Methods to solve non-smooth problems with high dimensionality

Filip Rozsypal*

May 22, 2016

1 Introduction

In this chapter, I describe how a projection algorithm can be extended to solve dynamic stochastic equilibrium models of high dimensionality and non-smooth decision rules. I show how this framework can be applied to solve a multi-sector business cycle model with endogenous growth such as the one in the first chapter of this dissertation.

I present and extend further the method of ergodic grids developed by Judd *et al.* (2012). Furthermore, I allow the shocks to have kinks in the way they affect the outcomes in the model. Having a methodology to solve models with kinks is important, because endogenous outcomes of standard shocks[1] can have two kinds of effects on outcomes: first *binary effects*, i.e. whether an action is taken or not, and second *intensity effects*, i.e. conditional on the action being taken, the size of the shock matters for the magnitude of the outcome. For example, following a low productivity draw, a firm might decide not to start exporting. In such a situation, conditional on the productivity being bad enough, the actual value of the productivity shock does not matter. On the other hand, if the productivity shock is above a certain threshold, the firm will start exporting and exports might be increasing the the size of the shock. The kink is then an outcome of the firm's optimisation and it has implications for aggregate outcomes.

*Centre for Macroeconmics, London Schoolof Economics. Email: f.rozsypal1@lse.ac.uk. I thank for useful comments to Wouter den Haan, Kenneth Judd and Serguei Maliar.

[1]Meaning shocks which have standard support, like normal shocks.

By construction, models with kinks cannot be solved by standard perturbation methods for two reason. First, linearisation smooths out the effect of the kinks. Second, the linearisation is performed around a steady state, which traditionally corresponds to the situation with the mean value of shocks. However, in the example with export, the mean value of productivity shock is either above or below the threshold, which means that in the non-stochastic steady state either all firms are exporting or there is no export at all. Therefore, it is necessary to use some global method instead of relying on local approximations.

The challenge of using global methods on large models is the curse of dimensionality caused by the grids and numerical quadrature which makes the requirements on computational time and memory grow exponentially. This chapter presents a framework which tackles these numerical problems. This chapter also contains simplified fragments of code to help understand the algorithms it describes.[2]

To test the methods described in this chapter, I consider a simple real business cycle model enriched with an extra source of non-linearity. Given that the goal of the exercise in this chapter is to highlight numerical properties of selected solution methods, this additional non-linearity makes an otherwise standard RBC model too complicated for standard solution methods.

1.0.1 RBC model with variable investment costs

Consider an economy with standard Cobb-Douglas aggregate production function

$$Y_t = \exp(z_t) K_t^{\alpha},$$

where the representative households maximises its lifetime utility of consumption

$$\max \sum_{t=0}^{\infty} \beta^t \mathsf{E}_0 \frac{C_t^{1-\sigma}}{1-\sigma}$$

subject to the following budget constraint:

$$C_t + K_{t+1} + \Gamma(K_{t+1}) = Y_t - (1-\delta)K_t.$$

[2]The actual codes used in this chapter are available for download from the author's webpage: http://people.ds.cam.ac.uk/fr282/chapter2_codes.zip

This budget constraint is standard apart from the investment function Γ. It is there to make the model more non-linear and therefore harder to solve. In the absence of this cost we know that the policy function of RBC model is very close to being linear (which is also why first-order perturbation works in the context of a basic RBC). To test the methods described in this chapter, I assume

$$\Gamma(K_t) = \kappa_0 \left(\frac{1}{1 + \exp(\kappa_1(K_t - \kappa_2))} + \frac{1}{1 + \exp(\kappa_3(K_t - \kappa_4))} \right)$$

where $\boldsymbol{\kappa}$ controls the location of two highly non-linear regions in the adjustment costs. Choosing $\kappa_0 = 0$ leads to $\Gamma(K) \equiv 0$ and the model becomes a plain vanilla real business cycle model. On the other hand, setting κ_1 and κ_3 high will introduce almost a kink at the locations given by κ_2 and κ_3. The shape and the value of the coefficients can be seen in figure 1. For the vast majority of time, the extra investment costs are essentially zero so the optimal behavior should be very close to the one of plain vanilla RBC model. However, for very low levels of capital it strongly incentivises the agent to save and for very high levels of capital to consume, relative to the plain RBC model. The effect of the Γ function on the simulated series can be seen in figure 10, page 23.

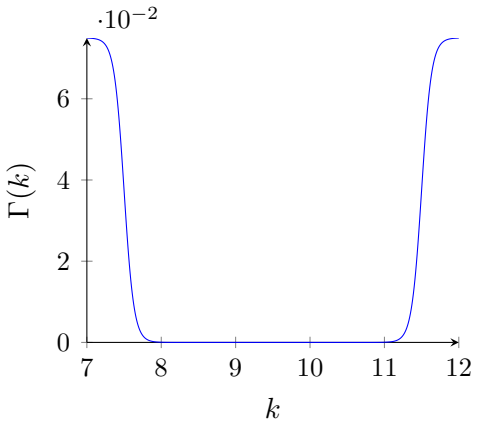


Figure 1: Extra investment costs occur for very low and very high levels of capital. This forces the agent to save relatively more for low levels of capital and consume more for high levels of capital compared to standard RBC model. The values generating this function are: $\kappa_0 = 0.075, \kappa_1 = 15 = -\kappa_3, \kappa_2 = 7.5, \kappa_4 = 11.5$.

Finally, the aggregate productivity follows an AR(1) process:

$$z_{t+1} = \rho z_t + \varepsilon_{t+1}$$

where $\varepsilon \sim N(0, \sigma_\varepsilon^2)$. The parameters in the model are $\kappa_0 = 0.075$, $\kappa_1 = 15$, $\kappa_2 = 7.5$, $\kappa_3 = -15$, $\kappa_4 = 11.5$, $\alpha = 0.33$, $\beta = 0.95$, $\delta = 0.025$, $\rho = 0.95$, $\sigma_\varepsilon = 0.02$ and $\sigma = 2$. The value for κs was chosen such that it mildly restricts where the model would live for $\kappa_0 = 0$. The solution of this model can be found in section 3.4.1 on page 22.

To solve this model by projection algorithm, I approximate the saving function. If *the order of the approximation* is equal to n, then the approximated policy function is a combination of polynomials of degree n in *both* capital k and productivity z and *all of the cross products*. The resulting function is

$$k_{t+1}(k_t, z_t) \equiv \phi_{\boldsymbol{\xi}}(k_t, z_t) = \boldsymbol{X}(k_t, z_t)\boldsymbol{\xi},$$

where $\boldsymbol{\xi}$ has 4 elements for $n = 1$ (constant, linear in k, linear in z and cross product between k and z). For a general order n, there are $(n+1)^2$ parameters in $\boldsymbol{\xi}$.

2 Selected existing literature on solving large models with global methods

In this section I review the recent contributions to the numerical methods literature for obtaining global solutions which are particularly useful for highly dimensional settings. Most notably, these are two recent contributions, Judd, Maliar and Maliar (2011) and Judd, Maliar and Maliar (2012)[3]

The key challenge for a global solution to large macroeconomic models is *the curse of dimensionality*, the fact that the number of points in a product grid grows exponentially with the dimension of the model and hence such a grid becomes quickly computationally infeasible. The idea to substitute the product grid with a set of points generated endogenously by simulating the model was pioneered by Marcet (1988) and den Haan and Marcet (1990). Their Parametrised expectation algorithm (PEA) used a simulated path of the model to implement Monte Carlo integration to evaluate the expectation in the Euler equation.

[3]For overview see , made significant contributions and opened the road for further research, which this chapter contributes to Maliar and Maliar (2014).

2.1 Numerically stable methods

Judd *et al.* (2011) improved PEA in the following way. They used the grid coming from the set of all simulated points. However, they evaluated the expectation in the Euler equation by quadrature and introduced a set of more robust regression approaches to recover the parameters of the parametrized policy function. Following the notation by Maliar and Maliar (2014, page 364), the solution (i.e. the policy function which solves the model) then has form of

$$\gamma_t = \boldsymbol{X}_t\boldsymbol{\xi} + \varepsilon_t, \tag{1}$$

where $\boldsymbol{X}\boldsymbol{\xi}$ is the approximation of true policy function γ and ε captures the difference between the approximated policy function and the true one. Just as in standard regression, a high degree of collinearity between individual components of state space representation $\boldsymbol{X}_t$ complicates the search for the value of parameters $\boldsymbol{\xi}$.

Judd *et al.* (2011) address this problem by suggesting alternatives to a standard OLS type of regression. First, they de-mean and normalise the variables in $\boldsymbol{X}$. They also use orthogonal polynomials rather than directly using the variables themselves. Finally, they discuss robust alternatives to ordinary least squares, including least absolute deviations and principal components. They show that the combination of these steps can increase the precision of the solution by several orders of magnitude.

