A Set of Matlab Codes for Calculating Shock Elasticities

Yiran Fan and Paymon Khorrami*

June 15, 2017

Contents

*Please email `paymon@uchicago.edu` with any comments or questions.

1 Introduction

This document serves as a handbook for the usage of Matlab codes to calculate shock elasticities, as well as some related auxiliary tools.

Assume we are given stochastic processes

$$\begin{aligned} dX_t &= \mu_X(X_t)dt + \sigma_X(X_t)dW_t \\ d\log C_t &= \mu_C(X_t)dt + \sigma_C(X_t)dW_t \\ d\log S_t &= \mu_S(X_t)dt + \sigma_S(X_t)dW_t \end{aligned} \tag{1}$$

where X is 1-dimensional, the Brownian motion dW_t has k dimension, and

$$\begin{aligned} \mu_X, \mu_C, \mu_S &: \mathbb{R} \mapsto \mathbb{R} \\ \sigma_X &: \mathbb{R} \mapsto \mathbb{R}^{1\times k} \\ \sigma_C, \sigma_S &: \mathbb{R} \mapsto \mathbb{R}^{1\times k} \end{aligned}$$

For a given process M that satisfies

$$d\log M_t = \beta(X_t)dt + \alpha(X_t)\cdot dW_t$$

Define

$$\varepsilon^1_M(x,t) = \nu(x)\cdot\left[\sigma_X'(x)\left(\frac{\partial}{\partial x}\log E\Big[\frac{M_t}{M_0}|X_0 = x\Big]\right) + \alpha(x)\right] \tag{2}$$

$$\varepsilon^2_M(x,t) = \frac{E\Big[\frac{M_t}{M_0}\nu(X_t)\cdot\alpha(X_t)|X_0 = x\Big]}{E\Big[\frac{M_t}{M_0}|X_0 = x\Big]} \tag{3}$$

where $\nu : \mathbb{R} \mapsto \mathbb{R}^k$ selects the shocks.

We are intersted in calculating

$$\varepsilon^1_C(x,t) - \varepsilon^1_{SC}(x,t) \tag{4}$$

and

$$\varepsilon^2_C(x,t) - \varepsilon^2_{SC}(x,t) \tag{5}$$

For a full discussion on shock elasticity, please see Borovička and Hansen (2016), Borovička, Hansen, and Scheinkman (2014), and Hansen (2012).

2 The Core Code

2.1 Methodology

Calculating (2) and (3) require solving the conditional expectations

$$E\Big[\frac{M_t}{M_0}|X_0 = x\Big] \tag{6}$$

$$E\Big[\frac{M_t}{M_0}\nu(X_t)\cdot\alpha(X_t)|X_0 = x\Big] \tag{7}$$

which can be done by a Feynman-Kac approach.

Define

$$\phi_t(x) \equiv E\Big[\frac{M_t}{M_0}\phi_0(X_t)|X_0 = x\Big]$$

Then, for one-dimensional x, $\phi_t(x)$ satisfies:

$$\frac{\partial\phi}{\partial t} = \Big(\beta(x) + \frac{1}{2}|\alpha(x)|^2\Big)\phi + \Big(\mu_X(x) + \sigma_X(x)\alpha(x)\Big)\frac{\partial\phi}{\partial x} + \frac{1}{2}|\sigma_X(x)|^2\frac{\partial^2\phi}{\partial x^2} \tag{8}$$

Let $\phi_0(x) = 1$ and $\phi_0(x) = \nu(x)\cdot\alpha(x)$ for (6) and (7) respectively.

2.2 Matlab Code: `SEimfd1.m`

The Matlab code `SEimfd1.m` is designed to calculate elasticities (4) and (5). The code takes inputs in the following order:

- `model`:

  A structure specifies $\mu_X, \sigma_X, \mu_C, \sigma_C, \mu_S, \sigma_S$ as defined in (1). The structure should have 6 fields, named as `muX, sigmaX, muC, sigmaC, muS`, and `sigmaS` respectively. Each field should be a function handle. Specifically, `muX, muC`, and `muS` take a vector of different values for x as input and return a vector having the same length. `sigmaX, sigmaC`, and `sigmaS` may return matrices when $k > 1$. In this case, they take a COLUMN vector as input, and the returned matrix should have the same number of rows as the input, where the first row corresponds to the results of the first element of the input, so on so forth.

- `domain`:

  A structure specifies the following fields:

  - `domain.x`: A two-element vector gives the lower and upper bounds of x.
  - `domain.nx`: The number of grids to discretize x.
  - `domain.dt`: The length of time stamp.

- `domain.T`: The termination time, or the number of total time periods wanted.

