Dynamic Programming Models and Algorithms for the Mutual Fund Cash Balance Problem

Juliana Nascimento, Warren Powell
Department of Operations Research and Financial Engineering, Princeton University, Princeton, New Jersey 08540
{juliana.m.nascimento@kcc.com, powell@princeton.edu}

Fund managers have to decide the amount of a fund’s assets that should be kept in cash, considering the trade-off between being able to meet shareholder redemptions and minimizing the opportunity cost from lost investment opportunities. In addition, they have to consider redemptions by individuals as well as institutional investors, the current performance of the stock market and interest rates, and the pattern of investments and redemptions that are correlated with market performance. We formulate the problem as a dynamic program, but this encounters the classic curse of dimensionality. To overcome this problem, we propose a provably convergent approximate dynamic programming algorithm. We also adapt the algorithm to an online environment, requiring no knowledge of the probability distributions for rates of return and interest rates. We use actual data for market performance and interest rates, and demonstrate the quality of the solution (compared to the optimal) for the top 10 mutual funds in each of nine fund categories. We show that our results closely match the optimal solution (in considerably less time), and outperform two static (newsrunner) models. The result is a simple policy that describes when money should be moved into and out of cash based on market performance.

Keywords: mutual fund cash balance; approximate dynamic programming

History: Received May 18, 2007; accepted December 12, 2009, by Michael Fu, stochastic models and simulation. Published online in Articles in Advance March 23, 2010.

1. Introduction

Mutual fund managers have to determine how much cash to keep on hand, striking a balance between the cost of meeting redemption requests against the opportunity cost of holding cash. The academic literature typically ignores several dimensions of the problem, such as the characteristics of the demands for redemptions. For example, the mutual fund has to consider both individual and institutional investors. Not only are redemption requests by institutional investors much larger, they have to be satisfied right away, producing short-term borrowing costs if there is insufficient cash. The manager has to consider transaction costs and bid-ask spreads. He also has to take into account whether the market is under- or over-performing long-term averages, as well as the pattern of deposits and redemptions that are correlated with both market performance and interest rates.

In Wermers (2000) and Edelen (1999), the authors present empirical evidence that supports the value of active fund management, as expenses and transaction costs are considerable factors in the fund net returns. In particular, these papers point out the significant cost of holding cash in a mutual fund.

Variations of the cash balance problem have been studied in the academic community in different forms. A survey of deterministic cash balance problems can be found in Srinivasan and Kim (1986). The deterministic problem assumes demands, market returns, and interest rates are known, but introduces issues such as fixed costs for a transaction and the modeling of continuous time processes. A similar deterministic version of this problem is the cash balance problem based on Sethi and Thompson (1970) and Sethi (1973), and later described in Sethi and Thompson (2000). The problem is represented by a system of differential equations and the optimal policy is determined studying the associated Hamiltonian and adjoint functions. Golden et al. (1979) and Jorjani and Lamar (1994) propose network flow optimization models to deal with efficient management of cash, investments, and short-term loans. One of the earliest investigations into stochastic cash management is given in Eppen and Fama (1971) (see also Eppen and Fama 1969), but these inventory-theoretic models of cash balance do not model the “state of the world” captured by market performance and interest rates (Bensoussan et al. 2009 is a more modern example of this). Schmid (2002) uses stochastic programming where uncertainty is represented in scenario trees. This approach can handle other types of information, but the scenario trees grow exponentially with the number of time periods, requiring complex algorithmic strategies.


A related problem is corporate cash holding, which is addressed in Kim et al. (2001), Almeida et al. (2004),
and Faulkender and Wang (2006). These papers present a qualitative analysis indicating that the more volatile the cash flow and the the higher the cost of finance, the more cash holding is observed. Almeida et al. (2004) also analyzes cash flow sensitivity.

A cash holding problem specifically applied to mutual funds is discussed in Yan (2006). Like the papers on corporate cash holdings, this paper only deals with qualitative aspects. The author develops a simplified static model for the problem and then proceeds with a regression analysis on actual mutual fund data that validates the model predictions. The author only hints, based on Constantinides (1986), that the holding of cash should be kept within a certain range and it should be adjusted only when the cash level is too low or too high.

Hinderer and Waldmann (2001) present a broader cash management model that includes a basic model of the mutual fund cash holding problem as a special case. The authors propose a dynamic programming formulation but do not address the computational problems that arise as a result of the curse of dimensionality.

The mutual fund cash holding problem can be viewed as a more complex inventory control problem (Graves et al. 1993). The complications arise because of the presence of side variables not present in a standard inventory context, such as interest rates and market rates of return. Even inventory problems with one-dimensional state spaces may lead to prohibitive computational requirements, because of the cardinality of the state space. In the context of a single-item stochastic lot-sizing problem, Halman et al. (2009) develops approximation algorithms to deal with it.

A related problem in the economic literature is the commodity storage problem (Wright and Williams 1982, 1984). This problem assumes an infinite planning horizon where a crop is harvested each year and a decision regarding the allocation between consumption and changes in inventory must be taken at each period. Both a dynamic programming analysis and a competitive equilibrium analysis are discussed in Judd (1998).

We provide a model of the mutual fund cash management problem that captures several dimensions of a realistic setting. In fact, our model is based on an e-mail sent by an actual fund manager describing the issues he was facing and that he could not find the answers in the literature to approach them. We propose both finite and infinite horizon models. We then describe several methods for solving these models: (a) we give two solutions based on newsvendor models suggested by the mutual fund manager in his e-mail, (b) we give an exact algorithm using backward dynamic programming (the most detailed version requires three days to solve), and (c) we provide an approximate dynamic programming (ADP) algorithm, which is based on the successive projective approximation routine (SPAR) for maintaining convexity of the value function. Our ADP algorithm, dubbed the “SPAR-Mutual” algorithm, is based on a procedure described in Powell et al. (2004) that adaptively learns a piecewise linear function giving the value of cash as a function of market return and interest rates. The SPAR-Mutual algorithm is provably convergent for finite horizon problems (the proof is given in Nascimento and Powell 2009), without requiring the use of an explicit exploration strategy. The algorithm can be used in a model-based implementation (where we assume that we know the distribution of market returns and interest rates) over a finite horizon or a model-free implementation if we are willing to assume steady state, using actual (rather than simulated) observations of the exogenous information.

2. Problem Description and Assumptions

Our problem is to determine how much cash to keep on hand to strike a balance between having enough cash to handle redemptions and maximizing the return on invested assets. We consider both finite and infinite horizon versions of this problem. The objective is to minimize discounted expected costs, where costs include early liquidation costs (if we have to liquidate assets to meet a redemption request), borrowing costs (if a redemption request has to be satisfied before assets can be liquidated), transaction costs, and the lost return on funds that are not invested in the market. During periods of market decline, the lost return is negative. We disregard the use of cash to pay management fees and other expenses and to make dividends and capital gains distributions, because they are typically a fixed percentage of the total assets under consideration.

There are two types of shareholders, namely, large institutional investors and small retail customers. We denote by \( D_i^t \) and \( D_r^t \) the demand for redemption at time \( t \) from institutional (large) and retail (small) investors, respectively. We denote by \( D_i \) the inflow of money from new investors, treated as a single, aggregate quantity. We assume that the stochastic processes describing \( D_i^t \), \( D_r^t \), and \( D_i \) are Markovian. Moreover, they are integer, bounded, and positive.

