, called the transition matrix, is the matrix representation of the state transition function. ***P*** can be entered as either a matrix or cell array. For many users, using ***P*** as a matrix will suffice; however, creating ***P*** as a cell array when it will contain many zero entries can improve computation time. When ***P*** is entered as a matrix, it must be an S x S x A matrix. The (i,j,k) element of ***P*** is the probability that the decision maker moves from the i-th state at time t to the j-th state in time t+1 when the k-th action is taken by the decision maker at time t. For example, the (3,4,2) element of ***P*** represents the probability of moving from the third state at time t to the fourth state at time t+1 when the decision maker chooses the second action at time t. An easy way to remember this is to remember that the rows of ***P*** represent the state at time t, the columns represent the state at time t+1, and the third dimension of ***P*** represents the actions. If ***P*** is entered as a cell array, it needs to be 1 x A, where each cell contains an S x S matrix that is possibly sparse. Each cell of the cell array represents a different action (the k-th cell represents the k-th action). Similar to when ***P*** entered as a matrix, the rows of each S x S matrix represent the state at time t, while the columns represent the state at time t+1. If there are many zeros in the transition matrix, entering ***P*** as a cell array with sparse S x S matrices can improve computation time.
R	(Required) ***R***, called the reward matrix, is the matrix representation of the reward function. ***R*** can be entered as either a matrix or cell array. For many users, using ***R*** as a matrix will suffice, however creating ***R*** as a cell array when it will contain many zero entries can improve computation time. If ***R*** is entered as a matrix, it can be either an S x A or S x S x A matrix; the choice will depend on the type of reward function being modeled. When ***R*** is S x A, the (i,j) element of the reward matrix represents the immediate reward that the decision maker will receive at time t given the decision maker is in the i-th state at time t and chooses the j-th action at time t. When ***R*** is an S x S x A matrix, the interpretation is slightly different. The (i,j,k) element of ***R*** represents the immediate reward that the decision maker will receive at time t given the decision maker is in the i-th state at time t, moves to the j-th state at time t+1 and the k-th action is played by the decision maker at time t. When ***R*** is an S x A matrix, it can be sparse, however it cannot be sparse when it is an S x S x A matrix. In the latter case if you require a sparse matrix create ***R*** as a cell array.

	If ***R*** is entered as a cell array, it needs to be 1 x A, where each cell contains an S x S matrix that is possibly sparse. Each cell of the cell array represents the action. Similar to previously, the rows of each S x S matrix (possibly sparse) represent the state at time t, while the columns represent the state at time t+1. If there are many zeros in the reward matrix, entering ***R*** as a cell array with sparse S x S matrices can improve computation time.
beta	(Required) ***beta*** is the discount factor and must be a number strictly greater than 0 and no greater than 1. (Caution: convergence of the algorithm may not occur when the discount factor is set to 1).
T	(Required) ***T*** is the number of periods in the model. This must at least 1.
h	(Optional) ***h*** (S x 1) is the terminal reward that the decision maker receives depending on the state they occupy at time ***T+1***. The terminal reward is the reward the decision maker receives in the terminal state as follows: at the ***T***-th period, the decision maker takes an action, receives an instantaneous reward determined by ***R*** and then transitions to a terminal state, where they receive the terminal reward. No actions are taken in the terminal state, and no reward from ***R*** is received. The default for ***h*** is a vector of zeros. To elect for the default value to be used, do not enter a value or enter the empty set [] for this input.
verbose	(Optional) Entering ***verbose*** as true displays to the command window selected output from each iteration, and whether convergence occurred or the maximum number of iterations was reached. The default value of ***verbose*** is true. To elect for the default value to be used, do not enter a value or enter the empty set [] for this input. Setting ***verbose*** to false will increase the speed of the algorithm.

Output Arguments

Up to five output arguments can be requested from RRbackwardinduction. These variables are described below.

Out	A structure array containing fields for all the output arguments described in this table as well as: • ***desc***: the title of the model (a string) • ***algorithm***: the type of algorithm used to find the solution (a string, in this case: 'backward induction') • ***Input*** (see Input Arguments above). This field is non empty only when an Input structure is used in RRbackwardinductione.g. RRbackwardinduction(Input). • ***date***: the time and date the backward induction algorithm was run (a date string). • ***note1***: empty. Can be filled in later with notes. • ***note2***: empty. Can be filled in later with notes.

	The values in Out can be retrieved the same way values can be retrieved from any structure array in MATLAB®. For example, the syntax: • Out.Vwill provide the value function • Out.Inputwill provide the structure array that was used to run RRvalueiteration if a structure array was used
V	S x (T+1) vector representing the value function. The columns of ***V*** represent the time periods, while the rows of ***V*** represent the state. The element in the s-th row and t-th column of ***V*** represents the maximum value of the decision maker's objective function given the decision maker is in the s-th state and t-th time period.
policy	S x T vector representing the optimal policy. The columns of ***policy*** represent the time periods, while the rows of ***policy*** represent the state. The element in the s-th row and t-th column of ***policy*** represents the optimal action for the decision maker when they are in the s-th state and t- th time period.
calculationtime	The number of seconds for which the algorithm ran.

Example

Using the following inputs to define a sequential decision problem, and the following syntax to run RRbackwardinduction, the value function and optimal policy can be found:

```
P(:,:,1)=          [0.9     0.1;
                    0.9     0.1];

P(:,:,2)=       [0.8       0.2;
                  0.8       0.2];

P(:,:,3)=       [0.7       0.3;
                  0.7       0.3];

R =[10          8       4;
      50        25      15];

beta = 0.9;

T = 3;

[Out, V, policy] = RRbackwardinduction(P,R,beta,T)

V =

    36.6920    24.2000    10.0000         0
    75.2360    62.6000    50.0000         0
```

```
policy  =

     2     2     1
     1     1     1
```

In the example, the value of ***policy*** indicates that if the decision maker is in the first state the decision maker's best action is to play the second action, and if the decision maker is in the second state the decision maker's best action is to play the first action, except when it is the last period, when the decision maker should play the first action no matter what state they are in.

The value of ***V*** indicates that, for example, if the decision maker's initial state is the first state, following the optimal ***policy*** gives the decision maker an expected discounted stream of rewards of 36.69. Similarly, if the decision maker's initial state is the second state, following the optimal ***policy*** gives the decision maker an expected discounted stream of rewards of 75.24.

See Also

RRvalueiteration| RRcheckinputs| RRtable

RRfindinvariantdist

Finds the invariant distribution of a Markov chain.

Syntax

dist = RRfindinvariantdist(markovChain)
dist = RRfindinvariantdist(markovChain, options)

Description

RRfindinvariantdist finds the invariant distribution of a Markov chain. The invariant distribution of a Markov chain is also sometimes referred to as the "limiting distribution" or "stationary distribution." The invariant distribution of a Markov chain can be interpreted as the long run proportion of time spent in each state of the Markov chain.

Mathematically, an invariant distribution, $\pi\pi$, satisfies: $\pi\pi = \pi$, where $\pi\pi$ is the (transition) matrix representation of the Markov chain and $\pi\pi$ is a row vector.

dist = RRfindinvariantdist(markovChain)
returns the invariant distribution, ***dist***, of ***markovChain***.

dist = RRfindinvariantdist(markovChain, options)
returns the invariant distribution, ***dist***, of ***markovChain*** using optional parameters ***options***. In ***options***, you can choose between two methods of finding the invariant distribution.

Input Arguments

Let ***S*** be the number of states and ***A*** be the number of actions.

markovChain	(Required) ***markovChain*** (S x S matrix) is the (transition) matrix representation of a Markov chain. The (i,j) element of ***markovChain*** is the probability that the system moves from the i-th state at time t to the j-th state in time t+1. For example, the (3,4) element of ***markovChain*** represents the probability of moving from the third state at time t to the fourth state at time t+1. An easy way to remember this is to remember that the rows of ***markovChain*** represent the state at time t, and the columns represent the state at time t+1. ***markovChain*** needs to be "irreducible" meaning that there are no subset of states that cannot be reached by another subset of states, e.g. ***markovChain*** should not contain absorbing states. *N.B. markovChain needs to be a 2 dimensional matrix, not a 3-dimenional matrix like the transition matrix that is used by RRvalueiterationor RRpolicyiteration.*
options	(Optional) ***options*** is a structure array that specifies optional parameters. The method field of ***options*** can be set to either 'direct' or 'power' to choose either the 'direct' method or 'power' method of finding the invariant distribution. If the power method is selected, the number of iterations for the power method can also be set in the iterations field of ***options***. For example, to set the method to 'direct', you can use the

	following code: options.method = 'direct'; Alternatively, to set the method to 'power' with 1000 iterations, you can use the following code: options.method = 'power'; options.iterations = 1000; The default is the 'power' method with 500 iterations.

Output Arguments

dist	1 x S vector representing the invariant distribution. The element in the s-th column of dist can be interpreted as the long run proportion of time that the Markov chain is in the s-th state.

Example

Using the following inputs, and the following syntax to run RRfindinvariantdist, the stationary distribution of the Markov chain can be found.

```
markovChain =  [0.8          0.2;
                    0.9      0.1];

options.method = 'direct';
dist = RRfindinvariantdist(markovChain,options) dist =

    0.8182        0.1818
```

See Also

RRpolicyiteration| RRcheckinputs| graphdist

RRpolicyiteration

Solves a sequential decision problem using policy iteration.

Syntax

Out = RRpolicyiteration(Input) *Recommendedsyntax [...] = RRpolicyiteration(Input)
[...] = ...
RRpolicyiteration(P,R,beta,policy0,maxiter,policyevalmethod,verbose) [Out,V,policy,iterations,calculationtime] = RRpolicyiteration(...)

Description

RRpolicyiterationsolves, using policy iteration, discrete-time infinite-horizon sequential decision problems where the decision maker has the objective of maximizing their expected discounted reward. The calculations used in this algorithm are described in Appendix B.

[...] = RRpolicyiteration(Input)
solves a sequential decision problem defined by ***Input***, which must be a structure array with fields for at least ***P***, ***R*** and ***beta***. Input can also contain fields for ***policy0***, ***maxiter***, ***policyevalmethod*** and ***verbose***, but these are not required.

[...] = RRpolicyiteration(P, R, beta, policy0, maxiter, policyevalmethod, verbose)
solves the sequential decision problem defined by ***P***, ***R*** and ***beta*** with the optional parameters, ***policy0***, ***maxiter***, ***policyevalmethod*** and ***verbose***.

[Out, V, policy, iterations, calculationtime] = RRpolicyiteration(...) solves a sequential decision problem and returns: ***Out***, a structure array containing fields for ***V***, ***policy***, ***iterations***, ***calculationtime***, and ***Input***; the value function, ***V***; optimal policy, ***policy***; the number of iterations required to reach the solution, ***iterations***; and the total computational time in seconds, ***calculationtime***.

Input Validation

Before running the policy iteration algorithm RRpolicyiterationvalidates its inputs. If an input error is found, policy iteration is stopped and messages identifying the error(s) are displayed in the command window.

Tips

An optional input can be entered only if all previous optional inputs have been entered. Optional inputs can be skipped by entering [] in place of the optional input, e.g. users can skip entering a value for ***policy0*** by using the following syntax:

[...] = RRpolicyiteration(P, R, beta, [], maxiter, policyevalmethod)

If only a subset of the output arguments is required, the arguments you do not require can be skipped by entering ~ in its place. For example, the following syntax can be used to skip ***V***, iterations and ***calculationtime***, but still request Out and ***policy***:

[Out,~,policy,~,~] = RRpolicyiteration(P, R, beta)

Input Arguments

Let S be the number of states and A be the number of actions.

Input	A structure array that contains fields for at least ***P***, ***R*** and ***beta*** and their values. Input can also contain optional fields for ***policy0***, ***maxiter***, ***policyevalmethod*** and ***verbose*** and their values.
P	(Required) ***P***, called the transition matrix, is the matrix representation of the state transition function. ***P*** can be entered as either a matrix or cell array. For many users, using ***P*** as a matrix will suffice; however, creating ***P*** as a cell array when it will contain many zero entries can improve computation time. When ***P*** is entered as a matrix, it must be an S x S x A matrix. The (i,j,k) element of ***P*** is the probability that the decision maker moves from the i-th state at time t to the j-th state in time t+1 when the k-th action is taken by the decision maker at time t. For example, the (3,4,2) element of ***P*** represents the probability of moving from the third state at time t to the fourth state at time t+1 when the decision maker chooses the second action at time t. An easy way to remember this is to remember that the rows of ***P*** represent the state at time t, the columns represent the state at time t+1, and the third dimension of ***P*** represents the actions. If ***P*** is entered as a cell array, it needs to be 1 x A, where each cell contains an S x S matrix that is possibly sparse. Each cell of the cell array represents a different action (the k-th cell represents the k-th action). Similar to when ***P*** entered as a matrix, the rows of each S x S matrix represent the state at time t, while the columns represent the state at time t+1. If there are many zeros in the transition matrix, entering ***P*** as a cell array with sparse S x S matrices can improve computation time.
R	(Required) ***R***, called the reward matrix, is the matrix representation of the reward function. ***R*** can be entered as either a matrix or cell array. For many users, using ***R*** as a matrix will suffice; however, creating ***R*** as a cell array when it will contain many zero entries can improve computation time. If ***R*** is entered as a matrix, it can be either an S x A or S x S x A matrix; the choice will depend on the type of reward function being modeled. When ***R*** is S x A, the (i,j) element of the reward matrix represents the immediate reward that the decision maker will receive at time t given the decision maker is in the i-th state at time t and chooses the j-th action at time t. When ***R*** is an S x S x A matrix, the interpretation is slightly different. The (i,j,k) element of ***R*** represents the immediate reward that the decision maker will receive at time t given the decision maker is in the i-th state at time t, moves to the j-th state at time t+1 and the k-th action is played by the decision maker at time t. When ***R*** is an S x A matrix, it can be sparse, however it cannot be sparse when it is an S x S x A matrix. In the latter case if you require a sparse

	matrix create ***R*** as a cell array. If ***R*** is entered as a cell array, it needs to be 1 x A, where each cell contains an S x S matrix that is possibly sparse. Each cell of the cell array represents the action. Similar to previously, the rows of each S x S matrix (possibly sparse) represent the state at time t, while the columns represent the state at time t+1. If there are many zeros in the reward matrix, entering ***R*** as a cell array with sparse S x S matrices can improve computation time.
beta	(Required) ***beta*** is the discount factor and must be a number strictly greater than 0 and no greater than 1. (Caution: convergence of the algorithm may not occur when the discount factor is set to 1).
policy0	(Optional) ***policy0*** (S x 1) is the initial value of the policy vector that is iterated on during policy iteration. The default for ***policy0*** is the policy from the solution to one application of the Bellman operator on ***V0*** as a vector of zeros. To elect for the default value to be used, do not enter a value or enter the empty set [] for this input.
maxiter	(Optional) Entering ***maxiter*** stops the algorithm if convergence has not occurred when ***maxiter*** iterations are reached. The default value of ***maxiter*** is 5000. To elect for the default value to be used, do not enter a value or enter the empty set [] for this input.
policyevalmethod	(Optional) entering ***policyevalmethod*** = 0 specifies that the "policy evaluation" step in the policy iteration algorithm is performed using Gaussian elimination with partial pivoting. Entering ***policyevalmethod*** = 1 specifies that the "policy evaluation" step is performed using the Jacobi method. The default value for ***policyevalmethod*** is 0. To elect for the default value to be used, do not enter a value or enter the empty set [] for this input.
verbose	(Optional) entering ***verbose*** as true displays to the command window output from each iteration, and whether convergence occurred or the maximum number of iterations was reached. The default value of ***verbose*** is true. To elect for the default value to be used, do not enter a value or enter the empty set [] for this input. Setting ***verbose*** to false will increase the speed of the algorithm.

Output Arguments

Up to five output arguments can be requested from RRpolicyiteration. These variables are described below.

Out	A structure array containing fields for all the output arguments described in this table as well as: • ***desc***: the title of the model (a string) • ***resultstable***: a table of the results (a cell array) • ***table***: a table of results using MATLAB's new tablefunction

	(MATLAB version R2013b or later only) • ***algorithm***: the type of algorithm used to find the solution (a string, either 'value function iteration' or 'policy iteration') • ***Input*** (see Input Arguments above). This field is non-empty only when an Input structure is used in RRvalueiteratione.g. RRvalueiteration(Input). • ***date***: the time and date the policy iteration algorithm was run (a date string). • ***note1***: empty. Can be filled in later with notes. • ***note2***: empty. Can be filled in later with notes. The values in Out can be retrieved the same way values can be retrieved from any structure array in MATLAB®. For example, the syntax: • Out.Vwill provide the value function • Out.Inputwill provide the structure array that was used to run RRvalueiteration if a structure array was used
V	S x 1 vector representing the value function, which along with the optimal policy forms the solution to a sequential decision problem. The element in the s-th row of ***V*** represents the maximum value of the decision maker's objective function given the s-th state is the first state the decision maker is in.
V_sa	S x A matrix representing the value of being in a specific state, represented by the row, and taking the action represented by the column, assuming that the optimal policy is followed in all future time periods.
policy	S x 1 vector representing the optimal policy, which along with the value function forms the solution to a sequential decision problem. The s-th element of ***policy*** represents the optimal action for the decision maker when they are in the s-th state.
iterations	The number of iterations of the value function that occurred before the solution was found.
calculationtime	The number of seconds for which the algorithm ran.

Examples

Using the following inputs to define a sequential decision problem, and the following syntax to run RRpolicyiteration, the value function and policy can be found:

```
P(:,:,1)=       [0.9     0.1;
                 0.9     0.1];

P(:,:,2)=     [0.8       0.2;
               0.8       0.2];

P(:,:,3)=     [0.7       0.3;
               0.7       0.3];
```

```
R =[10      8      4;
    50      25     15];

beta = 0.9;


[Out, V, policy] = RRpolicyiteration(P,R,beta)

V =

          75.44
         113.97


policy =

           2.00
           1.00
```

In the example, the value of ***policy*** indicates that if the decision maker is in the first state the decision maker's best action is to play the second action, and if the decision maker is in the second state the decision maker's best action is to play the first action.

The value of V indicates that if the decision maker starts in the first state and follows the optimal policy, the decision maker can expect a discounted stream of rewards worth 75.44. Similarly, if the decision maker's initial state is the second state, following the optimal policy gives the decision maker an expected discounted stream of rewards of 113.97.

