Journal of Statistical Software

June 2014, Volume 58, Issue 11. *http://www.jstatsoft.org/*

Optimal Asset Pricing

Rolf Turner
The University of Auckland

Pradeep Banerjee
The University of New Brunswick

Rayomand Shahlori
The University of Auckland

Abstract

We describe an R package for determining the optimal price of an asset which is "perishable" in a certain sense, given the intensity of customer arrivals and a time-varying price sensitivity function which specifies the probability that a customer will purchase an asset offered at a given price at a given time. The package deals with the case of customers arriving in groups, with a probability distribution for the group size being specified. The methodology and software allow for both discrete and continuous pricing. The class of possible models for price sensitivity functions is very wide, and includes piecewise linear models. A mechanism for constructing piecewise linear price sensitivity functions is provided.

Keywords: perishable assets, price sensitivity, elasticity of demand, airline seats, arrival intensity, group arrivals, discrete pricing, continuous pricing, piecewise linear functions.

1. Introduction

It is a familiar (and potentially infuriating) phenomenon that people traveling on the same flight (in the same class or cabin) will often have paid very different fares. Generally the closer to departure time that one purchases one's ticket, the more one pays. This characteristic of airline ticket prices can be modeled in terms of time varying "elasticity of demand" or "price sensitivity". As a rule, demand is more elastic (more sensitive to price) a long time prior to departure, and is less elastic (less sensitive) immediately before departure. To take advantage of cheap fares, trips must be planned, and prior arrangements made. People who "fly at the last minute" often *have to* fly, irrespective (to a large extent) of what it costs. Or their company is paying for the ticket, whence cost is no object.

Airline seats on a given flight form the prototypical example of a "perishable asset" – one whose value drops to zero after a given deadline or "expiry time" (i.e., when the flight takes off). Those selling the assets are assumed to be free to vary the price quoted for an item to any desired value, at any time. The vendor's objective is (presumably) to choose the pricing policy in such a way as to optimize, i.e., maximize, (expected) revenue. In particular the vendor does not want to be left holding a stock of valueless items that *could* possibly have been sold at some non-zero price.

Problems of the sort alluded to above constitute part of the class of optimization problems termed "asset selling" or "asset pricing" problems which was introduced by Karlin (1962). In such problems a vendor has a stock of a finite number of items which must be sold by a given deadline, otherwise their value becomes zero (or possibly diminishes to some relatively small "salvage value"). Examples of such assets include seats on a given airline flight, rooms in a hotel for a given night, and advertising slots on a particular television program. In Karlin's initial formulation of the problem, customers arrive at the point of sale according to a given stochastic process and make offers to buy a certain number of items. The analyst's objective is to determine if the customer's offer is acceptable or not.

Since Karlin's initial work many versions and extensions of this idea have been studied. Particular emphasis has been given to the problem of pricing airfares. See Gerchak, Parlar, and Yee (1985), Wollmer (1985), Mamer (1986), Belobaba (1989), Feng and Xiao (1999), and Slyke and Young (2000).

In reality, especially in the airfare context, it is rare for customers to make offers to the vendor. Instead a customer arriving at the point of sale is quoted a price by the vendor. The probability that the customer will choose to purchase the asset in question depends upon the quoted price and the time that the customer arrives at the point of sale. Thus instead of determining a policy for the acceptance of offers from customers, the vendor must decide on a policy of what prices to quote to customers. Restructuring the problem in this way alters the form of its solution remarkably. This version of the problem was addressed in the paper by Banerjee and Turner (2012). Other recent and important work on this more realistic variant of the asset selling problem includes that of Feng and Xiao (2000a,b), and of Zhao and Zheng (2000).

In the newer and more realistic version of the problem, the objective of the vendor is to choose a price, dependent on time and stock-level, so that the resulting pricing policy maximizes expected revenue. The R (R Core Team 2014) **AssetPricing** package (Turner 2014) permits the vendor to work out, under suitable assumptions, the optimal pricing policy. The suitable assumptions are that:

(a) groups of customers arrive according to an (inhomogeneous) Poisson process, with known intensity $\lambda(t)$,

(b) the size j of a customer group is determined by a known probability function $\pi(j)$,

(c) the probability that a group of size j, arriving at time t, will purchase items offered at price x is given by a price sensitivity function $S_j(x, t)$.

In the simplest case, where the customers always arrive singly, there is a single price sensitivity function (denoted simply by $S(x, t)$) and the group size probability function is just $\pi(1) = 1$ ($\pi(j) = 0$ for $j > 1$).

We remark that in work of this nature it is conventional to let t represent *residual* time, i.e., the time remaining until the "expiry date". Consequently the expiry date is at $t = 0$.

In the remainder of this paper we give (Section 2) a brief summary of the background on how optimal prices are calculated, referring in part to Banerjee and Turner (2012). The calculation procedure differs depending on whether prices are considered to be discrete or continuous.

In Banerjee and Turner (2012) it was assumed, in the context of continuous prices, that the price sensitivity function is smooth, i.e., twice differentiable. In Section 3 we discuss how the package is used to determine optimal pricing policies both for discrete prices and for continuous prices when the price sensitivity function is smooth.

In Section 4 we describe a new method for determining an optimal pricing policy when price is continuous and the price sensitivity function is piecewise linear in price. Constructing a valid piecewise linear price sensitivity function can be a rather delicate task so the **AssetPricing** package provides a function `buildS()` which builds such functions in an automatic manner, and likewise automatically does all of the required checking to assure the validity of the result. We provide details about the use of `buildS()` and the use of the **AssetPricing** package to calculate an optimal pricing policy given a valid piecewise linear price sensitivity function.

In Section 5 we give several illustrative examples in both the discrete and continuous price cases and show how the `plot()` method provided by the **AssetPricing** package can be used to plot the solutions. A detailed example of the construction and use of piecewise linear price sensitivity functions is also given. We again provide (in this last context) illustrations of displaying the results of the calculations graphically. We also compare the results from using the piecewise linear model with those from a discrete price approximation to the problem.

A brief summary and some discussion are given in the concluding section (Section 6).

2. Some theoretical background

Denote by $v_q(t)$ the expected revenue from a stock of size q (items) at time t. A pricing policy consists of a collection of functions $x_q(t)$ where $x = x_q(t)$ is the price offered to customers arriving at the point of sale at time t when there are q items remaining in stock. We might also wish to allow the price offered to depend on the size of the arriving customer group. In this case the pricing policy is a doubly indexed collection of functions $x_{qj}(t)$ giving the price offered to a group of size j at time t when there are q items remaining in stock.

Given a pricing policy, we can derive a system of differential equations which can be solved for the expected revenue functions $v_q(t)$. The key to deriving these differential equations is a basic property of Poisson processes: the probability of an arrival in a (small) interval $(t, t+h]$ is equal to $\lambda(t)h + o(h)$, and the probability of more than one arrival is $o(h)$. Define $R_q(x,t)$ to be the conditional expected revenue from a stock of size q at time t when the quoted price is x, *given* an arrival at time t. In the single arrivals setting, $R_q(x_q(t),t) = (x_q(t)+v_{q-1}(t))S(x_q(t),t)+v_q(t)(1-S(x_q(t),t))$. It is more complicated in the group arrivals setting. For convenience, set V equal to the expected revenue from the stock when there is more than one arrival between $t+h$ and t. Observe that V is bounded by $q \times x_{\max}$ where $x_{\max}$ is the maximum of all prices under consideration.

We can work out that

$$\begin{aligned} v_q(t+h) &= (1-\lambda(t)h + o(h))v_q(t) + \\ &\qquad (\lambda(t)h + o(h)) \times R_q(x_q(t),t) + \\ &\qquad\quad o(h) \times V, \end{aligned} \tag{1}$$

when the offered price is $x_q(t)$. Rearranging Equation 1 and dividing by h we get:

$$\begin{aligned}\frac{v_q(t+h) - v_q(h)}{h} &= \lambda(t)(-v_q(t) + R_q(x_q(t),t)) \\ &\qquad + (v_q(t) + R_q(x_q(t),t) + V) \times \frac{o(h)}{h}\end{aligned}$$

In the limit, using $o(h)/h \to 0$ as $h \to 0$, the forgoing equation becomes the differential equation:

$$\dot{v}_q(t) = \lambda(t)[-v_q(t) + R_q(x_q(t),t)] \tag{2}$$

Observe that the resulting system is "coupled" since $R_q(x_q(t),t)$ depends on $v_{q-1}(t)$. The system can be solved readily via numerical methods. The **AssetPricing** package uses the `ode()` function from the **deSolve** package (Soetaert, Petzoldt, and Setzer 2010). By default the `"lsoda"` method is used.

In a setting in which prices take values in a (small) discrete set (e.g., a regular price, a sale price, and a bargain basement price) we can optimize over price in a very straightforward manner. We simply set $x_q(t) =$ that value of x which maximizes (over the small discrete set of x-values) the expression $R_q(x,t)$, on the right hand side of Equation 2. This can be done at each step of the numerical solution procedure. The relevant function (`xsolve.disc()`) in the **AssetPricing** package is structured so that the values of the optimal prices as well as the values of $v_q(t)$ and $\dot{v}_q(t)$ are returned.