We denote by \( R \) the cash level at period \( t \) after new deposits \( D_i^t \) have been added. If the total demand for redemption \( D_i^t + D_r^t \) at \( t \) is larger than \( R \), then part of the fund portfolio must be liquidated to meet the demand. The cost involved will be denoted by \( \rho^h \), the shortfall cost, given by a deterministic fraction of the liquidating amount. We note that we can easily accommodate increasing costs for larger transactions,
which might arise from the cost of liquidating illiquid assets. If the institutional demand alone is higher than \( R_i \), then a financial cost (interest rate), denoted by \( P_i^t \), is also charged, because the demand must be satisfied immediately through short-term loans while securities are liquidated (a process that can require several days). On the other hand, if too much cash is maintained, the portfolio is losing investment opportunities. This loss is measured using the fund’s portfolio rate of return, denoted by \( P_i^t \). The prices \( P_i^t \) and \( P_i^t \) are Markov processes (not necessarily independent) that are exogenous to the system. We also assume they have finite support. Continuous processes could be discretized appropriately to apply the algorithms.

We let \( W_t = (P_i^t, P_i^t, D_i^t, D_i^t, D_i^t) \) be the vector of exogenous information that becomes available at every time period \( t \). We assume that \( W_t \) is independent of the cash level. Moreover, \( (P_i^t, P_i^t) \) is conditionally independent of \( (D_i^t, D_i^t, D_i^t) \) given \( W_{t-1} \). That is, \( E[P_i^t, P_i^t | D_i^t, D_i^t, D_i^t, W_{t-1}] = E[P_i^t, P_i^t | W_{t-1}]E[D_i^t, D_i^t, D_i^t | W_{t-1}] \). We note that more general structures, other than the Markovian assumption, could be considered for the processes. As a consequence, we would have to augment \( W_t \) with additional features.

Knowing \( W_t \) and \( R_i \), the cash rebalancing decision \( x_t = (x_{t1}, x_{t2}) \) is made, where \( x_{t1} \) is the amount of money to transfer from the portfolio to the bank account, whereas \( x_{t2} \) is the amount of money to transfer from the bank account to the portfolio. There is a limit on the amount of the portfolio that can be liquidated in one period, so we impose the constraint \( 0 \leq x_{t1} \leq M \), where \( M \) is a deterministic bound. Normally, \( M \) would simply be chosen large enough that it never constrains the optimal solution, but it can also be justified by the 1940 Investment Company Act (http://www.sec.gov/about/laws.shtml#invcoact1940), which allows redemptions to be satisfied using shares of stock instead of cash if a redemption request is too large.

It is obvious that \( 0 \leq x_{t2} \leq \max(0, R_i - D_i^t - D_i^t) \). We denote by \( \pi(W_t, R_i) \) the feasible region for \( x_t = (x_{t1}, x_{t2}) \). The fund incurs transaction costs \( \rho_i^t \) for each dollar moved into or out of the fund, which means that total transaction costs are given by \( \rho_i^t(x_{t1} + x_{t2}) \). Moreover, we assume that the shortfall cost \( \rho_i^s \) is larger than the transaction cost \( \rho_i^t \). Finally, we assume that \( E[P_i^t | P_i^{t-1}] \) is positive and \( E[P_i^t | P_i^{t-1}] \) is greater than \(-\rho_i^s\) and \(-\rho_i^t\). Clearly, \( \rho_i^s \) and \( \rho_i^t \) must be positive.

We denote by \( R_i^t \) the cash level at the end of period \( t \) after the decision is taken. This means that \( R_i^t = \max(0, R_i - D_i^t - D_i^t) + x_{t1} - x_{t2} \) and \( R_{t+1} = R_i^t + D_i^t + D_i^t \). Note that because we assume that \( D_i^t, D_i^t, D_i^t \) are integer and bounded, then \( R_i, x_{t1}, x_{t2}, R_i^t \) are also integer and bounded. The state of the system before the decision is made is denoted by \( S_t = (W_t, R_i^t) \), whereas the state of the system after the decision is made is denoted by \( S_i^t = (W_t, R_i^t) \).

The one period cost is given by

\[
C_i(S_t, x_t) = \rho_i^s(D_i^t + D_i^t - R_i)1_{(D_i^t + D_i^t \geq R_i)} + P_i^t(D_i^t - R_i)1_{(D_i^t \geq R_i)} + P_i^t(R_i - D_i^t - D_i^t)1_{(D_i^t + D_i^t < R_i)} + \rho_i^t(x_{t1} + x_{t2}).
\]

The first term is the shortfall cost, whereas the second term is the financial cost. The third term is the opportunity cost and the last term is the transaction cost.

Let \( X_i^t(S_t) \) be a decision function that determines \( x_t \) given the information in \( S_t \). We assume that we have a family of functions \( X_i^t, \pi \in \Pi \). We consider both stationary and nonstationary policies in this paper, so for the purpose of properly representing nonstationary policies, the decision function is indexed by time. When we refer to a policy \( \pi \), we mean the decision function \( X_i^t \) for some policy \( \pi \in \Pi \). Later, we provide more specific meaning to a specific policy.

Our problem is to solve

\[
\inf_{\pi \in \Pi} \sum_{t=0}^{T} \gamma^t C_i(S_t, X_i^t(S_t)),
\]

where \( \gamma \) is a discount factor between 0 and 1. There are two strategies we can use to solve the problem (we test both in our experimental work). The first is an offline strategy where we use prior history to update our forecasts at each time period \( t \), from which we generate what is typically a nonstationary forecast of the future. For example, we may feel that a sudden drop in the market will be followed by a fairly fast return to normal levels over the next few days. We would implement such a strategy using a finite horizon model (\( T \) might be 5 or 10 days).

The second strategy is an online implementation where we assume that all processes are stationary. In this case, we would use an infinite horizon objective (\( T = \infty \)). We demonstrate how a steady-state model such as this can be implemented very easily, without requiring an explicit update to short-term forecasts.

3. Mutual Fund Cash Holding Models

We first introduce a dynamic programming model where decisions at time \( t \) consider their impact on the future. This model is compared to two static models that use myopic policies. The first one is a straightforward simplification of the dynamic model, in the sense that the only difference between them is that the value function will not play a role when a decision is taken. This way, we are able to measure the importance of the role of the value function, because it adds significant computational burden to solve the problem. The second static model, proposed by Yan (2006),
is even simpler. It does not consider transaction costs, and there is no distinction between the two types of demand.

The policy obtained from minimizing the cost is the same as the one obtained from maximizing the profit. Because the original approximate algorithm deals with profit maximization, we use this terminology when describing the models and algorithmic strategies.

### 3.1. The Dynamic Model

We start this section discussing the optimal value functions for the cash holding problem. In §2, we introduced the state of the system, at time $t$, before and after the decision is taken, denoted by $S_t = (W_t, R_t)$ and $S_t^* = (W_t, R_t^*)$, respectively. We let $V_t^*(S_t)$ be the optimal value of being in state $S_t$, where

$$V_t^*(S_t) = \max_{x \in \mathcal{X}(W_t, R_t)} -C_t(W_t, R_t, x)$$

$$+ \sup_{\pi \in \Pi} \mathbb{E} \left[ \sum_{t'=t}^T -\gamma^{t'-t} C_t(S_{t'}, X_{t'}^*(S_{t'})) \right] \{S_t, x\}.$$  

Similarly, we let $V_t^*(S_t^*)$ be the optimal value of being in the postdecision state $S_t^*$, where

$$V_t^*(S_t^*) = \sup_{\pi \in \Pi} \mathbb{E} \left[ \sum_{t'=t}^T -\gamma^{t'-t} C_t(S_{t'}, X_{t'}^*(S_{t'})) \right] \{S_t^*\}.$$  

Equivalently, the optimal value functions can be defined recursively. Using the value functions around the pre- and postdecision states, we break Bellman’s equation into two steps:

$$V_{t-1}(W_{t-1}, R_{t-1}) = \mathbb{E}[V_t^*(W_t, R_t) \{W_{t-1}, R_{t-1}\}]$$ (2)

$$V_t^*(W_t, R_t) = \max_{x \in \mathcal{X}(W_t, R_t)} -C_t(W_t, R_t, x)$$

$$+ \gamma V_{t+1}^*(W_t^*, R_t^*).$$ (3)

At time $t = T$, because it is the end of the planning horizon, $V_T^*(W_T, R_T^*) = 0$. At time $t = T-1$, for $t = T, \ldots, 1$, the value of being in any postdecision state $(W_{t-1}, R_{t-1})$ does not involve the solution of an optimization problem; it involves only an expectation, because the next predecision state $(W_t, R_t)$ depends only on the exogenous information that first becomes available at $t$, as in (2). On the other hand, the value of being in any predecision state $(W_t, R_t)$ at $t$ does not involve expectations, because the next postdecision state $(W_t, R_t^*)$ is a deterministic function of $W_t$; it requires only the solution of an optimization problem, as in (2).

