

**Price Discovery Analysis in SAS
Version 1.0**

Joel Hasbrouck

Department of Finance
Stern School of Business
New York University
44 West 4th St. Suite 9-190
New York, NJ 10012-1126
212.998.0310

jhasbrou@stern.nyu.edu

web site: www.stern.nyu.edu/~jhasbrou

January 10, 2002

Preliminary draft
Not for attribution
Comments welcome

Abstract

This paper describes a suite of SAS programs to analyze price discovery high-frequency market microstructure price data. The most recent versions of the program, data and this paper are available at my web site.

Key words: price discovery, cointegration, vector error correction, market microstructure.

I. Introduction

This paper describes the estimation of price discovery models using a suite of SAS programs.

There are three programs:

- CointTest01.sas implements an illustrative analysis using simulated data.
- Coint04.sas implements the price discovery analysis described in Hasbrouck (2001), for the S&P 500 index futures contract, the exchange traded fund (ETF) and the E-Mini futures contract.
- A third program (Build01.sas) performs data setup for Coint04.sas.
- A list and description of key files is given in the Appendix.

Other notes:

- I'm not distributing this code and documentation in the spirit of: "Here's a ready-to-go package that anyone with no experience of SAS or price discovery models can use." The programs are instead properly viewed as working analyses that are reasonably complete and documented. At best, they will serve as a starting point for a researcher applying these techniques to a new dataset. I've found that it's usually easier to modify someone else's code, rather than write the whole thing from the ground up.
- These programs are distributed at no charge as shareware. Although I believe the code and programs to be correct, I make no guarantees.
- While I welcome comments, feedback and inquiries regarding these programs, I can't promise a response. The programs are distributed "as is" and unsupported.
- The results reported in Hasbrouck (2001) are based on estimations performed with a suite of Matlab programs. The complexity of these programs is such that I doubt their general usefulness to others. The routines described in this document are written in SAS. Relative to the Matlab programs, they are simpler, though less computationally efficient. The results obtained in the SAS programs are substantially similar, though not identical to the Matlab results.
- The SAS code is also available (at my web site) for the cointegration analyses reported in Hasbrouck (2002). The code for Hasbrouck (2002) illustrates some stylized models. It is somewhat simpler than the present code, but not well suited to the analysis of real (as opposed to simulated) data.
- These programs use the basic SAS data handling and reporting tools. Estimation and forecasting is performed using PROC MODEL from the SAS/ETS (Economic

Time Series) package. Random-walk analysis is conducted using SAS/IML (Interactive Matrix Language). I also make extensive use of the SAS macro facility. SAS/ETS, SAS/IML and the macro facility are documented in separate SAS manuals and online documentation.

II. Principles

Price discovery analysis is based on the econometrics of cointegrated vector autoregressions. Hamilton (1994) provides an excellent textbook discussion of cointegration. Hasbrouck (1995) and Hasbrouck (1996) discusses microstructure applications. Recent discussions of alternative views of price discovery are presented in Baillie et al. (2002), de Jong (2002), Harris, McNish, and Wood (2002), Hasbrouck (2002) and Lehmann (2002)

Consider a price vector $p_t = [p_{1t} \ p_{2t} \ \dots \ p_{nt}]'$, where the p_i all refer to the “same” security (e.g., bid, ask and/or transaction prices at one or more market venues). A vector correction model of order K can be written as:

$$\Delta p_t = A_1 \Delta p_{t-1} + \dots + A_K \Delta p_{t-K} + \gamma (z_{t-1} - \mu_z) + u_t \quad (1)$$

where the A_i are square matrices of order n . The covariance matrix of the disturbances is

$$\text{Cov}(u_t) = E u_t u_t' = \Omega \quad (2)$$

The $\gamma(z_t - \mu_z)$ term contains the error correction coefficients. There are $n-1$ cointegrating vectors:

$$z_t = \begin{bmatrix} p_{1t} - p_{2t} \\ p_{1t} - p_{3t} \\ \vdots \\ p_{1t} - p_{n-1,t} \end{bmatrix} = F p_t, \text{ where } F = [I \quad -I_{n-1}] \quad (3)$$

for I_{n-1} the identity matrix of order $n-1$ and I a vector of ones; μ_z is the mean vector for the deviations; the elements of γ are the error correction coefficients.