- `bc`: boundary conditions for PDE (8)

  Each boundary condition should be in the following form:

$$0 = a_0(t) + a_1(t)\phi(x) + a_2(t)\frac{d}{dx}\phi(x) + a_3(t)\frac{d^2}{dx^2}\phi(x) \tag{9}$$

  `bc` is a structure that have two fields correspond to the boundary conditions for $M = C$ and $M = SC$ respectively, named as `bc.C` and `bc.SC`. `bc.C` and `bc.SC` then specify lower and upper boundary conditions in subfields `l` and `u`, which both have 4 more subfields called `const, level, deriv1`, and `deriv2` that correspond to a_0, a_1, a_2, and a_3 in (9) respectively.

  For instance, if the user wants to specify $\phi'(x) = 0$ at the lower boundary for $M = C$ case, one can type

```
bc.C.l.const = @(t) 0;
bc.C.l.level = @(t) 0;
bc.C.l.deriv1 = @(t) 1;
bc.C.l.deriv2 = @(t) 0;
```

- `param`:

  A structure contains all the models' parameters.

- `X0`:

  The initial points at which the user wants to calculate the elasticities. If multiple, `X0` should a COLUMN vector.

- `Nu`:

  A function handle corresponds to the $\nu(x)$ defined in (2) and (3). If $k > 1$, it takes a COLUMN vector as input, and return a matrix. The returned matrix should have the same number of rows as the input, where the first row corresponds to the results of the first element of the input, so on so forth.

- `modelname`:

  A string provides a directory where the automatically generated elasticity plots should be saved to. If empty, the plots will not be saved.

`SEimfd1.m` will return two structures: `elas1` (for ε_M^1) and `elas2` (for ε_M^2). Each contains three fields: `g`, `sg`, and `out`, corresponding to ε_C, ε_{SC}, and $\varepsilon_C - \varepsilon_{SC}$ respectively. Each field is a matrix with the length of `X0` rows and one plus $\frac{\texttt{domain.T}}{\texttt{domain.dt}}$ columns, where the ith row tracks the elasticity for the ith element in `X0` over the time periods [0 : `domain.dt` : `domain.T`]. The code will also autometically generate four plots for ε_C^1, $\varepsilon_C^1 - \varepsilon_{SC}^1$, ε_C^2, and $\varepsilon_C^2 - \varepsilon_{SC}^2$.

The user should define `model` and `param` "cleverly". Here is an example. (A full example will be given in Section 2.3) Suppose a model says

$$\begin{aligned}
\sigma_X(x) &= x\Big(\alpha(x) - 1\Big)\sigma \\
\alpha(x) &= \begin{cases} \frac{1}{1-\lambda(1-x)}, & \text{if } x > x^* \\ \frac{1}{(1+m)x}, & \text{if } x \le x^* \end{cases} \\
x^* &= \frac{1-\lambda}{1-\lambda+m} \\
\lambda &= 0.6 \\
m &= 4 \\
\sigma &= 0.09
\end{aligned}$$

One can code in the following way:

```
param.lambda = 0.6;
param.m = 4;
param.sigma = 0.09;
param.xstar = (1-param.lambda) ./ (1-param.lambda+param.m);
param.alpha = @(x) Alpha(x,param);

model.sigmaX = @(x) x.*(param.alpha(x)-1).*param.sigma;
```

In another file, we define:

```
function alpha = Alpha(x,param)

x1 = x(x<=param.xstar);
alpha1 = 1./(1+param.m)./x1;
x2 = x(x>param.xstar);
alpha2 = 1./(1-param.lambda.*(1-x2));

if (iscolumn(x))
        alpha = [alpha1;alpha2];
else
        alpha = [alpha1 alpha2];
end
```

An even simpler way to get rid of additional file is to replace the previous `param.alpha` line to be:

```
param.alpha = @(x) ...
        (x<=param.xstar) .* (1./(1+param.m)./x)...
        + (x>param.xstar) .* (1./(1-param.lambda.*(1-x)));
```

`SEimfd1.m` exploits another file, `CMimfd1.m`, to solve the PDEs. The next section will discuss `CMimfd1.m`. But for terminal users, no knowledge for `CMimfd1.m` is required.