If we model price as varying *continuously* a different approach is required. We differentiate $R_q(x,t)$ with respect to x and set this derivative equal to 0 to locate the value of the price yielding the maximal expected return. We then differentiate both sides of the resulting equation with respect to t, giving an equation which involves $\dot{x}(t)$. Solving for $\dot{x}(t)$, rewritten as $\dot{x}_q(t)$, yields a system of differential equations for $x_q(t)$ (the *optimal* prices). Like the system for $v_q(t)$, this system can be solved numerically. Again the `ode()` function from the **deSolve** package is used to effect the solution.

The details of the differential equations for $x_q(t)$ get a bit messy, and are even worse when the model involves group arrivals. The doubly indexed and singly indexed price structures must also be allowed for. It is, nevertheless, all feasible to implement. More details are given in Banerjee and Turner (2012). Note that the procedure discussed there requires that the price sensitivity functions be smooth, i.e., differentiable – once with respect to t, twice with respect to x. The mixed partial derivative, with respect to x and t jointly, must also exist. In the current paper we discuss a technique for handling price sensitivity functions which are piecewise linear in price (and hence not smooth). The development of this technique is presented in Section 4.

3. Using the package

The essential components of the package, from the user's point of view, are the functions `xsolve()`, `vsolve()`, `buildS()`, and the `plot()` method for objects of class '`AssetPricing`'. (This method calls upon an underlying method for objects of class '`flap`' to do the real work. The name for class '`flap`' signifies "function list for asset pricing".) We discuss `xsolve()`, `vsolve()`, and the `plot` method for '`AssetPricing`' objects in the following subsections. The function `buildS()` is dealt with in Section 4. The key function is `xsolve()`.

3.1. Determining optimal prices

The function `xsolve()` effects the determination of an optimal pricing policy. The major inputs (arguments) to this function are:

- the price sensitivity function(s) `S`,
- the arrival intensity function `lambda`,
- the group size probability function `gprob`,
- the maximum residual time (time until the deadline) over which the solution procedure is considered, `tmax`,
- the number `qmax` of items available for sale at residual time `tmax`,
- a vector `prices` of possible prices to be used in the discrete pricing case.

The function `xsolve()` examines its arguments and invokes the appropriate solution procedure depending on their nature. In any circumstance the argument `S` must be an expression, a function, a list of expressions, or a list of functions. If the argument `prices` is specified, discrete pricing is assumed and the work is dispatched to `xsolve.disc()`. In this case `S` must be a function or a list of functions (otherwise an error is given). If `prices` is *not* specified, and if `S` is an expression or a list of expressions it is assumed that the expression or expressions determine smooth functions and the work is dispatched to `xsolve.cont()`. Finally if `S` is a function and this function has an attribute `funtype` which is equal to `"pwl"` then the work is dispatched to `xsolve.pwl()`. Except in very exceptional circumstances piecewise linear price sensitivity functions should be built using `buildS()`. We discuss how the package handles piecewise linear price sensitivity functions in Section 4.

Other inputs to `xsolve()` are:

- A parameter `nout` determining the number of (equispaced) points on the interval [0, `tmax`] at which the solutions to the differential equation are evaluated. These points and the solution values are passed to `splinefun()` or (after some processing) to `stepfun()`, thus providing the solutions as functions.
- A parameter `type` specifying whether the quoted prices are permitted to depend on the size of the customer group. If not we have singly indexed prices, `type = "sip"`; if so then we have doubly indexed prices, `type = "dip"`.
- A parameter `alpha` (between 0 and 1) determining the outcome when the size of the customer group exceeds the size `q` of the remaining stock. In such a situation it is assumed that the group will, with probability `alpha` consider an offer at price `x` as if the group were of size `q` (and will purchase the entire remaining stock if the decision is to accept the offer). With probability `1 - alpha` the group will decline to consider an offer at all, and will simply “walk away”.
- An argument `salval` specifying the “salvage value” of each item of remaining stock at residual time 0; this argument defaults to 0.

Probably the most important, and the most intricate, argument is `S`. In the case of discrete prices this must be either a function, or a list of functions, with arguments `x` (price) and `t` (residual time). If `S` is a single function it is assumed to be the price sensitivity function $S(x, t)$ for a single customer. It is also assumed that for a group of size j an offer will be accepted if and only if all members of the group (independently) accept the offer whence the price sensitivity function for a group of size j is $S(x, t)^j$. If `S` is a list it is assumed that the jth entry of that list is the price sensitivity function $S_j(x, t)$ for a group of size j.

In the case of continuous prices, with the price sensitivity function assumed to be smooth, the argument `S` must be an *expression* or list of expressions. These expressions must in effect specify functions with arguments `x` and `t` and must be amenable to differentiation with respect to these arguments via the R function `deriv3()`. If `S` is a single expression it is assumed to yield the price sensitivity function $S(x, t)$ for a single customer. In this case the price sensitivity function for a group of size j is taken to be $S(x, t)^j$. If `S` is a list of expressions the jth entry of that list is an expression giving the price sensitivity function $S_j(x, t)$ for a group of size j.

Note that the expressions in `S` may depend on any number of *parameters* in addition to their arguments `x` and `t` but these parameters are *not* passed as function arguments. The values of the parameters are given in an attribute, called `"parvec"`, of the expression. This attribute is a *named* vector (the names being the parameter names) and the parameters are assigned in the *environments* of the price sensitivity functions and their derivatives (rather than being passed as function arguments).

If prices are assumed to be continuous and the price sensitivity to be piecewise linear, then `S` is only permitted to be a single function (not a list). In this situation the price sensitivity function for a group of size j is taken to be $S(x, t)^j$.

The object returned by `xsolve()` is a list with three components:

- `x`, giving the optimal pricing policy,
- `v`, giving the expected values of stocks under this policy,
- `vdot`, the derivatives (with respect to residual time) of `v`.

Each of these components is an object of class '`flap`', and is fundamentally a list of functions. The functions in `x` are returned by `splinefun()` in the case of continuous prices and by `stepfun()` in the case of discrete prices. The functions in `v` and `vdot` are always returned by `splinefun()`.

If `type` is equal to `"sip"` then `x` is of length `qmax` and `x[[q]](t)` is the optimal price to charge at residual time `t` when the stock size is `q`. If `type` is equal to `"dip"` then `x` is of length `(jmax - 1) * (qmax - jmax/2) + qmax`, where `jmax` is the maximum possible size of a customer group. We remark that as a matter of convenience it is assumed in the code that the largest possible group size is less than or equal to `qmax`. Thus `jmax` is taken to be the minimum of `qmax` and the largest value of `j` such that the probability of a group of size `j` (as determined by `gprob`) is numerically distinguishable from 0.

Also, if `type` is equal to `"dip"`, `x[[i]](t)` is the optimal price to charge at residual time `t` when the stock size is `q` and the size of the arriving customer group is `j`, where `i = (j - 1) * (qmax - j/2) + q`.

The v and vdot lists are always of length qmax. The value of v[[q]](t) is the expected value (under pricing policy x) of a stock of size q at residual time t, and the value of vdot[[q]](t) is the derivative of v[[q]] at residual time t.

3.2. Determining expected values for given prices

It is sometimes of interest to consider a *given* pricing policy and to determine the expected value of various stock sizes at various times, under the given policy. The means of doing so is provided by vsolve(). This function has an argument list similar to that of xsolve(), a notable difference being that vsolve() has an argument x, the given pricing policy, and does *not* have arguments qmax, or type, the values of these being determined by x.

The argument x must be an object of class 'flap' (such as the price component of an object returned by xsolve()). In particular x must be a list of functions and must have attributes tlim, ylim, qmax, and jmax. The function or functions specified by S must be capable of being evaluated at all points in $[0, \texttt{ylim}] \times [0, \texttt{tlim}]$. Note that S may be an expression or list of expressions as is the case with xsolve() and that such expressions must be amenable to differentiation by deriv3(). The differentiability is not actually used by vsolve(); the requirement is imposed because the expressions are turned into functions by deriv3() in the same manner as when they are used by xsolve().

The argument tmax, if specified, must be less than or equal to the upper endpoint of the tlim attribute of x, otherwise an error is given. If the prices specified by x depend upon customer group size as well as upon the number of items remaining to be sold (effectively if x was created with type set equal to "dip") then the largest possible size for an arriving group of customers, as determined by the gprob argument, must be less than or equal to the jmax attribute of x. The calculations involved in solving the system of differential equations require the evaluation of $x_{qj}(t)$ where $x_{qj}(\cdot)$ is given by one of the entries of x, for all q and j. If j exceeds the jmax attribute of x then $x_{qj}(\cdot)$ does not exist.

The object returned by vsolve() is essentially of the same nature as that returned by xsolve() except that the x component does not consist of optimal prices but is rather the x argument provided to vsolve(). This argument may have been modified to have a tlim attribute with upper endpoint equal to tmax if the tmax argument was specified and was smaller than the original upper endpoint of the original tlim attribute.

3.3. Plotting

The nature of the function lists returned by xsolve() and vsolve() is often best examined through graphical methods. However these lists are usually long and intricate, and creating a perspicuous graphical display can be difficult and tedious. The plot method for 'AssetPricing' objects is provided to make the task of creating useful and informative plots much easier.

The basic argument x to the plot method for objects of class 'AssetPricing' should be an object returned by xsolve() or vsolve(). The specification of which of the components (x, v, or vdot) of the object in question should be dealt with is effected by the witch argument of the function. This argument may take values "price", "expVal" or "vdot". These character strings may be abbreviated (e.g., to "p", "e", or "v").