2.2 Ergodic grid construction

Judd *et al.* (2012) improve on their previous method by using *ergodic grids*. In the simulation, there are some areas which are visited much more often than others. This suggests that the expectation is computed at very similar points unnecessarily many times when using the whole set of the simulated points, like in the PEA algorithm.

To address this problem, Judd *et al.* (2012) introduced a clustering algorithm which selects a grid from all the simulated points in the following way:

1. compute the distance among the points[4]
2. cut the outliers (points in the simulation where the number of other points in a δ neighborhood is less than some threshold)

[4]The distance is defined over standardised variables, i.e. the simulation data is transformed using a principal components approach to make the deviations along different directions comparable.

3. construct the grid

   (a) pick a random point and discard all points which are closer than a given value δ

   (b) repeat until there are no points left either to selected or to discard

4. transform the resulting grid back by inverting the normalisation done in the first step

Possible implementation of this algorithm can be seen in figure 2.

As another step to facilitate the solution of multidimensional models, Judd *et al.* (2012) also suggest a procedure based on iteration of the Euler equation instead of using some standard Newton algorithm based solver. The reason is that the iteration is very fast even for very large models, whereas Newton derivative-based optimisation algorithms require the knowledge of the Jacobian matrix, which is typically obtained by finite differences which is very costly in high dimension.

The Euler equation can be written as $1 = \beta \mathsf{E} \frac{u'(c_{t+1})}{u'(c_t)} R_{t+1}$. However, during the process of finding the best approximation of the policy function, the Euler equation is not going to hold.[5] Judd *et al.* (2012) use the following observation: if $1 > \beta \mathsf{E} \frac{u'(c_{t+1})}{u'(c_t)} R_{t+1}$, then it would be optimal to save more, because the interest rate does not compensate enough for the difference in marginal utilities. This means that tomorrow's capital needs to be higher than what it is suggested by the (current iteration) of the policy function. The trick is then to use this suggested difference to update the policy function.

If one multiplies both sides of the Euler equation by k_{t+1}, and using tilde to acknowledge the fact that the Euler equation does not hold perfectly, then in the context of the simple RBC model from the beginning of this chapter, it would be as follows:

$$\tilde{k}_{t+1} = \beta \mathsf{E} \frac{u'(c_{t+1})}{u'(c_t)} R_{t+1} k_{t+1} = \rho k_{t+1} \tag{2}$$

and then project the left-hand side $\tilde{k}_{t+1}$ onto the state vector (instead of k_{t+1}) to obtain the policy rule parameters, using the fact that the values of $\rho > 1$ are associated with savings which is too low: marginal utility tomorrow is higher than marginal utility today, which implies lower consumption tomorrow.

In what follows, when I mention *iterative solution*, I mean the solution obtained by iterating equation (2). In contrast, a *solver solution* uses a solver to find $\boldsymbol{\xi}$ which minimize

[5] By not holding I mean that the error is higher than the one given by the best possible coefficients for given shape of the policy function approximation.

```
function Grid = get_grids_sim(Σ,minN,maxN,ddelta)
%% get distance and density, following JMM(2015)
Sigma_demean = (Sigma-repmat(mean(Sigma),size(Sigma,1),1))./ ...
    repmat(sqrt(var(Sigma)),size(Sigma,1),1);
[~,~,V] = svd(Sigma_demean); %singular value decomposition
Sigma_normalised = Sigma_demean*V;
D_mat = distmat(Sigma_normalised); %compute distance matrix
g_hat = ...; % density as defined in JMM(2012), equation 2

%% cut off outliers
if ddelta>0
        sort Σ according to g_hat and cut outliers corresponding to delta
end

%% select the grid
multiplier=1; N_points_chosen=0 % loop initialisation
%outer loop chooses the right threshold
while (N_points_chosen<=max_points) && (N_points_chosen>=min_points)
    threshold = median(median(D_mat))*multiplier; %distance threshold to be eliminated
    final_set_normalised = zeros(size(Sigma_normalised));

    %set of points to be considered, at the start all points need to be considered
    consider = ones(size(Sigma_normalised,1),1); %1 for points still to be considered,
                                                 %0 for the points already eliminated
    in_set = zeros(size(Sigma_normalised,1),1);  %set of already chosen points in the grid

    %inner loop pickes the grid for a particular value of diameter threshold
    while ~all(consider==0) %stop if all points have been considered
        index          = find(consider,1,'first'); %find the first point to consider
        consider(index) = 0;                       %do not consider it anymore
        in_set(index)   = 1;                       %add it to the chosen set
        consider = consider.*(D_mat(index,:)>threshold)'; %keep considering only points
            further than threshold away
    end
    N_points_chosen = sum(in_set);
    if N_points_chosen>max_points % adjust the multiplier if needed
        multiplier = multiplier*1.15 %too many points, increase the threshold
    else N_points_chosen<min_points
        multiplier = multiplier*0.95; %too few points, decrease the threshold
    end
end
Grid = Sigma(find(in_set),:);  %find the points in the original Sigma
```

Figure 2: Algorithm to find ergodic grid. Inputs are simulated series $\mathbf{\Sigma}$, dimension of the grid (minN,maxN) and how to deal with outliers (ddelta).

the errors on the grid directly. My implementation of the iterative solution method is outlined in figure 3.

```
while cont
    %% simulate the model only if needed
    if simulate
        simulate = 0;
        update_grids = 1;

        simulate_the_model;

        %check whether new grid is needed
        if simulation moments close to the previous ones
            update_grids = 0;
        end
    end
    %% update grids
    if update_grids
        update_grids = 0;
        [...] = get_grids_sim(...); % generate grids
        [...] = renormalize(...);% renormalize policy function
        k_change_cum = zeros(size(grid)); %re-initialize cumulative capital counter
    end

    %% update policy function parameters
     y = βE[u_c(Σ_{t+1},ξ)/u_c(Σ_t,ξ)R_{t+1}(Σ_t,ξ)] k_{t+1}(Σ_t,ξ)    %left hand side
    X = f_X(Σ); %regressors
    ξ_new=(X'X)^{-1}X'y; %new guess
    ξ=λξ_new+(1-λ)ξ; %homotopy

    %check grid is consistency with the behaviour implied with the policy function
    if saving not balanced
      move the grid
    end

    %stop only if the simulation has not changed
    if converged(ξ_new,ξ_old)
        cont = 0; % solution converged
    end

    %check for divergence
    if ξ not converging
        if increase in grid might help %try with larger grid or give up
            make grid larger next time
            update_grids = 1;
        else
            cont = 0; %convergence not achieved, stopping
        end
    end

    k_change_cum = k_change_cum +k(grid,ξ_new)-k(grid,ξ_old); %change in policy funciton
    if max(abs(k_change_cum)) is too large
        simulate = 1; %re-simulate
        k_change_cum = 0; %re-initialize counter
    end
end
```

Figure 3: Core loop which solves the model iteratively with adaptive endogenous grids.

3 New methods

3.1 Elimination of loops for projection problems

MATLAB programming environment is a popular tool for solving macroeconomic models. However, it is known to be relatively slow compared to lower level programming languages like Fortran or C++. Aruoba and Fernandez-Villaverde (2014) demonstrate this fact using a value function iteration algorithm to solve a standard neoclassical growth model. In particular, loops are known to be computationally very slow in MATLAB relative to other programming languages.[6]

One particularly bad implementation of value function iteration can be seen in figure 4. This code is going to be very slow because of the presence of loops. This means that the odds are stacked against MATLAB when comparing the speed of finding a solution using value function iteration as in Aruoba and Fernandez-Villaverde (2014).

```
for each point k_i on capital grid k
  for each point z_j on productivity grid z
    for each possible saving decision k'_m
      for each Gauss–Hermite quadrature node ε_n ∈ ε
        compute V(k_i, k'_m, z_j, ε_n)
      end
      V(k_i, k'_m, z) = Σ_{n=1}^{N} w_n V(k_i, k'_m, z_j, ε_n)
    end
    V(k_i, z_j) = max_{k'} V(k_i, z_j)
  end
end
check convergence of V(k, z)
```

Figure 4: Value function algorithm. This particular implementation is very slow in MATLAB because of the pervasive loops.

On the other hand, MATLAB is very fast when using matrices. The popular term *vectorisation* describes the implementation of loop-heavy code in a way which eliminates loops and substitutes them (where possible) with matrix multiplications. Because projection algorithms are more suitable to vectorisation than value function iteration, MATLAB is likely not to lag behind lower level languages as much when the speed is evaluated in this setting rather than value function iteration. In fact, projection algorithms can be implemented in a way that eliminates all loops apart from the one which controls the convergence.

[6] Aruoba and Fernandez-Villaverde (2014) found found MATLAB to be approximately 10 times slower than Fortran or C++.

