Shock Elasticities Toolbox

Yiran Fan, Paymon Khorrami, Joseph Huang *

January 19, 2018

Contents

*Please email `paymon@uchicago.edu` or `jhuang12@uchicago.edu` with any comments or questions.

1 Introduction

The shock elasticities toolbox is a set of user-friendly MATLAB functions that compute shock elasticities defined as follows (for a full discussion on shock elasticities, please refer to Borovička and Hansen (2016), Borovička, Hansen, and Scheinkman (2014), and Hansen (2012)).

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 n-dimensional, the Brownian motion dW_t has k shocks (hence k dimensions), and

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

In the equations above, C_t is the cash flow or consumption process and S_t is the stochastic discount factor. We are interested in computing the **shock exposure elasticities** and **shock price elasticities**.

For a given cash flow or consumption process, its shock **exposure elasticity** can be computed as

$$\varepsilon_C^1(x,t) = \nu(x)\cdot\left[\sigma_X'(x)\left(\frac{\partial}{\partial x}\log E\Big[\frac{C_t}{C_0}|X_0=x\Big]\right)+\sigma_C(x)\right] \tag{2}$$

$$\varepsilon_C^2(x,t) = \frac{E\Big[\frac{C_t}{C_0}\nu(X_t)\cdot\sigma_C(X_t)|X_0=x\Big]}{E\Big[\frac{C_t}{C_0}|X_0=x\Big]} \tag{3}$$

where $\nu : \mathbb{R} \mapsto \mathbb{R}^k$ encodes the direction of the shocks.

To compute the price elasticities for a given cash flow process C_t and stochastic discount factor S_t, we first compute the shock cost elasticities by substituting process C in (2) and (3) with a new process SC (and define them as ε_{SC}^1 and ε_{SC}^2). To get the **price elasticities**, we compute:

$$\varepsilon_C^1(x,t) - \varepsilon_{SC}^1(x,t) \tag{4}$$

and

$$\varepsilon_C^2(x,t) - \varepsilon_{SC}^2(x,t) \tag{5}$$

All expressions in (2) and (3) can be computed explicitly, with the exception of conditional expectations

$$E\Big[\frac{C_t}{C_0}|X_0 = x\Big] \tag{6}$$

$$E\Big[\frac{C_t}{C_0}\nu(X_t)\cdot\sigma_C(X_t)|X_0 = x\Big] \tag{7}$$

We compute the conditional expectations by using the Feynman-Kac formula.

Define

$$\phi_t(x) \equiv E\Big[\frac{C_t}{C_0}\phi_0(X_t)|X_0 = x\Big]$$

Then $\phi_t(x)$ satisfies:

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

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

2 Installation

2.1 Adding Toolbox

After downloading the toolbox, unzip and place the scripts in a folder. Afterwards, add the folder to your MATLAB search path by doing [1]

```
addpath('yourpath');
```

where `yourpath` is the path of the toolbox folder.

2.2 Creating MEX Function

We use the finite differences method to solve the PDE (8). To improve performance, the PDE solver is built in C++ and a C++ package called Eigen developed by Guennebaud, Jacob, et al. (2010). However, no knowledge of C++ is required, as the toolbox can pass data from MATLAB into C++ through a MEX function. All you need to do is to build the MEX function by calling

```
mex constructSolve.cpp
```

After that, you should see the following message:

```
MEX completed successfully.
```

The installation is complete. The next step is optional.

2.3 Linking with Pardiso (Optional)

To solve higher dimensional problems [2], it would be useful to employ Pardiso, a high performance matrix solver to solve PDE (8). To link with Pardiso, first go to www.pardiso-project.org and download the library by requesting an academic license for free, after which you should receive an email with the download link and a license key (a long string of numbers and letters). Follow the steps below:

- Copy and paste the license key into a plain file and save it as `pardiso.lic` in the path of the toolbox.
- Download the correct Pardiso library on the download page based on your operating system and compiler; save it in the path of the toolbox.
- Execute the following command in MATLAB

[1] (Optional): To avoid repeating this command in each session that this toolbox is employed, try to add the folder onto the search path permanently (link to instructions).

[2] Pardiso could be complicated to set up initially, so if your model is less than two dimensions, it may not be necessary to use Pardiso.

```
mex constructSolvePardiso.cpp
-L[path of toolbox] -l[name of library]
-llapack -lblas -lpthread
```

Note: Replace `[path of toolbox]` with the path of the toolbox and `[name of library]` with the file name of the library downloaded **excluding** `lib` **in the beginning and** `.dylib` **at the end** . For example, if the toolbox is located in `/Users/Yourname/shockElas` and the filename of the library is `libpardiso500-MACOS-X86-64.dylib`, compile with the following command:

```
mex constructSolvePardiso.cpp
-L/Users/Yourname/shockElas -lpardiso500-MACOS-
   X86-64
-llapack -lblas -lpthread
```

After that, you should see:

```
MEX completed successfully.
```

This step is then complete.