See Also

RRvalueiteration | RRcheckinputs | RRtable

RRoptimalpolicymc

Finds a Markov chain that represents how a sequential decision problem transitions from state to state, assuming the decision maker follows the prescribed policy.

Syntax

markovChain = RRoptimalpolicymc(P, policy)

Description

RRoptimalpolicymc finds a Markov chain that represents how a sequential decision problem transitions from state to state if the decision maker follows the prescribed policy. This can be useful in calculating the long run proportion of time the decision maker spends in each state in a sequential decision problem (see example below).

markovChain = RRoptimalpolicymc(P, policy)
finds a Markov chain, ***markovChain***, that represents how a sequential decision problem transitions from state to state if the decision maker takes the actions prescribed by ***policy***.

Input Arguments

Let S be the number of states and A be the number of actions.

P	(Required) ***P***, called the transition matrix, is the matrix representation of the state transition function. ***P*** can be entered as either a matrix or cell array. For many users, using ***P*** as a matrix will suffice; however, creating ***P*** as a cell array when it will contain many zero entries can improve computation time. When ***P*** is entered as a matrix, it must be an S x S x A matrix. The (i,j,k) element of ***P*** is the probability that the decision maker moves from the i-th state at time t to the j-th state in time t+1 when the k-th action is taken by the decision maker at time t. For example, the (3,4,2) element of ***P*** represents the probability of moving from the third state at time t to the fourth state at time t+1 when the decision maker chooses the second action at time t. An easy way to remember this is to remember that the rows of ***P*** represent the state at time t, the columns represent the state at time t+1, and the third dimension of ***P*** represents the actions. If ***P*** is entered as a cell array, it needs to be 1 x A, where each cell contains an S x S matrix that is possibly sparse. Each cell of the cell array represents a different action (the k-th cell represents the k-th action). Similar to when ***P*** entered as a matrix, the rows of each S x S matrix represent the state at time t, while the columns represent the state at time t+1. If there are many zeros in the transition matrix, entering ***P*** as a cell array with sparse S x S matrices can improve computation time.
policy	(Required) S x 1 vector representing the a policy. The s-th element of ***policy*** represents the action the decision maker takes when they are in the s-th state.

Output Arguments

markovChain	S x S matrix is the matrix representation of a Markov chain that respresents how a sequential decision problem transitions from state to state if the decision maker follows the actions prescribed by policy. The (i,j) element of ***markovChain*** is the probability that the system moves from the i-th state at time t to the j-th state in time t+1. For example, the (3,4) element of ***markovChain*** represents the probability of moving from the third state at time t to the fourth state at time t+1. An easy way to remember this is to remember that the rows of ***markovChain*** represent the state at time t, and the columns represent the state at time t+1.

Example

In the following example, a sequential decision problem is created and solved. Then using the optimal policy from the solution to the sequential decision problem and the transition matrix, the Markov chain is found that respresents how a sequential decision problem transitions from state to state if the decision maker follows the actions prescribed by policy.

```
P(:,:,1)    =   [0.9          0.1;
                 0.9          0.1];

P(:,:,2)    =  [0.8         0.2;
                0.8         0.2];

P(:,:,3)    =  [0.7         0.3;
                0.7         0.3];

R = [10          8        4;
     50         25       15];

beta = 0.9;

[~, ~, policy] = RRvalueiteration(P,R,beta) policy =

              2.00
              1.00

  markovChain  =  RRoptimalpolicymc(P,policy)

  markovChain  =

       0.8000       0.2000
       0.9000       0.1000
```

After finding the Markov chain, you can find the long run proportion of time the system spends in each state by finding the invariant distribution:

```
dist = RRfindinvariantdist(markovChain) dist =

    0.8182    0.1818
```

See Also

RRfindinvariantdist| graphdist| RRpolicyiteration

generalchart

Creates and displays a customizable bar chart (but may be updated in future to supply other chart types).

Syntax

[bars,f] = generalchart(barHeights, 'ParameterName', 'ParameterValue')

Description

generalchart will display a bar chart of the values in the input vector barHeights with customizable orientation, order, colors for each bar, transparency of all bars, plot title, x and y labels, x label rotation, and bar value text location and format.

generalchart(barHeights)
will display a generalchart with of the vaues stored in barHeights. The corresponding values to two decimal points will be displayed above each bar, and the bar colors will rotate through five colors: Supported Intelligence blue, red, green, yellow, and magenta.

generalchart(barHeights, 'labels', labels)
will display a label (stored in labels) under or alongside each bar. The order of labels must match the order of barHeights.

generalchart(barHeights, 'colors', colors)
specifies the bar color for each value in barHeights. generalchart will cycle through the colors in colorsif fewer than the number of values in barHeightsare specified.

generalchart(barHeights, 'transparency', transparency)
specifies the transparency of all bars. A value of 1 corresponds to opaque.

generalchart(barHeights, 'title', title)
sets the title of the plot to the string stored in title.

generalchart(barHeights, 'xlabel', xlabel)
sets the x-axis label of the plot to the string stored in xlabel.

generalchart(barHeights, 'ylabel', ylabel)
sets the y-axis label of the plot to the string stored in ylabel.

generalchart(barHeights, 'orientation', orientation)
sets the orientation (either horizontal or vertical) of the plot to the string stored in orientation.

generalchart(barHeights, 'order', order)
specifies the order (descending, ascending, or same) for the values in barHeights to be plotted to the string stored in order.

generalchart(barHeights, 'textlocation', textlocation)
specifies the location (above, inside, or none) of the text displaying the value corresponding to each bar relative to the bar to the string stored in textlocation.

generalchart(barHeights, 'textformat', textformat)
sets the format of the text specifying the value corresponding to each bar to the string stored in textformat, which will be interpreted by sprintf.

generalchart(barHeights, 'xtickrotation', xtickrotation) specifies the angle in degrees to rotate the bar labels on the x-axis to number stored in xtickrotation.

Input Argument

barHeights	A vector of values to display on the bar chart.

Optional Parameters

'labels'	A cell array of string labels for each value in barHeights.
'colors'	A cell array of color identifiers for each value in barHeights.
'transparency'	The transparency level for all bars.
'title'	The title of the bar chart.
'xlabel'	The x axis label.
'ylabel'	The y axis label.
'orientation'	The orientation of the bar chart. Available options are 'vertical' and 'horizontal'. Default value is 'vertical'.
'order'	The order in which to display the values in barHeights on the chart. Available options are 'ascend', 'descend' and 'same'. Default value is 'same'.
'textlocation'	The location of the value text for each bar relative to the bar in the bar chart. Available options are 'above', 'inside', and 'none'. Default value is 'above'.
'textformat'	String format used as an input to sprint. Default value is '%.2f', corresponding to a two decimal place number.
'xtickrotation'	The angle in degrees to rotate the bar labels. Default value is 0.

Output Arguments

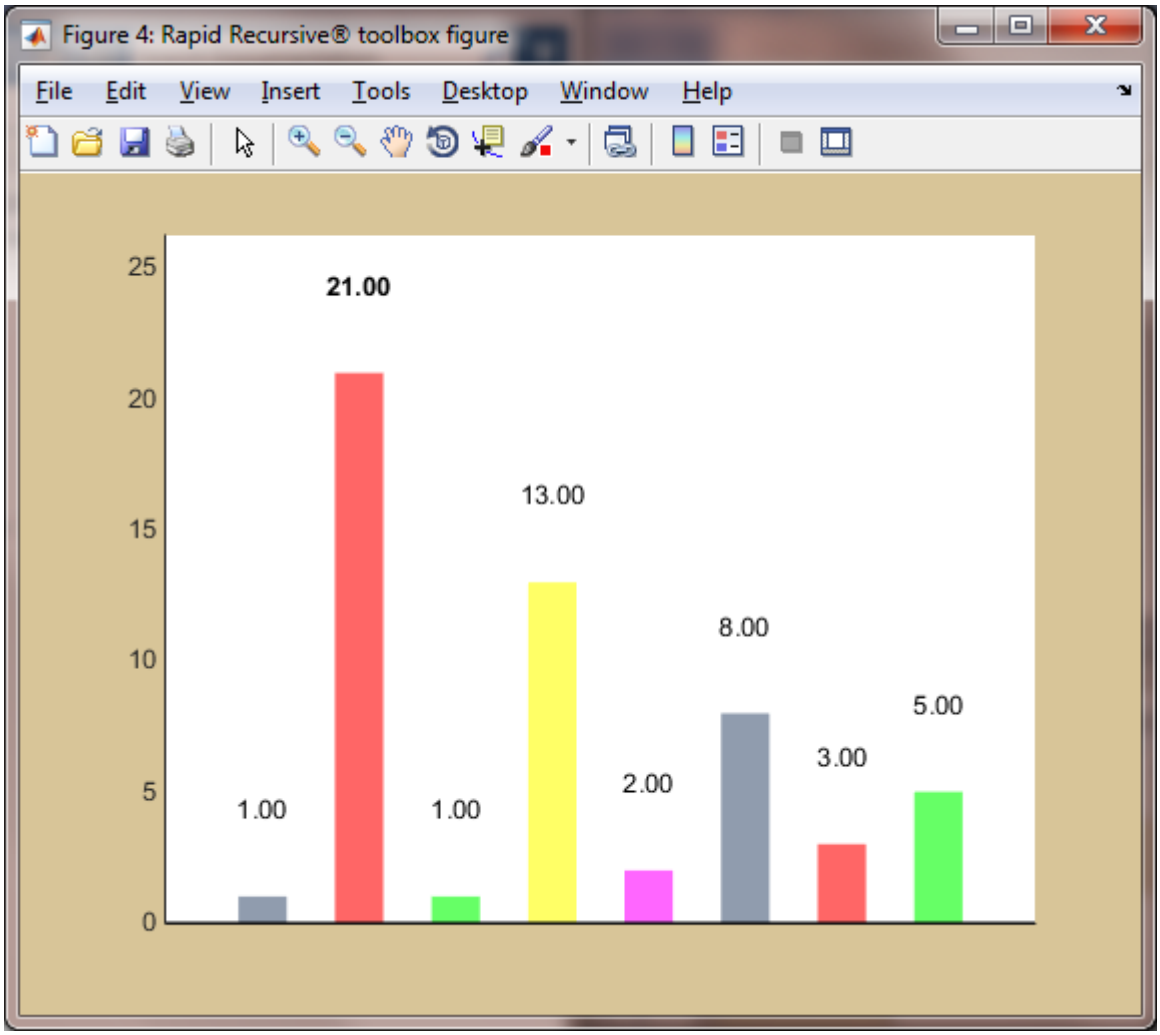
bars	A cell array of bar objects for each value in barHeights.

f	The figure on which the bar chart is displayed.

Examples

barHeights = [1,21,1,13,2,8,3,5];

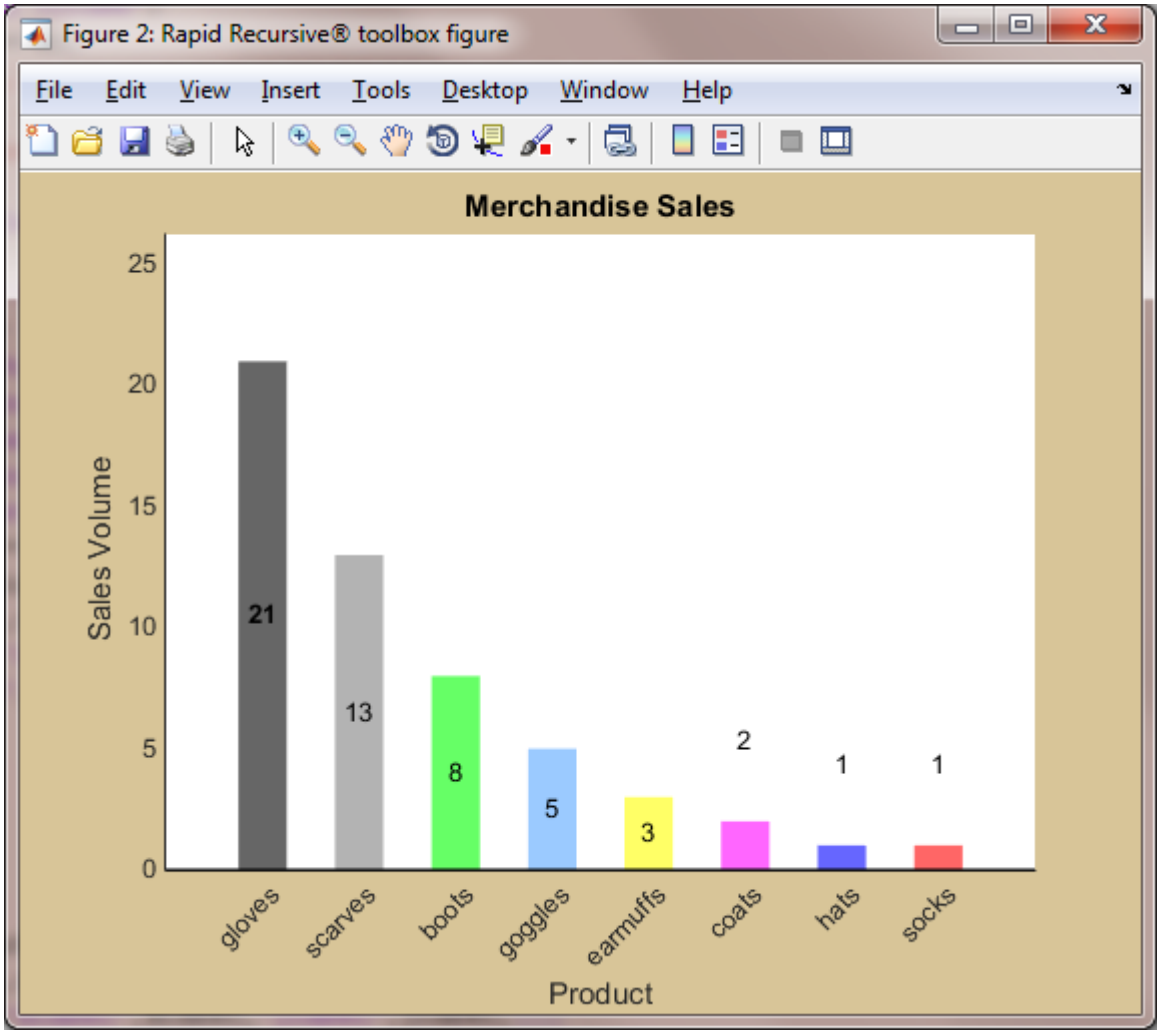
generalchart(barHeights);



Custom rotated labels, colors, title, x and y axis labels, and descending order:

```
labels = {'hats','gloves','socks','scarves','coats','boots','earmuffs',... 'goggles'};
colors = {'b','k','r',[0.5,0.5,0.5],'m','g','y',[0.35,0.65,1]};
textlocation = 'inside';
textformat = '%.0f';
order = 'descend';
title = 'Merchandise Sales';
xlabel = 'Product';
ylabel = 'Sales Volume'; xtickrotation = 45;
```

generalchart(barHeights,'labels',labels,'colors',colors,...
'textlocation',textlocation,'textformat',textformat,'order',order,... 'title',title,'xlabel',xlabel,'ylabel',ylabel,...
'xtickrotation',xtickrotation);



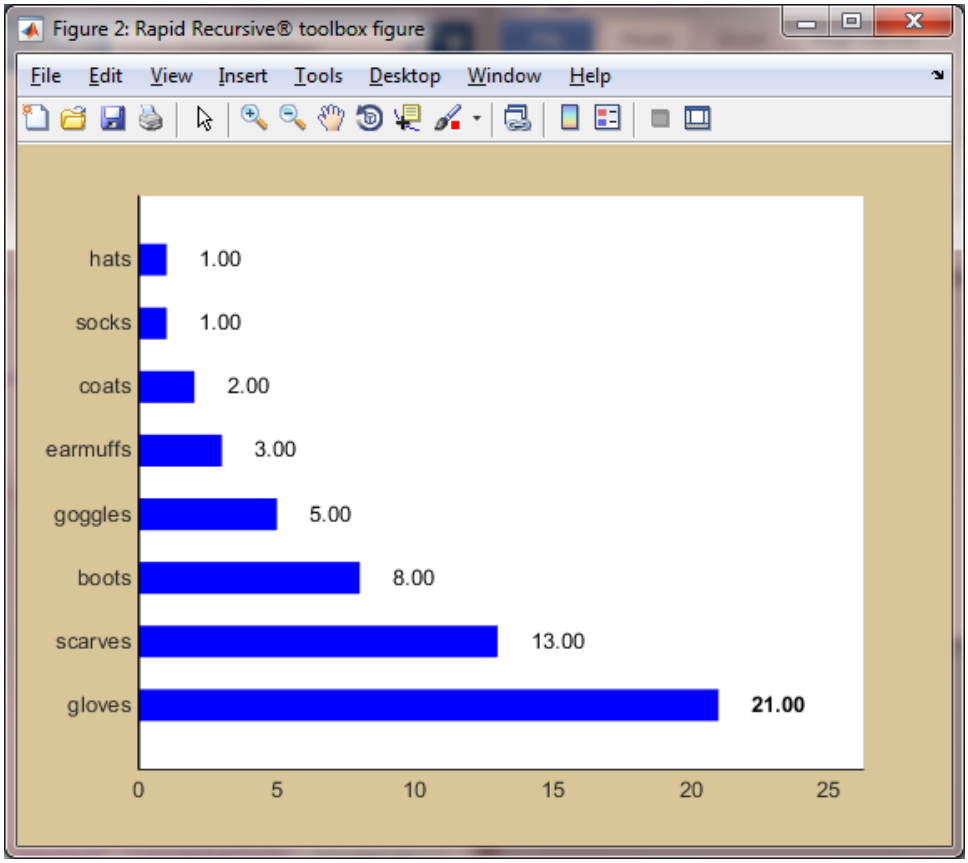
Horizontal Bar Chart with labels, custom transparency, single color, and ascending order

transparency = 1;

orientation = 'horizontal';

colors = 'blue';
order = 'ascend';

[f,bars] =
generalchart(barHeights,'colors','blue','order','ascend','orientation',orientation,'labels',labels,'transparency',transparency)



RRlikelypath

Using the solution of a Rapid Recursive® model, project forward a likely path assuming that the most likely random events occur and the subject person follows the optimal policy every time.

Syntax

paths = RRlikelypath(Out) *Recommended Syntax

paths = RRlikelypath(Out, 'ParameterName', 'Parameter Value')

Description

The syntax above returns the likely evolution of states in ***Spath***, the likely evolution of rewards in ***Rpath,*** and the optimal policy in each time period in ***Ppath***.