A projection algorithm parametrizes a policy function (for example the saving function) and then minimizes errors induced by optimality of the model (for example Euler equation)

$$\boldsymbol{\xi} = \arg\min |E_{\boldsymbol{\xi}}(\boldsymbol{k}, \boldsymbol{z})|$$

where the error function E is

$$E_{\boldsymbol{\xi}}(k_t, z_t) = \mathsf{E}\left[\frac{u'(c_{\boldsymbol{\xi}}(k_{t+1}, z_{t+1}))}{u'(c_{\boldsymbol{\xi}}(k_t, z_t))}(1 + r_{\boldsymbol{\xi}}(k_{t+1}, z_{t+1}) - \delta)\right] - 1 \tag{3}$$

and $|.|$ is the standard Euclidean norm. The policy function for saving is approximated using some flexible functional form parametrized by a set of parameters $\boldsymbol{\xi}$: $k_{t+1} = f_{\boldsymbol{\xi}}(k_t, z_t)$ and consumption then follows from the budget constraint: $c_{\boldsymbol{\xi}}(k_t, z_t) = \exp(z_t)k_t^{\alpha} + (1-\delta)k_t - f_{\boldsymbol{\xi}}(k_t, z_t)$ and the interest rate is equal to the marginal productivity of capital $r(k_t, z_t) = \alpha \exp(z)k_t^{\alpha-1}$. The expectation operator is then numerically approximated using Gauss-Hermite quadrature of order N giving nodes ε_i and weights $w_i, i = 1, \ldots, N$ (assuming that the errors are normally distributed). Then the error function at a particular point (k_t, z_t) becomes

$$E_{\boldsymbol{\xi}}(k_t, z_t) = \sum_{i=1}^{N} w_i \left(\frac{u'\left(\exp(\rho z_t + \varepsilon_i)(f_{\boldsymbol{\xi}}(k_t, z_t))^{\alpha} + (1-\delta)f_{\boldsymbol{\xi}}(k_t, z_t) - f_{\boldsymbol{\xi}}(f_{\boldsymbol{\xi}}(k_t, z_t), \rho z_t + \varepsilon_i)\right)}{u'(\exp(z_t)k_t^{\alpha} + (1-\delta)k_t - f_{\boldsymbol{\xi}}(k_t, z_t))} \times \right.$$
$$\left. \times (1 + r(f_{\boldsymbol{\xi}}(k_t, z_t), \rho z_t + \varepsilon_i) - \delta) \right) - 1 \tag{4}$$

The error of the Euler equation $E_{\boldsymbol{\xi}}(k_t, z_t)$ is computed on a grid for capital $\boldsymbol{k}$ and productivity $\boldsymbol{z}$ and the $\boldsymbol{\xi}$ which minimizes $|E_{\boldsymbol{\xi}}(\boldsymbol{k}, \boldsymbol{z})|$ is selected. The most straightforward implementation of this approach uses loops, as demonstrated in figure 5.

```
for each point k_i on capital grid k
  for each point z_j on productivity grid z
      for each gauss-hermite quadrature node ε_n ∈ ε
        compute E_ξ(k_i, z_j, ε_n)
      end
      E_ξ(k_i, z_j) = Σ_{n=1}^{N} w_n E_ξ(k_i, z_j, ε_n)
  end
end
ξ = arg min |E_ξ(k, z)|
```

Figure 5: Projection algorithm with loops

However, the loops can be eliminated. The trick is to re-write the error function as a function of future realisations of uncertainty and to accept vectors instead of just scalars. First, let $\tilde{E}_{\boldsymbol{\xi}} : \mathbb{R}^3 \to \mathbb{R}$ so the value of the Euler equation error is computed for a given quadrature node (on top of given capital and productivity)

$$\tilde{E}_{\boldsymbol{\xi}}(k_t, z_t, \varepsilon_i) = \left(\frac{u'\left(\exp(\rho z_t + \varepsilon_i)(f_{\boldsymbol{\xi}}(k_t, z_t))^\alpha + (1-\delta) f_{\boldsymbol{\xi}}(k_t, z_t) - f_{\boldsymbol{\xi}}(f_{\boldsymbol{\xi}}(k_t, z_t), \rho z_t + \varepsilon_i)\right)}{u'(\exp(z_t) k_t^\alpha + (1-\delta) k_t - f_{\boldsymbol{\xi}}(k_t, z_t))} \times \right.$$
$$\left. \times \left(1 + r(f_{\boldsymbol{\xi}}(k_t, z_t), \rho z_t + \varepsilon_i) - \delta\right) \right) - 1 \tag{5}$$

The second step is to code $\tilde{E}_{\boldsymbol{\xi}}(k_t, z_t, \varepsilon_i)$ to accept vectors as arguments. If $\boldsymbol{k_t}, \boldsymbol{z_t}$ and $\boldsymbol{\varepsilon}$ are vectors of the same length M, then $\tilde{E}_{\boldsymbol{\xi}}(\boldsymbol{k}, \boldsymbol{z}, \boldsymbol{\varepsilon})$ is a vector of the same length M and the error on every point on the grid can be computed as $\boldsymbol{W}\tilde{E}_{\boldsymbol{\xi}}(\boldsymbol{k}, \boldsymbol{z}, \boldsymbol{\varepsilon})$, where $\boldsymbol{W}$ is a weighting matrix. The input vectors and $\boldsymbol{W}$ can be constructed as described in figure 6.

The construction of input vectors is straightforward; all that is needed is a function `comb_vector` which uses two inputs $\boldsymbol{x} = [x_1, \ldots, x_N]$ and $\boldsymbol{y} = [y_1, \ldots, x_M]$ to generate two output vectors $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ of length MN such that

$$\tilde{\boldsymbol{x}} = [\overbrace{\underbrace{x_1 \quad \ldots \quad x_N}^{N} \quad \ldots \quad x_1 \quad \ldots \quad x_N}^{NM}]$$
$$\tilde{\boldsymbol{y}} = [\overbrace{\underbrace{y_1 \quad \ldots \quad y_1}^{N} \quad \ldots \quad y_M \quad \ldots \quad y_M}^{NM}]$$

The code uses this function twice; first, to combine grids for capital and productivity and second, to combine these grids with nodes for the quadrature. The resulting vectors are as follows:

$$\tilde{\boldsymbol{k}} = [\overbrace{\underbrace{k_1 \quad \ldots \quad k_1}^{J} \quad \ldots \quad \underbrace{k_N \quad \ldots \quad k_N}^{J}}^{NJ} \quad \ldots \quad \overbrace{k_1 \quad \ldots \quad k_1}^{J} \quad \ldots \quad \overbrace{k_N \quad \ldots \quad k_N}^{J}]^{NMJ}$$
$$\tilde{\boldsymbol{z}} = [\overbrace{\underbrace{z_1 \quad \ldots \quad z_1}^{J} \quad \ldots \quad \underbrace{z_1 \quad \ldots \quad z_1}^{J}}^{NJ} \quad \ldots \quad \overbrace{z_M \quad \ldots \quad z_M}^{J} \quad \overbrace{z_M \quad \ldots \quad z_M}^{J}]^{NMJ}$$

```
% precompute thequadrature nodes
[nodes, weights] = GaussHermite_2(J); %generate nodes and weights of order J
nodes   = sqrt(2)*sig_a*nodes;
weights = pi^(-0.5)*weights;

while ~converged
  %% simulate the model the get series for capital and productivity
  Σ =simulate_model(z̃,ξ)
  [k̃,z̃] = get_grids_sim(Σ,...) % get grids

  %% combine grids
  [k_grid1,z_grid1] = comb_vector(k̃,z̃); % basic rectangular grid for states
  %combine rectangular grids with quadrature nodes
  [ε̃,k̃] = comb_vector(nodes',k_grid1);
  [~,z̃] = comb_vector(nodes',z_grid1);
  [weights_grid,~] = comb_vector(weights',z_grid1);
  %construct weighting matrix W
  lk = length(k_grid1); lkkk = length(k̃); ln = length(nodes');
  W = zeros(lkkk,lk);
  for i=1:lk
      W(1+(i-1)*ln:i*ln,i) = weights_grid(1+(i-1)*ln:i*ln)';
  end

  %% update the solution
  ξ̃ = arg min |W E_ξ(k̃,z̃,ε̃)| %find update
  if ξ̃ is close to ξ̃
    converged = 1;
  else
    ξ = λξ̃ + (1-λ)ξ
  end
end
```

Figure 6: Projection algorithm with only one loop (only the inside of the outer `while` loop controling the overall convergence).

$$\tilde{\boldsymbol{\varepsilon}} = [\overbrace{\underbrace{\overbrace{\varepsilon_1 \quad \dots \quad \varepsilon_J}^{J} \quad \dots \quad \overbrace{\varepsilon_1 \quad \dots \quad \varepsilon_J}^{J}}_{NJ} \quad \dots \quad \overbrace{\varepsilon_1 \quad \dots \quad \varepsilon_J}^{J} \quad \dots \quad \overbrace{\varepsilon_1 \quad \dots \quad \varepsilon_J}^{J}}^{NMJ}]$$

The weighting matrix $\boldsymbol{W}$ then has to have the following shape:

$$\boldsymbol{W} = \overbrace{\left.\begin{bmatrix} \boldsymbol{w} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{w} & \dots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{0} & \boldsymbol{w} \end{bmatrix}\right\} NM}^{NMJ}$$

```
function [xx,yy] = comb_vector(x,y)
  %get the vecotr lengths
  lx = length(x);    ly = length(y);

  % first vector repeats itself ly times
  xx = repmat(x,1,ly);

  % the second vector repeats subsequentely each element in y exactly lx times
  yy = [];
  for i=1:ly
      yy = [yy repmat(y(i),1,lx)];
  end
end
```

Figure 7: `combine_vectors` function to combine two vectors.

where $\boldsymbol{w} = [w_1 \quad \dots \quad w_J]$ and $\mathbf{0} = [0 \quad \dots \quad 0]$ are vectors of length J.