Implementation note: The Matlab routines used to produce the estimates in Hasbrouck (2001) employed a two-step estimation procedure. In the first step, μ_z was estimated as the sample average \bar{z}_t . This estimate was then plugged in to equation (1) and the remaining parameters were determined via OLS. In the SAS programs described here, all parameters are estimated jointly using nonlinear least squares.

The vector moving average (VMA) representation of the model is:

$$\Delta p_t = B_0 u_t + B_1 u_{t-1} + B_2 u_{t-2} + \dots, \text{ where } B_0 = I \quad (4)$$

The B coefficients may be calculated by “forecasting” the system subsequent to a unit perturbation. For example, suppose that $\Delta p_t = 0$ and $z_t = \mu_z$ for $t = -1, -2, \dots$. At time $t = 0$, suppose that there is a shock of $u_0 = [1 \ 0 \ \dots \ 0]'$. Then,

$$\begin{aligned} \Delta p_0 &= \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ z_0 &= \mu_z + F \Delta p_0 \\ \Delta p_1 &= A_1 \Delta p_0 + \gamma z_0 \\ z_1 &= z_0 + F \Delta p_1 \\ \Delta p_2 &= A_1 \Delta p_1 + A_2 \Delta p_0 + \gamma z_1 \\ &\vdots \end{aligned} \quad (5)$$

The first column of B_0 is Δp_0 ; the first column of B_1 is Δp_1 , and so forth. To obtain the second column of B_0, B_1 , etc., we forecast the system subsequent to an initial shock of $u_0 = [0 \ 1 \ 0 \ \dots \ 0]$ and so forth. In this setting, we are primarily interested in the cumulative price changes:

$$C_k = \sum_{i=0}^k B_i \quad (6)$$

These are the cumulative impulse response functions. The first columns of the C_k describe the prices subsequent to a shock in the first price. It is useful to study the graphs of these. Of particular importance in the present setting is

$$C = \lim_{k \rightarrow \infty} C_k \quad (7)$$

(When the B 's are written as the lag polynomial $B(L)$, C is equivalent to $B(1)$.) The rows of C are all identical. Let c be any row of C . Then the variance of the (common) random-walk component of the prices is:

$$\sigma_w^2 = c \Omega c' \quad (8)$$

If Ω is diagonal,

$$\Omega = \begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ & & \ddots \\ 0 & 0 & \sigma_n^2 \end{bmatrix}. \quad (9)$$

The information share of the i th market is defined as

$$IS_i = \frac{c_i^2 \sigma_i^2}{\sigma_w^2} \quad (10)$$

If Ω is not diagonal, then IS_i is not uniquely defined. Instead, we determine lower and upper bounds \overline{IS}_i and \underline{IS}_i by considering the Cholesky factorizations of all of the rotations (permutations) of the disturbances.

III. Implementation considerations

This section describes various issues that arise in the SAS implementation of the model described above.

A. Calculating the impulse response functions

The model actually specified in the code is the model given in equation (1) transformed by moving the lagged price level on the r.h.s., to obtain:

$$p_t = p_{t-1} + A_1 \Delta p_{t-1} + \dots + A_K \Delta p_{t-K} + \gamma(z_{t-1} - \mu_z) + u_t \quad (11)$$

The impulse response functions are computed by setting $p_{-1} = p_{-2} = \dots = \begin{bmatrix} 0 \\ -\mu_z \end{bmatrix}$. That is, the system is “at rest”. At time $t = 0$, the system is shocked with a unit impulse to one of the prices. A unit shock to the j th price corresponds to setting

$$p_0^j = e_j - \begin{bmatrix} 0 \\ \mu_z \end{bmatrix} \quad (12)$$

where e_j is a vector of zeros with a “1” in the j th place. The system is then stepped forward from this point, with $u_1 = u_2 = \dots = 0$, to obtain p_1^j, p_2^j, \dots . Each of the p_t^j is an $n \times 1$ vector that specifies the prices at time t implied by the initial shock at time 0. We normalize these as:

$$p_t^{*j} = p_t^j + \begin{bmatrix} 0 \\ \mu_z \end{bmatrix} \quad (13)$$

The impulse response functions are then given as

$$C_k = \begin{bmatrix} p_k^{*1} & p_k^{*2} & \cdots & p_k^{*n} \end{bmatrix} \quad (14)$$