The argument to the plot methods for 'AssetPricing' objects which is most important in facilitating the creation of useful and informative plots is `groups`. This argument specifies which of the entries of the basic argument `x` are to be plotted, and how they are to be grouped together (in panels of multi-panel figures). The argument `groups` is a data frame with columns `group`, `q`, and possibly `j`.

Each row of `groups` specifies a trace to be plotted. The entries of the `group` column should be positive integers running from 1 to some maximal value. The traces are specified by the entries of the `q` and `j` columns. Those traces corresponding to each unique entry of the `group` column will all be plotted in a single panel. The value of `j` can differ from 1 only if the object being plotted is the price component of the object returned by `xsolve()` or `vsolve()` and then only if the `type` is equal to `"dip"`. If the `j` column is not present then an error is given unless the maximal possible value of `j` is 1, in which case the `j` column is internally set equal to a vector of 1's.

There is also a facility for adding a "marginal gloss" to be plotted at the right hand endpoint of each one (or some) of the traces, so as to identify these traces. The gloss consists by default of the value of `q` and possibly of `j` if the maximal value of `j` is greater than 1. The gloss may however consist of arbitrary character strings from a vector of length equal to the number of rows of `groups`.

If `gloss` is constructed in an automated manner (as it is in the default case) then it may be convenient to specify that only a subset of the traces have their corresponding gloss plotted. This prevents cluttering and over-plotting of the gloss. The facility for doing this is provided by the `glind` ("gloss index") argument which is a logical vector, of length equal to the number of rows of `groups`. A trace has its gloss plotted only if the corresponding entry of `glind` is `TRUE`.

The plotted traces are grouped together in panels as specified by the `groups` argument. To obtain a single plot showing all traces the `group` column of `groups` should be set to be identically equal to 1. Clearly setting `group` to be identically 1 (this is the default when the argument `groups` is not specified) is sensible only if a relatively small number of traces is being plotted. At the other extreme, to get traces plotted in individual panels (one trace per panel) the `group` column should be set to be the sequence from 1 to `n` where `n` is the total number of traces being plotted (equal to `nrow(groups)`).

The dimensions of the array(s) of panels to be plotted is determined by the `mfrow` argument. This defaults to either `c(1, 1)`, `c(2, 2)`, or `c(3, 2)` depending on the total number of groups. The user however may set these dimensions to be anything he or she chooses.

4. Piecewise linear price sensitivity functions

The procedure implemented by the **AssetPricing** package, for determining an optimal pricing policy, requires that the arrival intensity and the price sensitivity function or functions be *known* functions. Translating informal knowledge about customer characteristics into an arrival intensity, and even more so, into a price sensitivity function is in general difficult. There are however certain scenarios in which such translation is reasonably simple.

These scenarios lead to price sensitivity functions which are piecewise linear in price. For such functions, the differentiability conditions previously assumed in the context of continuous prices do not hold. A different approach to determining the optimal pricing policy must be taken. A broadly applicable approach has been implemented in the **AssetPricing** package.

This approach is capable of dealing with functions which are piecewise linear in *price* but more or less arbitrary with respect to time. That is, it can be applied to functions of the form

$$S(x,t) = \alpha_k(t) + \beta_k(t)x \tag{3}$$

(where x represents price and t represents residual time) for $x_{k-1} \leq x \leq x_k$. In the forgoing $0 = x_0 < x_1 < \ldots < x_K$ are the "knots" (change points) of the piecewise linear function of price, whose coefficients are time dependent. The optimization which is involved in dealing with such functions circles back, to some extent, to the discrete price scenario.

In order to be sure that it makes sense, we require the function $S(x,t)$ to be:

(a) Continuous in x (i.e., $\alpha_k(t) + \beta_k(t)x_k = \alpha_{k+1}(t) + \beta_{k+1}(t)x_k$ for $k = 1, \ldots, K-1$ and all t).

(b) Non-increasing in x (i.e., $\beta_k(t) \leq 0$ for all k and all t).

(c) Equal to 1 at $x = 0$ (i.e., $\alpha_1(t) \equiv 1$ for all t).

(d) Non-negative (i.e., $\alpha_K(t) + \beta_K(t)x_K \geq 0$ for all t).

Aside from that, the coefficient functions $\alpha_k(t)$ and $\beta_k(t)$ may be more or less arbitrary, although in practice we would probably want them to be continuous.

4.1. Constructing the functions

Building a function such as is described by Equation 3 is not completely straightforward and checking that it is "valid" (satisfies the four conditions listed above) is a bit delicate. The **AssetPricing** package contains a function `buildS()` which helps to alleviate this problem. This function returns as its value a legitimate (checked) price sensitivity function. The function `buildS()` takes as its input (arguments) lists of coefficient functions `alpha` and `beta`, the knots `kn`, and the maximal value of `t`, `tmax`. This last is needed in respect of the "for all t" conditions that S must satisfy. In effect this means "for all t such that $0 \leq t \leq \text{tmax}$".

The value returned by `buildS()` is a function of two arguments `x` (price) and `t` (residual time). Note that the arguments `alpha`, `beta`, `kn`, and `tmax` to `buildS()` are assigned *in the environment* of the function returned by `buildS`. Thus the returned price sensitivity function "carries these objects around with it" so that it can make use of them, although they are not directly visible to the user.

In the sort of scenario which originally motivated the implementation of piecewise linear price sensitivity functions, customers are divided into a number of types. Each type has a (time invariant) price sensitivity, about which informal knowledge might yield sufficient insight to represent it as a piecewise linear function of price. Similarly each type has a different arrival intensity. Again informal knowledge might be sufficient to construct reasonable representations of these intensities.

In such a scenario we would assume that customer class ℓ has a (time invariant) price sensitivity function $S_\ell(x) = a_{k\ell} + b_{k\ell}x$ for $x_{k-1} \leq x \leq x_k$, where the x_k are "change points" or "knots" (without loss of generality we may assume that the knots are the same for all ℓ). We would also assume that customers from class ℓ arrive according to a Poisson process with intensity

$\lambda_\ell(t)$. The overall intensity would then be $\lambda(t) = \sum_\ell \lambda_\ell(t)$ and the overall price sensitivity function would be:

$$S(x,t) = \frac{1}{\lambda(t)} \sum_\ell \lambda_\ell(t)(a_{k\ell} + b_{k\ell}(t)x) \text{ for } x_{k-1} \le x \le x_k \quad .$$

This function is of the required form (Equation 3) with

$$\alpha_k(t) = \left(\sum_\ell a_{k\ell}\lambda_\ell(t)\right)/\lambda(t) \qquad \text{and} \qquad \beta_k(t) = \left(\sum_\ell b_{k\ell}\lambda_\ell(t)\right)/\lambda(t) \quad .$$

4.2. Maximizing expected revenue

For a piecewise linear price sensitivity function, price varies continuously but such a function is not smooth. (There are discontinuities in the first derivative with respect to x.) Consequently the maximization technique previously applied when price is continuous is not applicable. Instead we implement a procedure which rests upon recognizing that the conditional expected value of a stock of size q, offered at price x at residual time t (denoted $R_q(x,t)$) is a *piecewise polynomial* in x.

These piecewise polynomials are quadratic in the setting in which customers arrive singly. They are of degree $j_{\max}+1$ in the setting in which customers arrive in groups, where $j_{\max}$ is the maximal customer group size. The maximum of $R_q(x,t)$ will therefore occur at either a zero of the derivative of one of these piecewise polynomials, or at one of the knots. Consequently there is a finite number of possible maxima. After determining the relevant zeroes (throwing away any such which do not fall between the appropriate pair of knots) it is possible to go back to the technique used in the discrete price setting and do a discrete maximization.

It turns out that the polynomial components of $R_q(x,t)$ can be built up as sums, in terms of "binomials" raised to a power: $(c_r + d_r x)(a + bx)^r$, with r running from 1 to a certain value $r_{\max}$. We can thereby apply the facilities of the **polynom** package (Venables, Hornik, and Maechler 2014) to put together each such polynomial. The code (part of the `turnPts()` function in the **AssetPricing** package) looks like this:

```
R> vq <- v[q]
R> ply <- 0
R> for (r in 1:rmax) {
+    vqmr <- if (r < q) v[q - r] else 0
+    p1 <- polynomial(c(vqmr - vq, r))
+    p2 <- polynomial(c(a, b))
+    # a and b are arguments to turnPts()
+    ply <- ply + Kpa[r] * p1 * p2^r
+ }
```

The resulting polynomial can be differentiated via:

```
R> dply <- deriv(ply)
```

and its zeroes found via:

```
R> zzz <- polyroot(dply)
```

It is necessary to check that there *are* any zeroes – it can turn out that `dply` is a constant. It is also necessary to discard zeroes with a non-trivial imaginary part and zeroes that do not fall between the relevant pair of knots. All this is reasonably straightforward to accomplish.