If $\boldsymbol{W}, \tilde{\boldsymbol{\varepsilon}}, \tilde{\boldsymbol{k}}$ and $\tilde{\boldsymbol{z}}$ are constructed as described and if $\tilde{E}_{\boldsymbol{\xi}}(\boldsymbol{k}, \boldsymbol{z}, \boldsymbol{\varepsilon})$ is a column vector, then $\boldsymbol{W}\tilde{E}_{\boldsymbol{\xi}}(\boldsymbol{k}, \boldsymbol{z}, \boldsymbol{\varepsilon})$ is a column vector of length $N \times M$ and hence minimising $|\boldsymbol{W}\tilde{E}_{\boldsymbol{\xi}}(\boldsymbol{k}, \boldsymbol{z}, \boldsymbol{\varepsilon})|$ with respect to $\boldsymbol{\xi}$ yield the same result as if the result was computed using loops.

Applying this method can generate significant speed-up, in some situation making the solution more than 100x faster, as documented in figure 8.

3.2 Policy function shape and the representation of the state space

The projection algorithm approximates the true policy function with some function $\varphi(\boldsymbol{X}, \boldsymbol{\xi})$, where $\boldsymbol{X}$ is some vector function of the state space. The policy function approximation hence requires two choices; what functional form to use and how to simplify the state space to make it conform with the choice of the policy function shape. Possible candidates for the policy function shape are:

- $X\boldsymbol{\xi}$
- $\exp(X\boldsymbol{\xi})$
- $\nu_0^{\varphi} + \frac{\nu_1^{\varphi}}{1+\exp(-\boldsymbol{X}\boldsymbol{\xi})}$ where ν_0^{φ} and ν_1^{φ} are model parameters which are not being optimised and the solution should not depend on their values as the extreme values should not be reached.

The advantage of the second and third specifications is that they restrict the outcome to lie within a certain interval (positive numbers for the second case and in the interval $(\nu_0^{\varphi}, \nu_1^{\varphi})$

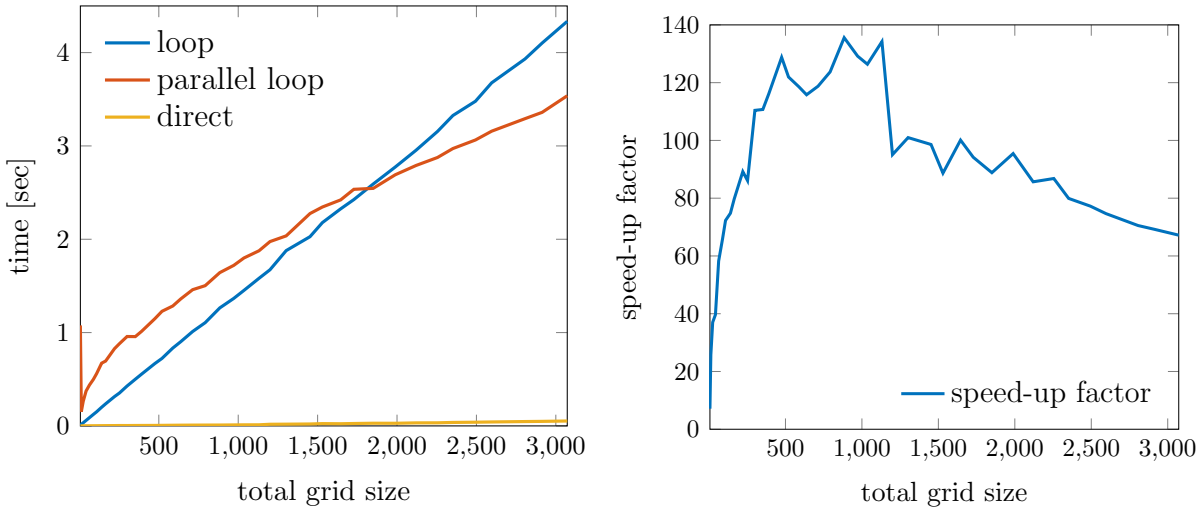


Figure 8: Vectorisation can yield significant benefits in computation time. In this example the Euler equation error (using 7 quadrature points) was evaluated on a grid for capital and productivity. Number of points in the capital grid was 4/3 of the points in the productivity grid, the total number, given by the product of the two, is depicted on the x-axis. Y-axis shows average time in seconds over 10 runs. The right panel shows the average speed-up using the direct method over the better of the two loop approaches. The computations are done on Intel i7-3930K 6 core CPU with 32 GB of RAM.

in the third case). This can help with numerical issues like negative consumption, which might be otherwise encountered during optimisation.[7]

The second choice to be made concerns the representation $\boldsymbol{X}$ of the state space $\boldsymbol{\Sigma}$. The fact that we approximate the true policy function with a polynomial means that we are introducing some error into the solution. The error comes from the limitation of the shape of the function, but it also depends on how well $\boldsymbol{X}$ captures $\boldsymbol{\Sigma}$.

Numerical stability of the process of recovering $\boldsymbol{\xi}$ depends on the so-called condition number of matrix $\boldsymbol{X}$. To see this, recall that in the setting of equation (1), the vector of interest $\boldsymbol{\xi}$ could be obtained by standard regression formula $(\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\gamma$. From regression perspective, ill-conditioned $\boldsymbol{X}$ corresponds to a situation of strong colinearity among columns of $\boldsymbol{X}$.

Due to multi-colinearity, it is not advisable to just use different orders of individual state variables. One way how to limit the colinearity is to use some family of orthogonal polynomials as a base instead. However, this approach is not automatically guaranteed to

[7]Some solvers might not respect the supplied constraints (like positive consumption or non-negative labour) during intermediate optimisation steps.

deliver $\boldsymbol{X}$ which is well conditioned. The reason is that the orthogonality is in this context defined with respect to a specific weighting function over a specific interval. Hence for the vectors of $\boldsymbol{X}$ to be uncorrelated, the domain of the weighting function has to correspond to the domain of the model.

This problem can be confronted directly. Realizing that scaling and normalising influences the numerical stability of the solution algorithm, I define *normalising parameters* to directly affect the construction of $\boldsymbol{X}$ and choose them such that the condition of $\boldsymbol{X}$ is minimised.

For example, suppose that the state vector consists of three variables, i.e. $\boldsymbol{\Sigma} = [\boldsymbol{\Sigma}_1 \quad \boldsymbol{\Sigma}_2 \quad \boldsymbol{\Sigma}_3]$. One simple choice would be

$$\boldsymbol{X} = [1 \quad \boldsymbol{\Sigma}_1 \quad \boldsymbol{\Sigma}_2 \quad \boldsymbol{\Sigma}_3].$$

Next, let:

$$\boldsymbol{X}(\boldsymbol{\nu}) = \begin{bmatrix} \mathbf{1} & \dfrac{\boldsymbol{\Sigma}_1 - \nu_1}{\nu_2} & \dfrac{\boldsymbol{\Sigma}_2 - \nu_3}{\nu_4} & \dfrac{\boldsymbol{\Sigma}_3 - \nu_5}{\nu_6} \end{bmatrix}$$

where $\boldsymbol{\nu} = [\nu_1 \dots \nu_6]$ is chosen as

$$\boldsymbol{\nu} = \arg\min \kappa(\boldsymbol{X}(\boldsymbol{\nu}))$$

where $\kappa(\boldsymbol{X}) = \|\boldsymbol{X}^{-1}\|\|\boldsymbol{X}\|$ is the condition number.[8] Then by construction, $\boldsymbol{X}(\boldsymbol{\nu})$ is better conditioned than $\boldsymbol{X}$. Note that after changing the scaling parameters $\boldsymbol{\nu}$, the policy function parameters $\boldsymbol{\xi}$ also have to be rescaled, so that

$$\forall \boldsymbol{\Sigma}: \qquad \boldsymbol{X}(\boldsymbol{\nu})\boldsymbol{\xi} = \boldsymbol{X}(\tilde{\boldsymbol{\nu}})\tilde{\boldsymbol{\xi}}.$$

Therefore the new policy function parameters $\tilde{\boldsymbol{\xi}}$ can be found easily as

$$\tilde{\boldsymbol{\xi}} = \left(\boldsymbol{X}(\tilde{\boldsymbol{\nu}})'\boldsymbol{X}(\tilde{\boldsymbol{\nu}})\right)^{-1} \boldsymbol{X}(\tilde{\boldsymbol{\nu}})'\boldsymbol{X}(\boldsymbol{\nu})\boldsymbol{\xi}$$

While formally both $\boldsymbol{\nu}$ and $\boldsymbol{\xi}$ enter the policy function in a similar way, the two sets of parameters are fundamentally different. The former are introduced to enhance numerical stability and accuracy, whereas only the latter capture the decisions of the economic agents.