B. Reducing the number of coefficients

If t indexes seconds then even a brief lag length will result in an enormous number of coefficients. A five minute maximum lag, for example, will result in $K = 300$. The total number of coefficients is n^2K , so the problem quickly becomes unworkable.

To reduce the number of coefficients, I use two expedients.

- Constraining a set of coefficients (say, the coefficients of Δp_2 in the Δp_1 equation, for lags 21-30) to be constant.
- Constraining a set of coefficients to lie on a polynomial function of the lag.

To illustrate both of these techniques, , consider the *univariate* ($n=1$) autoregression:

$$\Delta p_t = a_1 \Delta p_{t-1} + a_2 \Delta p_{t-2} + \cdots + a_K \Delta p_{t-K} + u_t \quad (15)$$

where K is large.

Important Note: What I have described here is most emphatically a computational expedient. There is nothing in the (hypothetical) handbook of multivariate time series that ensures I can get away with this. The only way to have any confidence in the results of such an analysis is to perform sensitivity testing on the lag structure.

1. Polynomial distributed lags (PDL's)

Here, we reduce the number of parameters by constraining the a_i to lie on polynomials (in i). That is:

$$a_i = c_0 + c_1 i + c_2 i^2 + \cdots + c_d i^d \quad (16)$$

where d is the maximum degree of the polynomial. (In practice, I've never used d higher than 2 or 3.) Rather than fit a high-degree polynomial to the full set of K coefficients, I prefer to break up the set of K into smaller ranges, and fit a polynomial within each range. I keep the ranges short at the near lags, and let the ranges become larger at the distant lags.

Polynomial distributed lags are implemented using the SAS %PDL macro

2. Constant coefficients (Moving averages)

Here, we constrain the a_i to be constant over pre-specified ranges, for example,

$$\begin{aligned}
a_1 &= a_2 = \cdots = a_5 \\
a_6 &= a_7 = \cdots = a_{10} \\
a_{11} &= a_{12} = \cdots = a_{20} \\
a_{21} &= a_{22} = \cdots = a_{30} \\
a_{31} &= a_{32} = \cdots = a_{60} \\
a_{61} &= a_{62} = \cdots = a_{120} \\
a_{121} &= a_{122} = \cdots = a_{180} \\
a_{181} &= a_{182} = \cdots = a_{240} \\
a_{241} &= a_{242} = \cdots = a_{300}
\end{aligned}$$

These intervals correspond to the first five seconds, the second five seconds, and so on. Notice that the ranges are shorter for the near lags. This is sensible because the price dynamics are likely to be most rapidly changing over short lags.

In programming this, it is efficient to make use of SAS's moving average function `movavg`. For a time series x_t , the SAS function `movavgk(x)` returns the moving average of order k , i.e.,

$$\text{movavg}k(x_t) = (x_t + x_{t-1} + \cdots + x_{t-k})/k$$

Consider the initial terms of the univariate autoregressive model:

$$\begin{aligned}
\Delta p_t &= a_1 (\Delta p_{t-1} + \cdots + \Delta p_{t-5}) + a_6 (\Delta p_{t-6} + \cdots + \Delta p_{t-10}) + \cdots \\
&= 5a_1 \text{movavg}5(\Delta p_{t-1}) + 5a_6 \text{movavg}5(\Delta p_{t-6}) + \cdots
\end{aligned}$$