5. Examples

Example 1: Suppose that the possible prices in a discrete price setting are \$60 (bargain-basement price), \$150 (sale price) and \$200 (regular price). Assume that the price sensitivity function is given by

$$S(x,t) = \begin{cases} 1 & \text{if } x = 60 \\ e^{-2t} & \text{if } x = 150 \\ e^{-4t} & \text{if } x = 200 \end{cases}$$

and that customers arrive according to a constant intensity Poisson process with intensity $\lambda = 100$. Suppose that time to departure is 1 time unit (month, say). Assume here that customers always arrive singly. We calculate the optimal pricing policy using **AssetPricing** as follows:

```
R> S <- function(x, t)
+    ifelse(x == 60, 1, ifelse(x == 150, exp(-2 * t), exp(-4 * t)))
R> optPrice01 <- xsolve(S = S, lambda = 100, tmax = 1, gprob = 1,
+    qmax = 100, prices = c(60, 150, 200), alpha = 1, type = "sip")
```

Traces of the optimal prices may be plotted using the `plot` method for '`AssetPricing`' objects. Since the prices are discrete, these traces are step functions whence plotting them all at once yields an incomprehensible tangle. Therefore we must specify a `groups` argument in the call to the plot method. Since there are 100 traces it is not feasible to plot all of them. Moreover many traces are identical to each other. We have picked out six reasonably representative (and interesting) traces to plot; these traces are shown in Figure 1. The call to produce this figure is:

```
R> grps <- data.frame(group = 1:6, q = c(10, 20, 40, 45, 50, 90))
R> mp <- paste("stock size =", grps$q)
R> plot(optPrice01, witch = "p", groups = grps, main.panel = mp)
```

As discussed in Section 3.3, the `plot` method for class '`AssetPricing`' chooses the dimensions of arrays of plots to be either $(1,1)$, $(2,2)$ or $(3,2)$ according to the total number of groups. Consequently we got the 3×2 array shown in Figure 1. If one wanted, say a 2×3 array rather than a 3×2 array, one would add the argument `mfrow = c(2, 3)` to the call to `plot()`.

The traces that appear in Figure 1 may seem rather counter intuitive (particularly that short sharp notch in the trace for $q = 40$) but they are correct. A plot of the expected values of stocks of various sizes, versus residual time, is shown in Figure 2.

It is interesting to compare the expected values for the optimal pricing policy with those for a suboptimal policy such as would be given by using the price function for $q = 90$ for all values of q:

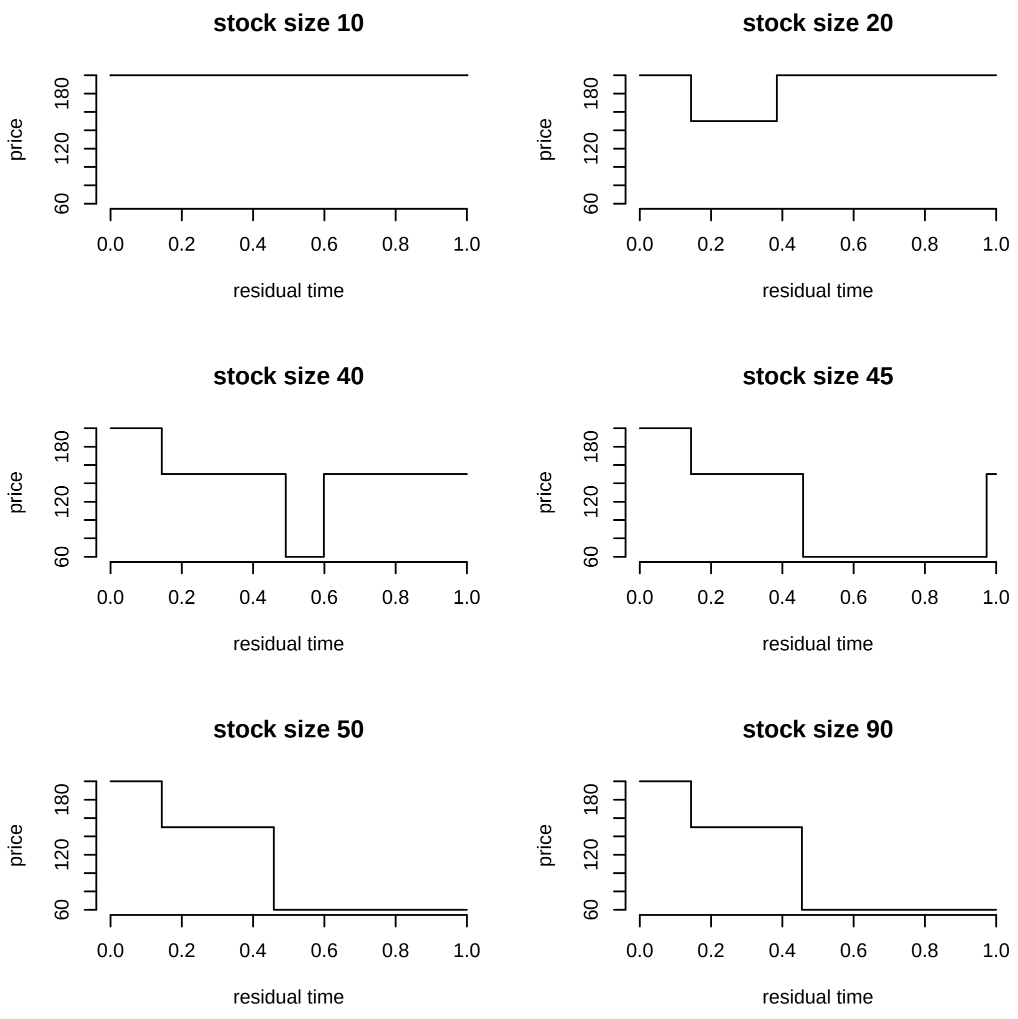

Figure 1: Traces of optimal prices for Example 1.

```
R> y <- rep(optPrice01$x[90], 100)
R> attributes(y) <- attributes(optPrice01$x)
R> comment(y) <- "Prices constant over q."
R> Cmpre <- vsolve(x = y, S = S, lambda = 100)
```

A comparison of some of the expected value traces under the optimal and suboptimal pricing policies is shown in Figure 3. We see that the traces for the maximal stock size, i.e., 100, are visually indistinguishable. Note that these traces give the expected value of the original stock size at each residual time value, i.e., the expected value that would pertain if no items of stock had been sold.

However suppose that you had sold half of the original number of items of stock by (for example) residual time 0.8. Under the suboptimal policy, the expected value of the remaining 50 items is

```
R> Cmpre$v[[50]](0.8)

[1] 4465.987
```

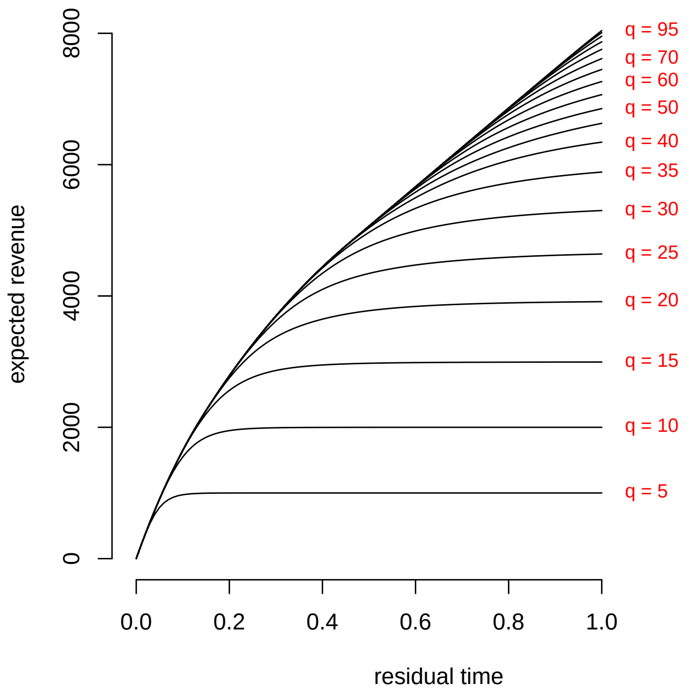

Figure 2: Traces of expected revenue for Example 1.

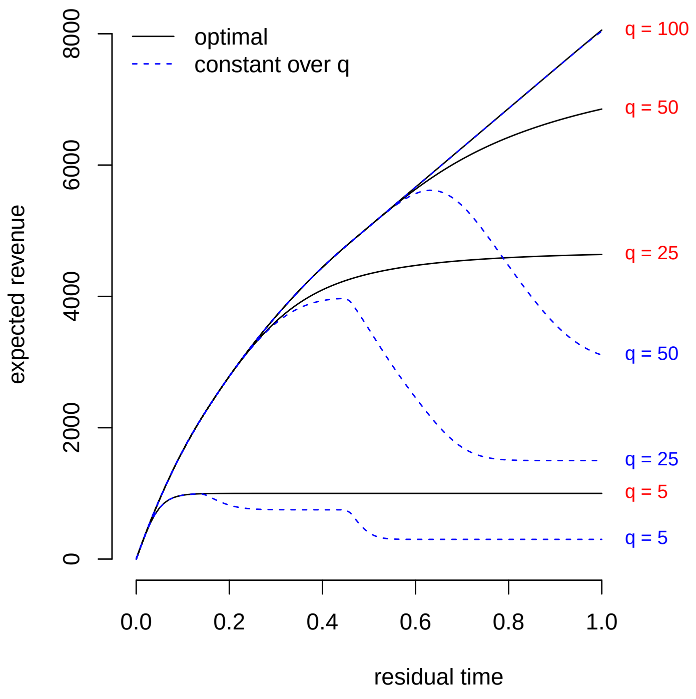

Figure 3: Comparison of expected revenue for optimal and suboptimal prices.

whereas under the optimal policy we get

```
R> optPrice01$v[[50]](0.8)
```

```
[1] 6423.743
```

almost a 44% increase over the suboptimal expected value.