If we substitute (3) into (2), we get

$$V_t^*(W_t, R_t) = \max_{x \in \mathcal{X}(W_t, R_t)} -C_t(W_t, R_t, x)$$

$$+ \gamma V_{t+1}^*(W_t^*, R_t^*) \{W_{t-1}, R_{t-1}\}.$$ (4)

For algorithmic reasons, throughout this paper, we only use (4), instead of (3) and (2), i.e., we only consider the value function around the postdecision state. Its main feature is the inversion of the optimization/expectation order in the value function formula. See Powell (2007, Chap. 4) for a complete discussion of postdecision states.

To simplify notation, we will just drop the superscript $x$ in the value function notation. We perform a qualitative analysis of (4) in order to provide insights about the optimal policy. This analysis is also the foundation for the SPAR-Mutual algorithm.

Even without computing the expectation in (4), given the integrality assumptions on $D_t^1$, $D_t^2$, and $D_t^3$, the function $V_t^*(W_{t-1}, \cdot)$: $[0, \infty] \rightarrow \mathbb{R}$ is piecewise linear with integer break points. Thus, disregarding its value at $(W_{t-1}, 0)$, the function can be identified uniquely by its slopes $(v_{t-1}^*(W_{t-1}, 1), v_{t-1}^*(W_{t-1}, 2), \ldots, v_{t-1}^*(W_{t-1}, R_{t-1}))$, where $R_{t-1}$ is the upper bound on the cash level $R_{t-1}^*$ and

$$v_{t-1}^*(W_{t-1}, R_{t-1}) = V_{t-1}^*(W_{t-1}, R_{t-1}) - V_{t-1}^*(W_{t-1}, R_{t-1} - 1).$$

Assuming the slopes are monotone decreasing in the cash level dimension, that is,

$$v_t(W_t, R_t^*) \geq v_t(W_t, R_t^* + 1),$$

for all states $(W_t, R_t^*)$, we give a simple and intuitive proof that the optimal cash holding policy only rebalances the cash holdings when the cash level goes outside a certain range. To describe the policy, we define the following ranges: $\mathcal{B}_1(W_t) = \{R: \gamma v_t(W_t, R) > \rho^{o}\}$, $\mathcal{B}_2(W_t) = \{R: -\rho^{o} \leq \gamma v_t(W_t, R) \leq \rho^{o}\}$, and $\mathcal{B}_3(W_t) = \{R: \gamma v_t(W_t, R) < -\rho^{o}\}$.

**Theorem 1.** Given a predecision state $(W_t, R_t^*)$, assume (5) for all possible cash levels. Regions $\mathcal{B}_1(W_t)$, $\mathcal{B}_2(W_t)$, and $\mathcal{B}_3(W_t)$ determine a decision rule for the optimization problem

$$\max_{x_t \in \mathcal{X}(W_t, R_t)} -C_t(W_t, R_t, x_t) + \gamma V_t(W_t, \bar{R} + x_{t+1} - x_{t+1}),$$

where

$$\bar{R} = \max(0, R_t - D_t^1 - D_t^2)$$

and

$$V_t(W_t, R) = \sum_{r=1}^R v_t(W_t, r).$$

The optimal policy is as follows.

**Rule 1.** If $\bar{R} \in \mathcal{B}_1(W_t)$, $x_{t+1}^* = \min(M, \max[x_{t+1}: \bar{R} + x_{t+1} \in \mathcal{B}_1(W_t)])$, $x_{t+2}^* = 0$.

**Rule 2.** If $\bar{R} \in \mathcal{B}_2(W_t)$, $x_{t+1}^* = 0$, $x_{t+2}^* = 0$.

**Rule 3.** If $\bar{R} \in \mathcal{B}_3(W_t)$, $x_{t+1}^* = 0$, $x_{t+2}^* = \max(\bar{R}, \min[x_{t+2}: \bar{R} - x_{t+2} \in \mathcal{B}_3(W_t)])$.  

Proof of Theorem 1. For any predecision state \((W_i, R_j)\), the optimal decision \(x_i^n\) for (6) is the same as the optimal decision \(x_i^*\) of the following linear problem:

\[
\begin{align*}
\max_{x_i, y_i, z_i} & -\rho' (x_{i1} + x_{i2}) + \gamma \sum_{i=1}^{M} v_i(W_i, \bar{R} + i) y_i \\
+ & \gamma \sum_{j=1}^{\bar{R}} v_j(W_i, \bar{R} - j)(1 - z_j) \\
\text{subject to} & \sum_{i=1}^{M} y_i = x_{i1}, \sum_{j=1}^{\bar{R}} z_j = x_{i2}, \ 0 \leq y \leq 1, 0 \leq z \leq 1.
\end{align*}
\]

Note that even without imposing integrality, each component of \(y\) and \(z\) is either equal to 0 or 1. Each \(y_i\) = 1, for \(i = 1, \ldots, M\), represents the decision to transfer one dollar from the portfolio to the bank account. On the other hand, each \(z_i\) = 1, for \(j = 1, \ldots, \bar{R}\) represents the transfer of one dollar from the bank account to the portfolio. Moreover, given (5), if for some \(i\), \(y_i' = 1\) then \(y_i'' = 1\) for all \(i' < i\), \(y_i'' = 0\) for all \(i' > i\) and the entire vector \(y\) is equal to 0. Furthermore, if for some \(j\), \(z_j' = 1\) then \(z_j'' = 1\) for all \(j' < j\), \(z_j'' = 0\) for all \(j' > j\) and the entire vector \(y\) is equal to 0. This follows the logical reasoning that if money is transferred from the portfolio to the bank account it makes no sense to transfer money from the bank account to the portfolio. Clearly, if \(\bar{R} \in \mathcal{R}_1(W_i)\), then the optimal vector \(z^*\) is equal to 0 and \(y_i' = 1\) for all \(i\) such that \(\gamma v_i(W_i, \bar{R} + i) > \rho''\). That is, we keep transferring money to the bank account while the transaction cost is smaller than the marginal value of having one more dollar in cash or until the upper bound \(M\) is reached. Symmetrically, if \(\bar{R} \in \mathcal{R}_2(W_i)\), then the optimal vector \(y^*\) is equal to 0 and \(z_i' = 1\) for all \(j\) such that \(\gamma v_j(W_i, \bar{R} - j) < -\rho''\), that is, we keep transferring money to the portfolio while the transaction cost is smaller than the marginal value of having one less dollar in cash or until there is no more cash to be transferred. Finally, if \(\bar{R} \in \mathcal{R}_3(W_i)\), then \(y_i'' = 0\) and \(z_i'' = 0\), since the transaction cost is too high to justify any transfer of money, proving Rules 1–3. □

We next give an expression for the optimal slopes and show that they are monotonically decreasing, a property that we exploit in the SPAR-Mutual algorithm. This property not only accelerates the rate of convergence of the algorithm, but it is also central to its pure exploitation nature, resulting in a simpler and faster procedure.