There is yet one more little programming trick. As written above, the MA specifications require the computation to carry (in a named variable) the sixth lagged price change Δp_{t-6} . We may eliminate this explicit reference by noting that (using the definition of the moving average function given above)

$$\text{movavg}5(x_{t-6}) = \frac{1}{5} (5 \times \text{movavg}5(x_{t-1}) + 10 \times \text{movavg}10(x_{t-1})),$$

i.e., $\text{movavg}5(x_{t-6})$ is a linear combination of $\text{movavg}5(x_{t-1})$ and $\text{movavg}10(x_{t-1})$. The constrained autoregression therefore has an equivalent representation, the first two terms of which are:

$$\Delta p_t = a_1^* \text{movavg}5(\Delta p_{t-1}) + a_6^* \text{movavg}10(\Delta p_{t-1}) + \cdots$$

C. Daily estimation and summary

The VECM price discovery model is generally valid only within a trading session. We would not normally expect overnight prices to follow the same dynamics. I therefore estimate the model (and all statistics) separately for each day. For any statistic, the upper bound of the first information share, for example, this results in a set

$$\overline{IS}_1^k \text{ for } k = 1, \dots, D$$

where D is the number of days in the sample. I then compute the mean estimate of \overline{IS}_1 averaged across days, and the usual standard error of this mean.

Alternatively, one might estimate jointly over all days. This would involve adding logic to the PROC MODEL specification that would set overnight lagged variables to “missing”.

IV. A sample program: CointTest.sas

CointTest.sas simulates a two-price system, estimates a VECM, computes impulse response functions, and performs a random-walk decomposition analysis (including information shares). It is a sample program designed to demonstrate the estimation procedure as cleanly as possible. It is therefore a stripped-down program, not a general one.

A. The model

The model involves a latent “true price” that follows a random-walk:

$$m_t = m_{t-1} + u_t \text{ where } u_t \overset{d}{\sim} N(0,1)$$

The first price is:

$$p_{1t} = m_t + e_{1t} \text{ where } e_{1t} \overset{d}{\sim} N(0,1)$$

The second price is:

$$p_{2t} = 4 + m_{t-2} + e_{2t} \text{ where } e_{2t} \overset{d}{\sim} N(0, (0.1)^2)$$

Thus, p_{2t} is lagged signal, but one with relatively higher precision. Where did the “4” come from? If you’d like a story, let’s call p_{1t} the bid price of the security; p_{2t} is the ask price; the ask adjusts somewhat later than the bid. Yes, it’s a simple story, but the point of this program is illustration rather than realism.

B. *CointTest01.sas: Notes*

1. The relevant files are:

CointTest01.sas	The program file
.\Log Files\CointTest01.log	The log output file
.\Listing Files\CointTest01.lst	The listing output file
.\SasMacros\RandomWalkAnalysis.sas	(see below).

I've put the log and listing files from my runs into separate directories, so that they won't be overwritten when you run the program.

2. The program contains some internal documentation.
3. The first line of the program is:

```
x 'cd c:\Active\SPY\Sp04';
```

On my machine, I run this program from the directory `c:\Active\SPY\Sp04`. The command sets this to the current directory. It is necessary because certain files (and in particular, the macro library) are located *relative* to the current directory. Obviously, you may wish to change this line to point to the directory that you are running from.
4. I make modest use of the SAS macro facility.

SAS macro variables can be identified by the “&” prefix, e.g., &nObs. SAS macro functions are prefixed by a “%”, e.g., “%RandomWalkAnalysis”.

V. A full analysis with real data

This section describes the programs used to perform the estimations reported in Hasbrouck (2001).

A. *Relevant files*

Build01.sas	program file
.\Listing Files\Build01.lst	listing file
.\Log Files\Build01.log	log file
Coint04.sas	program file
.\Log Files\Coint04.log	log output file
.\Listing Files\Coint04.lst	listing output file

B. *General conventions*

1. See the notes to CointTest01.sas (above)
2. I make extensive use of the SAS macro facility. SAS macro variables can be identified by the “&” prefix, e.g., &TempLib. SAS macro functions are prefixed by a “%”, e.g., %ModelVECM.