Inputs

Out	Out structure, containing fields for: ***V, policy,*** and ***Input***; which contains fields for ***R*** and ***P***, as well as ***beta***

Required Fields

Out.V	S x 1 vector containing the value in each state for a solved decision problem.
Out.policy	S x 1 vector containing the optimal policy for each state in a solved decision problem.
Out.Input.R	S x A matrix containing the reward for each state-action pair in a decision problem.
Out.Input.P	S x S x A matrix containing transition probabilities for every state-action pair in a decision problem.

Optional Parameters

slist	A vector or cell array of selected states for which to calculate a likely path. By default if not entered, the first three states form the *slist*. All states are included in the analysis of possible movements, but only the *slist* starting states are calculated.
alternate	Dictates the handling of states where multiple transitions are more than 30% likely. Possible values are: 0: no alternating. The first most likely transition

	is returned every time period. 1: Alternate Scheme 1. If the likelihood of the second most likely transition is greater than or equal to 30% and the time index is 3, 6, or 9, the second most likely transition is returned. 2: Alternate scheme 2. If the likelihood of the second most likely transition is greater than or equal to 30% and the path is already in the most likely state, the second most likely transition is returned.
T	An integer representing the desired number of periods to stimulate forward. Default = 10

Output

paths	A structure containing the following fields: • **slist:** a row vector containing the indices of all starting states. • **Spath:** an NxT matrix of state paths, each row representing the state evolution across time for a particular starting state and the actions given in Ppath. • **Rpath:** an NxT matrix of rewards assuming the states and actions given by Spath and Ppath respectively. • **Ppath:** an NxT matrix of actions assuming the state evolutions given by Spath.

Examples

After running the RentalPropertyManagement template:

paths = RRlikelypath(Out) paths =

slist: [1.00 2.00 3.00]
Spath: [3x10 double] Rpath:
[3x10 double] Ppath: [3x10
double]

paths2 = RRlikelypath(Out, 'slist', [5 6 9]) paths2 =

slist: [1.00 5.00 6.00 9.00]
Spath: [4x10 double] Rpath:
[4x10 double] Ppath: [4x10
double]

disppaths(paths2)

Starting in State: 5

	State	Policy	Reward
Period 1	5	2	'$316.67'
Period 2	5	2	'$316.67'
Period 3	5	2	'$316.67'
Period 4	5	2	'$316.67'
Period 5	5	2	'$316.67'
Period 6	5	2	'$316.67'
Period 7	5	2	'$316.67'
Period 8	5	2	'$316.67'
Period 9	5	2	'$316.67'
Period 10	5	2	'$316.67'

Starting in State: 6

	State	Policy	Reward
Period 1	6	2	'$416.67'
Period 2	6	2	'$416.67'
Period 3	6	2	'$416.67'
Period 4	6	2	'$416.67'
Period 5	6	2	'$416.67'
Period 6	6	2	'$416.67'
Period 7	6	2	'$416.67'
Period 8	6	2	'$416.67'
Period 9	6	2	'$416.67'
Period 10	6	2	'$416.67'

Starting in State: 9

	State	Policy	Reward
Period 1	9	2	'$716.67'
Period 2	9	2	'$716.67'
Period 3	9	2	'$716.67'
Period 4	9	2	'$716.67'
Period 5	9	2	'$716.67'
Period 6	9	2	'$716.67'
Period 7	9	2	'$716.67'
Period 8	9	2	'$716.67'
Period 9	9	2	'$716.67'
Period 10	9	2	'$716.67'

RRgraphtransitions

Creates a directed graph representation of the transition matrix for a specific action, for a properly composed RR decision model that includes states, actions, and a transition matrix. Here, the states are the nodes, and the edges between the nodes represent probabilistic transitions among the states. Such a representation can be made for a selected action.

Syntax

[NodeTable, EdgeTable, h, p] = RRgraphtransitions(Input)

Description

[NodeTable, EdgeTable, h, p] = RRgraphtransitions(Input) [NodeTable, EdgeTable, h, p] = RRgraphtransitions(Input) [NodeTable, EdgeTable, h, p] = RRgraphtransitions(Input)

Input Arguments

Input	(Required) A structure summarizing a composed RR decision model, and containing the following non-empty fields: • ***S***, the number of states in the problem • ***statelabels***, a cell array containing a string representation of each state indicated by S • ***A***, the number of actions in the problem • ***actionlabels***, a cell array containing a string representation of each action indicated by A • ***P***, a valid transition matrix sized S x S x A
selectedframe	(Optional) An integer representing the number of the action for which the graph will be created. The default value is 1.
verbose	(Optional) A logical indicating whether the function should run with extended documentation printed to the screen (TRUE), or without (FALSE, default).

Output Arguments

Output	• ***NodeTable*** and ***EdgeTable***, summarizing the network representation of the state transitions. • ***H***, a figure handle to the graphical representation of the state transitions. • ***P***, a handle to the plot within the figure.

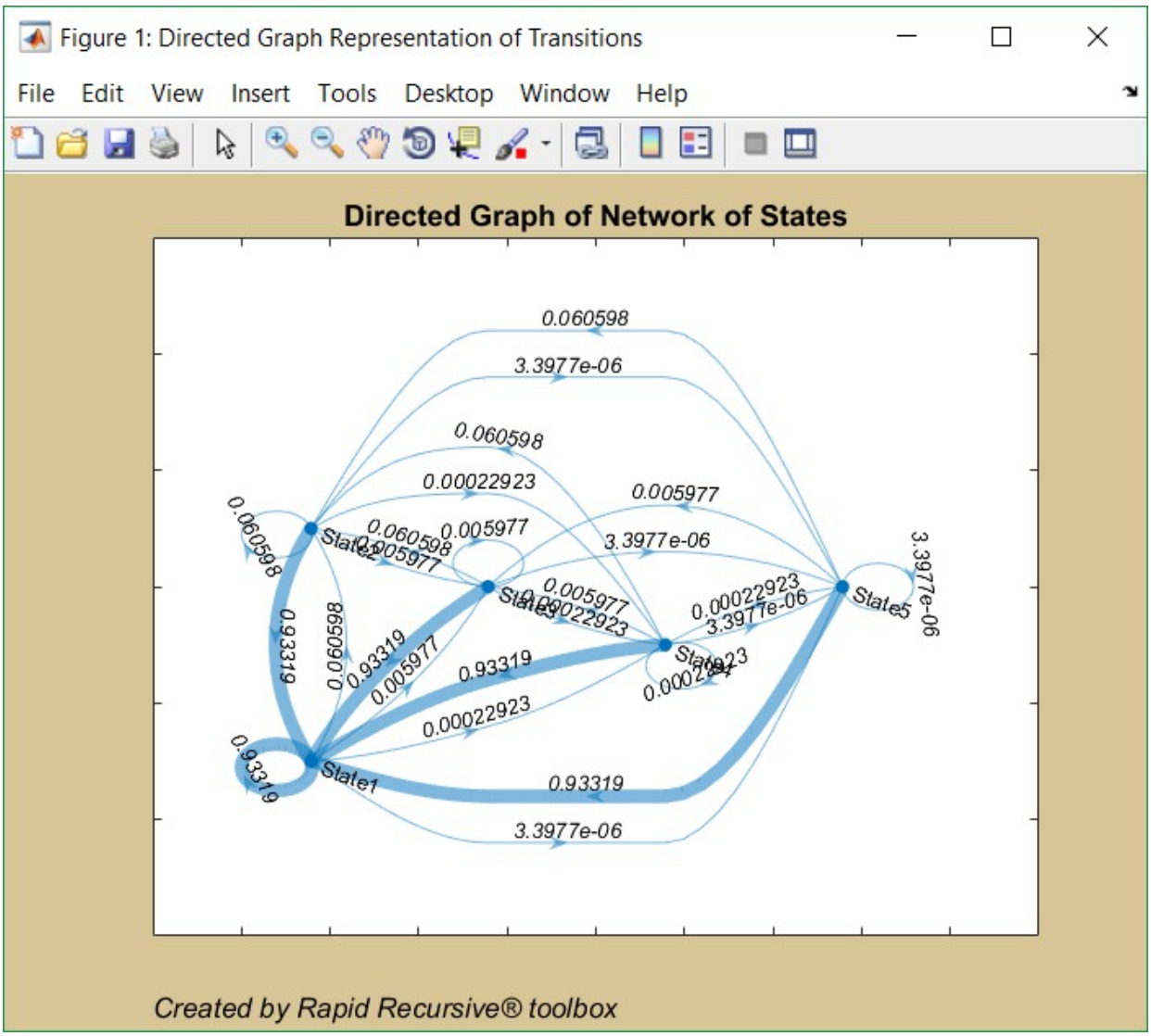
Examples

Input.S = 5;

Input.A = 2;

Input.P = repmat(RRcreatetransition([1:5], 'N'), 1, 1, 2); Input.actionlabels = {'Action1', 'Action2'};

Input.statelabels = {'State1', 'State2', 'State3', 'State4', 'State5'}; RRgraphtransitions(Input)



RRgraphlikelypath

Illustrates the likely path calculated for the solution of a Rapid Recursive® model, showing the likely state evolution, the optimal actions (policies), and expected rewards at each time period going forward. The likely path is generated assuming that the most likely random events occur and the subject person follows the optimal policy every time.

Syntax

[h1, h2, h3] = RRgraphlikelypath(paths)

[h1, h2, h3] = RRgraphlikelypath (paths, 'ParameterName',
'Parameter Value')

Description

The syntax above returns the handles to the graph of the likely state evolution in ***h1***, the graph of optimal actions in ***h2***, and the graph of expected rewards in ***h3.***

Inputs

paths	A structure containing the following fields: • ***slist*:** a row vector containing the indices of all starting states. • ***Spath*:** an NxT matrix of state paths, each row representing the state evolution across time for a particular starting state and the actions given in Ppath. • ***Rpath*:** an NxT matrix of rewards assuming the states and actions given by Spath and Ppath respectively. • ***Ppath*:** an NxT matrix of actions assuming the state evolutions given by Spath. This structure is the output of the RRlikelypath function.

Optional Parameters

statelabels	A cell array containing a string label for each state. A label should be provided for every state in the problem, not only those included in ***Spath***. Numeric labels will be used if this parameter is left blank
actionlabels	A cell array contianing a string label for each action. Numeric labels will be used if this parameter is left blank.
plots	A three-element vector indicating which plots to create. For example, [1 0 1] would create the

	state and reward plots, but not the policy plot.
handles	A three-element cell array containing valid axis handles on which the plots will be created. If you wish to have any of the plots created in a new figure window, simply enter [] in the corresponding element. By default each plot will be created in its own new figure window.

Outputs

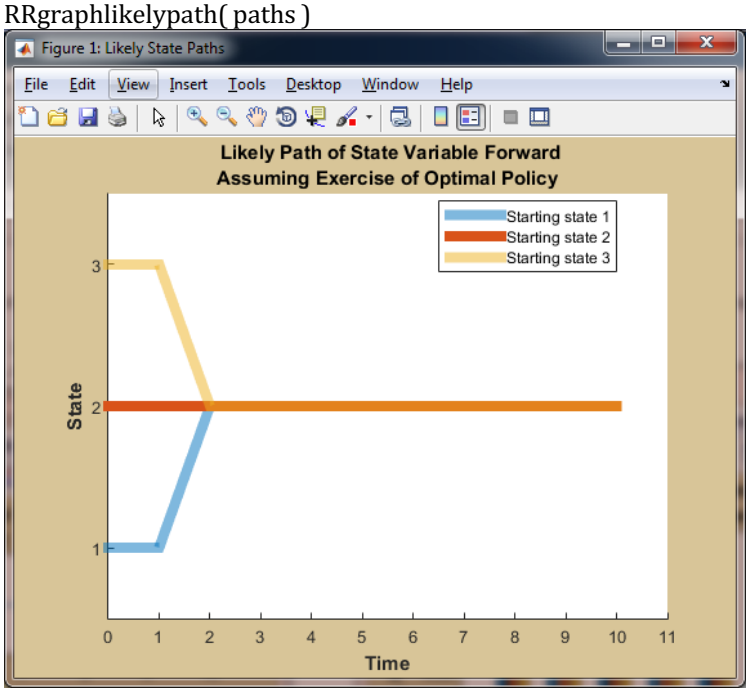
h1	Handle object for the graph of likely state evolution.
h2	Handle object for the graph of optimal actions.
h3	Handle object for the graph of expected future (undiscounted) rewards.

Examples

After running the RentalPropertyManagement template:

paths = RRlikelypath(Out, 'slist', 1:5); [h1, h2, h3] =

RRgraphlikelypath(paths)

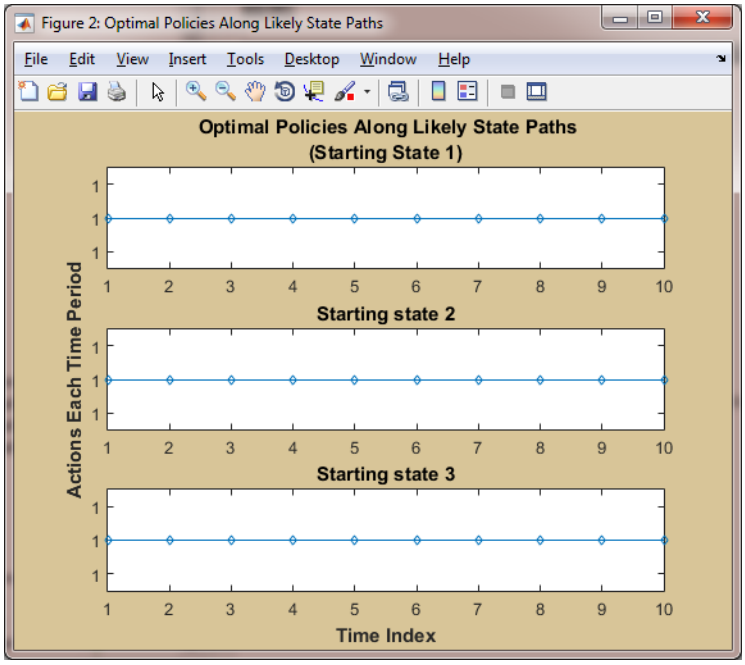


h1 =

Axes (Likely Path of State Variable Forward, Assuming Exercise of Optimal Policy) with properties:

```
            XLim: [0 11.00]
            YLim: [0.50 3.50]
          XScale:'linear' YScale:
          'linear'
    GridLineStyle: '-'
         Position: [0.13 0.11 0.78 0.78]
            Units: 'normalized'
```

Show all properties

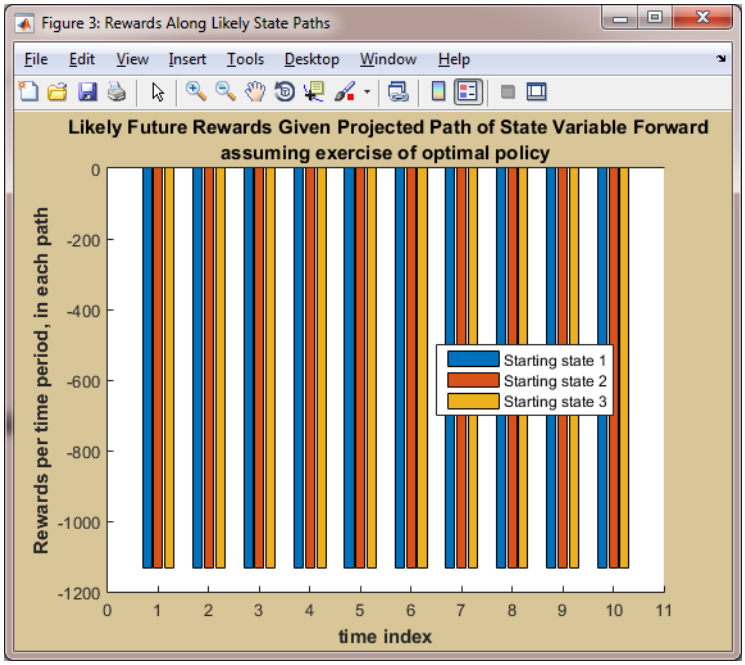


h2 =

Axes (Optimal Policies Along Likely State Paths, (Starting State 1)) with properties:

```
            XLim: [1.00 10.00]
            YLim: [0.70 1.30]
          XScale:'linear' YScale:
          'linear'
    GridLineStyle: '-'
         Position: [0.13 0.71 0.78 0.22]
            Units:'normalized' Show
```