Note that if the original stock size were 50, and if the pricing policy were to use the price function for $q = 90$ for all values of q, then there would be a 120% increase, at residual time = 1, when using the optimal policy:

```
R> EVOP <- optPrice01$v[[50]](1)
R> EVSOP <- Cmpre$v[[50]](1)
R> 100 * (EVOP - EVSOP)/EVSOP
```

```
[1] 120.576
```

Example 2: Let us consider an example in which the price sensitivity function is smooth:

$$S(x,t) = \frac{\exp(-\kappa x)}{1 + \gamma \exp(-\beta t)},$$

where κ, γ and β are positive parameters. Observe that this function is infinitely differentiable with respect to both x and t. The probability of purchase decreases as x increases and diminishes to zero as x tends to infinity. Its derivative with respect to x (price) increases in absolute value as t increases – demand is more elastic at times which are distant from departure time.

For this example we shall consider an arrival intensity which increases as t tends to 0: $\lambda(x) = 84(1-t)$. We shall assume that groups arrive in sizes determined by a Poisson distribution with mean equal to 5. In this example we shall take $\kappa = 10$, $\gamma = 5$ and $\beta = 3$.

The code to fit this model is as follows:

```
R> S <- expression(exp(-kappa * x/(1 + gamma * exp(-beta * t))))
R> attr(S, "parvec") <- c(kappa = 10, gamma = 5, beta = 3)
R> Lam <- function(t) 84 * (1 - t)
R> Gpr <- function(n) dpois(n, 5)
R> optPrice01 <- xsolve(S = S, lambda = Lam, tmax = 1, gprob = Gpr,
+    qmax = 100, alpha = 0.5, type = "dip")
```

Note that since `S` is a single expression (not a list of such) it is assumed that the price sensitivity function for a group of size j is $S(x,t)^j$. Plots of some of the optimal prices are shown in Figure 4. Salient features are the tendency of the optimal prices to decrease as the group size increases and to increase as the number of remaining items of stock decreases.

The code to produce Figure 4 is:

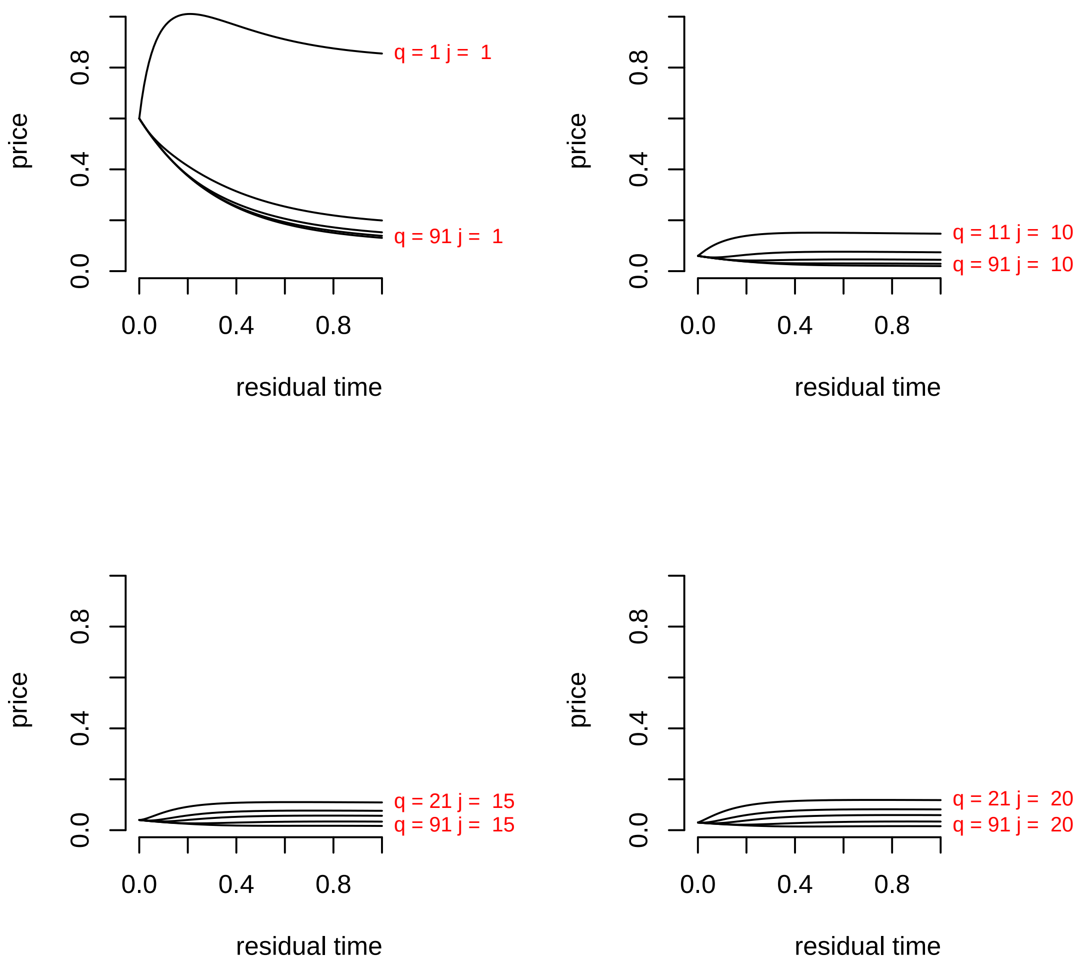

Figure 4: Traces of optimal prices for Example 2.

```
R> grps <- data.frame(group = rep(1:4, each = 5),
+    q = c(1, 21, 51, 71, 91, 11, 31, 51, 71, 91, 21, 31, 41, 61,
+      91, 21, 31, 41, 61, 91), j = rep(c(1, 10, 15, 20), each = 5))
R> glnd <- rep(c(TRUE, FALSE, FALSE, FALSE, TRUE), 4)
R> plot(optPrice02, witch = "p", groups = grps, gloss = TRUE, glind = glnd,
+    xlab = "residual time", ylab = "asking price", extend = 0.4,
+    main.panel = "", col.gloss = "red")
```

Some plots of expected values of stocks of various sizes, versus residual time, are shown in Figure 5. The code to produce Figure 5 is:

```
R> grps <- data.frame(group = 1, q = 5 + 10 * (1:9))
R> glnd <- rep(TRUE, 9)
R> glnd[7:8] <- FALSE
R> plot(optPrice02, witch = "e", groups = grps, glind = GLND,
+    col.gloss = "red", gloss = TRUE)
```

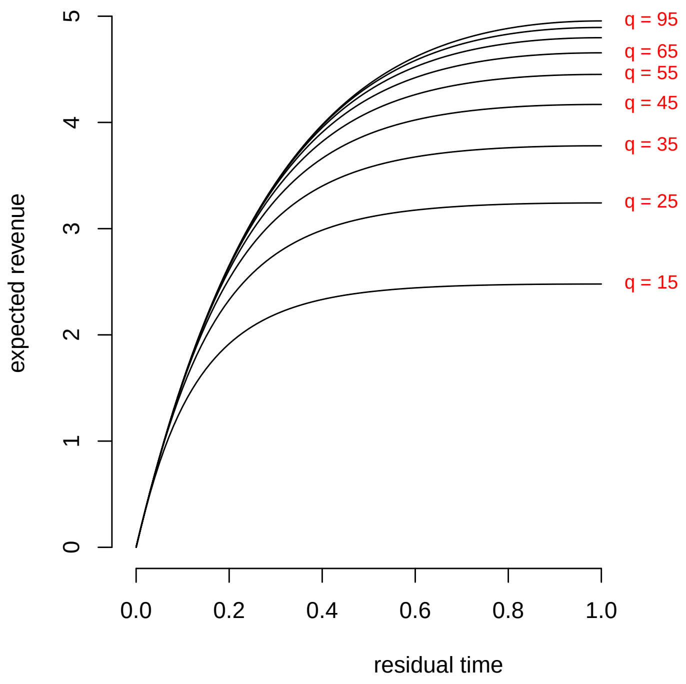

Figure 5: Traces of expected revenue for Example 2.

Example 3: Often there is likely to be a strong periodic effect in the arrival intensity. For instance there may well be a weekly periodicity in customer behavior. A somewhat artificial example of such a periodic intensity (based on “Model A” from Kutoyants 1998, Equation 2.67, p. 78) is:

$$\lambda(t) = \frac{a}{2}[1 + \cos(\omega t + \phi)] + \lambda_0 \ .$$

To produce an explicit example we take $a = 74$, $\omega = 8\pi$, $\phi = 6\pi/7$ and $\lambda_0 = 5$. This would roughly correspond to a weekly periodicity over a sales period of four weeks starting on a Wednesday (i.e., 3/7 of a week). The total expected number of arrivals over a sales period is 42. Using the same price sensitivity function and the same group size probability function as in Example 2, and setting `type = "sip"` for simplicity, we obtain plots of optimal prices and corresponding expected values as shown in Figures 6 and 7. We omit the code for the calculation and plotting, since it is similar to that used in Example 2, but it is available in the supplementary material. The periodicity of the arrival intensity has a readily apparent impact upon the resulting optimal prices and corresponding expected values, which manifests itself in the undulating nature of the curves. The impact is more apparent for the optimal prices when the number q of items available for purchase is small and for the corresponding expected values when the number of items is large.