3. I have set the system options to NOMPRINT and NONOTES to minimize the log output. If you'd like to see the SAS code generated by the macros, use "MPRINT".
4. Some of my SAS macro routines have a provision for diagnostic output. This is controlled by the PrintLevel variable in the calling sequence. The default is generally 1 (printout), but for the iterations after the first, I set PrintLevel=0 (to suppress output).
5. Some of the macro subroutines are defined within the main program. Others are placed in a subdirectory called "SasMacros".
6. The program constructs a large number of temporary datasets. I place these in a library (directory) called &TempLib.
7. Variable names
 - a) The prices are denoted p1, p2, ..., p&nPrice.
 - b) The z_i are denoted z2,...,z&nPrice .
 - c) The μ_z are denoted zmean2,...,zmean&nPrice.

C. Overview of the programs (Figure 1)

1. The original dataset (bdata.dat) is a binary file with fixed length records. The program BUILD01.sas generates a SAS dataset named Prices. This dataset is sorted by date. BUILD01.sas also generates a dataset PriceMap. PriceMap has one record per date, and contains pointers to the data. (If you're familiar with the TAQ index files, you'll note the similarity.)

To modify BUILD01 for your data, you'll probably have to change the infile statement and the input statement.
2. COINT04 actually runs the analysis and generates the summary statistics. The impulse response functions are saved in a dataset called.

D. COINT04

This program is arranged as follows. Ultimately, the program will loop over all of the days in the sample. The key macros are generally written, though, to perform analysis for one day only. The macros and their functions are:

1. %DayBuild extracts the price data for a single day and builds a dataset (matrix) in which observation (row) corresponds to one second. The columns are the prices.

Using the index information from the PriceMap dataset, DayBuild first extracts (from the Prices dataset) all of the data for the day. The transpose step results in a

dataset that is a matrix with a row for each second of the day and a column for each price. At this stage, some of the prices are missing:

Obs	time	FES	FSP	SPY
868	10:00:04	1378.50	.	.
869	10:00:06	1379.75	.	.
870	10:00:07	1379.50	.	.
871	10:00:08	.	1379.50	.
872	10:00:13	1378.25	.	.
873	10:00:15	1379.00	1379.80	.
874	10:00:18	1378.25	.	.
875	10:00:19	.	1380.00	.
876	10:00:22	.	.	1380.78
877	10:00:23	1378.00	.	.

The key step in %DayBuild is a call to PROC EXPAND. PROC EXPAND is a SAS/ETS procedure that replaces the missing values with the most recent observation. After PROC EXPAND has run, the dataset looks like this:

Obs	time	p3	p2	p1
1800	10:00:01	1379.50	1379.80	1381.09
1801	10:00:02	1379.50	1379.80	1381.09
1802	10:00:03	1379.50	1379.80	1381.09
1803	10:00:04	1378.50	1379.80	1381.09
1804	10:00:05	1378.50	1379.80	1381.09
1805	10:00:06	1379.75	1379.80	1381.09
1806	10:00:07	1379.50	1379.80	1381.09
1807	10:00:08	1379.50	1379.50	1381.09
1808	10:00:09	1379.50	1379.50	1381.09
1809	10:00:10	1379.50	1379.50	1381.09
1810	10:00:11	1379.50	1379.50	1381.09
1811	10:00:12	1379.50	1379.50	1381.09
1812	10:00:13	1378.25	1379.50	1381.09
1813	10:00:14	1378.25	1379.50	1381.09
1814	10:00:15	1379.00	1379.80	1381.09
1815	10:00:16	1379.00	1379.80	1381.09
1816	10:00:17	1379.00	1379.80	1381.09
1817	10:00:18	1378.25	1379.80	1381.09
1818	10:00:19	1378.25	1380.00	1381.09
1819	10:00:20	1378.25	1380.00	1381.09

2. %GraphPrices produces a price plot for the given day. It is not generally called. I use it as a diagnostic/utility routine.
3. %ModelVECM estimates the VECM
4. %ImpulseResponse constructs the vector moving average (VMA) representation of the model. The impulse response functions are computed using the PROC MODEL solve statement.
5. %GraphImpulseResponse produces impulse response plots.