3 Using the Toolbox

3.1 Computing Shock Elasticities: `computeElas`

This is the function that computes the shock exposure and price elasticities (both the first and second types). It takes in the following arguments:

- `domain`: An $S \times n$ matrix that has all the grid points in the state space. n is the number of dimensions and S is the total number of grid points initialized. The grid points must be vectorized from MATLAB's `ndgrid` format[3]. For example, suppose the model has two state variables and each state variable has 100 grid points. The `domain` argument would be a $10,000 \times 2$ matrix.

- `model`:
  A structure that contains the information described in (1). Specifically, it has the following fields

  - `muX`: μ_X as in (1) computed for each of the grid points in `domain`; an $S \times n$ matrix;
  - `sigmaX`: A cell of matrices that contains σ_X computed for each of the grid points in `domain`; each element of the cell, a matrix, corresponds to the numerical values of the volatility of the state variable at the grid points specified in `domain`. For example, suppose there are two state variables and three shocks. `sigmaX` would be a cell of two $S \times 3$ matrices.
  - `muC`: μ_C as in (1) computed for each of the grid points in `domain`; an $S \times 1$ matrix;
  - `sigmaC`: σ_C as in (1) computed for each of the grid points in `domain`; an $S \times k$ matrix (assuming there are k shocks);
  - `muS`: μ_S as in (1) computed for each of the grid points in `domain`; an $S \times 1$ matrix;
  - `sigmaS`: σ_S as in (1) computed for each of the grid points in `domain`; an $S \times k$ matrix (assuming there are k shocks);
  - `dt`: The length of time step;
  - `T`: Total number of time periods. Suppose `dt = 1/12` implies one month and `model.T = 120`. The total time that the elasticities are solved for is 10 years.

- `bc`:
  A structure that specifies the boundary conditions in this form:

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

[3] For example, `[X,Y] = ndgrid(1:2:19,2:2:12); domain = [X(:) Y(:)]`.

where a_0 is a scalar, a_1, a_2, and a_3 are $1 \times n$ vectors (n being the number of dimensions). The user should configure `bc` in this way:

- `a0`: a scalar that corresponds to a_0 in (9)
- `level`: a $1 \times n$ that corresponds to a_1 in (9)
- `first`: a $1 \times n$ that corresponds to a_2 in (9)
- `second`: a $1 \times n$ that corresponds to a_3 in (9)
- `natural`: a boolean (`true` or `false`) that determines whether to use natural boundaries[4]; if `true`, the fields above would be ignored.

For example, to implement the boundary conditions $0 = \frac{\partial}{\partial x}\phi(x)$ (i.e. first derivatives are set to zero), one should set up `bc` by doing:

```
bc.a0 = 0; bc.first = [1 1]; bc.second = [0 0];
bc.level = [0 0]; bc.natural = false;
```

- `x0`: A matrix that contains all the starting points (i.e. x in (2)). For l starting points, `x0` should be an $l \times n$ matrix (n being the number of dimensions).
- `optArgs` (optional): A structure that contains optional arguments. It has two fields -
  - `usePardiso`: a boolean that determines whether to use Pardiso to solve high dimensional and large matrix systems when solving the PDE (8). It is `false` by default. To use this feature, one must complete section 2.3.
  - `priceElas`: a boolean that determines whether to compute price elasticities. By default, the function returns both exposure and price elasticities (i.e. `priceElas` is set to `true` by default).

The function returns exposure and price elasticities. Suppose the user inputs l starting points (i.e. `x0` has l rows) and T time periods and chooses to compute price elasticities (the default option), the function will return two cells:

- `expoElas`: a cell of l elements where each element is a structure that contains two fields `firstType` and `secondType`. Each field is a $T \times n$ matrix (n being the number of dimensions). As the name suggests, `firstType` is the first type shock exposure elasticities for the ith starting point (the ith row in input `x0`) up to T time periods, whereas `secondType` is the second type shock exposure elasticities[5].
- `priceElas`: similar to `expoElas`, but contains the price elasticities.

[4]Natural boundaries simply take the limit of PDE (8), where second derivatives at the boundary are approximated by one grid point inside the boundary.