Theorem 2. For \(t = T, \ldots, 1\) and all states \((W_{t-1}, R_{t-1})\), the optimal slopes are given by

\[
v_{t-1}(W_{t-1}, R_{t-1}) = \mathbb{E}[\hat{G}(W_t, R_t, v_i^*) | (W_{t-1}, R_{t-1})],
\]

where

\[
\hat{G}(W_t, R_t, v_i^*) = \rho^{sh} \mathbb{I}_{[D_{t+1}^i + D_{t+1}^j \geq R_t]} + \mathbb{I}_{[D_t^j \geq R_t]} - \rho^{sh} \mathbb{I}_{[D_t^j < R_t]} + \mathbb{I}_{[D_{t+1}^i = R_t]} - \mathbb{I}_{[D_{t+1}^i < R_t]} - \rho^{sh} \mathbb{I}_{[D_{t+1}^j = R_t]} + \mathbb{I}_{[D_{t+1}^j < R_t]},
\]

and

\[
\max_{x_{i1}} \mathbb{E}[-C_{t+1}(W_{t+1}, R_{t+1}) + D_t^i] | (W_t, R_t),
\]

where

\[
R_t^i = \max(0, R_t - D_t^i - D_t^j + x_{i1} - x_{i2}).
\]

4. Algorithmic Strategies

In this section, we present algorithms to actually compute the decisions implied by each cash holding model. For the dynamic setting, once we compute the slopes of the value functions, a solution is easily determined following the decision rule described in Theorem 1. We propose two approaches to find the...
slopes. The first one is traditional backward dynamic programming. Even though this approach is simple and straightforward, it is computationally very demanding.

The second approach is through ADP. The ADP algorithm replaces the computation of the expected value by iteratively observing sample path realizations and, most importantly, by exploiting the structural property of the optimal slopes, namely, the monotone decreasing property. Of course, ADP replaces the computational burden of the exact solution with the statistical errors of a Monte Carlo based procedure. The upside is that the slopes do converge to the optimal ones in the limit. The idea is that if the number of iterations is large enough, the approximate solutions are very close to the optimal ones. The ADP approach is discussed in full in the next section.

The optimal decision for the static models is obtained in a similar fashion as the optimal decision for the dynamic model at time period $T-1$, because no downstream effects are taken into consideration at the end of the planning horizon $T$. Therefore, our procedure to determine the optimal decision for the static models follows the reasoning of the proof of Theorem 1, i.e., we make the decision comparing the transaction cost with the marginal value of transferring one dollar to/from the bank account.

Given the predecision state $(W_t, R_t)$, let $f_1(W_t, R_t^1)$ and $f_2(W_t, R_t^1)$ denote the marginal value of transferring one dollar to the bank account for the first and second static models, respectively, when the cash level is $R_t^1 = \max(0, R_t - D_t^1 - D^1_t) + x_{11} - x_{21}$. From (8) and (9), we can easily obtain

$$f_1(W_t, R_t^1) = E'P[D_{t+1}^1 + D^1_t - D^1_t = R_t^1 | (W_t, R_t)]$$

$$+ (\rho^{1\prime} + E')P[D_{t+1}^1 + R_t^1 \geq R_t^1 | (W_t, R_t)] - E'$$

$$f_2(W_t, R_t^1) = (E' + \rho^{1\prime} + E')P[D_{t+1}^1 \geq R_t^1 | (W_t, R_t)] - E'$$

where $E' = E[\rho^{1\prime} | W_t]$, $E' = E[\rho^{1\prime} | W_t]$, and $D_{t+1} = D_{t+1}^1 + D^{1\prime}_t - D^1_t$. Symmetrically, the marginal value of transferring one dollar from the bank account is given by $-f_1(W_t, R_t)$ and $-f_2(W_t, R_t)$, respectively.

The optimal decision for the first model can be found using the following procedure:

STEP 0: Initialize $x_{11} = x_{12} = 0$.

STEP 1: While $f_1(W_t, R_t^1) > \rho^{1\prime}$ and $x_{11} < M_t$ do $x_{11} = x_{11} + 1$.

STEP 2: While $f_1(W_t, R_t^1) < -\rho^{1\prime}$ and $x_{12} < \max(0, R_t - D^1_t - D^1_t)$ do $x_{12} = x_{12} + 1$.

Because there is no transaction cost involved in the second model, its solution is determined using a similar procedure, replacing $\rho^{1\prime}$ by 0 and $f_1(W_t, R_t^1)$ by $f_2(W_t, R_t^1)$.

5. The Approximate Dynamic Programming Approach

The main idea of the algorithm is to construct concave and piecewise linear function approximations $\tilde{V}_i^n(W_{r_t}, \cdot)$, its slopes $\tilde{v}_i^n(W_{r_t}, R_{r_t}^n)$, $\tilde{v}_i^n(W_{r_t}, R_{r_t}^n)$, $\tilde{v}_i^n(W_{r_t}, R_{r_t}^n)$ over the iterations. At each iteration, our decision function looks like

$$X^n_t(S^n_t) = \arg\max_{x_{11} < (W^n_t, R^n_t)} -C_x(S^n_t, x_{11}) + \gamma \tilde{V}_i^{n-1}(W^n_t, R^n_t)$$

where $\tilde{V}_i^{n-1}(W_{r_t}, \cdot)$ is the piecewise linear value function approximation computed using information up through iteration $n-1$. We now see that our policy is parameterized by the slopes $\tilde{v}_i^{n-1}(W_{r_t}, r)$, $r = 1, 2, \ldots$, for each possible value of $W_t$. Thus, when we refer to a policy $\pi$, we are actually referring to a specific set of slopes.

The catch is that the algorithm does not try to learn the slopes for the whole state space, but only for parts close to optimal cash levels, which are determined by the algorithm itself. Figure 1 illustrates the idea.

At each iteration $n$ and time $t$, instead of computing the expectation in (4), the algorithm observes one sample realization of the information vector. After that, the sample realization and the current value function approximation are used to take a decision $x^n_t$, leading the system to a postdecision state. Sample information around the new postdecision state is gathered and is used to update the approximate slopes $\tilde{v}_i^{n-1}$. A projection operation is then performed in case a violation of the concavity property occurs.

5.1. The SPAR-Mutual Algorithm

Before we present the algorithm, some notation is necessary. A general postdecision state at $t$ is denoted by $S^n_t$ or $(W^n_t, R^n_t)$. The two of them are used
STEP 0: Algorithm initialization:

STEP 0a: Initialize \( \hat{v}_n(R_n^t) \) for all \( t \) and \( (W_n, R_n) \) monotonically decreasing in \( R_n^t \).

STEP 0b: Pick \( N \), the total number of iterations.

STEP 0c: Set \( n = 1 \).

STEP 1: Planning horizon initialization: Observe the initial cash level \( R_n^t \).

Do for \( t = 0, \ldots, T \):

STEP 2: Sample/Observed \( p_t^{+, n}, p_t^{-, n}, D_t^{, n}, D_t^{-, n}, \) and \( F_t^{+, n} \).

STEP 3: Compute the predecision cash level: \( R_n^t = R_n^{t-1} + D_t^{, n} \).