2.3 Matlab Code: `SEimfd1.m`, An Example

Suppose the model (See He and Krishnamurthy (2013)) says

$$\begin{aligned}
\mu_X(x) &= x\Big[-\frac{\ell}{1+\ell}\rho + \big(\alpha(x)-1\big)^2\sigma^2\Big] \\
\sigma_X(x) &= x\Big(\alpha(x)-1\Big)\sigma \\
\mu_S(x) &= -\frac{\rho}{1+\ell} - g + \alpha(x)\sigma^2 - \frac{1}{2}\alpha(x)^2\sigma^2 \\
\sigma_S(x) &= -\alpha(x)\sigma \\
\mu_C(x) &= -\mu_S(x) - \rho \\
\sigma_C(x) &= -\sigma_S(x)
\end{aligned}$$

where

$$\begin{aligned}
\alpha(x) &= \begin{cases} \frac{1}{1-\lambda(1-x)}, & \text{if } x > x^* \\ \frac{1}{(1+m)x}, & \text{if } x \le x^* \end{cases} \\
x^* &= \frac{1-\lambda}{1-\lambda+m}
\end{aligned}$$

and

$$\begin{aligned}
\lambda = 0.6; \quad m = 4; \quad \sigma = 0.09 \\
g = 0.02; \quad \ell = 1.84; \quad \rho = 0.04
\end{aligned}$$

One can use the following code to calculate the shock elasticity

```
%% Define Parameters

param.g = 0.02; param.sigma =0.09; param.m = 4;
param.lambda = 0.6; param.rho = 0.04; param.l = 1.84;
```

```
param.xstar= (1-param.lambda) / (1-param.lambda+param.m);
param.alphaX = @(x) 1./(1-param.lambda*(1-x)) ...
          .*(x>param.xstar) ...
           + 1/(1+param.m)./x.*(x<=param.xstar);


%% Model Construction

model.muX = @(x) x.*( ...
          -param.l/(1+param.l).*param.rho ...
          + (param.alphaX(x)-1).^2.*param.sigma.^2);
model.sigmaX = @(x) x.*(param.alphaX(x)-1).*param.sigma;
model.muS = @(x) -param.rho/(1+param.l) ...
          -param.g+param.alphaX(x).*param.sigma.^2 ...
           -0.5.*param.alphaX(x).^2*param.sigma.^2;
model.sigmaS = @(x) -param.alphaX(x).*param.sigma;
model.muC = @(x) -model.muS(x) - param.rho;
model.sigmaC = @(x) -model.sigmaS(x);


%% Define Boundary Conditions

bc.C.l.const = @(t) 0; bc.C.l.level = @(t) 0;
bc.C.l.deriv1 = @(t) 1; bc.C.l.deriv2 = @(t) 0;

bc.C.u.const = @(t) 0; bc.C.u.level = @(t) 0;
bc.C.u.deriv1 = @(t) 1; bc.C.u.deriv2 = @(t) 0;

bc.SC.l.const = @(t) 0; bc.SC.l.level = @(t) 0;
bc.SC.l.deriv1 = @(t) 1; bc.SC.l.deriv2 = @(t) 0;

bc.SC.u.const = @(t) 0; bc.SC.u.level = @(t) 0;
bc.SC.u.deriv1 = @(t) 1; bc.SC.u.deriv2 = @(t) 0;


%% Define Domain

domain.x = [0 1]; domain.nx = 10000;
domain.dt = 0.25; domain.T = 100;


%% Select Shock

Nu = @(x) 1 + 0*x;
```

```
%% Calculate

[elas1, elas2] = SEimfd1(model,domain,bc,param,X0,Nu,[])
```

One line that may need more explanation is

```
Nu = @(x) 1 + 0*x;
```

Usually, if any of $\mu_X, \sigma_X, \mu_S, \sigma_S, \mu_C, \sigma_C, \nu$ in the model is a constant, please write it as the constant plus `0*x`. In this way, when a vector is passed in, the function will give a vector out rather than a constant.

2.4 Matlab Code: `CMimfd1.m`

Exploiting the implicit finite difference scheme, the code `CMimfd1.m` is designed to solve a generalized version of PDE (8):

$$\frac{\partial \phi}{\partial t} = C(x)\phi + B(x)\frac{\partial \phi}{\partial x} + A(x)\frac{\partial^2 \phi}{\partial x^2} \tag{10}$$

The code takes inputs in the following order:

- `xx`: The grid of x for the finite difference scheme, must be a COLUMN vector and equally spaced.
- `tt`: The grid of t for the finite difference shceme, equally spaced, starting from the first non-zero value.
- `A,B,C`: The coefficients correspond to PDE (10). Each should have the same length as `xx`.
- `phi0`: The COLUMN vector corresponds to the initial condition $\phi_0(x)$. It should have the same length as `xx`.
- `X0`: The points of X_0 are under watch, must be a COLUMN vector.
- `bc`: The boundary condition for PDE (10), which is discussed in Section 2.2.