[8] In MATLAB, matrix norm corresponds to maximum singular value of the matrix.

For example, in the first chapter of this thesis I use the following functional form for *aggregate labour* in production sectors:

$$l^p = \nu_0^{\varphi} + \frac{\nu_1^{\varphi}}{\exp(-\boldsymbol{X}\boldsymbol{\xi})}$$

with

$$\boldsymbol{X} = \Bigg[1 \quad \frac{k-\nu_{\bar{k}}}{\nu_{\sigma_k}} \quad \left(\frac{k-\nu_{\bar{k}}}{\nu_{\sigma_k}}\right)^2 - 1 \quad \frac{1}{N}\sum_{i=1}^{N}\frac{e_i-\nu_{\bar{e}}}{\nu_{\sigma_e}} \quad \left(\frac{1}{N}\sum_{i=1}^{N}\frac{e_i-\nu_{\bar{e}}}{\nu_{\sigma_e}}\right)^2 - 1$$
$$\frac{1}{N}\sum_{i=1}^{N}\frac{\mu_i-\nu_{\bar{\mu}}}{\nu_{\sigma_\mu}} \quad \left(\frac{1}{N}\sum_{i=1}^{N}\frac{\mu_i-\nu_{\bar{\mu}}}{\nu_{\sigma_\mu}}\right)^2 \quad \frac{1}{N}\sum_{i=1}^{N}\frac{\tilde{a}_i-\nu_{\bar{\tilde{a}}}}{\nu_{\sigma_{\tilde{a}}}} \quad \frac{k-\nu_{\bar{k}}}{\nu_{\sigma_k}}\frac{1}{N}\sum_{i=1}^{N}\frac{\tilde{a}_i-\nu_{\bar{\tilde{a}}}}{\nu_{\sigma_{\tilde{a}}}}\Bigg]$$

where $[\nu_0^{\varphi}, \nu_1^{\varphi}, \nu_{\bar{\mu}}, \nu_{\sigma_\mu}]$ is set to $[0, 0.75, \bar{\mu}, \sigma_\mu]$ and all the other parameters are chosen to minimize the condition number of matrix $\boldsymbol{X}$. The problem of the household is such that only the aggregate state matters. The distribution of individual sector variables (current innovation step e_i, relative productivity $\tilde{a}_i$ and quality of research ideas μ_i for $i = 1, \ldots, N$) matters only as much as it helps to predict the future aggregate state.

The policy function for research labour (labour used in one given research firm), however, depends also on the variables of this particular sector. Therefore, the following functional form is used

$$q_i^P = \min\{\nu_0, \exp(\boldsymbol{X}\boldsymbol{\xi})\}$$

with

$$\boldsymbol{X} = \Bigg[1 \quad \frac{k-\nu_{\bar{k}}}{\nu_{\sigma_k}} \quad \frac{1}{N}\sum_{i=1}^{N}\frac{\tilde{a}_i-\nu_{\bar{\tilde{a}}}}{\nu_{\sigma_{\tilde{a}}}} \quad \frac{1}{N}\sum_{i=1}^{N}\frac{e_i-\nu_{\bar{e}}}{\nu_{\sigma_e}} \quad \frac{1}{N}\sum_{i=1}^{N}\frac{\mu_i-\nu_{\bar{\mu}}}{\nu_{\sigma_\mu}} \quad \ldots$$
$$\frac{e_i-\nu_{\bar{e}}}{\nu_{\sigma_e}} \quad \left(\frac{e_i-\nu_{\bar{e}}}{\nu_{\sigma_e}}\right)^2 - 1 \quad \left(\frac{e_i-\nu_{\bar{e}}}{\nu_{\sigma_e}}\right)^3 - 3\frac{e_i-\nu_{\bar{e}}}{\nu_{\sigma_e}} \quad \ldots$$
$$\frac{\mu_i-\nu_{\bar{\mu}}}{\nu_{\sigma_\mu}} \quad \left(\frac{\mu_i-\nu_{\bar{\mu}}}{\nu_{\sigma_\mu}}\right)^2 - 1\Bigg]$$

3.3 Quadrature with truncated outcomes

In many situations, different realisations of a shock might change the chosen action both quantitatively and qualitatively. For example, for low realisations of productivity, a firm might decide not to produce at all and it might start producing only after productivity is above some threshold. In such a setting, the firm might be earning zero profits for realisations less or equal to the threshold and then the profits might be increasing in productivity. I call the effect of a shock *truncated*, if the shock induces *both* a discrete action and it has continuous effects if the action is taken.

In the first chapter of this thesis, the research firm receives an imperfect signal about the quality of the project it can work on and it chooses the labour input upon observing this information. This is implemented by assuming two shocks. First, there is the quality of the research project μ, which is public knowledge at the time of decision. Second, there is a luck factor ε, which is realised only after all decisions are made. The final outcome of the innovation process is then

$$\mu + f(l^r) + \varepsilon$$

where l^r is the labour input of the research firm. This formulation simplifies a more realistic signal extraction problem where the research firm observes a noisy signal about the quality of its project.

Conditional on μ and the action l^r, the outcome of innovation activity hence depends on the luck factor ε. It turns out that there is a threshold value of the luck shock ε denoted by $\underline{\varepsilon}$ such that for realisations $\varepsilon \leq \underline{\varepsilon}$, the research effort is not successful (hence it does not matter if ε is just marginally below the threshold or if the threshold is missed by a lot). For $\varepsilon > \underline{\varepsilon}$ the research is successful and the research firm will enter its production stage next period. Furthermore, conditional on $\varepsilon > \underline{\varepsilon}$, the size ε matters as it changes future mark-ups and hence profits. In other words, the size of ε matters only as long as it is large enough. The research firms are well aware of the implications of this setting when choosing its optimal effort l^r, as higher l^r implies lower value of the threshold $\underline{\varepsilon}$.

In general, consider a multi-sector setting where there are two shocks affecting each sector i, one which has truncated effects, ε_i, and one which does not, μ_i. Here I show how this structure can be exploited to reduce computation costs of computing expectations $\mathsf{E}[g(\boldsymbol{\Sigma}_t, \boldsymbol{\varepsilon}_{t+1}, \boldsymbol{\mu}_{t+1})]$ where $\boldsymbol{\Sigma}_t$ represents the state of the world at time t. The truncated

nature of the effect of $\boldsymbol{\varepsilon}$ formally written is the fact that $\exists \underline{\boldsymbol{\varepsilon}}$ such that $\forall \boldsymbol{\varepsilon}' < \underline{\boldsymbol{\varepsilon}}$

$$g(\boldsymbol{\Sigma}_t, \boldsymbol{\mu}_{t+1}, \boldsymbol{\varepsilon}') = g(\boldsymbol{\Sigma}_t, \boldsymbol{\mu}_{t+1}, \underline{\boldsymbol{\varepsilon}})$$

In other words, the realisation of ε has to be *large enough* for ε *to start* to have any impact on the outcome. This threshold value can depend on the state of the economy

$$\underline{\boldsymbol{\varepsilon}}_{t+1} = \underline{\boldsymbol{\varepsilon}}(\boldsymbol{\Sigma}_t)$$

This introduces a kink in conditional outcomes and non-linearity in expected outcomes. Note that it is possible that the threshold depends on the behavior of the agents (which in turn depends on the state). In such a setting, the true value of the threshold depends on the true policy function. Therefore during the solution, the value of the threshold also depends on the current parameters governing the approximation of the policy function.