STEP 4: Slope update procedure:

\[
\text{If } t > 0 \text{ then:}
\begin{align*}
\text{STEP 4a: Observe } \hat{v}_n'(R_n^{t-1}) \text{ and } \hat{v}_n''(R_n^{t-1}) + 1. \\
\text{STEP 4b: For all possible states } S_{n-1}:
\end{align*}
\]

\[
\hat{v}_n''(S_{n-1}) = (1 - \hat{v}_n''(S_{n-1}))(\hat{v}_n'(S_{n-1}) + \hat{v}_n''(S_{n-1})).
\]

STEP 4c: Perform the projection operation

\[
\hat{v}_n''(W_n, R_n^t) = \Pi_{H_e(W_n, R_n^t), \hat{v}_n''(S_{n-1})}(\hat{v}_n''(S_{n-1})).
\]

STEP 5: Find the optimal solution \( x^*_n \) of

\[
\max_{x_n} x_n(W_n, R_n^t) - C(S_n, x) + \gamma V_{n-1}(W_n, R_n^t).
\]

STEP 6: Compute the postdecision cash level:

\[
R_n^{t-1} = \max(0, R_n^t - D_t^{, n} + D_t^{-, n} + x_n^* - x_n^t).
\]

STEP 7: If \( n < N \) increase \( n \) by one and go to Step 1.

Else, return \( \hat{v}_n'' \).

interchangeably. We use \( S_{n-1}^t \) to denote the actual postdecision state visited by the algorithm at iteration \( n \) and time \( t \). The same notation convention holds for the predecision states. At iteration \( n \) and time \( t \), the actual decision taken by the algorithm is denoted by \( x_n^t \), whereas the value function approximation is denoted by \( \hat{V}_n^t(W_n, R_n^t) \). The corresponding slopes are denoted by \( \hat{v}_n''(W_n, R_n^t) \).

The SPAR-Mutual algorithm is presented in Figure 2. As described in Step 0, the algorithm inputs are piecewise linear value function approximations represented by their slopes \( \hat{v}_n''(W_n, R_n^t) \). The initial slopes must be decreasing in the cash level dimension. A slope vector that is equal to zero for all states and time periods is a valid input. Because we know that \( v_n''(W_n, R_n^t) = 0 \) for all states \( (W_n, R_n^t) \), then we use \( \hat{v}_n''(W_n, R_n^t) = 0 \) for all iterations \( n \).

At each iteration \( n \), the algorithm starts by observing the initial cash level, denoted by \( R_n^{t, n} \), as in Step 1. The initial cash level must be a positive integer. After that, the algorithm proceeds over time periods \( t = 0, \ldots, T \). At the beginning of time period \( t \), the algorithm generates a sample of the interest rate \( p_t^{+, n} \), rate of return \( p_t^{-, n} \), money inflow \( D_t^{, n} \), institutional redemption \( D_t^{-, n} \), and retail redemption \( D_t^{, n} \), as in Step 2. These are Monte Carlo samples following the probability distribution of the information process \( W_n \) given that \( W_{n-1} = (p_t^{+, n}, p_t^{-, n}, D_t^{, n}, D_t^{-, n}, D_t^{, n}) \). After that, the predecision cash level \( R_n^t \) is computed, as in Step 3.

Before the decision at time period \( t \) is taken, the algorithm uses the sample information to update the slopes of time period \( t-1 \). Steps 4a–4c describes the procedure, and Figure 3 illustrates it. We first observe slopes relative to the predecision states \( (W_{n-1}, R_{n-1}) \) and \( (W_{n-1}, R_{n-1} + 1) \) (see Step 4a and Figure 3(a)). Then, these sample slopes are used to update the approximation slopes \( \hat{v}_n''(S_{n-1}) \) through a temporary slope vector \( \hat{v}_n''(S_{n-1}) \) (see Step 4b and Figure 3(b)). This step requires the use of a step-size rule that is state dependent, denoted by \( \alpha_n(S_{n-1}) \). We have that \( \alpha_n(S_{n-1}) = \alpha_n^+ \frac{1}{(1 + r_n^{+})^1 + (1 + r_n^{-})^1} \), where \( 0 < \alpha_n^+ \leq 1 \) and \( \alpha_n^+ \) can depend only on information that became available up through iteration \( n \) and time \( t \). For example, it is valid to use \( \alpha_n^+ = 1/(N(S_{n-1}^t)) \), where \( N(S_{n-1}^t) \) is the number of visits to state \( S_{n-1}^t \) up until iteration \( n \). The updating scheme may produce slopes that violate the property of being monotonically decreasing. In this case, a projection operation is performed to restore the property and obtain the updated approximation slopes \( \hat{v}_n''(S_{n-1}) \) (see Step 4c, illustrated in Figure 3(c)).

Next, a decision \( x_n^t \) is made given the current state at time \( t \). This decision is the optimal solution with respect to the current predecision state \( (W_n^t, R_n^t) \) and value function approximation \( \hat{V}_n^{t-1}(W_n^t, R_n^t) \), as stated in Step 5. This decision can be easily calculated following the decision rule described in Theorem 1. We just need to consider the current predecision state \( (W_n^t, R_n^t) \) and the current slope approximation \( \hat{v}_n''(W_n^t) \).

Finally, the postdecision state \( R_n^t \) is computed, as in Step 6, and we advance the clock to time \( t + 1 \). As the algorithm reaches the planning horizon \( t = T \), if the number of iterations has not reached its limit \( N \), then the iteration counter is incremented, as in Step 7, and a new iteration is started from Step 1. Otherwise, the algorithm is finished returning the current slope approximation \( \hat{v}_n''(W_n^t, R_n^t) \) for all \( t \) and \( (W_n^t, R_n^t) \)

We obtain sample slopes by replacing the expectation and the optimal slope \( \hat{v}_n''(W_n^t, R_n^t) \) by the sample realization \( w_n^t \) and the current slope approximation \( \hat{v}_n''(S_{n-1}) \), respectively. Thus, for \( t = 1, \ldots, T \), the sample slope is \( \hat{v}_n''(W_n^t, R_n^t) = \hat{G}(W_n^t, R_n^t, \hat{v}_n''(S_{n-1})) \).

The projection operator \( H_{H_e}, \hat{v}_n''(S_{n-1}) \) maps a vector \( z_n^t \), that may not be monotone decreasing in the cash level dimension, into another vector \( \hat{v}_n''(S_{n-1}) \) that has this structural property. The operator imposes the property by forcing the newly updated slope at \( (W_n^t, R_n^t) \) to be greater than or equal to the newly updated slope at \( (W_n^t, R_n^t + 1) \) and then forcing the other violating slopes to be equal to the newly updated ones. For any
Figure 3  Slopes Update Procedure, Where $F_{t-1}(S_{t-1}^n, V_{t-1}^n, x) = -C_{t-1}(S_{t-1}^n, x) + \gamma V_{t-1}^{n-1}(W_{t-1}^n, R_{t-1}^n)$

(a) Current approximate function, optimal decision and sampled slopes

(b) Temporary approximate function with violation of concavity

(c) Level projection operation: updated approximate function with convexity restored

state $(W_t, R_t^z)$, the projection is given by

$$
\Pi_{\psi, W_t^n, R_t^{z,n}}(z)(W_t, R_t^z) = \begin{cases} 
\frac{z(W_t^n, R_t^{z,n}) + z(W_t^n, R_t^{z,n} + 1)}{2} & \text{if C1,} \\
\Pi_{\psi, W_t^n, R_t^{z,n}}(z)(W_t^n, R_t^{z,n}) & \text{if C2,} \\
\Pi_{\psi, W_t^n, R_t^{z,n}}(z)(W_t^n, R_t^{z,n} + 1) & \text{if C3,} \\
z(W_t, R_t^z) & \text{otherwise,}
\end{cases}
$$

(10)

where the conditions C1, C2, and C3 are

C1:  $W_t = W_t^n$,  $R_t^z = (R_t^{z,n} \text{ or } R_t^{z,n} + 1)$,

$z(W_t^n, R_t^{z,n}) < z(W_t^n, R_t^{z,n} + 1)$;

C2:  $W_t = W_t^n$,  $R_t^z < R_t^{z,n}$,

$z(W_t^n, R_t^z) \leq \Pi_{\psi, W_t^n, R_t^{z,n}}(z)(W_t^n, R_t^{z,n})$;

C3:  $W_t = W_t^n$,  $R_t^z > R_t^{z,n} + 1$,

$z(W_t^n, R_t^z) \geq \Pi_{\psi, W_t^n, R_t^{z,n}}(z)(W_t^n, R_t^{z,n} + 1)$. 
### 5.2. Convergence Analysis

We state two theorems that prove that the algorithm converges to an optimal policy, that is, the algorithm does learn the optimal decision to be taken at each state that can possibly be reached by an optimal policy. The proofs are based on more general results in Nascimento et al. (2007); hence, only a sketch is provided here.