Example 4: Consider a setting in which the assets under consideration are seats on airline flights and in which customers may be divided into three types: (1) “Bargain hunters”, (2) “Tourists”, and (3) “Business travelers”. Business travelers’ arrival times will tend to be concentrated relatively close to the departure time of the flight. Tourists (who need to plan holidays well in advance) will arrive at times concentrated earlier in the sales period. Bargain

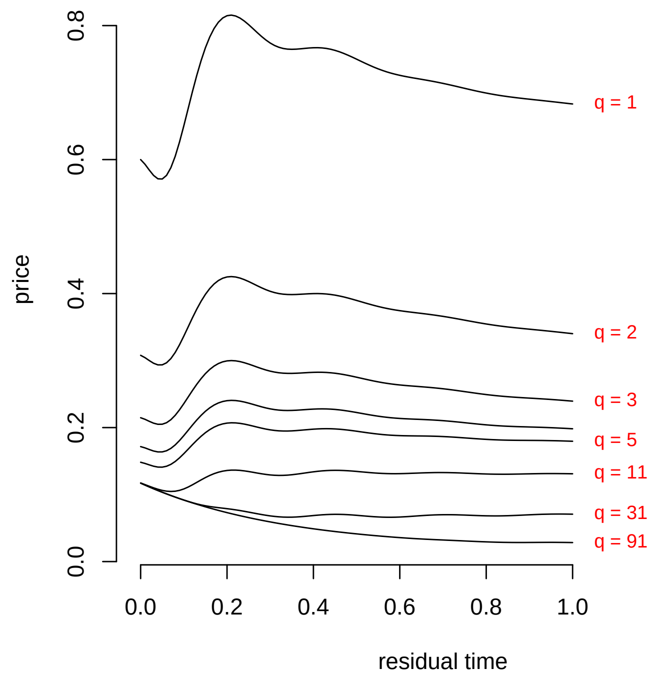

Figure 6: Traces of optimal prices for Example 3.

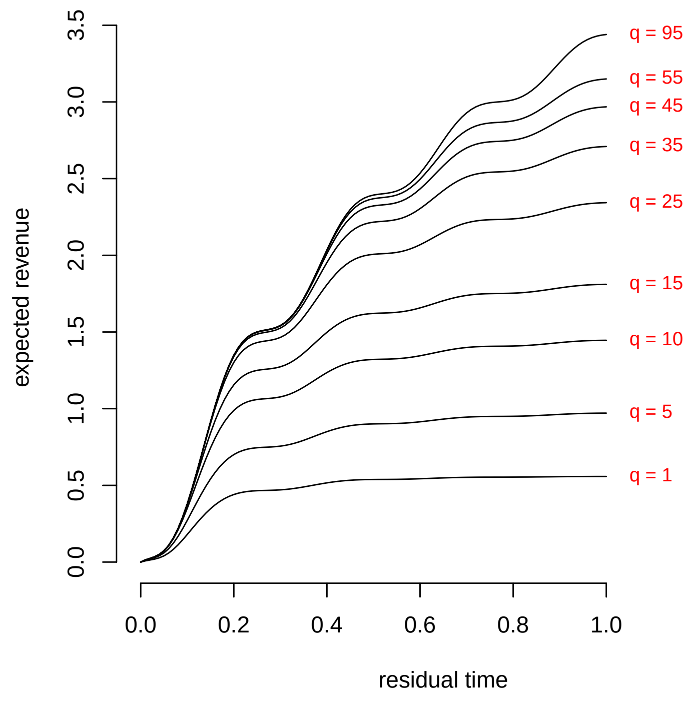

Figure 7: Traces of expected revenue for Example 3.

hunters may be assumed to arrive more "uniformly" but with a tendency to be sparse toward departure time. Bargain hunters will be relatively sensitive to the offered price. Tourists will be less, but still substantially, sensitive to price. Business travelers will be the least sensitive.

Explicitly we assume price sensitivity functions for the various customer classes as follows:

$$S_{\text{Bhnt}}(x) = \begin{cases} 1 & \text{for } 0 \le x \le 2 \\ 1.495 - 0.2475x & \text{for } 2 \le x \le 6 \\ 0.01 & \text{for } 6 \le x \le 10 \\ 0.01 & \text{for } 10 \le x \le 14 \end{cases}$$

$$S_{\text{Tour}}(x) = \begin{cases} 1 & \text{for } 0 \le x \le 2 \\ 1 & \text{for } 2 \le x \le 6 \\ 2.485 - 0.2475x & \text{for } 6 \le x \le 10 \\ 0.01 & \text{for } 10 \le x \le 14 \end{cases}$$

$$S_{\text{Busi}}(x) = \begin{cases} 1 & \text{for } 0 \le x \le 2 \\ 1 & \text{for } 2 \le x \le 6 \\ 1 & \text{for } 6 \le x \le 10 \\ 3.475 - 0.2475x & \text{for } 10 \le x \le 14 \end{cases}$$

We assume the arrival intensities for the customer classes are:

$$\lambda_{\text{Bhnt}}(t) = \begin{cases} 12t & \text{for } 0 \le t \le 1 \\ 12 & \text{for } 1 \le t \le 2 \\ 12 & \text{for } 2 \le t \le 3 \\ 12 & \text{for } 3 \le t \le 4 \end{cases}$$

$$\lambda_{\text{Tour}}(t) = \begin{cases} 0 & \text{for } 0 \le t \le 1 \\ 16(t-1) & \text{for } 1 \le t \le 2 \\ 16 & \text{for } 2 \le t \le 3 \\ 64 - 16t & \text{for } 3 \le t \le 4 \end{cases}$$

$$\lambda_{\text{Busi}}(t) = \begin{cases} 20 & \text{for } 0 \le t \le 1 \\ 10(3-t) & \text{for } 1 \le t \le 2 \\ 10(3-t) & \text{for } 2 \le t \le 3 \\ 0 & \text{for } 3 \le t \le 4 \end{cases}$$

These intensity functions are also piecewise linear, but this is of no particular consequence in respect of the ideas being developed. Plots of the price sensitivity functions are shown in Figure 8 and of the arrival intensity functions in Figure 9. A plot of the resulting overall price sensitivity function is shown in Figure 10.

Initially we took the minimum values of the price sensitivity functions to be 0 rather than 0.01 (with the consequently simpler value of -0.25 for the slope coefficients and 1.5, 2.5, and 3.5 for the respective intercept coefficients of the non-constant segments). However the 0 values induced a "flat spot" in the overall price sensitivity function whereby the probability of any purchase was 0 for $t > 3$ and $x > 10$. As a consequence the optimum price became indeterminate for $t > 3$ which induced a numerical instability in the system. We therefore adjusted the minimal value up to 0.01 so as to alleviate this instability. The code has since been revised to cope with the indeterminacy, but we have retained the example which has the slightly more complicated coefficients. In this example the maximum number of assets (seats) for sale was set equal to 30, and *single* arrivals were assumed, to keep things simple.

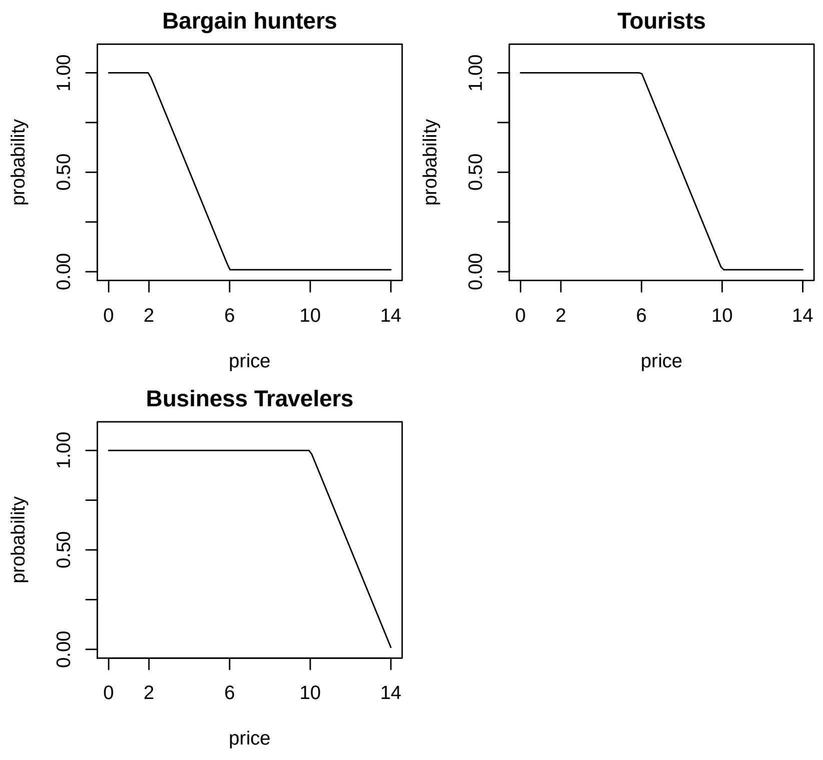

Figure 8: Example of piecewise linear price sensitivity functions.