The truncated nature of the outcomes allows for an effective way of implementing Gaussian quadrature. For any function g, the following holds (omitting the $\boldsymbol{\Sigma}_t$)

$$\begin{aligned}\mathsf{E}[g(\boldsymbol{\varepsilon}_{t+1}, \boldsymbol{\mu}_{t+1})] = \mathsf{E}[g(\varepsilon_{it+1}, \boldsymbol{\varepsilon}_{-it+1}, \boldsymbol{\mu}_{t+1}) | \varepsilon_{it+1} \leq \underline{\varepsilon}_{it+1}] \mathsf{P}(\varepsilon_{it+1} \leq \underline{\varepsilon}_{it+1}) \\ + \mathsf{E}[g(\varepsilon_{it+1}, \boldsymbol{\varepsilon}_{-it+1}, \boldsymbol{\mu}_{t+1}) | \varepsilon_{it+1} > \underline{\varepsilon}_{it+1}] \mathsf{P}(\varepsilon_{it+1} > \underline{\varepsilon}_{it+1}).\end{aligned}$$

Using the no effect below the threshold property and $\mathsf{P}(\varepsilon_{it+1} \leq \underline{\varepsilon}_{it+1}) = \Phi(\underline{\varepsilon}_{it+1})$:

$$\begin{aligned}\mathsf{E}[g(\boldsymbol{\varepsilon}_t, \boldsymbol{\mu}_{t+1})] = \mathsf{E}[g(\underline{\varepsilon}_{it+1}, \boldsymbol{\varepsilon}_{-it+1}, \boldsymbol{\mu}_{t+1})] \Phi(\underline{\varepsilon}_{it+1}) \\ + \mathsf{E}[g(\varepsilon_{it+1}, \boldsymbol{\varepsilon}_{-it+1}, \boldsymbol{\mu}_{t+1}) | \varepsilon_{it+1} > \underline{\varepsilon}_{it+1}] (1 - \Phi(\underline{\varepsilon}_{it+1}))\end{aligned}$$

For two sectors, there are two shocks ε which are independent, hence

$$\begin{aligned}\mathsf{E}[g(\boldsymbol{\varepsilon}_t, \boldsymbol{\mu}_{t+1})] = \mathsf{E}[g(\underline{\varepsilon}_{1t+1}, \underline{\varepsilon}_{2t+1}, \boldsymbol{\mu}_{t+1})] \Phi(\underline{\varepsilon}_{1t+1}) \Phi(\underline{\varepsilon}_{2t+1}) \\ + \mathsf{E}[g(\varepsilon_{1t+1}, \varepsilon_{2t+1}, \boldsymbol{\mu}_{t+1}) | \varepsilon_{2t+1} > \underline{\varepsilon}_{2t+1}] \Phi(\underline{\varepsilon}_{1t+1}) (1 - \Phi(\underline{\varepsilon}_{2t+1})) \\ + \mathsf{E}[g(\varepsilon_{1t+1}, \varepsilon_{2t+1}, \boldsymbol{\mu}_{t+1}) | \varepsilon_{1t+1} > \underline{\varepsilon}_{1t+1}] (1 - \Phi(\underline{\varepsilon}_{1t+1})) \Phi(\underline{\varepsilon}_{2t+1}) \\ + \mathsf{E}[g(\varepsilon_{1t+1}, \varepsilon_{2t+1}, \boldsymbol{\mu}_{t+1}) | \varepsilon_{1t+1} > \underline{\varepsilon}_{1t+1}, \varepsilon_{2t+1} > \underline{\varepsilon}_{2t+1}] (1 - \Phi(\underline{\varepsilon}_{1t+1})) \times \\ \times (1 - \Phi(\underline{\varepsilon}_{2t+1}))\end{aligned}$$

The same logic can be applied to extend the model to any dimension.

Taking advantage of the truncated distribution of outcomes allows to obtain very precise

numerical approximation of the expectations while keeping the number of nodes low. This result follows from the fact that the approximation is computed over a much smaller interval ($[\underline{\varepsilon}, \infty]$) and hence fewer nodes are needed compared to standard quadrature. In other words, only one node is needed to perfectly evaluate the expectations of the outcomes over $(-\infty, \underline{\varepsilon})$.

The nuisance is that the threshold $\underline{\varepsilon}$ depends on the state and the policy function parameters and hence it has to be recomputed for each point in the grid separately after every update of the research policy function. The faster solution is to compute the weights and nodes on a very fine equidistant grid for thresholds and then to do a fast linear interpolation.[9]

Figure 9 demonstrates the benefits of truncated quadrature. Suppose that the goal is to approximate the expectations $\mathsf{E}f(x)$, where x is normally distributed with mean zero and standard deviation equal to one and

$$f(x) = \begin{cases} \exp(-\frac{(x-1).^2}{4} - 1) & \text{for } x > \underline{x} \\ 0 & \text{otherwise} \end{cases}$$

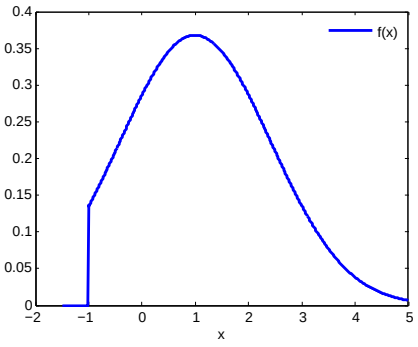
with 3 thresholds $\underline{x} \in \{-1, 0, 1\}$. The truncated quadrature is very precise even with just a single quadrature node, whereas standard quadrature is not precise even with one hundred nodes.[10]

If the shocks are independent, then the nodes can be constructed independently and combined later to get the multidimensional approximation to the expectations. This setting allows for choosing different precision for different dimensions. For example, the outcomes in a sector are likely to be much more affected by the shocks to the own sector than by the shocks to the other sectors. Then, if the computation speed is a concern and a high number of nodes cannot be applied to all dimensions, the independence of the shocks allows for a high number of nodes for high precision with respect to own shocks and reduced number of nodes for shocks of the other sectors to keep the computational costs reasonable.

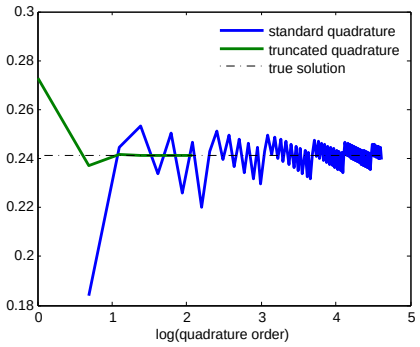
The standard approach to numerically approximate the expectation in highly dimensional problems is to apply monomial nodes rather than product quadratures. If $\boldsymbol{\varepsilon}$ and $\boldsymbol{\mu}$ are independent, it is possible to combine the monomial nodes for $\boldsymbol{\mu}$'s and truncated

[9] John Burkardt provides a code to compute the nodes and weights for an arbitrary truncated interval, http://people.sc.fsu.edu/ jburkardt/

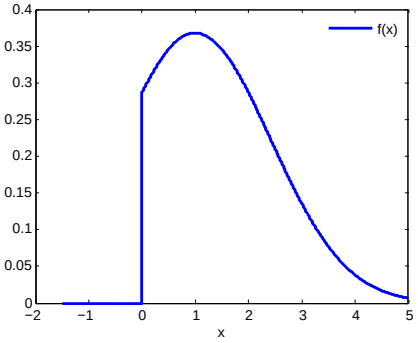
[10] In real life situations it is not practical to have more than 30 quadrature nodes. Even if the function is fast to evaluate so that the number of nodes does not represent a computationally expensive problem, the weights get too small for algebra in double precision.



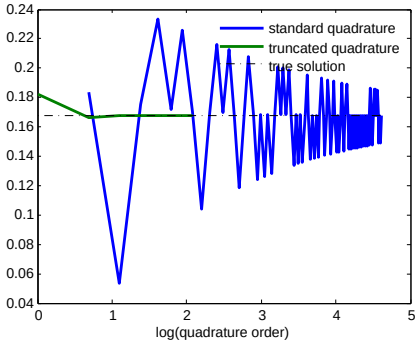
(a) threshold=-1



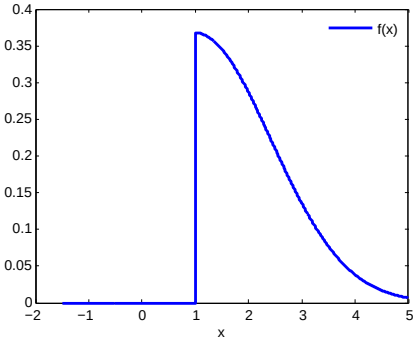
(b) threshold=-1



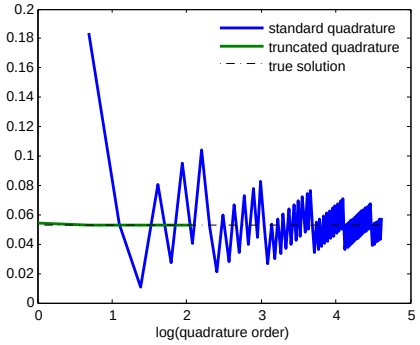
(c) threshold=0



(d) threshold=0



(e) threshold=1



(f) threshold=1

Figure 9: Truncated quadrature precision

Gauss-Hermite nodes for $\boldsymbol{\varepsilon}$'s. In the first chapter of this thesis I use monomial nodes to approximate future $\boldsymbol{\mu}$ together with the threshold value and two truncated Gaussian nodes for other sectors ε_{-i} and 3 nodes for own ε_i.