We start with some remarks. We denote by \(\{S^t\}_{t=0} = \{(W^n_t, R^n_t)\}_{t=0}^{\infty}\) the sequence of state visited by the algorithm at time \(t\). Likewise, we denote by \(\{x^t\}_{t=0}^{\infty}\) the sequence of decisions taken by the algorithm, and by \(\{(S^t_n)\}_{t=0}^{\infty} = \{(W^n_t, R^n_t)\}_{t=0}^{\infty}\) the sequence of postdecision states visited by the algorithm. Each one of these sequences has at least one accumulation point. This result derives from the fact that the information process is stationary. We denote by \(\bar{\alpha}^t(W^t, R^t)\) an accumulation point of this sequence.

**Theorem 3.** On the event that \((W^t, R^t)\) is an accumulation point of \(\{(W^n_t, R^n_t)\}_{t=0}^{\infty}\), if
\[
\sum_{t=0}^{\infty} \bar{\alpha}^t(W^t, R^t) = \infty
\]
and
\[
\sum_{t=0}^{\infty} (\bar{\alpha}^t(W^t, R^t))^2 < B < \infty \quad a.s.,
\]
then
\[
\bar{\alpha}^t(W^t, R^t) \to \alpha^*(W^t, R^t)
\]
and
\[
\bar{\alpha}^t(W^t, R^t + 1) \to \alpha^*(W^t, R^t + 1) \quad a.s. \quad (11)
\]

As a byproduct of the previous theorem, we obtain the next theorem:

**Theorem 4.** For \(t = 0, \ldots, T\), on the event that \((W_t, R_t, \bar{\alpha}^t, x_t)\) is an accumulation of \(\{(W^n_t, R^n_t, \bar{\alpha}^t_{n-1}(*))\}_{n=1}^{\infty}\), if the step-size condition of Theorem 3 is satisfied, then, with probability one, \(x^t\) is an optimal solution of
\[
F_t(W^t, R^t, V^t, x_t), \quad x_t
\]
\[
= -C_t(W^t, R^t, x_t) + \gamma V^t
\]
\[
+ (W^t, \max(0, R^t - D^t* - D^t*) + x_{t1} - x_{t2}). \quad (12)
\]
Equation (12) implies that the algorithm has learned an optimal decision for all states that can be reached by an optimal policy. This implication can be easily justified as follows. We start with \(t = 0\). For each accumulation point \((W^t, R^t)\) of \(\{(W^n_t, R^n_t)\}_{t=0}^{\infty}\) (12) tells us that the accumulation points \(x^t_0\) of \(\{x^t\}_{t=0}^{\infty}\) along the iterations with initial predecision state \((W^t, R^t)\) are in fact an optimal decision for period 0 when the predecision state is \((W^t, R^t)\). This implies that all accumulation points \(R^t_0 = \max(0, R^t - D^t* - D^t* + x^t_0)\) of \(\{R^t_0\}_{t=0}^{\infty}\) are postdecision cash levels that can be reached by an optimal policy. When \(t = 1\), for each \(R^t_1*\) and each accumulation point \(W^t\) of \(\{W^t\}_{t=0}^{\infty}\), \(R^t_1 = R^t_1* + D^t*\) is a predecision cash level that can be reached by an optimal policy. Once again, (12) tells us that the accumulation points \(x^t_1\) of \(\{x^t\}_{t=0}^{\infty}\) along the iterations with \((W^t, R^t) = (W^t, R^t)\) are indeed an optimal decision for period 1 when the predecision state is \((W^t, R^t)\). As before, the accumulation points \(R^t_1* = \max(0, R^t - D^t* - D^t* + x^t_1)\) of \(\{R^t_1\}_{t=0}^{\infty}\) are postdecision cash levels that can be reached by an optimal policy. The same reasoning can be applied for \(t = 2, \ldots, T\).

### 6. The Infinite Horizon Cash Holding Problem

The infinite horizon problem arises when we believe that the exogenous information processes (prices, interest rates, deposits, and redemption) are stationary. The only change is that we drop the indexing by time of all variables. We adjust the algorithms to the infinite horizon case in a fairly straightforward way. For the static models, we use the procedure described in §4, dropping the time index.

The ADP algorithm for the infinite horizon case is similar to the SPAR-Mutual algorithm described in Figure 2, except that we do not loop over the different time periods and again the time index is dropped. The modified SPAR-Mutual algorithm for the infinite horizon method brings a nice twist. It can be considered an online algorithm in the sense that probability distributions do not need to be known or estimated. For the finite horizon case, the sample realizations for the interest rate, rate of return, cash inflow, money, and demand for redemption can be used for the infinite horizon case, without any need to estimate the probability distribution underlying them. Moreover, as new daily information becomes available, it can be used to update the current slope approximations. We do not have a proof of convergence for this online algorithm, but we show that the policies produced by this algorithm outperform the static ones.
7. Numerical Experiments

The purpose of this section is to analyze the behavior of the different algorithmic approaches. We study the effect of discretization on central processing unit (CPU) time and solution quality for the exact and the ADP algorithms. Moreover, we quantify how much we gain by considering the impact a decision has on the future, instead of just using a myopic policy.

Finally, for the infinite horizon case, we can observe the behavior of an online algorithm and the errors incurred in constructing probability distributions and estimating their parameters.

We want to make sure we compare the algorithmic strategies when they are applied to realistic environments. To achieve this goal, the probability distributions are estimated using real data, including bank prime loan rates, rates of return of top-performing funds, total asset values, and redemption rates of a broad range of funds.

We start by describing the instances considered and the data used to construct them. After that, we discuss implementation details. We then present and analyze the results for the finite and infinite horizon cases.

7.1. Problem Instances

We start by describing the instances considered and the data used to construct them. After that, we discuss implementation details. We then present and analyze the results for the finite and infinite horizon cases.

We identify each set using the first letters of the corresponding category and style. For example, LB represents the set of large blend funds.

Daily rates of return (from July 2001 to June 2006) for each fund were collected from the Center for Research on Security Prices (CRSP) Mutual Fund Database (http://www.crsp.com/) and were averaged across the 10 funds on each style/category. The resulting daily rates and the MLE method were used to estimate the parameters of the Vasicek process.

For each time period $t$, we assume that the arrival of shareholders investing (redeeming) money follows a compound Poisson process with rate $\lambda_t^i$ ($\lambda_t^r$). Moreover, $s_i\%$ of these shareholders are institutions investing $d_i$ (redeeming $d_i$) units of dollars, whereas $100-s_i\%$ are retail shareholders investing (redeeming) one unit of dollar. Data from 8,460 different mutual funds that have both institutional and retail investors and that did not have any mergers or acquisitions from July 2005 to June 2006 were collected from the CRSP Mutual Fund Database. They show that it is reasonable to consider $\lambda_t^i$ ($\lambda_t^r$) as a linear function of the rate of return $P_t^i$; see Figure 4(a).