The code outputs a matrix with the length of `X0` rows and one plus the length of `tt` columns, where the ith row tracks the solution of ϕ for the ith element in `X0` over the time periods [0 `tt`].

3 Auxiliary Tools

3.1 Stationary Density

3.1.1 Methodology

Consider the SDE

$$dX_t = \mu(X_t)dt + \sigma(X_t)dW_t$$

Then the stationary density $h(x)$ satisfies the Fokker-Planck equation

$$0 = -\frac{d}{dx}\Big[\mu(x)h(x)\Big] + \frac{1}{2}\frac{d^2}{dx^2}\Big[D(x)h(x)\Big] \tag{11}$$

where

$$D(x) \equiv |\sigma(x)|^2$$

and $x \in [\underline{x}, \bar{x}]$.

Define

$$J(x) \equiv \mu(x)h(x) - \frac{1}{2}\frac{d}{dx}\Big[D(x)h(x)\Big]$$

Then by construction, Equation (11) can be expressed as

$$\frac{d}{dx}J(x) = 0$$

Obviously, this has solution

$$J(x) = J^*$$

where J^* is any constant.

Thus, the two conditions needed for the original second-order ODE (11) are: i) $\int_{\underline{x}}^{\bar{x}} h(x) = 1$, and ii) a prespecified J^*. In this way, the original ODE will turn into the following first-order ODE:

$$J^* = \mu(x)h(x) - \frac{1}{2}\frac{d}{dx}\Big[D(x)h(x)\Big] \tag{12}$$

with condition $\int_{\underline{x}}^{\bar{x}} h(x) = 1$.

A numerical way for solving ODE (12) will be:

- Step 1: Is one of the boundaries reflecting? If yes, set $J^* = 0$ and fix arbitrarily $\check{x}$ and $h(\check{x})$ and go Step 3. If no, specify another condition, say $h(\hat{x}) = 0$, (For example, $h(\bar{x}) = 0$) and go Step 2.
- Step 2: Fix an arbitrary number $\check{x} \in [\underline{x}, \bar{x}]$ and an arbitrary value $h(\check{x})$. Guess a J^* and integrate Equation (12) from $\check{x}$ to $\hat{x}$. And then shoot $h(\hat{x}) = 0$ by varying J^*.
- Step 3: Solve Equation (12) using the resulting J^* and the same $h(\check{x})$. Then rescale by $\int_{\underline{x}}^{\bar{x}} h(x) = 1$. (The scaling coefficient will give also scale the resulting J^* to the right value)

3.1.2 Matlab Code: `Density.m`

The Matlab code `Density.m` is designed to implement the numerical method described above for solving ODE (12). The current version is for $k = 1$.

The code takes inputs in the following order:

- `process`: A structure specifies μ_X and σ_X. Both are function handles, taking a vector in and return a vector of the same length.
- `xx`: The grid of x, must be equally spaced.
- `cutoff`: A value corresponds to $\check{x}$.
- `param`: A structure contains all the model's parameters.
- `imposing0`: A value corresponds to $\hat{x}$. If empty, the code will set $J^* = 0$ and skip Step 2 described above.
- `modelname`: A string provides a directory where the automatically generated elasticity plots should be saved to. If empty, the plots will not be saved.

The code will return in the following order:

- `den`: A structure contains the solved $h(x)$, where `den.d` are the values of the density function and `den.x` are the values of x's at which the function is evaluated.
- `cumden`: A structure contains the solved cumulative, where `cumden.cd` are the values of the density function and `cumden.x` are the values of x's at which the function is evaluated.

The code will also generate a plot for the density function automatically.

3.1.3 Expectation under the Stationary Density: Matlab Code `Expectation.m`

The Matlab code `Expectation.m` calculates the unconditional expectation of a function given the density. It takes the inputs in the following order:

- `den`: A structure comes from the output of `Density.m`, containing the density.
- `integrand`: A function handle that the user wants to take expectation, taking a vector in and return a vector of the same length.

The code will output a value corresponds to the unconditional expectation.

3.2 Multiplicative Functional Martingale Decomposition

3.2.1 Methodology

For a given process M that satisfies

$$d\log M_t = \beta(X_t)dt + \alpha(X_t)\cdot dW_t$$

and

$$dX_t = \mu(X_t)dt + \sigma(X_t)dW_t$$

Define

$$\mathbb{B}f = \Big(\beta(x) + \frac{1}{2}|\alpha(x)|^2\Big)f + \Big(\mu_X(x) + \sigma_X(x)\alpha(x)\Big)\frac{\partial f}{\partial x} + \frac{1}{2}|\sigma_X(x)|^2\frac{\partial^2 f}{\partial x^2}$$

The decomposition requires solving the following Sturm-Liouville equation

$$\mathbb{B}e = \eta e \tag{13}$$

where $e(x)$ should be a strictly positive function.