all properties

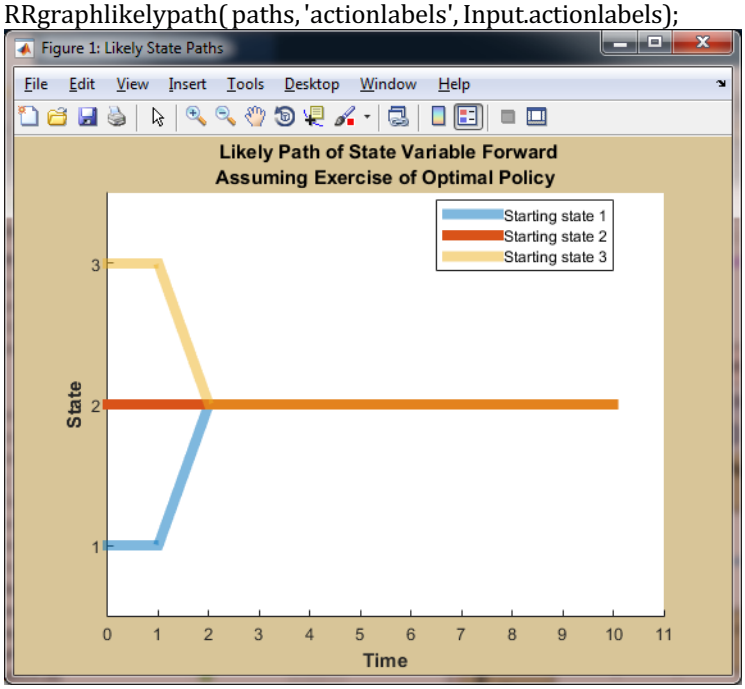


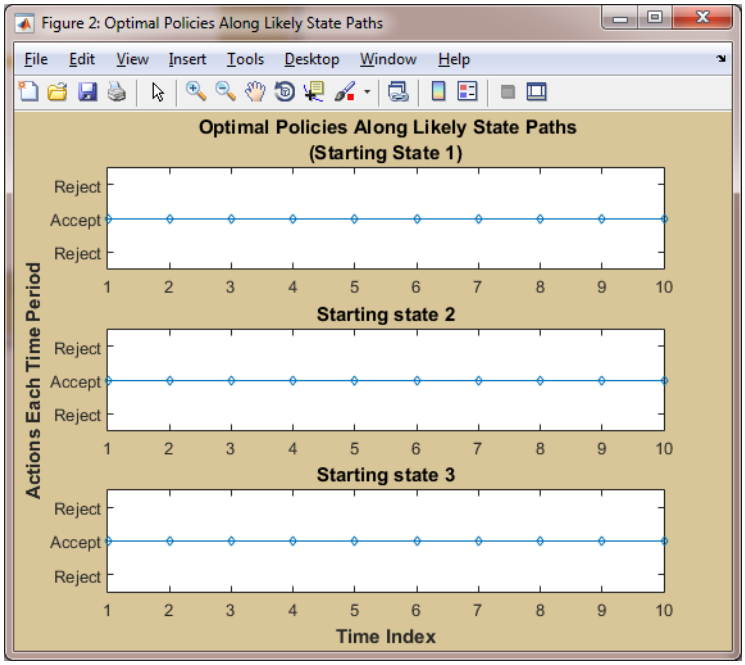
h3 =

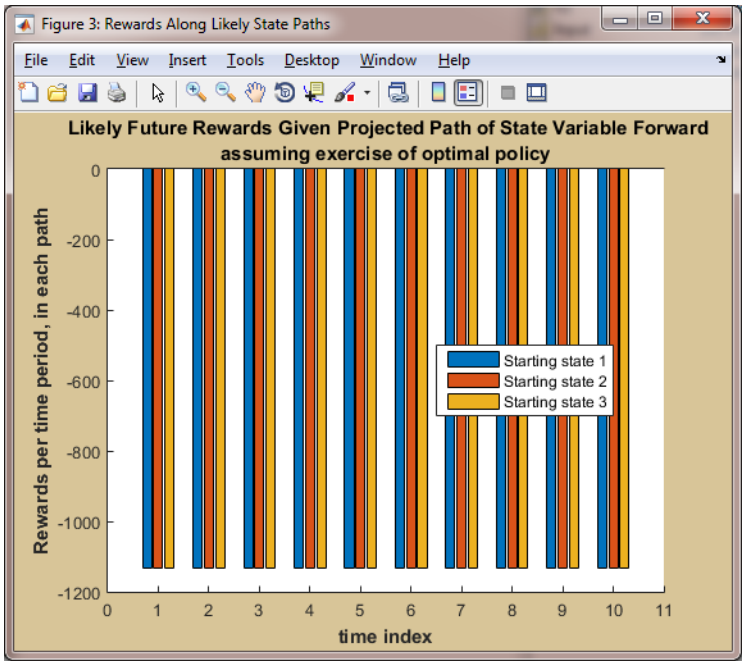
Axes (Likely Future Rewards Given Projected Path of State Variable Forward, assuming exercise of optim...) with properties:

XLim: [0 11.00]
YLim: [-1200.00 0]
XScale: 'linear' YScale: 'linear'
GridLineStyle: '-'
Position: [0.13 0.11 0.78 0.78]
Units: 'normalized'

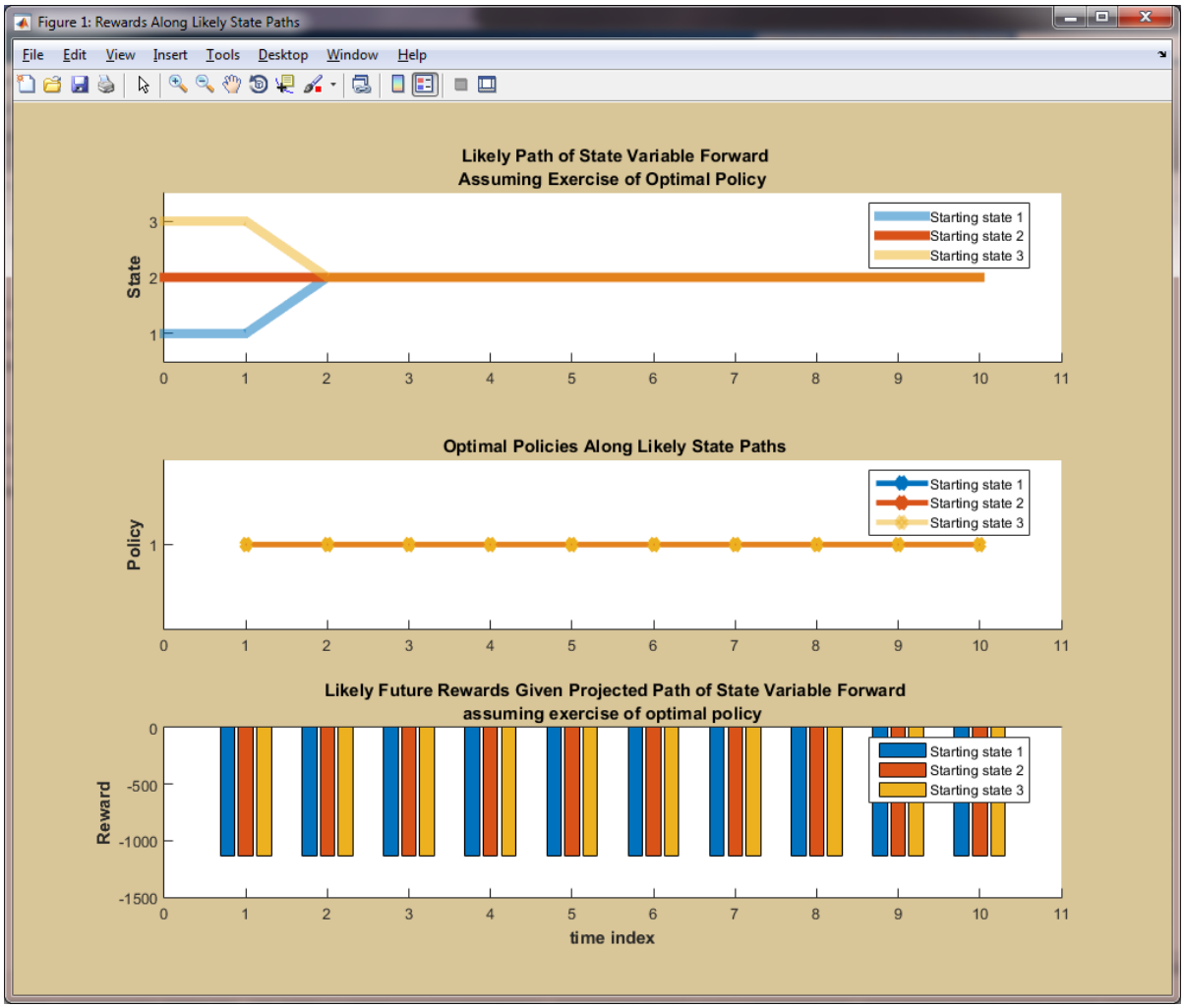
RRgraphlikelypath(paths, 'actionlabels', Input.actionlabels);







```
RRfigure;
ax1 = subplot(3, 1, 1);
ax2 = subplot(3, 1, 2);
ax3 = subplot(3, 1, 3);
RRgraphlikelypath(paths, 'handles', {ax1, ax2, ax3});
```



RRvalueiteration

Solves a sequential decision problem using value function iteration.

Syntax

Out = RRvalueiteration(Input) *Recommendedsyntax [...] = RRvalueiteration(Input)
[...] = RRvalueiteration(P, R, beta, epsilon, maxiter, V0, verbose) [Out,V,policy,iterations,calculationtime] = RRvalueiteration(...)

Description

RRvalueiteration solves, using value function iteration, discrete-time infinite-horizon sequential decision problems where the decision maker has the objective of maximizing their expected discounted reward. The calculations used in this algorithm are described in Appendix B.

[...] = RRvalueiteration(Input)
solves a sequential decision problem defined by ***Input***, which must be a structure array with fields for at least ***P***, ***R*** and ***beta***. ***Input*** can also contain fields for ***epsilon***, ***maxiter***, ***V0*** and ***verbose***, but these are not required.

[...] = RRvalueiteration(P, R, beta, epsilon, maxiter, V0, verbose) solves the sequential decision problem defined by ***P***, ***R*** and ***beta*** with the optional parameters, ***epsilon***, ***maxiter***, ***V0*** and ***verbose***.

[Out, V, policy, iterations, calculationtime] = RRvalueiteration(...) solves a sequential decision problem and returns: ***Out***, a structure array containing fields for ***V***, ***policy***, ***iterations***, ***calculationtime***, and ***Input***; the value function, ***V***; optimal policy, ***policy***; the number of iterations required to reach the solution, ***iterations***; and the total computational time in seconds, ***calculationtime***.

Input Validation

Before running the value function iteration algorithm RRvalueiterationvalidates its inputs. If an input error is found, value function iteration is stopped and messages identifying the errors are displayed in the command window.

Tips

An optional input can be entered only if all previous optional inputs have been entered. Optional inputs can be skipped by entering [] in place of the optional input, e.g. users can skip entering a value for ***epsilon*** by using the following syntax:

[...] = RRvalueiteration(P, R, beta, [], maxiter, V0)

If only a subset of the output arguments is required, the arguments you do not require can be skipped by entering ~in their place. For example, the following syntax can be used to skip ***V***, ***iterations*** and ***calculationtime***, but still request ***Out*** and ***policy***:

[Out,~,policy,~,~] = RRvalueiteration(P,R,beta)

Input Arguments

Let S be the number of states and A be the number of actions.

Input	A structure array that contains fields for at least ***P***, ***R*** and ***beta*** and their values. Input can also contain optional fields for ***epsilon***, ***maxiter***, ***V0*** and ***verbose*** and their values.
P	(Required) ***P***, called the transition matrix, is the matrix representation of the state transition function. ***P*** can be entered as either a matrix or cell array. For many users, using ***P*** as a matrix will suffice; however, creating ***P*** as a cell array when it will contain many zero entries can improve computation time. When ***P*** is entered as a matrix, it must be an S x S x A matrix. The (i,j,k) element of ***P*** is the probability that the decision maker moves from the i-th state at time t to the j-th state in time t+1 when the k-th action is taken by the decision maker at time t. For example, the (3,4,2) element of ***P*** represents the probability of moving from the third state at time t to the fourth state at time t+1 when the decision maker chooses the second action at time t. An easy way to remember this is to remember that the rows of ***P*** represent the state at time t, the columns represent the state at time t+1, and the third dimension of ***P*** represents the actions. If ***P*** is entered as a cell array, it needs to be 1 x A, where each cell contains an S x S matrix that is possibly sparse. Each cell of the cell array represents a different action (the k-th cell represents the k-th action). Similar to when ***P*** entered as a matrix, the rows of each S x S matrix represent the state at time t, while the columns represent the state at time t+1. If there are many zeros in the transition matrix, entering ***P*** as a cell array with sparse S x S matrices can improve computation time.
R	(Required) ***R***, called the reward matrix, is the matrix representation of the reward function. ***R*** can be entered as either a matrix or cell array. For many users, using ***R*** as a matrix will suffice; however, creating ***R*** as a cell array when it will contain many zero entries can improve computation time. If ***R*** is entered as a matrix, it can be either an S x A or S x S x A matrix; the choice will depend on the type of reward function being modeled. When ***R*** is S x A, the (i,j) element of the reward matrix represents the immediate reward that the decision maker will receive at time t given the decision maker is in the i-th state at time t and chooses the j-th action at time t. When ***R*** is an S x S x A matrix, the interpretation is slightly different. The (i,j,k) element of ***R*** represents the immediate reward that the decision maker will receive at time t given the decision maker is in the i-th state at time t, moves to the j-th state at time t+1 and the k-th action is played by the decision maker at time t. When ***R*** is an S x A matrix, it can be sparse, however it cannot be sparse when it is an S x S x A matrix. In the latter case if you require a sparse matrix create ***R*** as a cell array.

	If ***R*** is entered as a cell array, it needs to be 1 x A, where each cell contains an S x S matrix that is possibly sparse. Each cell of the cell array represents the action. Similar to previously, the rows of each S x S matrix (possibly sparse) represent the state at time t, while the columns represent the state at time t+1. If there are many zeros in the reward matrix, entering ***R*** as a cell array with sparse S x S matrices can improve computation time.
beta	(Required) ***beta*** is the discount factor and must be a number strictly greater than 0 and no greater than 1. (Caution: convergence of the algorithm may not occur when the discount factor is set to 1).
epsilon	(Optional) ***epsilon*** is the threshold for the maximum difference between the value function found by this algorithm and the true value function. ***epsilon*** must be strictly greater than 0. The default for ***epsilon*** is 0.01. To elect for the default value to be used, do not enter a value or enter the empty set [] for this input.
maxiter	(Optional) Entering ***maxiter*** stops the algorithm if convergence has not occurred when ***maxiter*** iterations are reached. The default value of ***maxiter*** is 5000. To elect for the default value to be used, do not enter a value or enter the empty set [] for this input.
V0	(Optional) ***V0*** is an S x 1 vector that serves as the starting point for value function iteration. The default is a vector of zeros. To elect for the default value to be used, do not enter a value or enter the empty set [] for this input.
verbose	(Optional) Entering ***verbose*** as true displays to the command window output from each iteration, and whether convergence occurred or the maximum number of iterations was reached. The default value of ***verbose*** is true. To elect for the default value to be used, do not enter a value or enter the empty set [] for this input. Setting ***verbose*** to false will increase the speed of the algorithm.

Output Arguments

Up to five output arguments can be requested from RRpolicyiteration. These variables are described below.

Out	A structure array containing fields for all the output arguments described in this table (***V***, ***policy*** etc) as well as: • ***desc***: the title of the model (a string) • ***resultstable***: a table of the results (a cell array) • ***table***: a table of results using MATLAB's new tablefunction (MATLAB version R2013b or later only) • ***algorithm***: the type of algorithm used to find the solution (a string, in this case 'value function iteration') • ***Input*** (see Input Arguments above). This field is non empty only when an Input structure is used in RRvalueiteratione.g.

	RRvalueiteration(Input). • ***date***: the time and date the value function iteration was run (a date string). • ***note1***: empty. Can be filled in later with notes. • ***note2***: empty. Can be filled in later with notes. The values in ***Out*** can be retrieved the same way values can be retrieved from any structure array in MATLAB®. For example, the syntax: • Out.Vwill provide the value function • Out.Inputwill provide the structure array that was used to run RRvalueiteration if a structure array was used
V	S x 1 vector representing the value function, which, along with the optimal policy, forms the solution to a sequential decision problem. The element in the s-th row of ***V*** represents the maximum value of the decision maker's objective function given the s-th state is the first state the decision maker is in.
V_sa	S x A matrix representing the value of being in a specific state, represented by the row, and taking the action represented by the column, assuming that the optimal policy is followed in all future time periods.
policy	S x 1 vector representing the optimal policy, which, along with the value function, forms the solution to a sequential decision problem. The s-th element of ***policy*** represents the optimal action for the decision maker when they are in the s-th state.
iterations	The number of iterations of the value function that occurred before the solution was found.
calculationtime	The number of seconds for which the algorithm ran.

Examples

Using the following inputs to define a sequential decision problem, and the following syntax to run RRvalueiteration, the value function and policy can be found:

```
P(:,:,1) =  [0.9     0.1;
             0.9     0.1];

P(:,:,2) = [0.8      0.2;
            0.8      0.2];

P(:,:,3) = [0.7      0.3;
            0.7      0.3];

R = [10     8     4;
     50     25    15];
```

```
beta = 0.9;

[Out, V, policy] = RRvalueiteration(P, R, beta)

V =

   75.44
  113.97


policy =

    2.00
    1.00
```

In the example, the value of ***policy*** indicates that if the decision maker is in the first state the decision maker's best action is to play the second action, and if the decision maker is in the second state the decision maker's best action is to play the first action.

The value of ***V*** indicates that if the decision maker's initial state is the first state, following the optimal policy gives the decision maker an expected discounted stream of rewards of 75.44. Similarly, if the decision maker's initial state is the second state, following the optimal policy gives the decision maker an expected discounted stream of rewards of 113.97.

See Also

RRpolicyiteration | RRcheckinputs | RRtable

convertdiscount

Converts the discount factor from an annual basis to the time index specified by the user.

Syntax

beta = convertdiscount(annualFactor, timeIndex)

Description

beta = convertdiscount(annualFactor, timeIndex)
will return the updated discount factor, ***beta.***

Note on Conversion: This function converts an annual discount factor into a discount factor for the specified time index according to the following formula:

$$\beta\beta = \sqrt[nn]{\frac{1+gg}{1+dd}}$$

where: β is the converted discount factor,
g is the annual growth rate,
d is the annual discount rate, and
n is the number of periods per year under the new time index.

Input Arguments

Input	An input structure from a Rapid Recursive® problem with fields for at least ***beta*** (the discount rate) and ***periodicity*** (the time index)
annualFactor	The annual discount factor for the problem
timeIndex	A string containing the time index for the problem. Supported time indices include: Year, Years, Yearly, Y Quarter, Quarters, Quarterly, Q Month, Months, Monthly, M Week, Weeks, Weekly, W Day, Days, Daily, D

Output

This function returns an updated discount factor, *beta*

Examples

```
beta = convertdiscount(0.364, 'M') beta =

    0.9192

beta = convertdiscount(0.364, 'D') beta =

    0.9972
```

displist

Displays a list in the command window.

Syntax

displist(list)

Description

displist(list)
displays to the command window the contents of each cell in the cell array list. The contents of each cell are displayed on a new line.

Input Argument

list	A cell array that is a vector. Each cell in list must contain a string or a scalar number.

Example 1

actionlabels = {'buy','sell','hold'}; displist(actionlabels)

```
 --List-- buy
      sell hold
  ------------
```

Example 2

statelabels = [1000:100:2000]; statelabels = num2cell(statelabels); displist(statelabels)

```
  --List--
     1000
     1100
     1200
     1300
     1400
     1500
     1600
     1700
     1800
     1900
     2000
  ------------
```

See Also

num2cell(a MATLAB® function) | RRtable

dispreward

Creates a table that contains labels for the states and actions next to the reward matrix.

Syntax

table = dispreward(R, statelabels, actionlabels)

Description

table = dispreward(R, statelabels, actionlabels)
creates a cell array ***table*** that contains the reward matrix ***R*** along with the labels for the states and actions, ***statelabels*** and ***actionlabels***.