The redemption rate, as reported by the Investment Company Institute website (http://www.ici.org/), is calculated by taking the sum of regular redemptions and redemption exchanges for a given period as a percentage of the average total net assets at the beginning and end of this period. Considering a group of 4,623 stock funds, the redemption rate for the July 2005–June 2006 period was 25.2%. Therefore, we approximate the redemption amount for each instance over this period by multiplying their total net assets by the redemption rate. The resulting redemption amount is displayed in Figure 4(b). Because the cash holding problem objective value is directly related to the redemption amount involved, we can use the numbers in Figure 4(b) to quantify the actual amount of dollars that can be saved when each algorithmic strategy is employed to produce a policy.

Finally, we picked the transaction cost $\rho^t$ and the shortfall cost $\rho^s$ to be equal to 0.1% and 0.2%, respectively. For the finite horizon case, the planning horizon is set to $T=5$ working days.

7.2. Implementation Issues

One of our goals is to compare the performance of an ADP-based algorithm to an exact solution obtained using traditional backward dynamic programming methods. This requires that we address the fact that the interest rate and rate of return processes are unbounded and continuous, but an exact solution requires that they be bounded and discrete.

To obtain an exact solution, we had to create a discretized version of the interest rate and return processes. We did this by first discretizing and truncating...
the exogenous changes to interest rate and return processes. Using this modified process, we created a probability distribution for these processes that reflected both the discretization and the truncation. We then chose the finest level of discretization that could be solved with reasonable execution times (we allowed run times to span days) and defined this to be the exact solution. This solution could then be compared to solutions obtained using backward dynamic programming for coarser levels of aggregation, as well as solutions obtained using the SPARMutual algorithm.

When using the SPARMutual algorithm, we simulated the original continuous state (that is, the interest rate and returns—movements of cash were always assumed to occur in discrete quantities). Sample observations of changes in interest rates and returns were generated by either using the original continuous distributions or sample observations from history (without discretization). Only the value function approximation was discretized. This strategy allowed us to estimate the errors produced using approximate dynamic programming in a realistic setting, but we still compared our performance against an “exact” model that assumed a very fine level of discretization.

A second important implementation issue was the choice of the right step-size rule, which is a key ingredient to faster rates of convergence. Even though the step-size rule \( \alpha_n = 1/N(S_{t,n}^r) \) produces a provably convergent algorithm, the associated rate of convergence is poor, because this rule goes to zero too quickly. The problem is that while learning the slope for one state, the updating procedure depends on another slope approximation, which is steadily changing, because the algorithm is also learning this other slope, leading to a nonstationary process. Moreover, the rate of convergence for each slope is different and, ideally, the step-size rule would reflect that. For example, the slower the convergence of a given slope, the larger its step size should be. Because we cannot tune a step-size rule for each slope, an adaptive step-size rule works best. In our implementation, we use the adaptive step-size rule proposed in George and Powell (2006). It is given by \( \alpha^*_t = 1 - (\tilde{\delta}_t^*(S_{t,n}^r))^2/(\tilde{\delta}_t^m(S_{t,n}^r)) \), where \( n \) is the number of visits to state \( S_{t,n}^r \) up to iteration \( n \). The variance \((\tilde{\delta}_t^*(S_{t,n}^r))^2\) is an estimate of the variance of the observation error, and \( \tilde{\delta}_t^m(S_{t,n}^r) \) is an estimate of the total squared variation between the observation and the estimate, which includes both bias (if our estimates are consistently high or low) as well as the observation error. These are computed using

\[
\tilde{\delta}_t^m(S_{t,n}^r) = (1 - \nu_{m-1}) \tilde{\delta}_t^{m-1}(S_{t,n}^r) + \nu_{m-1}(\tilde{\gamma}_t^{m-1}(S_{t,n}^r) - \tilde{\delta}_t^{m-1}(S_{t,n}^r))^2 ;
\]

\[
\tilde{\beta}_t^m(S_{t,n}^r) = (1 - \nu_{m-1}) \tilde{\beta}_t^{m-1}(S_{t,n}^r) + \nu_{m-1}(\tilde{\nu}_t^{m-1}(R_{t,n}^r) - \tilde{\beta}_t^{m-1}(S_{t,n}^r))^2 ;
\]

\[
(\tilde{\delta}_t^*(S_{t,n}^r))^2 = \frac{\tilde{\delta}_t^*(S_{t,n}^r) - (\tilde{\beta}_t^*(S_{t,n}^r))^2}{1 + \lambda_t^{m-1}(S_{t,n}^r)}.
\]

Here, \( \nu_m = 10/(10 + m) \) is a step-size formula.

The other ingredient to faster rates of convergence is the choice of discretization increments. It is intuitive to expect that coarser discretization increments (smaller state space) lead to faster rates of convergence in the initial iterations but poorer results in the long run, whereas finer discretization increments (larger state space) lead to slower rates of convergence in the initial iterations but more accurate results in the long run. The reasoning behind this intuition is that, when the discretization is fine, in the initial iterations, most of the state space (in the information

---

**Figure 4 Net Flow and Redemptions**

(a) Average net flow

(b) Approximate redemption amount

*Note.* LB, Large blend; LG, large growth; LV, large value; MB, mid blend; MG, mid growth; MV, mid value; SB, small blend; SG, small growth; SV, small value.
dimension $W_t$) does not have enough observations to produce a reasonable slope approximation. Because the SPAR-Mutual algorithm depends on the slope approximation to make a decision and this decision determines the further course of the algorithm, the rate of convergence can suffer from this lack of information.

We start the SPAR-Mutual algorithm (for the infinite horizon case) considering a coarse discretization increment, for the rate of return. Then, after $N_1$ iterations, we switch the discretization increment, increasing the state space by a factor of two. We repeat the same procedure after $N_2$ and $N_3$ iterations. The values of $N_1$, $N_2$, and $N_3$ are a rough estimate of the number of iterations necessary to observe a wide spectrum of values of $W_t$.

We close this section describing how the numerical experiments were conducted. All the algorithms were implemented in Java on a 3.06 GHz Intel P4 machine with 2 GB of memory running Linux. To evaluate the policies generated by the different algorithmic strategies, we created, for each instance, a unique testing set. For the finite horizon case, the testing set consists of 1,000 different sample paths that were randomly generated following the processes described in §7.1. For the infinite horizon case, the testing set consists of actual daily prime rates and rates of return from July 2005 to June 2006. It is worth mentioning that the testing set is not used as part of the sample data required to learn the slopes.

Unless otherwise noted, we adopt as optimal the policy obtained using traditional dynamic programming with discretization increments 0.001 and 0.0001 for the interest rate and rate of return, respectively. For $i = 1, \ldots, 1,000$, let $F^i$ be the value of following policy named $\pi$ for the $i$th sample path $\omega_i$ in the testing set, given by

$$F^i_t = \sum_{t=0}^{T} \gamma^t C_t(S_t(\omega_t), X_t^\pi(S_t(\omega_t))).$$

Let $F^i_t$ be the value of following the optimal policy labeled by $\pi^*$. Moreover, when the approximate algorithms are considered, we add the superscript $n$ to the notation, indicating that $\hat{F}^n_t$ is measuring the policy obtained after $n$ iterations of the algorithm. To take into account the randomness involved in the approximate approaches, the policy considered to obtain $\hat{F}^n_t$ is in fact an average over 10 runs of the SPAR-Mutual algorithm, each starting with a different random seed.

Finally, we measure our distance from the optimal policy using

$$\eta^n = 100 \times \frac{\sum_{i=1}^{1,000} (\hat{F}^n_t - \hat{F}^i_t)}{\sum_{i=1}^{1,000} \hat{F}^i_t}.$$  \hspace{1cm} (13)
cost $\rho^T$ is too high compared to the marginal gain of increasing/decreasing the cash level. Although not depicted in the figure, the interest rate plays a less significant role in the cash holding decision. When the interest rate is increased, the optimal slopes increase slightly, indicating that it is a little bit more valuable to hold cash. It is consistent with the intuition that higher financial costs lead to higher cash balances.