3.4 More on ergodic grids

Here I discuss some further benefits of using ergodic grids. The great insight of Judd *et al.* (2012) is that correlation among the state variables induced by the agent behavior allows to reduce the grid significantly compared to the standard product grid. However, there are also accuracy and speed benefits.

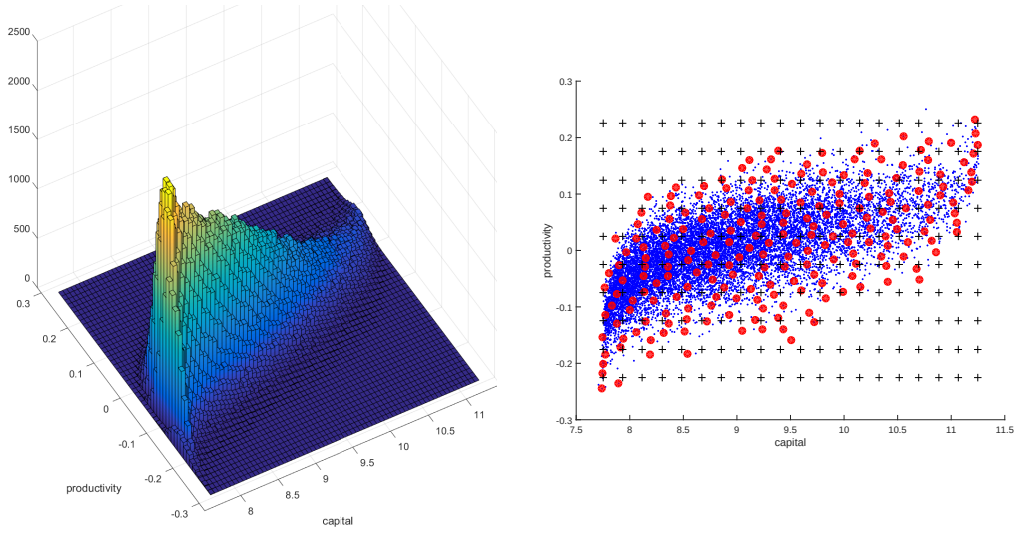
I also show how the intermediate information during the solution can be used in order to speed-up the computation and how to construct more robust ergodic grids. I conclude this section with some other observations about ergodic grids.

3.4.1 Benefits for accuracy of solution

Figure 10 demonstrate the size-reduction effect of ergodic grids. Clearly, the optimal behaviour in the model induces strong positive correlation between the level of capital and productivity; because the productivity process is persistent, higher levels of capital are accumulated for high levels of productivity and vice versa. The left panel shows the histogram of the RBC model descibed in the beginning of this chapter, the right panel shows the set of points generated by simulation against the ergodic grid and standard product grid.

The ergodic grid captures the cloud of simulated points very closely, whereas the product grid covers areas, namely very high level of capital with very low level productivity and vice versa, which are essentially never visited in the simulation. The fact that we can exclude such points in the ergodic grid has implication for accuracy of the solution. The reason is due to the way the policy function is approximated in the projection algorithm, in particular, due to polynomial approximation. Polynomial approximation is prone to bad behaviour outside of the approximation range. In the context of different grids, a solution f_1 which achieves very high accuracy on a smaller grid (in terms of area covered) is likely to misbehave outside of this grid. However this also implies that a different approximated function f_2 (of the same order) over a larger grid is highly unlikely to achieve the same accuracy over the smaller grid as f_1.

In theory, this would not be a problem if it would be possible to use high enough



(a) histogram (b) grid

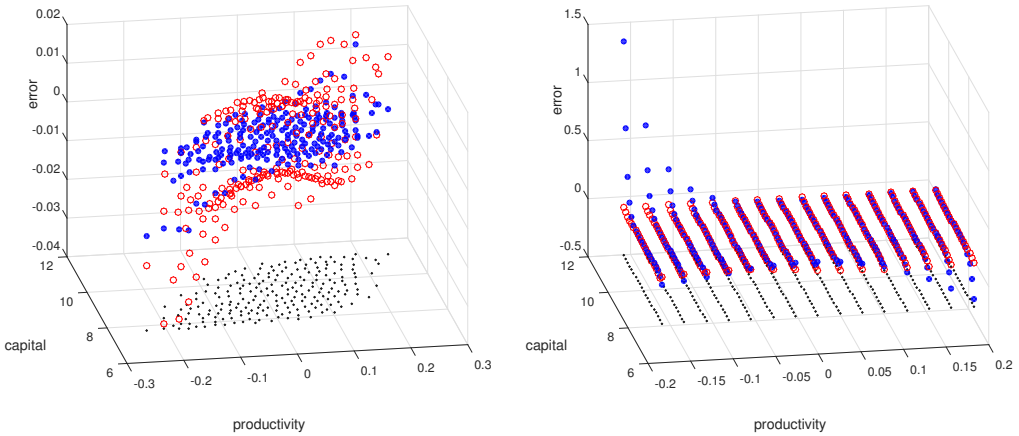
Figure 10: Simulated solution of a RBC model. Left panel shows the histogram of model simulation (600000 periods). The model generates high correlation between the two state variables. In panel (b), blue points represent the simulated series (every 50th point from the simulation), red points represent the ergodic grid and the black crosses represent the product grid. Clearly, the product grid covers areas (in top left and bottom right) which are never visited by the simulated model.

order of polynomial approximation. However, higher approximation is likely to be more computationally expensive and even if computation time is not a problem, increase in dimensionality of the solution increase the degree of collinearity problems which makes the higher solution less accurate. Increasing the order of the polynomials is thus impractical once the degree gets larger that 5.

Figure 11 demonstrates this observation visually. I take two solutiona of order 4, one for an ergodic grid and one for a fixed product grid and I evaluate how well the Euler equation holds (using Gauss-Hermite quadrature of degree 7). As predicted, the accuracy is better for the solution which was computed over the restricted grid.

Table 1 demonstrates this point further. This table compares the average absolute value of the errors obtained from the previous exercise. While evaluating the solution obtained on a product grid over a smaller ergodic grid actually increases accuracy (decreases the average absolute error), evaluating the solution obtained on the ergodic grid on the product grid leads to drastic increase of the average absolute errors.

An alternative way how to evaluate the accuracy is to simulate the model and examine the realised (absolute value of) errors from the Euler equation. The results of this exercise



(a) ergodic grid (b) larger product grid

Figure 11: Solution accuracy and size of grid. In both panels, blue dots represent the solution on a product grid, whereas red circles represent solution obtained on an ergodic grid. Both solution are of order 4. The left panel shows the behaviour of the two solutions on a smaller ergodic grid. The solution represented by red circles was solved on a similar grid and hence is more accurate,although neither solution is particularly imprecise. However, the right panel shows that the approximation obtained on the ergodic grid, which was indeed very precise over the smaller ergodic grid, is very imprecise over the larger product grid. Note that the accuracy is particularly bad exactly at the areas which are further away from the ergodic grid, i.e. high capital with low productivity and *vice versa*.

are capture in figure 12. In addition to the iterative solution solved on the ergodic grid and the direct/solver solution obtained over a product grid, I also examine a solution obtained from a solver on a final ergodic grid from the iterative solution. This is to evaluate the accuracy of the iterative solution against a standard derivative free solver when keeping the grid the same.

First, using the ergodic grid increases the accuracy, in particular it compresses the tail of the error distribution. Second, on the ergodic grid, the accuracy of the solution obtained iteratively and the one obtained by a solver seems to be very close.

It is not straightforward to compare the solution time. The reason is that even when using a product grid, the correct location is typically not obvious and hence some time has to be spent simulating the model, in fact in a similar way as using the iterative solution on the ergodic grid.

One benefit of the iterative solution is that the time to evaluate the function grows more slowly with the dimensionality of the problem than the time any solver needs and hence the iterative procedure is relatively more suitable for larger problems. At the same

		grid	
		ergodic	product
solution method	solver on p-grid	0.007	0.008
	iterative on e-grid	0.001	0.024

Table 1: Average value of absolute errors for model with 4th-order policy function on different grids. Errors are computed from the Euler equation with 7th order quadrature (p-grid = product grid, e-grid = ergodic grid).

time, in my experience, solvers are more robust to having badly conditioned problems.