Input Arguments

R	A reward matrix. Must be S x A where S is the number of states and A is the number of actions.
statelabels	1 x S cell array containing labels for the states in the sequential decision problem, where S is the number of states in the sequential decision problem. Each cell statelabels{j}should contain a string that represents the name of the j- th state in the sequential decision problem.
actionlabels	1 x A cell array containing labels for the states in the sequential decision problem, where A is the number of states in the sequential decision problem. Each cell actionlabels{j}should contain a string that represents the name of the j- th action in the sequential decision problem.

Example

R = [-50 50; -100 100];
actionlabels = {'buy','sell'}; statelabels =
{'low', 'high'};
table = dispreward(R,statelabels,actionlabels); disp(table)

The following is displayed to the command window:

```
'Reward Matrix'    'buy'     'sell'
'low'              [ -50]    [  50]
'high'             [-100]    [ 100]
```

See Also

num2cell(a MATLAB® function) | RRtable

dispcurrency

Converts a number to a string formatted as a currency value, according to the conventions of the specified currency.

Syntax

currencyStr = dispcurrency(num)
currencyStr = dispcurrency(num, 'currency')

Description

currencyStr = dispcurrency(num)

currencyStr = dispcurrency(num, 'currency')
converts a number to a string formatted as a currency value, according to the conventions of the specified currency. The default currency is USD.

Input Arguments

num	(Required) The number to be converted. This may also be a vector or cell array of numbers to convert.
Currency	(Optional) A string identifying the desired currency format. Supported currencies include: US Dollar (USD); Canadian Dollar (CAD); Australian Dollar (AUD); British Pound (GBP); Euro (EUR); Japanese Yen (JPY); Swiss Franc (CHF).

Output Arguments

currencyStr	The number in a string format as a currency value, according to the conventions of the specified currency. If no currency is specified, USD will be used as the default currency. If **num** is entered as a vector, this will be a cell array of formatted currency strings.

Examples

currencyStr = dispcurrency(34)

currencyStr =
$34

currencyStr = dispcurrency([34, 52], 'EUR') currencyStr =
'€34' '€52'

dispwelcome

Displays to the command window a welcome message and outputs two formatting variables.

Syntax

[shortline, line] = dispwelcome

dispwelcome

Description

[shortline, line] = dispwelcome;
prints to the command window the traditional Rapid Recursive welcome message and assigns the formatting variables shortline and line.

dispwelcome
simply displays the welcome message without outputting any formatting variables.

Output Arguments

shortline	A string consisting of 10 dashes, useful for formatting.
line	A string consisting of 59 dashes. Often used to visually divide different sections of output from Solution Templates.

Example

[shortline, line] = dispwelcome;

```
----------
Rapid Recursive® toolbox version 1.7.0
----------
```

graphdist

Graphs a probability mass function.

Syntax

f = graphdist(dist)

Description

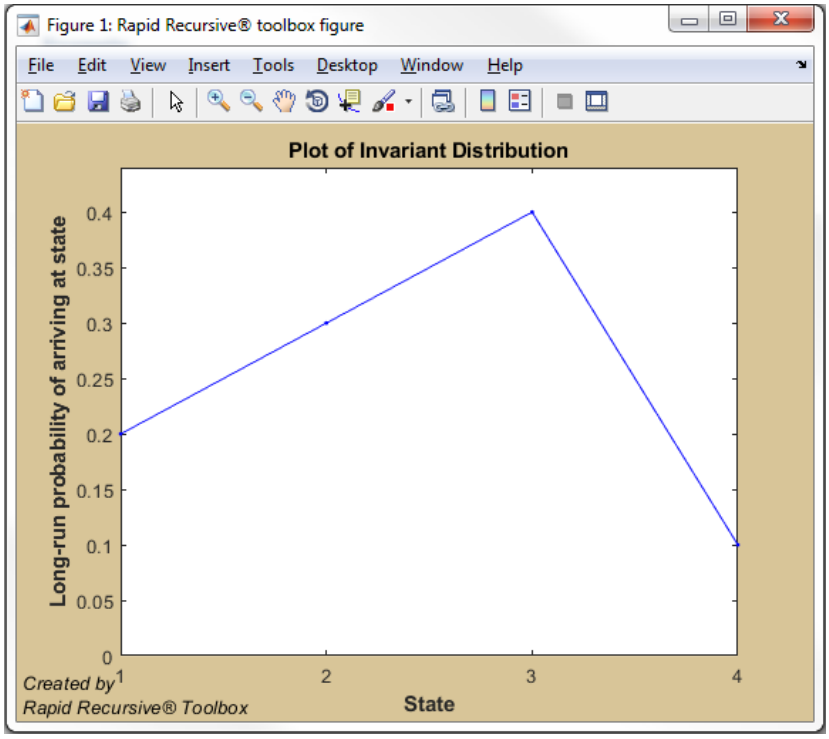
f = graphdist(dist)
graphs the probability mass function represented by ***dist*** and returns ***f*** the handle of the figure. The title and labels of the graph are tailored to graph the stationary distribution of a Markov chain (also known as invariant distribution), but these can be easily changed after the graph has been created by editing the figure's properties.

Input Argument

dist	The probability mass function. ***dist*** must be a vector where each element is between 0 and 1, and the sum of all elements must equal 1.

Example

```
dist = [0.2 0.3 0.4 0.1];
graphdist(dist)
```



See Also

RRtable

RRresultsreport

Displays the results of a solved sequential decision problem in an interactive format.

Syntax

RRresultsreport(Out)

Description

RRresultsreport(Out)
will display the results of a solved sequential decision problem in an interactive format in a new MATLAB figure window.

Input Arguments

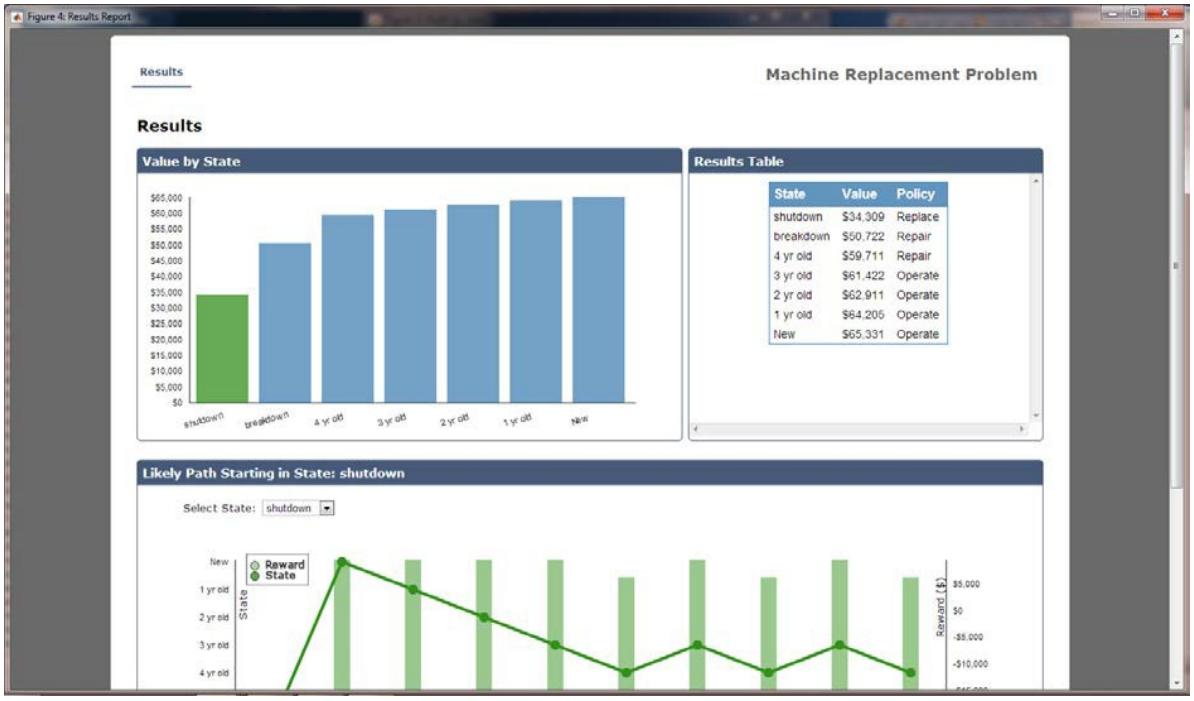
Out	(Required) An Out structure from a solved Rapid Recursive model. This must have at least the following fields: • ***V***, an S x 1 vector representing the value function. • ***policy***, an S x 1 vector representing the optimal policies. • ***Input***, an *Input* structure modelled after the sequential decision problem. • ***Input.S***, the number of states. • ***Input.statelabels***, an S x 1 vector representing the labels of each state. • ***Input.A***, the number of actions. • ***Input.actionlabels***, an S x 1 vector containing the labels of each action.

Output Arguments

None. This function simply displays the results in a new figure window.

Examples

MachineReplacement; RRresultsreport(Out)



RRsimulateresults

Performs a Monte Carlo simulation of a solved sequential decision problem following the optimal policy.

Syntax

Output = RRsimulateresults(Out, 'ParameterName', ParameterValue)

Description

Sim = RRsimulateresults(Out)
performs a 100 Monte Carlo simulations of the solved sequential decision problem in stored in Out for 10 periods.

Sim = RRsimulateresults(Out, 'nSims', nSims)
sets the number Monte Carlo simulations to the value stored in nSims.

Sim = RRsimulateresults(Out, 'nPeriods', nPeriods)
sets the number of periods to simulate the sequential decision problem to the value stored in nPeriods.

Input Arguments

Out	An Out structure (as returned by RRvalueiteration, RRpolicyiteration, or RRbackwardinduction) with at least the following fields: • ***Input***: A valid Input Structure accepted by one of the solution algorithms above • ***policy***: A vector of length 1 x S specifying the optimal policy for each state • ***algorithm***: A string specifying solution algorithm used to solve the sequential decision problem. Accepted values are: 'Value iteration','Backward induciton', or 'Policy iteration'.

Optional Parameters

nSims	The number of unique simulations to perform. The default value is 100.
nPeriods	The number of periods to simulate the sequential decision problem. The default value is 10.

Output Arguments

Sim	A structure containing the following fields:

	• ***simV***: an S x 1 x nSims matrix containing the simulated value for each state in each simulation. • ***averageV***: an S x 1 vector containing the simulated value for each state averaged over all simulations. • ***Spath***: an S x nPeriods+1 x nSims matrix containing the simulated state path for each starting state in each simulation. • ***Ppath***: an S x nPeriods x nSims matrix containing the simulated policy taken in each state in each period in each simulation. • ***Rpath***: an S x nPeriods x nSims matrix containing the simulated reward received in each state in each period in each simulation. • ***seed***: the seed of the random number generator prior to the simulation

Examples StartUpEntreprenuer

```
close all
Sim = RRsimulateresults(Out) Sim =

            simV: [7x1x100 double]
        averageV: [7x1 double]
           Spath: [7x11x100 double]  Ppath:
           [7x10x100      double]      Rpath:
           [7x10x100 double]
            seed:       [1x1       struct]

Sim.Spath(:,:,1)

ans =

    Columns 1 through 4

         1.00         1.00         1.00         1.00
         2.00         3.00         5.00         6.00
         3.00         5.00         6.00         6.00
         4.00         3.00         3.00         5.00
         5.00         5.00         6.00         6.00
         6.00         6.00         4.00         4.00
         7.00         5.00         6.00         6.00

    Columns 5 through 8
```

```
    1.00    1.00    1.00    1.00
    4.00    4.00    3.00    5.00
    4.00    4.00    4.00    4.00
    6.00    6.00    6.00    6.00
    6.00    4.00    4.00    4.00
    4.00    4.00    4.00    4.00
    6.00    6.00    6.00    6.00

  Columns 9 through 11

    1.00    1.00    1.00
    1.00    1.00    1.00
    4.00    4.00    4.00
    4.00    4.00    4.00
    4.00    4.00    3.00
    4.00    4.00    4.00
    6.00    4.00    4.00

Sim = RRsimulateresults(Out,'nSims',20)
---------------
   The RR Toolbox has checked the Input structure for conformance, convergence,
   and validation of key inputs. Results of these tests are positive. The problem is now
   ready to be formulated mathematically and solved.
---------------

Sim =

        simV:    [7x1x20    double]
    averageV: [7x1 double]
       Spath: [7x11x20 double]  Ppath:
       [7x10x20    double]    Rpath:
       [7x10x20 double]
         seed: [1x1 struct]

Sim = RRsimulateresults(Out,'nPeriods',20)
---------------
   The RR Toolbox has checked the Input structure for conformance, convergence,
   and validation of key inputs. Results of these tests are positive. The problem is now
   ready to be formulated mathematically and solved.
---------------

Sim =

        simV:    [7x1x100    double]
    averageV: [7x1 double]
       Spath: [7x21x100 double]  Ppath:
       [7x20x100    double]    Rpath:
       [7x20x100 double]
         seed: [1x1 struct]
```

RRsvplot

Creates and displays a bar chart of the value in each state of a solved sequential decision problem.

Syntax

h = RRsvplot(Out)
h = RRsvplot(Out, 'ParameterName', 'ParameterValue') h = RRsvplot(V)
h = RRsvplot(V, 'ParameterName', 'ParameterValue')

Description

RRsvplotcreates and displays a bar chart of the value in each state of a sequential decision problem, and returns the handle to the figure containing this chart.

h = RRsvplot(Out)
uses an ***Out*** structure from the Rapid Recursive® Toolbox to determine the number of states, their labels, and the corresponding values. By default, the plot title uses the description contained in the ***Input*** structure that is a field of the ***Out*** structure.

h = RRsvplot(Out, 'ParameterName', 'ParameterValue')
performs the same tasks as the syntax above, but allows the user to define custom parameter values (see below).

h = RRsvplot(V)plots the state values contained in ***V***.

h = RRsvplot(V, 'ParameterName', 'ParameterValue')
performs the same tasks as the syntax above, but allows the user to define custom parameter values (see below).

Input Validation

Before attempting to create any figures, RRsvplotvalidates its inputs. If an error is found, RRsvplotis stopped and messages identifying the error(s) are displayed in the command window.

Input Arguments

Let S be the number of states and A be the number of actions. **Note that only one of the two Input arguments below is required to call this function.**

Out	A structure that contains fields for at least ***V*** and ***Input***. In turn, ***Input*** is a structure that must contain fields for at least ***S***.
V	A vector containing the values for each of the states in the problem.

Parameters

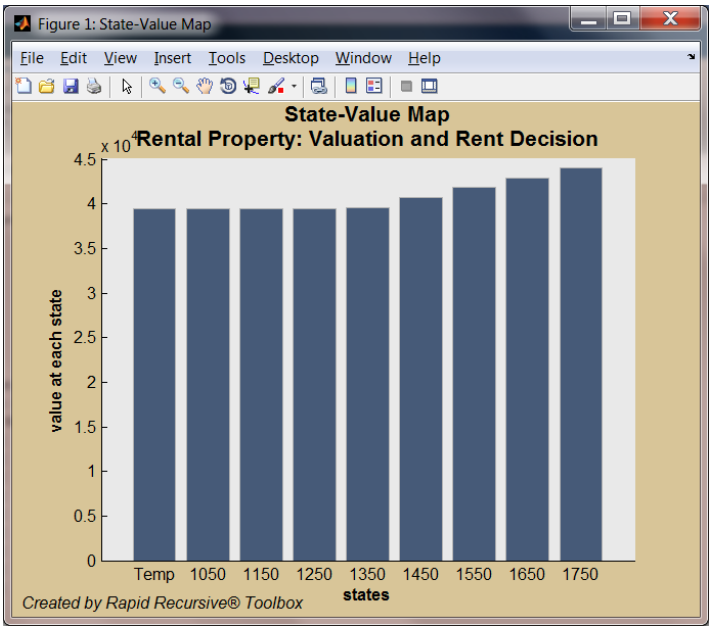
The following are all optional parameters. They may be added to the function call in any order and combination.

title	A user-defined string to use as part of the title on the chart. The title will read "***State- Value Plot:*** *TITLE*".
statelabels	S x 1 cell array containing a string argument for the label of each state. If no value is provided, the function attempts to use ***Input.statelabels*** in the ***Out*** structure. If ***Input.statelabels*** is not found or no ***Out*** structure exists, default numeric labels are used.
spline	Parameter used to toggle the presence of a spline[2] approximation to the continuous value function. Permitted values are 'on' and 'off'. The default value is 'off'.
slist	A row vector containing the indices of states for which the values will be plotted. By default, all states in ***Out*** or ***V*** are plotted.

Examples

In these examples, the ***Out*** structure from the "RentalPropertyManagement" solution template is used to generate a state-value plot.