Table 1 shows the percentage distance and the associated CPU time for selected discretization levels for the rate of return. The interest rate increment was fixed at 0.001, because policies obtained at this level are comparable to the ones obtained at 0.0005 and the computational time is reduced. As expected, the quality of the solution improves as the discretization of the rate of return process is made finer. Of course, this accuracy comes at a cost. The curse of dimensionality is clearly observed, because the less accurate policy (smallest state space) is computed in less than 20 seconds, whereas it takes almost a day to compute the optimal one (largest state space). It is important to note that in this problem class, an improvement of 0.1% implies that millions of dollars can be saved in one year, because savings are proportional to the redemption amount as depicted in Figure 4(b).

Table 2 presents the percentage distance from the optimal policy for static and ADP approaches. The ADP algorithm was run for 300 thousand iterations, and the reported results are for the last iteration. The discretization increment for the interest rate was 0.001 and for the rate of return was 0.0005.

Figure 6 shows the rate of convergence as a function of CPU time and as a function of number of iterations for the large style funds. These graphs illustrate that the algorithm has very fast initial convergence with a long tail. However, the convergence path is quite stable.

The substantial gap between the two static methods shows that the second model is oversimplified and indicates that it is important to distinguish the two types of demand (individual and institutional) in addition to considering transaction costs. Traditional dynamic programming has running times comparable with the ADP approach when the discretization level (the discretization of prices and interest rates) is 0.005. However, the accuracy of its policy is inferior to the one produced by an ADP algorithm.

7.4. Infinite Horizon Results

In the finite horizon case, we compared the performance of different algorithms using an assumed probability distribution for the portfolio return and interest rates. In the infinite horizon case, the ADP algorithm is applied without any knowledge of the distributions, because actual data can be used in an online learning setting. By contrast, the exact infinite horizon model and the static model require that parameters first be estimated from the model. Actual data is also used to measure the policies, independently of the algorithm. Therefore, modeling and fitting the parameters of the underlying distributions does introduce errors in the resulting policies when value iteration and the static approaches are considered.

Table 3 presents the percentage of increased cost when the static models are considered relative to the ADP approach. Formula (13) multiplied by $-1$ was used to obtain the numbers. Even though the ADP algorithm was run for 200 thousand iterations and the total CPU time was around six seconds, it converged after 25,000 iterations. Because we used actual rates of return and interest rates from July 2001 to June 2005, we did not have 200 thousand sample realizations as required by the ADP algorithm. We circumvented this problem by going back to the July 2001 data after the June 2005 data was reached.

We can infer from this table that the savings using a dynamic policy instead of a static one were even more dramatic for the infinite horizon case. Other than the intrinsic dynamic versus static factor, we can infer
that the main reason behind the bigger difference is modeling errors. We conclude that real gains can be observed when an online algorithm is used instead of one that requires the estimation of the underlying distributions.

8. Insights and Conclusions

We proposed static and dynamic models for the mutual fund cash balance problem as well as algorithmic strategies to solve them. We were able to develop a provably convergent approximate dynamic programming algorithm that replaces the computation of expected values with the observation of sample realizations. Moreover, in an infinite horizon environment, our approach is an online algorithm that requires no estimation of probability distributions.

We were able to show experimentally that the SPAR-Mutual approximate dynamic programming algorithm delivers policies very close to the optimal ones in a short amount of time. Moreover, it is very robust relative to state space size, eliminating the curse of dimensionality. For the infinite horizon case, the SPAR-Mutual algorithm does not require the knowledge of the underlying probability distributions, eliminating the errors introduced by model selection and parameter estimation.

We proved that the optimal value functions are piecewise linear and concave in the cash level dimension, implying that dealing with the monotone decreasing value function slopes is equivalent to dealing with the function itself. This structural result allowed us to prove in a very intuitive way that the optimal policy does not produce transactions unless the cash level falls outside of a specific range. However, this range depends on the performance of the market and, to a lesser degree, interest rates.

We found that the rate of return is more influential than the interest rate when making a decision. Furthermore, transaction costs and differentiating redemptions originating from institutional and retail investors are two factors that should be taken into consideration when solving the problem, because the policy obtained from the model that disregards these factors is inferior to the policies produced by the models that do consider them. Finally, dynamic policies outperform static ones by a significant margin, implying that the downstream effect of the decisions does play an important role and the increased computational burden introduced with the consideration of the value functions does pay off.

Acknowledgments
The authors are grateful for the very helpful comments of the reviewers, as well as additional comments by the department editor, Michael Fu, which clarified this paper. This work was also supported in part by Air Force Office of Scientific Research Grant FA9550-06-1-0496.

Appendix

Proof of Theorem 2. We prove the theorem using a backward induction on \( t \). We start with \( t = T - 1 \). By definition, we have that

\[
\begin{align*}
V_{t+1}(S_{t-1}) &= V_{t+1}(W_{t-1}, R_{t-1}) - V_{t+1}(W_{t-1}, R_{t-1}^-) - V_{t+1}(W_{t-1}, R_{t-1}^+). \\
&= \mathbb{E}[-C_t(W_t, R_t, 0) + C_t(W_t, R_t - 1, 0) | (W_{t-1}, R_{t-1})] \\
&= \mathbb{E}[\rho^t \mathbf{1}_{(D_t + D_{t-1} + R_t) \geq R_t} + p_t^T \mathbf{1}_{D_t \geq R_t} \\
&\quad - p_t^T \mathbf{1}_{D_t \geq R_t} | (W_{t-1}, R_{t-1})].
\end{align*}
\]
because the optimal decision \( x^*_T \) inside the expectation in (4), given any predecision state \((W_t, R_t^v)\), is equal to 0, because the transaction cost \( \rho^v_x \) is charged for all decisions different from 0 and the optimal value function at \( T \) is 0 for all states \((W_t, R_t^v)\). Based on the assumption that \( E[P_t^f | P_{t-1}^f] \) is positive and \( E[P_t^r | P_{t-1}^r] \) is greater than \(-\rho^v_x\) and \(-\rho^v_x\), it is easy to see that \( v^*_t(W_{t-1}, R_{t-1}^v) \geq v^*_t(W_{t-1}, R_{t-1}^v + 1) \). 

Now we state the induction hypothesis. Suppose, for any \( t = T - 2, \ldots, 1 \) and all postdecision states \((W_t, R_t^v)\) that \( v^*_t(W_t, R_t^v) \geq v^*_t(W_t, R_t^v + 1) \). We shall prove that \( v^*_t(W_{t-1}, R_{t-1}^v) \) is given by (7) and \( v^*_t(W_{t-1}, R_{t-1}^v) \geq v^*_t(W_{t-1}, R_{t-1}^v + 1) \) for all postdecision states \((W_{t-1}, R_{t-1}^v)\).

To obtain (7), we plug in (4) in the definition of \( v^*_t \). Because the first three terms of the cost function \( C_t(W_t, R_t, x) \) are independent of the decision \( x \), the first three terms of (7) are obtained in the same fashion as we did for \( t = T - 1 \). The last three terms of (7) are obtained plugging in the solution determined by Rules 1–3 of Theorem 1.

Finally, \( v^*_t(W_{t-1}, R_{t-1}^v) \geq v^*_t(W_{t-1}, R_{t-1}^v + 1) \) follows from the fact that \( \rho^v_x \) is monotone decreasing and the assumption that \( \rho^v_x \geq \rho^v_x \).

## References