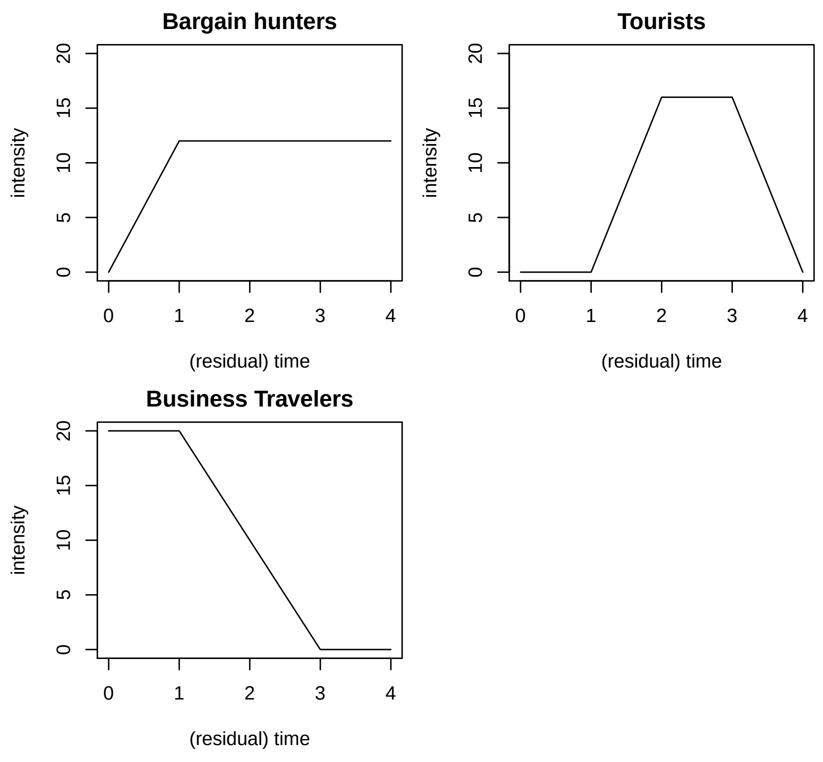

Figure 9: Example of arrival intensities linked to piecewise linear price sensitivity functions.

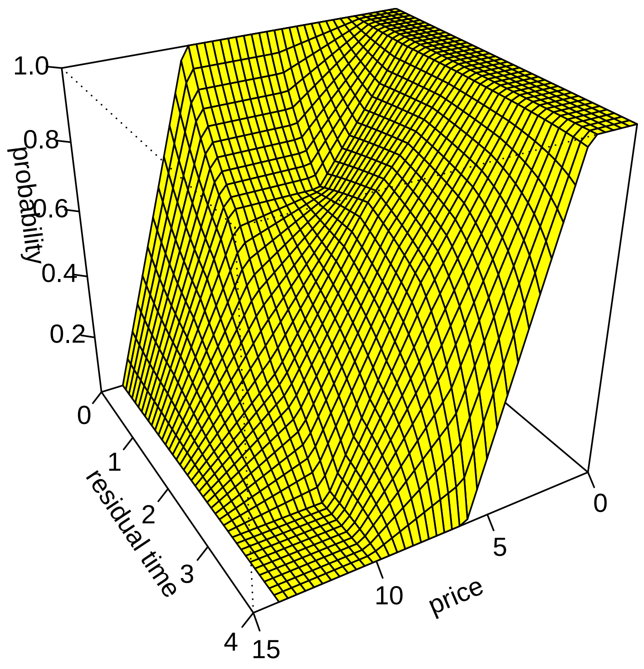

Figure 10: Price sensitivity function built from components in Figures 8 and 9.

We found that we had to increase the value of `nout` from its default value of 300 in the call to `ode()`, otherwise some anomalous "jagged" fluctuations appeared in the traces of the optimal prices.

To produce the traces of optimal prices shown in Figure 11 we set `nout = 1000`. (This had a substantial impact on the computing time, increasing it from 2.6 minutes to about 7.5 minutes.) Traces of the corresponding expected values of stocks are shown in Figure 12. We remark that increasing the value of `nout` had no visually discernible impact on these latter traces. As a check the model was fitted using the *discrete* pricing paradigm, with the "discrete prices" taken to be an equispaced sequence (of 51 points) on $[0, 14]$. The prices from the discrete model, corresponding to those shown in Figure 11, are shown in Figure 13 and the corresponding expected values are shown in Figure 14. The optimal prices from the discrete pricing approximation agree well with those from the piecewise linear model. The expected values from the two models are virtually indistinguishable.

6. Discussion

The **AssetPricing** package provides a convenient facility for determining the optimal pricing policy for a number of "perishable" assets (assets with a fixed expiry date). The optimal policy is determined by solving a coupled system of differential equations. The solution is effected numerically using the `ode()` function from the **deSolve** package. The **AssetPricing** package also has the capability to determine the expected value of a number of assets given a specified (not necessarily optimal) pricing policy. Determination of the expected value is also effected via the solution of a system of differential equations (using the `ode()` function). Finally, the package provides means of producing useful and informative plots of the calculated results in

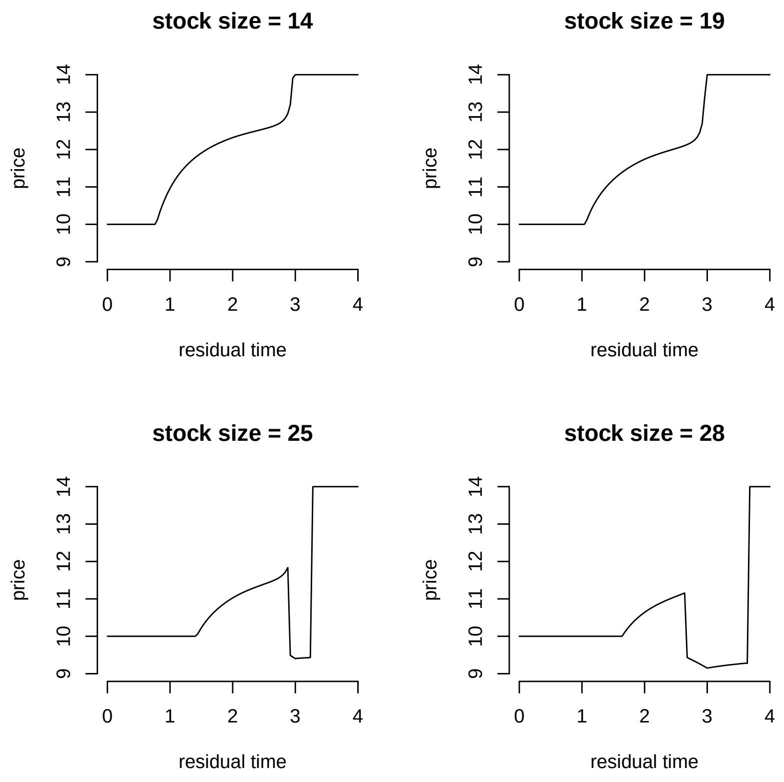

Figure 11: Optimal prices from a piecewise linear model.

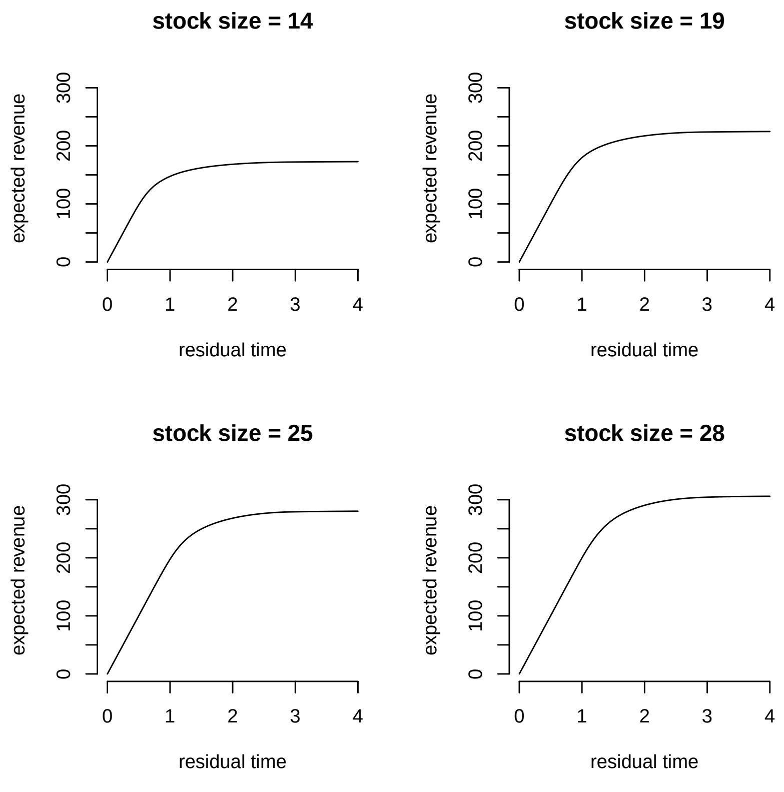

Figure 12: Expected revenue from a piecewise linear model.