Numerically, this problem can be converted to an eigenvalue-eigenvector problem via a finite difference scheme.

3.2.2 Matlab Code: `Decompose.m`

The Matlab code `Decompose.m` solves the above problem (13) in a numerical way. The code takes inputs in the following order:

- `process`:

  A structure specifies μ_X, σ_X, μ_M, and σ_M as in equation (1) notation. The structure should have 4 fields, named as `muX, sigmaX, muM`, and `sigmaM` respectively. Each field should be a function handle. See Section 2.2 for detailed discussion.

- `domain`:

  A structure specifies the following fields:

  - `domain.x`: A two-element vector gives the lower and upper bounds of x.
  - `domain.nx`: The number of grids to discretize x.

- `bc`:

  boundary conditions for ODE (13). See Section 2.2 for detailed discussion. Here `bc` should contain the following

```
bc.l.const;   bc.l.level;   bc.l.deriv1;   bc.l.deriv2;
bc.u.const;   bc.u.level;   bc.u.deriv1;   bc.u.deriv2;
```

- `param`:

  A structure contains all the models' parameters.

The code returns in the following order:

- `eta`: a value corresponds to the solve η.
- `e`: a vector corresponds to the solved function $e(x)$.
- `xx`: a vector corresponds to the x's at which the solved function $e(x)$ is evaluated.

CAUTION: In theory the all-positive eigenfunction $e(x)$ should correspond to the smallest (in absolute value) eigenvalue. However, current version of code will go through the first 10 eigenvalues to see if there exists one eigenvalue whose eigenfunction is all positive (and stop there). A message will be sent on which the eigenvalue the program picks. If the message says, "quit in 1 eigenvalue", then it should be safe. But if it quits on any other eigenvalues, use with caution. If the program cannot find any all positive eigenvector in the first 10, it will send out an error message.

3.2.3 Calculating Expecatations for Shock Elasticity under the Changed Measure: Matlab Code `SEimfd1e.m`

The Matlab code `SEimfd1e.m` calculates shock elasticity exploiting the change of measure. It has exactly the same inputs and outputs as `SEimfd1.m` described in Section 2.2, except it requires another input `eig` which is a structure contains the following:

- `eig.C.eta`: The decomposed η for $M = C$.
- `eig.C.e`: The decomposed eigenfuction $e(x)$ for $M = C$. Notice that this should be function handle instead of a vector. It takes a vector and returns a vector of the same length.
- `eig.C.dloge`: A function handle corresponds to $\frac{d}{dx}\log e(x)$ for $M = C$.
- `eig.SC.eta`: The decomposed η for $M = SC$.
- `eig.SC.e`: The decomposed eigenfuction $e(x)$ for $M = SC$. Notice that this should be function handle instead of a vector. It takes a vector and returns a vector of the same length.
- `eig.SC.dloge`: A function handle corresponds to $\frac{d}{dx}\log e(x)$ for $M = SC$.

3.3 Term Structure

3.3.1 Methodology

Given specifications in (1), the term structure can be calculated by

$$-\frac{1}{t}\log E\Big[\frac{S_t}{S_0}|X_0 = x\Big] \tag{14}$$

where the expectation can be calculated by the PDE approach described in Section 2.1.

3.3.2 Matlab Code: `TSimfd1.m`

The Matlab code `TSimfd1.m` calculates the term structure. It takes inputs in the following order:

- `model`: A structure specifies $\mu_X, \sigma_X, \mu_S, \sigma_S$ as defined in (1). See Section 2.2 for a detailed discussion.
- `domain`: Same as described in Section 2.2.
- `bc`: Same as described in Section 2.2.
- `param`: Same as described in Section 2.2.
- `X0`: Same as described in Section 2.2.
- `modelname`: Same as described in Section 2.2.

The code outputs a matrix with the length of `X0` rows and the length of `tt` columns, where the ith row tracks the term structure for the ith element in `X0` over the time periods [`domain.dt` : `domain.dt` : `domain.T`].

References

Borovička, Jaroslav, and Lars Peter Hansen, 2016, Term Structure of Uncertainty in the Macroeconomy, *working paper*.

——— , and José A. Scheinkman, 2014, Shock elasticities and impulse responses, *Mathematics and Financial Economics*.

Hansen, Lars Peter, 2012, Dynamic Valuation Decomposition within Stochastic Economies, *Econometrica*.

He, Zhiguo, and Arvind Krishnamurthy, 2013, Intermediary Asset Pricing, *The American Economic Review*.