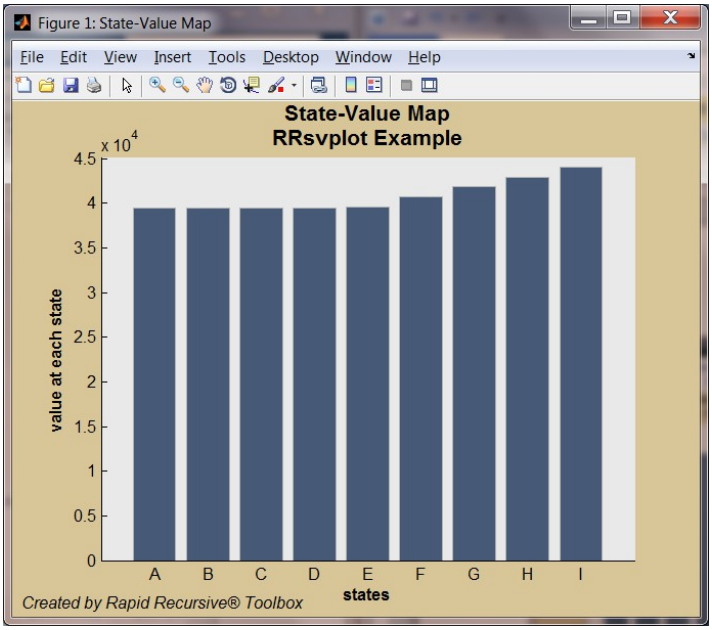
h = RRsvplot(Out)



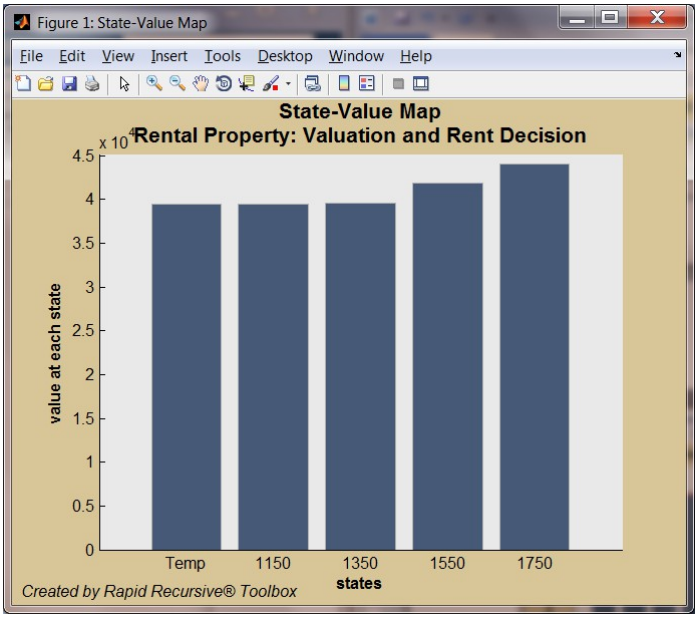
[2] Mathematically, a spline is a polynomial function that smoothly approximates the relationship between data points. Spline interpolation treats these data points as knots and draws a smooth curve that passes through each knot.

h = 1

RRsvplot(Out,'title','RRsvplot Example','statelabels',...
{ 'A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I'});

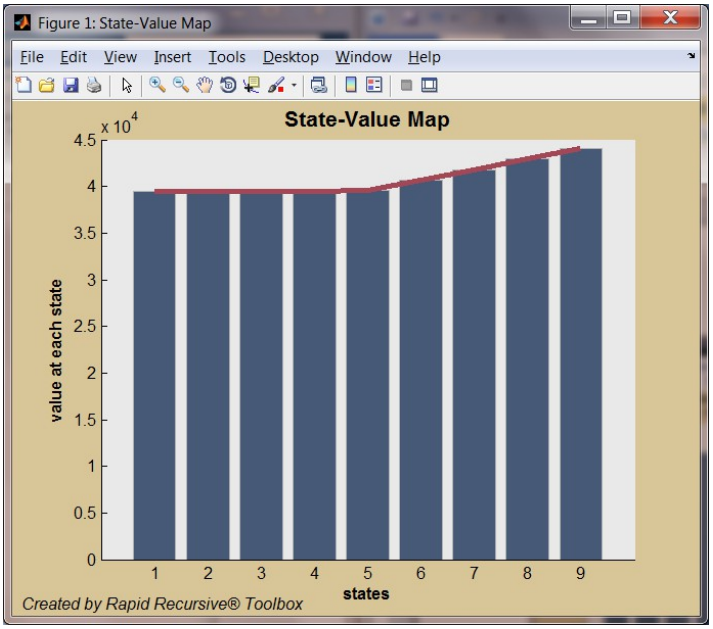


RRsvplot(Out,'slist',[1 3 5 7 9]);



V = Out.V;

RRsvplot(V,'spline','on');



See Also

RRtable| displist

RRtable

Creates a formatted uitable with results from a sequential decision problem.

Syntax

f=RRtable(Out,statelabels,actionlabels,desc)**(recommended syntax)**
… = RRtable(Out, statelabels, actionlabels)
… = RRtable(Out, Input)
… = RRtable(Out)
[f t] = RRtable(...)

Description

…=RRtable(Out,statelabels,actionlabels,desc)**(recommended syntax)**
creates a formatted uitable with results from the ***resultstable*** field of ***Out***. The table will use labels for the states and actions from ***statelabels*** and ***actionlabels*** respectively, and a title from ***desc***.

… = RRtable(Out, statelabels, actionlabels)
creates a formatted uitable with results from the ***resultstable*** field of ***Out***. The table will use labels for the states and actions from ***statelabels*** and ***actionlabels*** respectively.

… = RRtable(Out, Input)
creates a formatted uitable with results from the ***resultstable*** field of ***Out***. The table will use labels for the states and actions from the ***statelabels*** and ***actionlabels*** fields, and a title from the ***desc*** field of ***Input***, if they exist.

… = RRtable(Out)
creates a formatted uitable with results from the ***resultstable*** field of Out. The table will use a title from the ***desc*** field of ***Out*** if it exists. The table will also use labels for the states and actions from the ***statelabels*** and ***actionlabels*** fields respectively of the Input field of Out, if it exists.

[f t] = RRtable(...)
creates a formatted uitable and returns f and t, the handles for the figure and the uitable respectively.

Input Arguments

Out	A structure array with fields for at least ***resultstable*** and optionally ***Input*** and ***desc***. To prevent errors, this variable should be taken directly from a run of RRvalueiteration or RRpolicyiteration.
statelabels	1 x S cell array containing labels for the states in the sequential decision problem, where S is the number of states in the sequential decision problem. Each cell statelabels{j}should contain a string that represents the name of the j- th state in the sequential decision problem.
actionlabels	1 x A cell array containing labels for the states in

	the sequential decision problem, where A is the number of states in the sequential decision problem. Each cell actionlabels{j}should contain a string that represents the name of the j- th action in the sequential decision problem.
desc	A string that will be turned into the title of the uitable.
Input	A structure array that contains fields for ***statelabels***, ***actionlabels***, ***desc*** and ***round***.

Tips

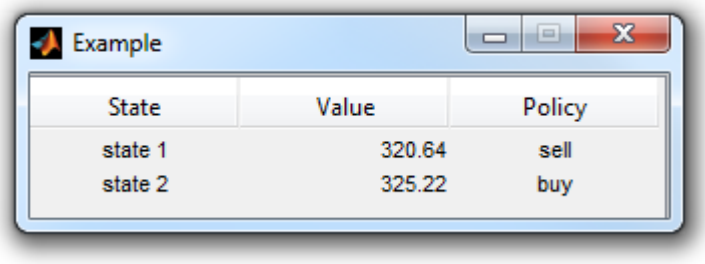
To prevent errors, this ***Out*** should be taken directly from a run of RRvalueiterationor RRpolicyiteration.

Users can set the number of decimals the "value" column in the table is round to by setting a value to the ***round*** field in the ***Input*** structure. Users have two options of when to set the value of the round field: users can either set the value of the ***round*** field to the ***Input*** structure before Input is used as the input to one of the solution algorithms (like in the example below) or users can set ***Out.Input.round*** after the solution has been obtained, but before use in RRtable. Note that for rounding to work, the ***Out*** structure must contain a field for ***V***.

Example

```
% Create a structure array with default fields Input =
RRcreateinputstruct('e')

% Fill in required field values before using it in RRvalueiteration P(:,:,1) = [0.6 0.4; 0.6 0.4];
P(:,:,2) = [0.5 0.5; 0.5 0.5];
Input.P = P;
Input.R = [20 30; 35 25];
Input.beta = 0.9;
Input.round = 2; % Round the value column to 2 decimal points

% Run value function iteration using Input Out =
RRvalueiteration(Input);

% Create state and action labels and a title statelabels = {'state 1',
'state 2'}; actionlabels = {'buy','sell'};
desc = 'Example';

% Create a formatted uitable RRtable(Out,statelabels,actionlabels,desc)
```

The following uitable is created:



See Also

RRvalueiteration | RRpolicyiteration | displist

installcheck

Checks installation was completed correctly.

Syntax

installcheck

Description

installcheckdisplays a positive message to the screen if installation was correctly completed. Otherwise, an error occurs.

Example

```
installcheck
---------------
    Congratulations! You have successfully installed the Rapid Recursive Toolbox.
---------------
```

See Also

RRvalueiteration | RRpolicyiteration | RRcheckinputs

RRclearworkspace

Clears the workspace according to conventions at the end of a Rapid Recursive® solution template, leaving only the variables that have names beginning with '***Input***' or '***Out***'.

Syntax

RRclearworkspace

RRclearworkspace(varname)

Description

RRclearworkspace
clears all variables from the workspace that do not have names beginning with '***Input***' or '***Out***'.

RRclearworkspace(varname)
clears the workspace, as above, also leaving any variables listed in ***varname*** (a string or cell array).

Input Arguments

varname	(Optional) A string or cell array containing the names of variables not to be cleared

Output Arguments

None

Examples

Input = RRcreateinputstruct('b'); Out =

RRcreateoutstruct;

test = 'A';

[a, b, c, d, e] = deal(1:5); who

Your variables are:

Input Out a b c d e test

RRclearworkspace

who

Your variables are:

Input Out

Input = RRcreateinputstruct('b'); Input2 =

RRcreateinputstruct('b'); Out =

RRcreateoutstruct;

test = 'A';

[a, b, c, d, e] = deal(1:5); who

Your variables are:

Input Input2 Out a b c d e test

RRclearworkspace({'b', 'd', 'e'})

who

Your variables are:

Input Input2 Out b d e

RRnormcdf

Calculates the value of the cumulative density function (CDF) for the normal distribution with mean mu and standard deviation sigma at the value x.

Syntax

p = RRnormcdf(x, mu, sigma, y)

Description

p = RRnormcdf(x)
calculates the value of the normal CDF with mean 0 and standard deviation 1 from –Infinity to x.

p = RRnormcdf(x, mu)
calculates the value of the normal CDF with mean mu and standard deviation 1 from –Infinity to x.

p = RRnormcdf(x, mu, sigma)
calculates the value of the normal CDF with mean mu and standard deviation sigma from –Infinity to x.

p = RRnormcdf(x, mu, sigma, y)
calculates the value of the normal CDF with mean mu and standard deviation sigma from y to x.

Input Arguments

x	(Required) The point at which the CDF will be evaluated
mu	(Optional) The mean for the normal distribution in question. Default value is 0.
sigma	(Optional) The standard deviation for the normal distribution in question. Default value is 1.
y	(Optional) The lower bound for the integral used to calculate the CDF. Default value is –inf.

Output Arguments

p	Value of the CDF of the normal distribution at x given mu and sigma.

Examples

x = 0;
mu = 1;
sigma = 2;
y = -1;

RRnormcdf(x) ans =

```
        0.50

RRnormcdf(x,mu) ans =

        0.16

RRnormcdf(x,mu,sigma) ans =

        0.31

RRnormcdf(x,mu,sigma,y) ans =

        0.15
```

See also

RRcreatetransition

superclear

Closes all windows, clears the command window and clears the MATLAB® memory.

Syntax

superclear

Description

superclearperforms the function of close all, clcand clear all. Thus, superclear deletes all figures whose handles are not hidden, clears the command window and removes all variables, global variables, functions and MEX-files from memory.

Example

superclear

ind2sub_mat

Convert linear indices of matrix to subscript indices. ind2sub_mat is used to determine the equivalent subscript values corresponding to a given linear index into an array.

Syntax

sub = ind2sub_mat(siz, ind)

Description

sub = ind2sub_mat(siz, ind)
converts linear indices stored in indto a matrix of subscript indices for a matrix of size siz. Each row of subcontains the subscript indices for the corresponding element of ind.

Input Arguments

siz	the size of the matrix for which to calculate multiple subscript indices. This must be a vector matching the output of size(matrix).
ind	the linear index (or indices) to be converted to subscript indices. This can be a scalar or vector. Ind2sub_mat will calculate subscript indices for a matrix of size siz for all elements of the vector.

Output Arguments

sub	a vector or matrix of subscript indices. Each row contains the subscript indices for the corresponding linear index element of ind.

Notes

For calculating subscript indices of matrices of different sizes, use cellfun.

ind2sub_mat is an extension of the built-in matlab function sub2ind.

Examples

A = reshape(1:16,4,4)

A =

```
     1     5     9    13
     2     6    10    14
     3     7    11    15
     4     8    12    16
```

sub = ind2sub_mat([4,4],[1,5,8,16]) sub =

```
     1     1
```

```
     1     2
     4     2
     4     4
```

A(1,2)

ans = 5

A(4,2)

ans = 8

See also

sub2ind_mat

sub2ind_mat

Convert subscript indices to linear index. Sub2ind_mat is used to determine the equivalent linear index corresponding to a given set of subscript values.

Syntax

ind = sub2ind_mat(siz, sub)

Description

ind = sub2ind_mat(siz, sub)
converts subscript indices of a matrix of size siz stored as rows in the matrix subscript notation to linear indices. Each element of indcorresponds to the subscript indices in the corresponding row of sub.

Input Arguments

siz	(Required) The size of the matrix from which the subscript indices will be converted to linear indices. This must be a vector matching the output of size(matrix).
sub	(Required) A vector or matrix of subscript indices. Each row contains the subscript indices for a single linear index.

Output Arguments

ind	The linear index (or indices) determined from the subscript indices in sub. This can be a scalar or vector.

Notes

For calculating subscript indices of matrices of different sizes, use cellfun.

sub2ind_mat is an extension of the built-in matlab function sub2ind.

Examples

A = reshape(1:16,4,4)

A =

```
     1     5     9    13
     2     6    10    14
     3     7    11    15
     4     8    12    16
```

```
sub = [1,1;
       1,2;
```

```
        4,2;
        4,4];

ind = sub2ind_mat([4,4],sub) ind =

     1     5     8    16

A(ind)

ans =

     1     5     8    16
```

See also

ind2sub_mat

RRver

RRverreturns information on the version of the Rapid Recursive® Toolbox.

Syntax

RRver

[ver, name, date] = RRver

[RR.ver, RR.name, RR.date] = RRver

Description

RRver returns the version number of the Rapid Recursive® Toolbox that is running on this computer.

[ver, name, date] = RRver
returns the version number, name, and date for the Rapid Recursive® Toolbox, as installed on this machine.

[RR.ver,RR.name,RR.date] = RRver
returns the above information in the structure 'RR'.

Input Arguments

There are no input arguments for this function

Output Arguments

ver	Version number of the current installation of the Rapid recursive® Toolbox
name	Name of the Rapid Recursive® Toolbox
date	Release date for the current version of the Rapid Recursive® Toolbox

Example

s RRver
ans ='2.0.0'

RRguide

RRguide accesses the Guide to Recursive Models.

Syntax

RRguide

Description

RRguide displays the **Guide to Recursive Models**. This Guide is intended to assist interested people in understanding the general power, use, and features of recursive models, and to provide instructive examples using the Rapid Recursive® toolbox. The Guide is structured to:

- Introduce the concepts of sequential decision problems and the recursive approach.
- Describe a step-by-step process to organize the information necessary to compose a sequential decision problem, solve it, and report the results.
- Present a set of models representing common decision problems (Solution Templates).

The function attempts to open the document titled GuideToRecursiveModels.pdf or guide_to_recursive_models.htm, depending on the user's MATLAB® version and whether the PDF is available.

Input Arguments

There are no input arguments for this function.

Output Arguments

This function does not return any variables.

RRhelp

RRhelp accesses the Rapid Recursive Toolbox reference and command syntax guide (User's Guide).

Syntax

RRhelp

Description

RRhelp displays the **User's Guide** for the Rapid Recursive® toolbox. The User's Guide serves as a reference for commands in the software, as well as for installation, licensing, and supported operating systems. Users should consult the User's Guide for information on the proper command syntax and usage, as well as troubleshooting and licensing.

The function attempts to open the document titled users_guide.htm or Rapid Recursive - Users Guide.pdf, depending on the user's MATLAB® version and file availability.

Input Arguments

There are no input arguments for this function.

Output Arguments

This function does not return any variables.

I. Release Notes

Version 2.0.0

Date: 13 November, 2025

The following new templates have been added:

- ***Automotive Customer Promotion Strategy:*** Models the decision-making process for a marketing strategy in the automotive retail industry.
- ***Basic Business Decision Model:*** This model provides a dynamic framework for analyzing the strategic value of a business whose performance evolves under uncertainty. Using the Sequential Decision Problem (SDP) approach, the model evaluates how optimal management actions — such as operating, investing, or selling — affect firm value across multiple states and over time.

  The model operates along three key state dimensions:

  a) General Economic Demand (Y): Captures the impact of changing macroeconomic or market conditions on profitability and valuation.

  b) Company Scale (M): Represents firm size or operating capacity, which can expand or contract through investment or divestment decisions.

  c) Idiosyncratic Factor (I): Reflects firm-specific characteristics such as restrictions on resale, liquidity limitations, or other unique operational risks.

  Key Features:

  1. Dynamic Valuation: Solves the recursive value functional equation (Bellman equation) to estimate the firm's equity value in every possible state.
  2. Optimal Policy Identification: Determines the value-maximizing decision (operate, invest, or sell) for each state, providing actionable managerial insight.
  3. Reward and Transition Matrices: Quantify immediate returns and probabilistic state transitions based on strategic choices.
  4. Marketability Discount (DLOM) Analysis:
     a. When the idiosyncratic dimension (I) represents a time-varying resale restriction, the model computes Discounts for Lack of Marketability (DLOM) by comparing restricted and unrestricted states.
     b. Outputs include both absolute value differences and percentage discounts, illustrating how restrictions reduce liquidity and value — and how those discounts decline as restrictions lapse.
  5. Likely Path Analysis: Projects the firm's probable evolution over time under optimal management, displaying sequences of states, policies, and rewards.
  6. Comparative Income Statement Integration: Links baseline profitability and cost assumptions to resulting firm values for representative companies (e.g., Company A vs. Company D).

Version 1.9.0

Date: 12 November, 2018

The following new functionality has been added:

- ***RRtrainreward:*** Creates and trains a reward matrix based on a data series of observed state-action combinations and the rewards associated with them that were received by decision makers.
- ***RRtraintransition:*** Creates and trains a transition matrix based on a data series of observed state-action combinations and the final states associated with them that were transitioned to by the

decision makers.

- ***RRslicestates:*** Allows users to specify which states from their multi-dimensional state space they would like to view from their BigState table. The function returns a subset of the BigState table with the rows corresponding to the specified states as well as a binary column vector whose entries denote which rows of the TBigState table satisfy the conditions of the array state slices. This particularly useful in viewing state-specific results from the solved sequential decision problems.
- ***ItoProject:*** A function to simulate certain Ito processes or stochastic integrals, which rely upon Brownian motion and can be described as Stochastic Differential Equations ("SDEs"). The SDEs that can be simulated with this function include simple Brownian motion, Brownian motion with drift, and Geometric Brownian motion.
- ***TrendProject:*** A function which projects baseline, high, and low growth trends given base revenue, a base growth rate, a deviation, and number of periods for which to project.
- ***ExtractStateDimension:*** Creates and returns sub-Input structures that correspond to each of the state dimensions of the Input structure. This function is useful for extracting and examining the individual dimensions that make up a multi-dimensional decision problem.
    - ***dispwelcome:*** Displays the traditional Rapid Recursive Toolbox welcome message, as well as the version. The function also returns the shortline and line variables, which are useful for formatting solution templates.