[5]For example, to access the first type shock exposure elasticity of the second shock at the third starting point (in the third row of input `x0`), type `expoElas{3}.firstType(:,2)`

3.2 Computing Stationary Distribution: `simStatDent`

The toolbox uses simulations to compute stationary density as it is an approach that has guaranteed success if implemented correctly. Given the stochastic process dX_t specified in (1), function `simStatDent` simulates by

1. Given a starting point x at $t = 0$ (labeled x_0), it determines the drift μ_X and σ_X;
2. Draw the Brownian vector $dW_t \sim \mathcal{N}(\mathbf{0}_k,\, dt \times \mathbb{I}_k)$ and compute dX_t, where $\mathbf{0}_k$ is a $k \times 1$ vector and $\mathbb{I}_k$ is the $k \times k$ identity matrix (k being the number of shocks) so that x at $t + dt$ can be determined;
3. Repeat the two steps above T/dt times so that it has a history of T/dt time periods.

For each starting point x_0, the function will generate a history of time paths. We recommend that the user use multiple starting points as each history is embarrassingly parallel (so parallel computing can be used to speed up the process)[6].

The function takes in the following inputs: Let n be the number of dimensions, S total number of grid points, k number of shocks, and l number of starting points

- `stateSpace`: a cell of grid vectors for each of the state variables. It mirrors the input for MATLAB's `ndgrid` function[7].
- `hist0`: an $l \times n$ matrix that contains the starting points for the simulations . Having l starting points will generate l number of time paths.
- `dt`: the length of time step. This corresponds to dt in (1) and the variance of the Brownian shock in the simulations.
- `T`: the total length of time for each time path. For each starting point, the function will simulate for T/dt time periods. If you have l starting points, the function will generate l time paths and $l \times (T/dt)$ time points in total.
- `drifts`: A cell that contains n elements where each element is either a function handle that computes the drift of the state variable or a column vector that contains the values of the drift for the state variable at each grid point of the state space;

[6] By default, the function uses `parfor` which employs all cores available on your computer, assuming that MATLAB's Parallel Computing Toolbox is installed. To check whether it is installed, simply type `ver` in the MATLAB command window and you should see Parallel Computing Toolbox listed. If not, go to this link for more information

[7] For example, `stateSpace = {0:1:10,10:5:50}` is a valid input and creates two state variables.

- `vols`: A cell that contains n elements where each element is either a function handle that computes the volatility of the state variable (a $1 \times m$ vector) or an $S \times m$ that contains the values of volatility for the state variable at each grid point of the state space;

The function outputs a cell of l elements, where each element is a $(T/dt) \times n$ matrix, corresponding to each history of points.

4 Examples

4.1 He and Krushnamurthy (2013)

This example borrows from He and Krishnamurthy (2013). The model specifications are as follows.

The state variable is characterized by

$$\begin{aligned}\mu_X(x) &= x\Big[-\frac{\ell}{1+\ell}\rho + (\alpha(x)-1)^2\sigma_Y^2\Big]\\ \sigma_X(x) &= x(\alpha(x)-1)\sigma_Y\end{aligned}$$

And the stochastic discount factor by:

$$\begin{aligned}\mu_S(x) &= \frac{\rho}{1+\ell} + \mu_Y - \alpha(x)\sigma_Y^2 + \frac{1}{2}\alpha(x)^2\sigma_Y^2\\ \sigma_S(x) &= \alpha(x)\sigma_Y\end{aligned}$$

We are interested in two cash flows G^1 (aggregate endowment) and G^2 (expert consumption):

$$\begin{aligned}\mu_{G^1}(x) &= \mu_Y - \frac{1}{2}\sigma_Y^2\\ \sigma_{G^1}(x) &= \sigma_Y\\ \mu_{G^2}(x) &= \mu_S(x) - \rho\\ \sigma_{G^2}(x) &= \sigma_S(x).\end{aligned}$$

The parameters in the equations above are either computed or set as:

$$\begin{aligned}\mu_Y = 0.02 \quad \sigma_Y = 0.09 \quad m = 4\\ \lambda = 0.6 \quad \rho = 0.04 \quad \ell = 1.84\\ \alpha(x) = \begin{cases}[1-\lambda(1-x)]^{-1}, & \text{if } x > x^*\\ (1+m)^{-1}x^{-1}, & \text{if } x \le x^*.\end{cases}\\ x^* = \frac{1-\lambda}{1-\lambda+m} \in (0,1).\end{aligned}$$

The file `HeKrushnamurthy.m` computes the shock elasticities for both cash flows and the stationary density through simulations.