6. %RandomWalkAnalysis2 computes the information shares and related statistics. This is perhaps the most complicated of the macros. The guts of the calculation is in the macro %RandomWalkAnalysis. This is a SAS IML (“Interactive Matrix Language”) program.
7. %DoAllDates loops over all the days in the sample. It calls (in order) the macros previously defined.
8. After %DoAllDates has run, there are invocations of PROC MEAN to summarize (across days in the sample) the information shares and impulse response functions.
9. A final call to %GraphImpulseResponse produces the impulse response functions (Figure 2).

E. Other notes

1. The last lines of Coint04.sas contain code to produce the impulse response function graphs. These lines are commented out.
2. The program regularly writes out the current date and time. On my machine (800 MHz Pentium, running Windows XP), the program took about an hour. (See the log file.)

VI. Modifying these programs for your own analyses: How to proceed.

A. Play around with CointTest01

This will give you a feel for how a price discovery VECM is estimated.

B. Modify the input statements of Build01.sas to conform to your data.

C. Modifying and running Coint04.sas

1. You should be able to use the principal estimation program, Coint04.sas without extensive modification. There are, however, some macro variables in the first part of the program that will almost certainly have to be changed. These include variables that name the price variables and point to temporary directories.
2. After you’ve modified Coint04.sas, try to run it for one day. After each macro definition is an invocation that has been commented out. Uncomment these and run to verify that each macro is functioning correctly.
3. Run %DoAllDates with &nDates set to 2. (This will execute the program for the first days in your sample.)
4. Run %DoAllDates for all dates.

5. Fine-tune the last part of the program, which produces summary statistics and graphs. %DoAllDates saves the key estimates in two datasets: RWSummary and IRSummary. You shouldn't have to rerun %DoAllDates (which is likely to be quite time consuming) when you're playing around with the summary reporting and graphing.

VII. References

- Baillie, R. T., Booth, G. G., Tse, Y., Zobotina, T., 2002. Price discovery and common factor models. *Journal of Financial Markets*, forthcoming.
- de Jong, F., 2002. Measures of contributions to price discovery: A comparison. *Journal of Financial Markets*, forthcoming.
- Hamilton, J.D., 1994. *Time Series Analysis*. (Princeton: Princeton University Press).
- Harris, F. H. d., McNish, T. H., Wood, R. A., 2002. Common factor components vs. information shares: alternative approaches to price discovery research. *Journal of Financial Markets*, forthcoming.
- Hasbrouck, J., 1995. One security, many markets: Determining the contributions to price discovery. *Journal of Finance* 50, 1175-99.
- Hasbrouck, J., 1996. Modeling microstructure time series. In: Maddala, G. S. and Rao, C. R. (Eds.), *Handbook of Statistics 14: Statistical Methods in Finance*. Elsevier North Holland, Amsterdam, 647-692.
- Hasbrouck, J., 2001. Intraday price formation in US equity index markets. Unpublished working paper. Stern School of Business, New York University.
- Hasbrouck, J., 2002. Stalking the efficient price in empirical microstructure specifications. *Journal of Financial Markets*, forthcoming.
- Lehmann, B. N., 2002. Some desiderata for the measurement of price discovery across markets. *Journal of Financial Markets*, forthcoming.

Appendix: List of files distributed

Directory	File	Description
Root directory (“.”)	CointTest01.sas	Sas program file
	Coint04.sas	“”
	Build01.sas	“”
	CointDocumentation01.pdf	This file
	Bdata.dat	Binary data file containing S&P 500 ETF and futures data
	Prices.sas7bdat	Sas dataset
	PriceMap.sas7bdat	“”
	.\SasMacros	ComputeCorrelations.sas
.\SasMacros	CurrentDate.sas	“”
	NumObs.sas	“”
	RandomWalkAnalysis.sas	“”
	RowDifference.sas	“”
	.\Listing Files	CointTest01.lst
.\Listing Files	Coint04.lst	“”
	Build01.lst	“”
	.\Log Files	CointTest01.log
.\Log Files	Coint04.log	“”
	Build01.log	“”