The following new solution templates have been added:

- **Possible HQ2 Value In Place ST:**
- **Possible FranchiseeValue ST:**
- **Possible MonetaryInvestment ST:**

Other improvements:

- ***GetFormattingVariables*** (now ***GetColor***) now supports outputting a structure of formatting variables for improved ease of use.
- The functions ***profitpower*** and ***profitlinear*** have been updated with more robust input checking to ensure they are being used properly in solution templates.
- All solution templates have been updated with improved formatting.

Version 1.7.0

Date: 18 August, 2017

The following new functionality has been added:

- ***RRspecialkron:*** Creates a large matrix from a newly defined kronecker-type product between one 2-dimensional matrix and another 3-dimensional matrix. This is especially useful in defining transition matrices for multiple state dimensions with statistical dependence.
- ***disppaths:*** Displays the information contained in the paths structure output by the RRlikelypath function. Information can be displayed by starting state or by path type (state path, reward path, etc…).
- ***CreateBigState:*** Captures multi-dimensional state information in a fashion that allows it to be used in other calculations (such as rewards) or in data visualizations.
- ***GetFormattingVariables*** *(now* ***GetColor****):* Returns a set of standard colors and line definitions used in various examples throughout the Rapid Recursive toolbox.
- ***RRgraphtransitions:*** Creates a directed graph representation of the transition matrix for a specific action, for a properly composed RR decision model that includes states, actions, and a transition matrix. Here, the states are the nodes, and the edges between the nodes represent probabilistic transitions among the states. Such a representation can be made for a selected action.
- ***RRcompareoptima:*** compares the profit maximizing solution of a sequential decision problem to the value maximizing solution of the same problem.
- ***poissoncdf:*** Calculates the value of the cumulative density function (CDF) for the poisson distribution with expected value lambda, at the point x.

The following new solution templates have been added:

- **AutoDriveOrSell:** The owner of an automobile or truck learns that the mileage or other performance characteristic is significantly different than he or she originally thought, due to misrepresentation or other cause. The owner decides whether to continue driving the vehicle, perhaps with lower usage; or to sell the vehicle and use the proceeds as partial payment on a new one.
- **HouseholdSaving:** The representative household in this Solution Template chooses the household's optimal saving or borrowing plan based on its employment state and asset holdings. This model is a workhorse of microeconomics, macroeconomics, and asset pricing theory. This version is based on the model outlined in Ljungqvist and Sargent's *Recursive Macroeconomic Theory* (MIT Press, 2012), Section 4.2.
- **JobSearchWithFiring:** This model is taken from Ljungqvist and Sargent's *Recursive Macroeconomic Theory* Section 6.3.4. It models a household that receives job offers from a particular distribution (which may be time-invariant or Markov) and decides whether to accept an offer (thus becoming employed) or to reject it and to continue searching. Employed workers will be fired with a certain probability. Also added from L-S to the model

is the opportunity for the worker to quit his job and search for another. It can be shown that the acceptance decision can be characterized by a reservation wage, and that the option to quit a job is never taken.

- **GradSchoolTutionPricing:** This template models the pricing decision of a graduate school. The school considers raising or lowering tuition in an attempt to increase its earnings (revenue less costs) on its graduate program. The school recognizes that some students (especially those close to graduation) are much less sensitive to price increases than those about to enter the school. In this model, the school seeks the value-maximizing solution to the decision problem, rather than following the conventional (neoclassical) profit maximizing pricing goal.
- **ForestManagement:** A forest is managed by two actions: Wait and Harvest. The manager has two sets of objectives in mind: to maintain the forest for wildlife, enjoyment of natural resources, and the growth in the value of the wood; and to maximize the value of the wood that can be harvested. The manager is aware that there is a possibility that a fire burns the forest (or another loss event occurs, such as a flood or disease) that results in the loss of the timber.

Other improvements:

- ***RRcreateinputstruct*** now has a new format option ('e'), containing new info and table fields
- ***RRchecktransition*** now supports sparse transition matrices
- ***RRcreatetransition*** now includes an option for the Poisson distribution
- All solution templates have been updated with cleaner formatting

Version 1.5.0

Date: 14 September, 2015

The following new functionality has been added:

- ***RRresultsreport:*** Displays the results of a solved sequential decision problem in an interactive format.
- ***RRcombinetransitions:*** Functionality has been extended to handle statistically independent transition matrices from N state dimensions.
- ***RRcombinerewards:*** Functionality of RRcreaterewardstwodim has been extended to handle reward parameters from N state dimensions.
- ***RRcreatetransition:*** Creates one frame of a transition matrix informed by the specified probability distribution or process.
- ***RRrewardmatrix:*** Calculates a reward matrix (reward function) from a matrices containing the states, applied actions, and received rewards for a population of observations taken over time.
- ***RRtransitionmatrix:*** Calculates a transition matrix from matrices containing the states and applied actions for a population of observations taken over time.
- ***RRsimulateresults:*** Performs a Monte Carlo simulation of a solved sequential decision problem following the optimal policy.

- ***RRtimetransition:*** Creates one frame of a transition matrix for time periods, where time moves forward to the next period with certainty.
- ***generalchart:*** Creates and displays a customizable bar chart.
- ***createlabels:*** A multipurpose label creator. createlabels will combine any number of groups of string labels via a Cartesian or Kronecker Product. createlabels will also convert numeric vectors to a group of string labels.
- ***disppaths:*** Displays the information contained in the paths structure output by the RRlikelypath function. Information can be displayed by starting state or by path type (state path, reward path, etc…).
- ***RRnormcdf:*** Calculates the value of the cumulative density function (CDF) for the normal distribution with mean mu and standard deviation sigma at the value x.
- ***ind2sub_mat:*** Convert linear index of matrix to multiple subscript indices. ind2sub_mat is used to determine the equivalent subscript values corresponding to a given single index into an array.
- ***sub2ind_mat:*** Convert subscript indices to linear index. Sub2ind_mat is used to determine the equivalent linear index corresponding to a given set of subscript values.

The following functions have been deprecated:

- ***RRcreaterewardtwodim*** has been renamed ***RRcombinerewards***
- ***RRreducetable***

Other improvements:

- a new format option ('d'), containing new info and table fields was added ***RRcreateinputstruct***.
- ***RRviewtransition*** now accepts a transition matrix as an input in addition to a full Input structure
- ***RRvalueiteration*** and ***RRpolicyiteration*** now return an Out structure with an additional field, *V_sa*, which reports the value of each action in each state.
- ***RRlikelypath*** now supports 3D reward matrices and outputs the more compact *paths* structure.
- Visualization of overlapping paths is improved in ***RRgraphlikelypath,*** and improved options for plot suppression and subplot display have been added.
- ***SToutline*** was reformatted with executable skeleton code, now offering a clearer and more informative command window report.

Version 1.3.0

Date: 1 October, 2014

The following new functionality has been added:

- ***RRviewreward:*** Displays one frame of the transition matrix for a recursive problem as a heat map of the transition probabilities.

- ***RRviewtransition:***Displays a ribbon chart visualization of the reward matrix, where each ribbon corresponds to a different state in the reward matrix and shows how rewards change as actions vary.
- ***RRlikelypath:*** Using the solution of a Rapid Recursive® model, project forward a likely path assuming that the most likely random events occur and the subject person follows the optimal policy every time.
- ***RRgraphlikelypath:*** Illustrates the likely path calculated for the solution of a Rapid Recursive® model, showing the likely state evolution, the optimal actions (policies), and expected rewards at each time period going forward. The likely path is generated assuming that the most likely random events occur and the subject person follows the optimal policy every time.
- ***newST:*** Creates a new solution template, with tips and hints to guide you through creating your own recursive model.
- ***RRclearworkspace:*** Clears the workspace according to conventions at the end of a Rapid Recursive® solution template, leaving only the variables that have names beginning with '***Input***' or '***Out***'.
- ***Dispcurrency:*** Converts a number to a string formatted as a currency value, according to the conventions of the specified currency.
- ***Convertdiscount:*** Converts the discount factor from an annual basis to the time index specified by the user.
- ***RRfigure:*** Creates a new figure with the default background color for the Rapid Recursive® Toolbox and a note indicating that the figure was created by the Rapid Recursive® Toolbox.
- ***RRver:*** returns information on the version of the Rapid Recursive® Toolbox.

All graphics have also been updated to ensure compatibility with the new graphics system found in MATLAB 2024+.

In addition, the following functions have been deprecated:

- ***createinputstruct*** has been renamed ***RRcreateinputstruct***
- ***cleaninputstruct*** has been renamed ***RRcelaninputsruct***
- ***createoutstruct*** has been renamed ***RRcreateoutstruct***

Version 1.2.0

Date: Aug 2013

The following new functionality has been added:

- ***RRinvesttransition***: creates a transition matrix that can be used in a reinvestment problem.
- ***RRcreaterewardtwodim***: creates the reward matrix for a sequential decision problem where the state vector contains two dimensions.
- ***RRcombinetransitions***: creates a transition matrix for a two-dimensional problem by combining two statistically independent transition matrices.
- ***RRfindinvariantdist***: finds the invariant distribution of a Markov chain.

- ***RRbackwardinduction***: solves finite time sequential decision problem using backward induction.
- ***RRshortincomestatement***: calculates and displays an income statement for a company whose business is described by a small set of key variables.
- ***RRoptimalpolicymc***: finds a Markov chain that represents how a sequential decision problem transitions from state to state if the decision maker follows the prescribed policy.
- ***RRreducetable***: reduces a results table into a table with fewer rows.
- ***RRsvplot***: creates and displays a bar chart of the value in each state for a sequential decision problem.
- GUIs have been added for two solutions templates, the beverages wholesaler valuation model and the rental property valuation model. These can be accessed by using the commands BeverageWholesalerGUIand RentalPropertyGUI.
- (R2013b or later only) A new field, ***table***, has been added to ***Out*** structures. The new field contains a table of results using MATLAB's new tablefunction. This allows for the easy display of a results table to the MATLAB command window.
- End users can now input their own models into the Compose Tool for visualization of their inputs by constructing their own ***Input*** structure with fields for all the required inputs ***P***, ***R***, ***beta*** as well as ***S*** and ***A***, and using the following syntax: RRcomposetool(Input).
- ***RRtable*** now allows end users to select the number of decimals they would like to round the "value" column to. End users do this by setting ***Input.round*** to the number of decimals they would like the numbers in the "value" column rounded it, before running a solution algorithm. Alternatively, users can set ***Out.Input.round*** after the solution has been obtained, but before use in RRtable. Note that for rounding to work, the ***Out*** structure must contain a field for ***V***.
- End users can now display help by typing "help" followed by the Rapid Recursive® Toolbox command they are interested in.
- There is an HTML version of the User's Guide.

Other changes:

- ***dispreward*** now automatically displays the reward matrix to the MATLAB command window without requiring the "disp" command.
- ***RRtable*** now has larger font size.
- ***RRvalueiteration*** and ***RRpolicyiteration*** now displays the number of scenarios considered.
- Installers have been code signed for both Mac and Windows operating systems.
- Improved a few error messages.

Version 1.1.1

Date: 15 May 2013

Adjustment that improves compatibility with earlier versions of MATLAB®.

Version 1.1.0

Date: 30 January 2013

Updated the End User License Agreement.

Version 1.0.0

Date: 23 December 2012

Appendix A. End User License Agreement

LICENSE AGREEMENT

This license is an agreement between Supported Intelligence, LLC ("Licensor") and you as the licensee ("you"). This license and the information you submitted as part of the purchase of a license in the Rapid Recursive Software collectively comprise the agreement ("Agreement") between us. When you download or use the Software this Agreement is immediately effective (and the date of use or acquisition is the "Effective Date"). If you do not consent to abide by this Agreement, you may not download or use the Software.

Do you agree to abide by the terms and conditions of this Agreement?

☐ **I agree.** ☐ **I do not agree.**

1.0 License.

1.1. Definitions. The term "Software" includes the Rapid Recursive® Software (inclusive of its object code and Solution Templates). The term "Documentation" refers to any instructional, informational, or help documentation that is available for download with the Software at Licensor's website, which is currently http://www.supportedintelligence.com/support. The term "Updates" refers to updates, bug fixes, patches, enhancements, improvements, upgrades, modifications and/or maintenance releases to the Software. The Software is designed to call up open-source software libraries. The term "Suppliers" refers to the third-party licensors who developed these open-source libraries.

The term "Software" does not include any printed summaries of output that you create using the Software ("Reports"), including any Reports generated by the Software using a Solution Template. The term "Software" also does not include any information you provide and include in an adaptation of a Solution Template.

1.2. License Grant. Subject to the terms of this Agreement, Licensor grants to you, and you accept, a non-exclusive and nontransferable perpetual license to install and use the Software on the number of servers, for the number of Users you selected when you purchased the Software. If you wish to increase the number of servers and/or number of concurrent Users per server, you agree to provide Licensor with written notice. Any such increases shall be subject to Licensor's approval and Licensor's then-current fees. You may make a reasonable number of copies of the Software and Documentation for backup or archival purposes only, so long as Licensor's and Suppliers' copyright notices and other proprietary markings, including a copy of this License, are reproduced or contained within such copies.

1.3. Limitations on Use. You may not copy, rent, lease, sell, sublicense, assign, loan, time-share or otherwise transfer or distribute copies of the Software or Documentation to others, except as set forth in this Agreement. You may not decompile, disassemble, decrypt, or otherwise reverse engineer the Software, and you agree to use your best efforts to prevent your employees (if any) and contractors (if any) from doing so. You may not modify, adapt, create a derivative work, merge, or translate the Software or the Documentation without the prior written consent of Licensor.

1.4. Adaptation of Solution Templates. Licensor hereby consents to your modification, adaptation, creation of a derivative work, merging, or translation of Solution Templates as desired. You may only share your altered Solution Templates with other licensed users of the Software.

1.5. Intellectual Property Rights. You acknowledge that Licensor and its Suppliers retain exclusive ownership of all copyrights, trademarks, patents and/or other intellectual property rights in the Software and the Documentation. You are not granted any rights in the Software or Documentation other than the license rights expressly set forth above. Licensor acknowledges that it does not have any ownership rights in your Reports.

1.6. Maintenance and Support. Software maintenance service ("Maintenance") is included in your license fee for the first full year after the Effective Date ("Initial Term"). Maintenance includes all Updates, updated Documentation pertaining to the Software, and Support services. "Support" refers to the customer assistance and service Licensor provides to address usage and licensing problems and enhance your experience with the Software. All Maintenance and Support policies and rates are listed on Licensor's website. You may need to download Updates and check for other relevant information by accessing Licensor's website, which is currently http://www.supportedintelligence.com/support.

You can continue uninterrupted service in subsequent years by renewing your Maintenance annually. Licensor projects Maintenance fees will increase from year to year by the greater of five percent (5%) or the most recent annual percentage increase in the general Consumer Price Index. Reinstatement of lapsed Maintenance is subject to your payment of all outstanding and unpaid Maintenance for prior years at Maintenance fees in effect on the date Maintenance is re-ordered.

Customized Support and consulting services, for issues not addressed through Maintenance, are available to you under policies and rates listed on Licensor's website.

2.0 Fees.

2.1. Payment. You agree to pay Licensor the license fees set forth on the applicable schedule on Licensor's website. From time to time, Licensor may revise its price schedule, and may set different fees applicable to group and individual licensees, as well as for academic, commercial, consulting, nonprofit, author, and government licensees. Licensor may also issue limited-term trial and test licenses at a nominal cost to specific individuals for purposes restricted to testing the Software.

2.2. Taxes. You agree to pay any sales, transfer, use or similar tax, as a result of your purchase of a license in this Software.

3.0 Term; Termination.

3.1. Term. The term of this Agreement is one year ("Initial Term") and is renewable on or around the first anniversary of the Effective Date upon payment of the applicable annual license fee.

3.2. Termination. You may terminate this Agreement at any time. However, you acknowledge that the intellectual property rights, limitations on use, limitation on liability, and other provisions survive any such termination, as stated in Section 7.12.

Should you terminate this Agreement, the license fee is nonrefundable except under the circumstances described below in Section 5.1. Except as otherwise provided in this Agreement, you agree that upon termination, you will either destroy (or permanently erase) the Software and Documentation and all copies thereof, and will provide confirmation of such actions to Licensor if requested.

4.0 Confidential Information.

4.1. Definition. "Confidential Information" shall mean all nonpublic information, whether in oral, written or other tangible form that you would consider to be confidential. However, Confidential Information does not include information that: (i) is or becomes generally available to the public other than (a) as a result of a disclosure by you or your employees (if applicable) or any other person who directly or indirectly receives such information from you or your employees or (b) in violation of a confidentiality obligation to you, or (ii) is known to Licensor on a non-confidential basis from a source which is entitled to disclose it to Licensor.

4.2. Nondisclosure and Nonuse Obligation. You agree not to disclose any Confidential Information in any Report, illustration, or example that you submit to Licensor as part of your Maintenance. If you unintentionally disclose such Confidential Information that you want to remain confidential, you will immediately notify Licensor in writing, describing the date time and information disclosed in sufficient detail that Licensor can locate it. Licensor will take such action as is reasonable and feasible to protect and/or destroy such Confidential Information.

5.0 Warranty.

5.1. Limited Warranty and Limited Remedy. Licensor warrants to you only that the Software will substantially conform to the Documentation and all specifications that reference the performance and functionality of the Software for a period of ninety (90) days from the Effective Date (the "Limited Warranty Period"), provided the Software is used with compatible computer hardware and operating systems. THIS LIMITED WARRANTY SHALL NOT APPLY TO SOLUTION TEMPLATES YOU HAVE MODIFIED. THIS LIMITED WARRANTY IS VOID IF FAILURE OF THE SOFTWARE HAS RESULTED FROM ACCIDENT, ABUSE, OR MISAPPLICATION. LICENSOR'S ENTIRE LIABILITY, AND YOUR SOLE AND EXCLUSIVE REMEDY SHALL BE, AT LICENSOR'S OPTION, EITHER TO (A) CORRECT OR HELP YOU WORK AROUND OR AVOID A REPRODUCIBLE ERROR, (B) REPLACE DEFECTIVE SOFTWARE OR (C) AUTHORIZE A REFUND, SO LONG AS THE SOFTWARE AND DOCUMENTATION ARE RETURNED WITHIN NINETY (90) DAYS OF THE EFFECTIVE DATE TOGETHER WITH A BRIEF WRITTEN STATEMENT DESCRIBING
THE ALLEGED ERROR. An "Error" is a defect in the Software that causes it not to perform substantially in accordance with the limited warranty described above. Any replacement Software or Documentation will be warranted for the remainder of the original warranty period only.