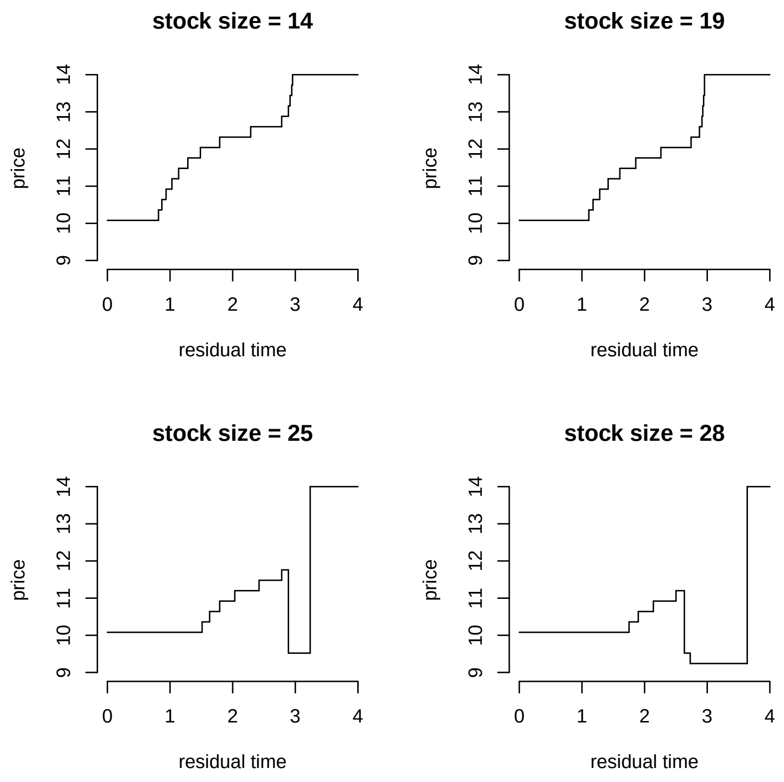

Figure 13: Optimal prices from the discrete price approximation.

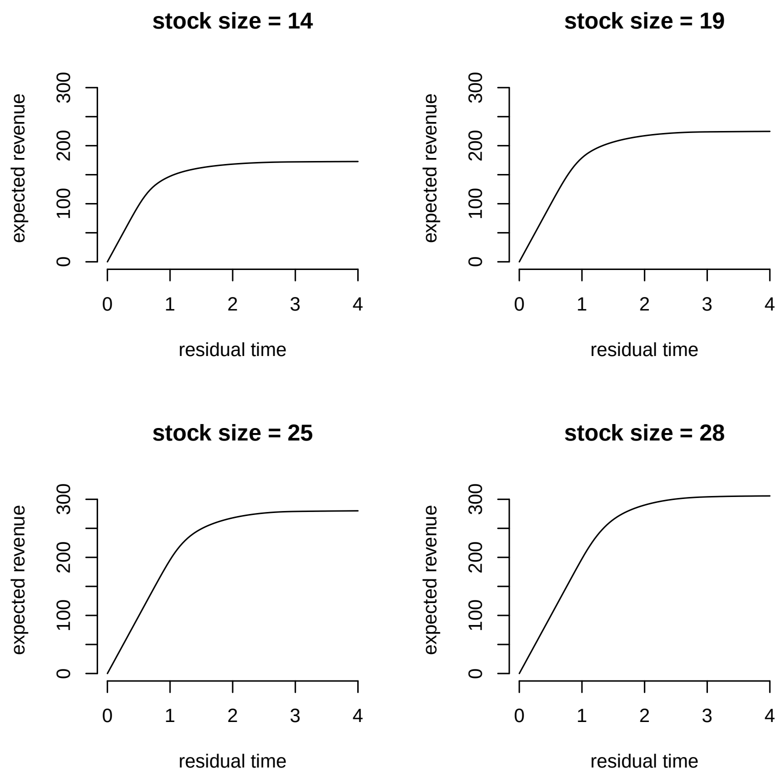

Figure 14: Expected revenue from the discrete price approximation.

a simple and convenient manner.

The techniques used in the package are largely based on the theoretical foundations described in Banerjee and Turner (2012). However a new methodological feature is also involved, and this is developed in the current paper. It permits dealing with price sensitivity functions (for continuous prices) which are piecewise linear in price and consequently not smooth. It is likely to be much easier to specify a piecewise linear price sensitivity function in terms of "informal knowledge" than it is to specify a smooth one. Given this capacity to construct and handle piecewise linear price sensitivity functions, the proposed technique for determining optimal pricing policies should be reasonably simple for users to apply to real problems and the technique should be able to deliver genuinely useful results in a practical and convenient manner.

When the price sensitivity function is piecewise linear, optimizing with respect to price requires constructing piecewise polynomials and finding the zeroes of their derivatives. This can be done surprisingly easily using the **polynom** package. The results of applying the technique agree well with a "discrete price" approximation.

The package is entirely written in "raw R" and hence runs somewhat slowly. It should be possible to recode the software in Fortran or C and thereby achieve a considerable increase in speed. Such an improvement to **AssetPricing** is planned for a future release of the package. The coding is likely to be somewhat intricate however, hence effecting the improvement will probably take some time.

It is of interest to remark here on a technical issue pertaining to the coding of the functions in the **AssetPricing** package. The differential equations to be solved take the form:

$$\dot{\boldsymbol{x}}(t) = \mathcal{G}(\boldsymbol{x}, t) \quad \text{(optimal prices)}$$

and

$$\dot{\boldsymbol{v}}(t) = \mathcal{F}(\boldsymbol{v}, t) \quad \text{(expected values)} \quad .$$

In the code, the functions $\mathcal{G}(\cdot,\cdot)$ and $\mathcal{F}(\cdot,\cdot)$ (and other functions as well) depend on a number of other "auxiliary" arguments. It turned out to be expedient to *assign* these auxiliary arguments in the *environments* of the relevant functions, rather than making them members of the argument lists of the functions. Example:

```
R> assign("alpha", alpha, envir = environment(scrG))
```

Applying this idea resulted in code that was much cleaner and simpler than would otherwise have been the case.

Acknowledgments

The authors would like to thank an anonymous Associate Editor of the journal for some extremely helpful advice which led to a substantial improvement in the paper. We would also like thank the authors of the package **deSolve** for providing such a remarkably powerful, useful and versatile piece of software.

References

Banerjee PK, Turner TR (2012). "A Flexible Model for the Pricing of Perishable Assets." *Omega*, **40**(5), 533–540.

Belobaba PP (1989). "Application of a Probabilistic Decision Model to Airline Seat Inventory Control." *Operations Research*, **37**(2), 183–197.

Feng Y, Xiao B (1999). "Maximizing Revenue of Perishable Assets with a Risk Factor." *Operations Research*, **47**(2), 332–343.

Feng Y, Xiao B (2000a). "A Continuous Time Yield Management Model with Multiple Prices and Reversible Price Changes." *Management Science*, **46**(5), 644–657.

Feng Y, Xiao B (2000b). "Optimal Policies of Yield Management with Multiple Predetermined Prices." *Operations Research*, **48**(2), 332–343.

Gerchak Y, Parlar M, Yee TKM (1985). "Optimal Rationing Policies and Production Quantities for Products with Several Demand Classes." *Canadian Journal of Administrative Sciences*, **2**(1), 161–176.

Karlin S (1962). "Stochastic Models and Optimal Policy for Selling an Asset." In K Arrow, S Karlin, W Scarf (eds.), *Studies in Applied Probability and Management Sciences*, pp. 148–158. Stanford University Press, Stanford, CA.

Kutoyants YA (1998). *Statistical Inference for Spatial Poisson Processes*. Springer-Verlag, New York.

Mamer JW (1986). "Successive Approximations for Finite Horizon, Semi-Markov Decision Processes with Application to Asset Liquidation." *Operations Research*, **34**(4), 638–644.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL `http://www.R-project.org/`.

Slyke RV, Young Y (2000). "Finite Horizon Stochastic Knapsacks with Applications to Yield Management." *Operations Research*, **48**(1), 155–172.

Soetaert K, Petzoldt T, Setzer RW (2010). "Solving Differential Equations in R: Package **deSolve**." *Journal of Statistical Software*, **33**(9), 1–25. URL `http://www.jstatsoft.org/v33/i09/`.

Turner R (2014). ***AssetPricing**: Optimal Pricing of Assets with Fixed Expiry Date*. R package version 1.0-0, URL `http://CRAN.R-project.org/package=AssetPricing`.

Venables B, Hornik K, Maechler M (2014). ***polynom**: A Collection of Functions to Implement a Class for Univariate Polynomial Manipulations*. R package version 1.3-8; S original by Bill Venables, packaged for R by Kurt Hornik and Martin Maechler, URL `http://CRAN.R-project.org/package=polynom`.

Wollmer RD (1985). "An Airline Reservation Model for Opening and Closing Fare Classes." *Internal report*, McDonnell-Douglas Corporation, Long Beach, CA.

Zhao W, Zheng Y (2000). "Optimal Dynamic Pricing for Perishable Assets with Nonhomogeneous Demand." *Management Science*, **46**(3), 375–388.

Affiliation:

Rolf Turner
Department of Statistics
University of Auckland
Private Bag 92019
Auckland 1142, New Zealand
E-mail: r.turner@auckland.ac.nz

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 58, Issue 11 *Submitted:* 2012-11-21
June 2014 *Accepted:* 2014-02-24