Figure 1

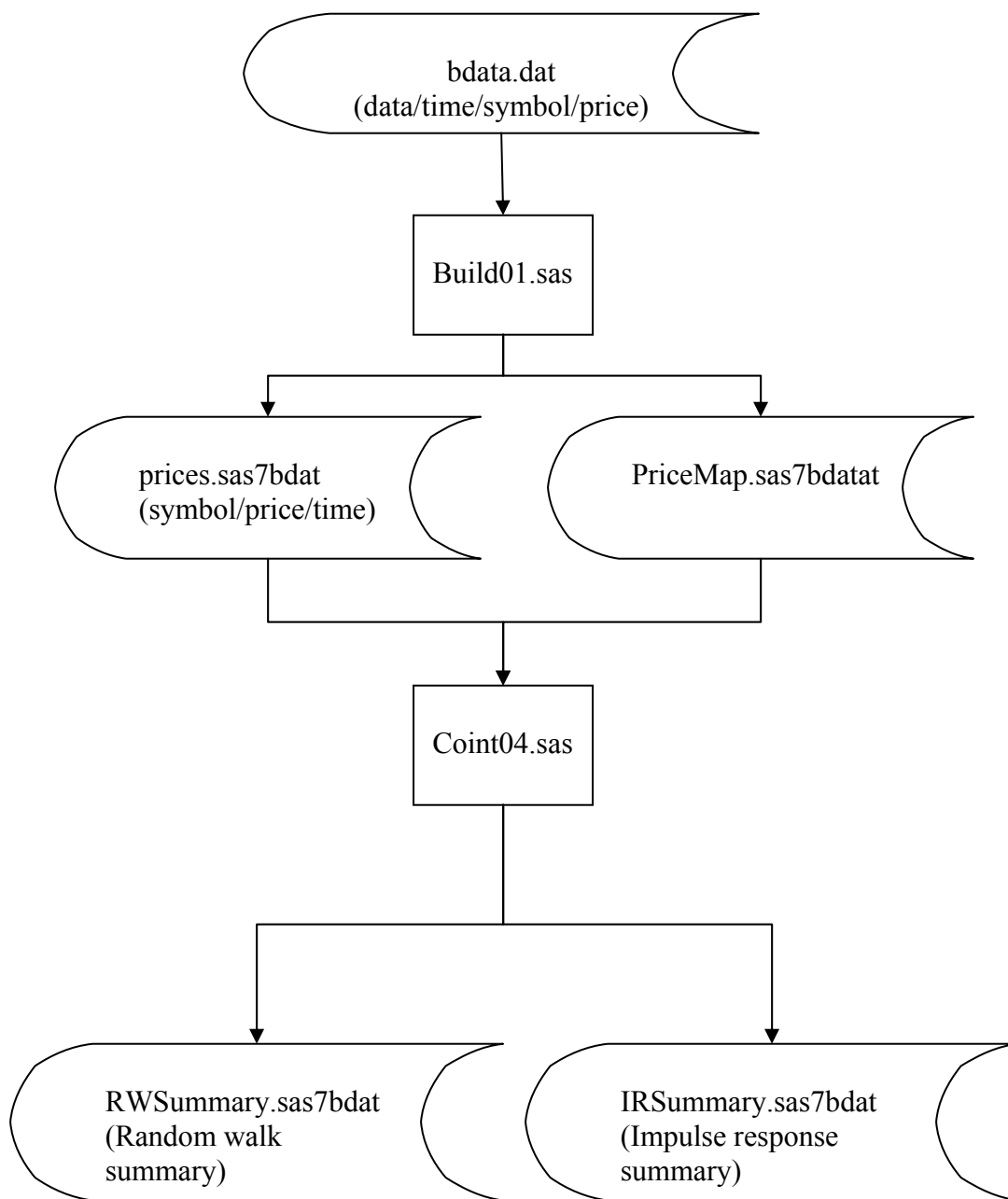


Figure 2. Impulse response functions