5.2. No Liability of Suppliers. YOU ACKNOWLEDGE THAT YOUR RIGHTS UNDER THIS AGREEMENT ARE SOLELY AGAINST LICENSOR. NO SUPPLIER MAKES ANY WARRANTY, ASSUMES ANY LIABILITY, OR UNDERTAKES TO FURNISH TO YOU ANY SUPPORT OR INFORMATION CONCERNING THE SOFTWARE AND/OR DOCUMENTATION. YOU RELEASE ALL SUPPLIERS FROM ANY CLAIMS, DAMAGES OR LOSSES ARISING FROM THE USE OF THE SOFTWARE AND/OR DOCUMENTATION, REGARDLESS OF THE FORM OF ACTION.

5.3. Disclaimer of Warranties. EXCEPT FOR THE LIMITED WARRANTY SET FORTH ABOVE ("LIMITED WARRANTY AND LIMITED REMEDY"), LICENSOR EXPRESSLY DISCLAIMS ALL OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT OF THIRD PARTY RIGHTS, OR THAT THE SOFTWARE'S FUNCTIONS WILL MEET YOUR REQUIREMENTS OR THAT ITS OPERATION WILL BE UNINTERRUPTED OR ERROR FREE. EXCEPT AS SET FORTH IN THIS AGREEMENT, INCLUDING WITHOUT LIMITATION LICENSOR'S MAINTENANCE OBLIGATIONS, THE ENTIRE RISK FOR THE QUALITY AND PERFORMANCE OF THE SOFTWARE AND THE DOCUMENTATION RESTS WITH YOU. IF THE SOFTWARE AND THE DOCUMENTATION PROVE DEFECTIVE AFTER THE EFFECTIVE DATE, AND IF NOT OTHERWISE COVERED BY LICENSOR'S MAINTENANCE OBLIGATIONS, YOU ALONE ASSUME THE ENTIRE COST OF SERVICE OR REPAIR.

6.0 Limitation of Liability.

6.1. Acknowledgment of Uncertainty and Risks Inherent in Use. Licensor's products are intended to assist you in evaluating possible future opportunities that involve judgments, predictions, and assessments of future economic, market, and other conditions. You acknowledge that neither you nor Licensor can know with certainty the future or predict with confidence all future conditions that may affect you. Furthermore, current economic, market, and other conditions will change in the future. If you wish to use the Software to assist in evaluating such an opportunity, you agree to exercise your best judgment when making any related decision, and acknowledge that the responsibility for such decision rests solely with you. Furthermore, you acknowledge that it is always advisable to retain competent assistance from business, legal, accounting, economic, actuarial, or other professionals as necessary when making important financial decisions.

6.2. Liability Exclusions and Limitations. IN NO EVENT SHALL EITHER PARTY OR ANY SUPPLIER BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES OF ANY KIND (INCLUDING LOST PROFITS, LOSS OF USE OR INTERRUPTION OF BUSINESS), OR LEGAL FEES, ARISING OUT OF THE USE OF THE SOFTWARE OR THE DOCUMENTATION, REGARDLESS OF THE FORM OF ACTION, WHETHER IN CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT PRODUCT LIABILITY OR OTHERWISE, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT WILL EITHER PARTY'S AGGREGATE LIABILITY FOR ANY CLAIM EXCEED THE LICENSE TOTAL FEES YOU HAVE PAID. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION.

7.0 General.

7.1. Governing Law; Jurisdiction; Venue; Entire Agreement. This Agreement shall be governed in all respects by the laws of the State of Michigan without regard to its conflict of laws principles. The parties exclude, in its entirety, the application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods. The parties stipulate and agree that any litigation between them concerning or relating to the Agreement and which includes federal statutory or federal common claims or causes of action for which exclusive subject matter jurisdiction is vested in a federal court shall be adjudicated in the United States District Court for the Eastern District of Michigan and that such Court may exercise supplemental jurisdiction over any and all state law claims or causes of action; provided, however, that any litigation which involves state law claims or causes of action, federal claims over which a state court has concurrent jurisdiction, or in the event that the designated federal court does not exercise supplemental jurisdiction over any state law claims or causes of action, shall be adjudicated in Washtenaw County Circuit Court (sitting in Ann Arbor, Michigan). You consent to personal jurisdiction in either or both the United States District Court for the Eastern District of Michigan and Washtenaw County Circuit Court. This Agreement constitutes the entire agreement between the parties with respect to the subject matter. This Agreement supersedes, and the terms of this Agreement govern, any prior agreements. This Agreement may only be changed by written mutual agreement of the parties.

7.2. Arbitration. Except with regard to disputes that may arise out of disputes involving any form of Intellectual Property, including but not limited to rights of patent, trademark, copyright or trade secret, any disputes, controversies and claims arising out of the terms of this Agreement shall be resolved by binding arbitration in accordance with the Commercial Arbitration Rules of the American Arbitration Association ("AAA"). Either party may initiate arbitration by filing a demand for arbitration. Disputes will be heard and determined by a single arbitrator mutually agreed upon and appointed by the parties. If the parties cannot agree on an arbitrator within one (1) month from the date arbitration is initiated, the arbitrator shall be appointed by the AAA. Neither party will communicate separately with any arbitrator. All communications between a party and any arbitrator will be directed to the AAA for transmittal to the arbitrator. Arbitration proceedings shall commence within sixty (60) days of the date arbitration is initiated, unless otherwise agreed by the parties. The arbitration location shall be: Washtenaw County, Michigan. Judgment upon the award rendered by the arbitrator may be entered in any court having jurisdiction. You agree to bear the expense of your own counsel, experts, witnesses, and preparation and presentation of proofs; except the arbitrator shall award to the prevailing party, if any, as determined by the arbitrator, all of its costs and fees. "Costs and fees" means all reasonable pre-award expenses of the arbitration, including the arbitrator(s)' fees, administrative fees, travel expenses, out-of-pocket expenses such as copying and telephone, court costs, witness fees and reasonable attorneys' fees.

7.3. Assignment. Your rights under this Agreement may not be assigned, whether by operation of law or otherwise, without the prior written consent of Licensor and any non-permitted assignment you perform will be void.

7.4. Notices. All notices permitted or required under this Agreement shall be in writing and sent by a commercial express carrier or prepaid registered or certified mail, return receipt requested, to: For Licensor: Supported Intelligence, LLC, 1555 Watertower Place, Ste. 300, East Lansing, MI 48823. For you: the contact information you supplied upon purchase of a license in the Software. All such notices shall be effective when received.

7.5. Injunctive Relief. It is understood and agreed that, notwithstanding any other provisions of this Agreement, your breach of this Agreement will cause Licensor irreparable damage for which recovery of money damages would be

inadequate, and that Licensor shall therefore be entitled to obtain timely injunctive relief to protect Licensor's rights under this Agreement in addition to any and all remedies available at law.

7.6. No Agency; Limitation on Action. Nothing contained in this Agreement shall be construed as creating any agency, partnership, or other form of joint enterprise between the parties. All legal actions against Licensor must be filed with the appropriate judicial jurisdiction within two (2) years after the cause of action first arose.

7.7. Waiver; Severability. The failure of either party to require performance by the other party of any provision hereof shall not affect the full right to require such performance at any time thereafter; nor shall the waiver by either party of a breach of any provision hereof be taken or held to be a waiver of the provision itself. All rights and remedies, whether conferred by this Agreement or by any other instrument or by law, shall be cumulative and may be exercised singularly or concurrently. If any provision of this Agreement is held unenforceable or invalid by any law, rule, order, or regulation of any government or by the final determination of any court of competent jurisdiction, such unenforceability or invalidity shall not render this Agreement unenforceable or invalid as a whole and, in such event, such provision shall be changed and interpreted so as to best accomplish the objectives of such provision within the limits of applicable law or applicable court decisions.

7.8. Vendor Selection. Licensor may retain certain third party service providers ("Vendors") to perform certain functions such as the processing of your license fee and the installation of the Software. You agree that Licensor bears no liability associated with the acts or omissions of such Vendors.

7.9. Headings. The Section headings appearing in this Agreement are inserted only as a matter of convenience and in no way define, limit, construe or describe the scope or extent of such Section, or in any way affect this Agreement.

7.10. Government Rights. If the Software is acquired under the terms of a Department of Defense or civilian agency contract, the Software is "commercial computer software" and "commercial computer software documentation," and as set forth in 48 C.F.R. 12.212 of the Federal Acquisition Regulations, and its successors, or 48 C.F.R. 227.7202 of the DoD FAR Supplement and its successors, as applicable. All U.S. Government end users acquire the Software and the Documentation with only those licenses and rights set forth in this Agreement.

7.11. Force Majeure. Neither party shall be liable by reason of any failure or delay in the performance of its obligations under this Agreement (except for the payment of money) on account of strikes, riots, insurrection, fires, flood, storm, earthquakes, explosions, war, acts or threats of terrorism, governmental action, labor conditions, material shortages, or any other cause, which is beyond the reasonable control of such party.

7.12. Survival. The following provisions will survive any termination or expiration of this Agreement: 1.3 ("Limitations on Use"), 1.4 ("Intellectual Property Rights"), 4 ("Confidential Information"), 5.2 ("No Liability of Suppliers"), 5.3 ("Disclaimer of Warranties"), 6 ("Limitation of Liability"), and 7 ("General").

7.13. Export Restrictions. You acknowledge that the laws and regulations of the United States restrict the export and re-export of certain commodities and technical data of United States origin, including the Software and the Documentation, in any medium. You agree that you will not export or re-export the Software or the Documentation in any medium without the appropriate United States and foreign government licenses. You further acknowledge that you are knowledgeable about U.S. export licensing requirements or that you will become so prior to engaging directly or indirectly in any export transaction including the Software and/or Documentation. You hereby agree to comply with the requirements of the U.S. Foreign Corrupt Practices Act (the "Act") and shall refrain from any payments to third parties that would cause you or Licensor or Licensor's licensors to violate the Act.

8.0 Licensed Libraries.

8.1. Use of Software Libraries. The Software uses several software libraries for which the source code was provided through a variety of Suppliers. The Suppliers, licenses and libraries are as follows:

8.2. BSD Licensed software from the Mathworks File Exchange. This product includes software developed using source code from BSD-licensed products obtained from the Mathworks File Exchange, found at: http://www.mathworks.com/matlabcentral/fileexchange, with license ID numbers 14317, 9909, 25786, 31272, and 45172, 29027. Supported Intelligence LLC wishes to acknowledge the original contributions toward these libraries of the following authors (with copyright dates): Yair Altman (2009), The MathWorks, Inc.(2009), INRA (2009), Jan Simon (2011), Kyle Andringa (2014), and Dirk-Jan Kroon (2010). The BSD license requires the inclusion of the following notice:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution:

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The BSD license template is provided by the Open Source Initiative, found at: http://www.opensource.org.

8.3 Open-source and BSD-Licensed software from INRA. This product includes software developed using source code from versions 1, 2, and 3 of the MDP toolbox created by the Unité de Biométrie et Intelligence Artificielle de Toulouse of INRA in Toulouse, France. Supported Intelligence LLC wishes to acknowledge the original contributions of authors Iadine Chadès, Marie-Josée Cros, Frédérick Garcia, and Régis Sabbadin of INRA. The BSD license notice is printed above.

8.4 Open Source and MIT-Licensed Software (d3.legend). Copyright © 2012 Ziggy Johnson. This software is subject to the following permission notice: THIS SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND , EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

8.5 Original software. The Software includes original software created by founders, employees, and contractors of Supported Intelligence LLC, with whom Licensor has an employment, subcontract, license, or similar agreement. Licensor wishes to acknowledge the contributions of the following such persons: Patrick L. Anderson, who developed the original prototype for the Software in 2010; David Quach, the lead developer for versions 1.0 through 1.2 during 2012 and 2013; Jeff Johnson, the lead developer for version 1.3 during 2014; and Neal Anderson, the lead developer for version 1.5 during 2015.

9.0 Application Pack Addendum

9.1. Expansion of Above Agreement. The license agreement above is hereby extended in the following manner: the term "Software" also includes Application Packs distributed by Licensor, and the term "Solution Templates" also includes open- code Caller Procedures included in Application Packs.

Appendix B. Algorithms

Blackwell's sufficient conditions and existence theorem

Unique solutions to sequential decision problems can be shown to exist under a variety of broad conditions. One of these conditions, often referred to as "Blackwell's sufficient conditions", establishes that a unique solution to a sequential decision problem exists under "discounting" and "monotonicity" (Blackwell, 1965).

Value Function Iteration

Value function iteration (also referred to as successive approximations, over-relaxation, backward induction or pre-Jacobi iteration) is, accordingly many sources including Puterman (2005) and Powell (2007), perhaps the most widely used algorithm for solving sequential decision problems.

The value function iteration algorithm used in the Rapid Recursive® Toolbox takes the following steps:

1. Choose a finite integer $N \geq 1$, the maximum number of times this algorithm will iterate.

2. Set the iteration counter, $k = 0$, and choose an initial (arbitrary) value function $V^0(s')$ and tolerance $\epsilon > 0$.

3. Apply the Bellman operator to V^k by computing:

$$V^{k+1}(s) = \max_{a \in A(s)} \left\{ r(s,a) + \beta \sum_{s' \in S} P(s' \mid s,a) V^k(s') \right\}$$

4. (Stopping criterion) If $k + 1 = N$ or $\| V^{k+1} - V^k \| < \epsilon \frac{1-\beta}{2\beta}$, proceed to Step 5. Otherwise, increase k by 1 and go back to Step 3.

Note: The norm is defined as $\| V^{k+1} - V^k \| = \sup_{s \in S} | V^{k+1}(s) - V^k(s) |$.

5. The value function is V^{k+1} and the optimal policy for each state $s \in S$ is:

$$a^*(s) \in \arg \max_{a \in A(s)} \left\{ r(s,a) + \beta \sum_{s' \in S} P(s' \mid s,a) V^{k+1}(s') \right\}$$

It turns out that under certain conditions—specifically when $0 < \beta < 1$—if the maximum iteration count N is chosen large enough, this value function iteration algorithm will produce a policy that is within ϵ of optimal (an ϵ-optimal policy), meaning its corresponding value function is within ϵ of the true value function.

Further, the algorithm will reliably converge for any initial value function V^0 selected from a reasonably large class of functions. The convergence occurs monotonically: if the starting guess V^0 is above the optimal solution, each successive update V^{k+1} will be less than the previous V^k until optimality is achieved. Conversely, if the initial guess is below the true

solution, each V^{k+1} will be greater than V^k at every iteration, approaching the optimal value from below.

Discretization and truncation

Although convergence of this value function iteration algorithm to the solution can be shown to occur for a broad state space, numerical use of the algorithm is only practical when s is finite. Similarly, numerical use of the algorithm is only practical when the action set is finite. Accordingly, the value function iteration algorithm in the Rapid Recursive® Toolbox requires both a finite set of states and a finite set of actions.

Non-finite sets can be turned into a finite set using discretization or truncation. A state space that is a continuous interval on the real number line from a to β can be discretized by creating a state space that consists of, for example, equally spaced points on the interval from a to β. Similarly a state space that is a continuous interval from *aa* to positive infinity can be truncated by placing an upper bound on the interval.

Policy Iteration

Policy iteration provides an efficient alternative to value function iteration for infinite time problems.

The policy iteration algorithm used in the Rapid Recursive® Toolbox takes the following steps:

1. **Choose a finite integer $N \geq 1$, the maximum number of times this algorithm will iterate.**
2. **Set the number of iterations, $k = 0$, and choose an initial (arbitrary) policy a_0.**
3. **Given a_k, express $r(s, a_k)$ and $\sum_{s'} P(s' \mid s, a_k)$ in matrix notation as r_k and P_k respectively.**
Perform the policy evaluation step to find V_k by solving the following system of equations:

$$(I - \beta P_k)V_k = r_k$$

4. **Perform the policy improvement step by finding a_{k+1}:**

$$a_{k+1}(s) \in \arg \max_{a \in A(s)} \left[r(s,a) + \beta \sum_{s'} P(s' \mid s, a) V_k(s') \right]$$

(It is possible for $a_{k+1} = a_k$).

5. **(Stopping criterion) If $k + 1 = N$ or $a_{k+1} = a_k$, stop. Otherwise, increase k by 1 and go back to Step 3.**

Step 3 of the policy iteration algorithm is referred to as the "policy evaluation" step and involves solving a system of linear equations. The Rapid Recursive Toolbox offers two

methods for this: Gaussian elimination with partial pivoting or the Jacobi iterative method, allowing flexibility based on user needs or computational constraints.

It can be shown that policy iteration will converge when the state and action sets are finite and the maximum iteration count N is selected sufficiently large. While numeric applications are only practical for finite sets, the algorithm's convergence properties also extend to broader classes—convergence can occur even with more general state and action sets, provided that the argmax in step 4 is well-defined. Thus, policy iteration is robust and widely usable for dynamic decision problems across a range of model complexities.

A similar monotone convergence result also holds for policy iteration under complete generality: the value function obtained in Step 3, V^{k+1}, is always higher than the value function obtained at the previous iteration V^k, i.e., $V^{k+1} \geq V^k$.

Again, for practical considerations, the policy iteration algorithm in the Rapid Recursive® Toolbox requires both a finite set of states and a finite set of actions.

Appendix C. References

Anderson, Patrick L. (2013). *The Economics of Business Valuation and Finance: Towards a Value Functional Approach.* Stanford University Press.

Blackwell, David (1965). "Discounted Dynamic Programming." *Annals of Mathematical Statistics* 36: 226–234.

Ljungqvist, Lars, and Thomas J. Sargent (2000 [2004]). *Recursive Macroeconomic Theory.* MIT Press. Lucas, Robert E., Jr., and Nancy L. Stokey (1989). *Recursive Methods in Economic Dynamics.* Harvard University Press.

Powell, Warren B. (2007). *Approximate Dynamic Programming*, John Wiley & Sons.

Puterman, Martin L. (2005). *Markov Decision Processes*, John Wiley & Sons.