4.2 Borovička, Hansen, Hendricks, Scheinkman (2010)

This example `BHHS.m` borrows from Borovička, Hansen, Hendricks, and Scheinkman (2011). Different from the example in 4.1, the model is two dimensional and

has three shocks. The example computes elasticities for six starting points. The state variables are characterized by

$$\mu_X = \begin{pmatrix} \lambda_g(\overline{g} - g) \\ \lambda_s(\overline{s} - s) \end{pmatrix}$$
$$\sigma_X = \sqrt{s} \begin{pmatrix} \sigma_g' \\ \sigma_s' \end{pmatrix}$$

And the cash flow and stochastic discount factor are characterized by

$$\mu_C = \iota(q) - \delta + g - \frac{1}{2} s \|\sigma_A\|^2$$
$$\sigma_C = \sqrt{s}\sigma_A$$
$$\mu_S = -r - \frac{1}{2}\|\pi\|^2$$
$$\sigma_S = -\pi$$

Finally, π and r are computed by

$$q = \frac{a + 1/\phi}{\rho + 1/\phi}$$
$$\pi = \sqrt{s}\left[(\gamma - 1)\left(\alpha_g \sigma_g + \alpha_s \sigma_s\right) + \gamma \sigma_A\right]$$
$$r = \rho + \iota(q) + g - \delta - s\left[\gamma\|\sigma_A\|^2 + (\gamma - 1)\sigma_A \cdot \left(\alpha_g \sigma_g + \alpha_s \sigma_s\right)\right]$$

where

$$\alpha_g = \frac{1}{\rho + \lambda_g}$$
$$\alpha_s = \left[\frac{(\gamma - 1)\sigma_s \cdot (\alpha_g \sigma_g + \sigma_A) + \rho + \lambda_s}{(\gamma - 1)\|\sigma_s\|^2}\right]\left[\sqrt{1 - \frac{\|\alpha_g \sigma_g + \sigma_A\|^2 - \frac{\|\sigma_A\|^2}{(\gamma - 1)}}{\left(\frac{(\gamma - 1)\sigma_s \cdot (\alpha_g \sigma_g + \sigma_A) + \rho + \lambda_s}{(\gamma - 1)\|\sigma_s\|}\right)^2}} - 1\right]$$

and the parameters are given in Table 1.

Lastly, we will then pick the following orthogonal shocks:

$$\left(\frac{\sigma_A}{\|\sigma_A\|}, \frac{\sigma_g}{\|\sigma_g\|}, \frac{\sigma_s}{\|\sigma_s\|}\right) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

.

In particular, some of the arguments for function `computeElas` are as follows

Let S be the total number of grid points:

Parameter	Value	Description
a	0.0117	TFP
$\|\sigma_A\|$	0.0078	TFP vol.
$\bar{g}$	0	Long run risk latent variable mean value
λ_g	0.0210	Mean-reversion speed latent variable
$\|\sigma_g\|$	0.00034	Latent variable vol. multiplier
$\bar{s}$	1	"Normalized" agg. TFP long run vol.
λ_s	0.0130	Mean-reversion speed agg. TFP vol.
$\|\sigma_s\|$	0.0078	"Normalized" agg. TFP vol. of vol.
ϕ	1	Adjustment cost
δ	0.0050	Depreciation rate
γ	10	Risk aversion
ρ	0.0042	Rate of time preference

Table 1: Calibration Parameters

- `model`
  - `muC`: an $S \times 1$ matrix
  - `sigmaC`: an $S \times 3$ matrix (remember, there are **3** shocks)
  - `muS`: an $S \times 1$ matrix
  - `sigmaS`: an $S \times 3$ matrix
  - `muX`: an $S \times 2$ matrix (there are **2** state variables)
  - `sigmaX` a cell that has 2 elements, with each element being an $S \times 3$ matrix
- `x0`: a 6×2 matrix (there are **6** starting points and **2** dimensions)

Also, the example plots out the elasticities and thus demonstrates how to access the computed results.

References

Borovička, Jaroslav, and Lars Peter Hansen, 2016, Term structure of uncertainty in the macroeconomy, *Handbook of Macroeconomics* 2, 1641–1696.

——— , Mark Hendricks, and José A Scheinkman, 2011, Risk-price dynamics, *Journal of Financial Econometrics*.

Borovička, Jaroslav, Lars Peter Hansen, and José A. Scheinkman, 2014, Shock elasticities and impulse responses, *Mathematics and Financial Economics* 8.

Guennebaud, Gaël, Benoît Jacob, et al., 2010, Eigen v3, .

Hansen, Lars Peter, 2012, Dynamic Valuation Decomposition within Stochastic Economies, *Econometrica* 80.

He, Zhiguo, and Arvind Krishnamurthy, 2013, Intermediary Asset Pricing, *The American Economic Review* 103(2).