3.4.2 Moving grid during solution

By construction, if $\varphi(\boldsymbol{\xi})$ is a bad representation of the true policy function p, then the ergodic grid is constructed over an area of the state space which might not correspond to the area which would be recovered should the model be simulated with the true p. This situation might occur when the parameters in $\boldsymbol{\xi}$ are far away from the solution. For example, if the initial guess was such that there is less capital than under the true solution, then the errors in the Euler equation will lead to updating $\boldsymbol{\xi}$ such that there is more capital (hence people work and save more). However, given the approximation error it might be the case that the solution for $\boldsymbol{\xi}$ will overshoot and in the next simulation there will be too much work and too much capital.

One advantage of using ergodic grids combined with an iterative solution is that it provides additional information about the shape of the solution even before the process has converged. This additional information can be used to accelerate the solution, by exploiting the saving behavior over the grid.

For a candidate solution $\phi(\tilde{\boldsymbol{\xi}})$, if for all points on the grid the solution implies that the agent is a net saver, then it is clear that either the grid does not represent the correct solution (assuming that $\tilde{\boldsymbol{\xi}}$ is the correct solution) or the solution is wrong (assuming that the grid is located at the right area of the state space), or most likely, a combination of both.

To see this, consider the first possibility. If the agent is saving at each point of the grid, then in the next period the agent will find herself out of the area covered by the grid for the point in the current grid with the most capital. Vice versa, if the agent is reducing her capital stock everywhere on the grid, then next period she will find herself with capital stock below the area covered by the current grid.

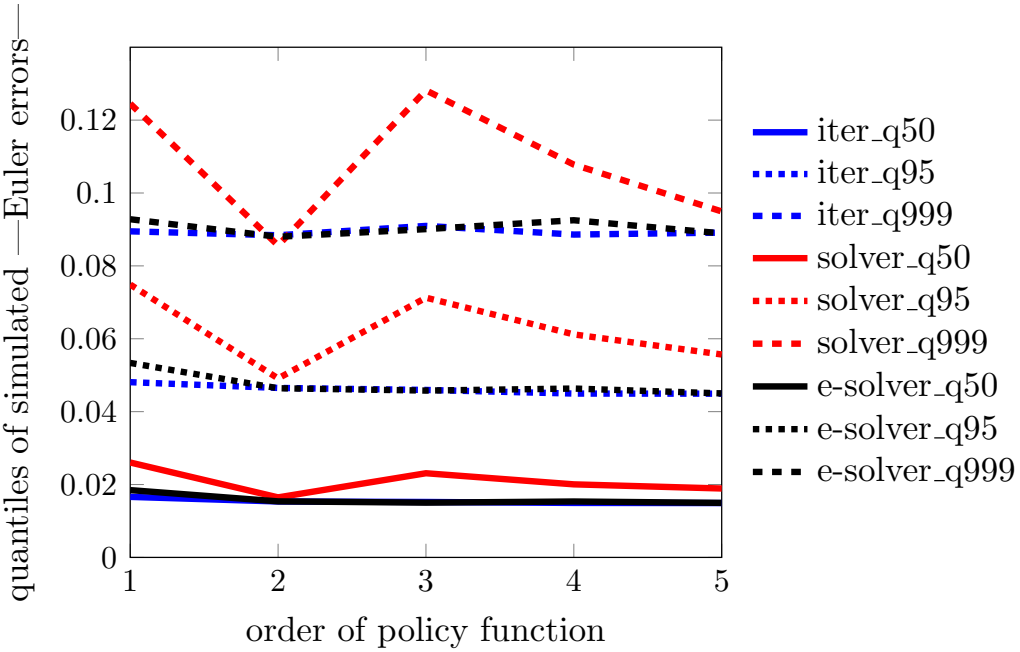


Figure 12: Errors of the Euler equation in the simulation. This figure captures the differences in solution accuracy for three methods, iterative using ergodic grid (blue lines), solver on a exogenous product grid (red lines) and a solver on ergodic grid (black lines). The style of line captures different quantiles; solid median, dotted 0.95th and dashed 0.999th percentile.The solver used is MATLAB's `fminsearch`, which is a derivative free algorithm using the NelderMead method.

At the same time, even if the current $\boldsymbol{\xi}$ is very close to the true solution *for the grid which would cover the area where the* correct *model lives*, search for the correct solution on a wrong grid will not converge to the correct value.

If this is the case and if the model is very time-consuming to simulate (which is often the case for highly dimensional settings), it is possible to save time by adjusting the grid directly without resorting to re-simulating during every iteration. In the case where the behavior implies saving at each point of the grid, I adjust the capital at each point of the grid by adding a small constant (and vice versa, reduce the capital stock at each point of the grid if the agent is reducing the capital stock at each point on the grid). That way, the correlation structure between the state variables is maintained while the grid is moved toward the area the intermediate solution suggests it should go to.

3.4.3 Grid stretching

Divergence in simulation can occur due to two possible reasons. First, the algorithm fails *on the grid*, meaning that due to all the reasons discussed above, the iterative algorithm or the solver diverges when looking for the parameters of the policy function $\boldsymbol{\xi}$. To address this possibility of algorithm failure, Maliar and Maliar (2014) introduce the robust regression methods.

However, secondly, there is also the possibility that the algorithm finds (converges to) a particular $\boldsymbol{\xi}$ *on a given grid*, but then the simulation step diverges when a new simulation is attempted. One reason why this can happen is that the solution found on the grid fails to represent well the true solution *in a neighborhood* of the convex hull defined by all the points in the simulation. This can happen just by bad luck in the process of randomly selecting the grid from the simulated points.[11] However, the chance of this problem occurring increases with the number of outlier points eliminated from the simulation before the grid is selected. This problem typically does not occur when the grid is exogenously set to cover a much wider range than where the model lives.

However, it is possible to exploit the advantages of both approaches. In *grid stretching* the ergodic grid is constructed using two sets of points. The first set comes from the simulation directly, just as in the standard approach pioneered by Judd *et al.* (2012). The second set is constructed from the simulation to cover a wider area while maintaining the correlation present in the data.

One simple approach can be to stretch points further from the median. It is also easy to stretch in a particular direction more than others and/or use nonlinear stretching as demonstrated in figure 13. Stretching is going to create new outliers and some points which used to be outliers in the simulation data will now be surrounded by many points from the stretched set. Combining stretching with cutting outliers allows us to avoid divergence by constructing dense and more uniformly covered grids.

The main motivation to use ergodic grids is to reduce the grid by exploiting the strong correlation among the state variables implied by the model. This allows to eliminate potentially large areas which the simulated model never visits. However, this also by construction introduces collinearity and hence potentially can make the solution more difficult to find. Consider a situation where there is perfect correlation between productivity and

[11] Typically, one makes a long simulation of hundreds of thousands of periods to explore all the places the model can go. However, it is very memory demanding to compute and store the distance matrix of all the points, so a reasonable step is to randomly pick only a fraction of points from the simulation and then construct the grid using only those selected points.

```
function Σ = stretching(k,z,stretch_z_up stretch_z_down,stretch_k_up stretch_k_down)
Sigma_in = [k,z]; %points from the simulation
%matrix of stretching coefficients
stretch_coef_up = repmat([stretch_k_up stretch_z_up ],size(Sigma_in,1),1);
stretch_coef_down = repmat([stretch_k_down stretch_z_down ],size(Sigma_in,1),1);
%construct the center point (median of the simulation)
Sigma_mean = repmat(median(Sigma_in,1),size(Sigma_in,1),1);

%apply stretching
Sigma_in_extra_up   = Sigma_in+stretch_coef_up.*(min(0,Sigma_in-Sigma_mean));
Sigma_in_extra_down = Sigma_in-stretch_coef_down.*(sqrt(abs(max(0,Sigma_in-Sigma_mean))));
%combine sets
Σ = [Sigma_in;Sigma_in_extra_up;Sigma_in_extra_down];
```

Figure 13: Stretching, allowing for different degree of ergodic grid expansion above and below the steady state of the simulation defined by the median point.

capital. In this economy, the ergodic grid would lie on a line. If this is the case, then the polynomials capturing capital are perfectly correlated with the polynomials capturing productivity.

This fact shows why the collinearity is more likely to be a problem with ergodic grids than with standard product grids. In my experience, this effect is particularly strong if a grid has one area where the points are distributed uniformly and then one outlier located away from the main cluster. Usually, it is possible to avoid such grids by a combination of eliminating the outliers and stretching at the same time.

3.4.4 Convergence and accuracy consideration

The ergodic grid is by construction a stochastic object. The parameters obtained thus depend on a particular realisation of the grid and hence are also stochastic. This is due to the fact that the coefficients $\boldsymbol{\xi}$ are specific to the particular shape of the policy function which changes every time the normalising parameters $\boldsymbol{\nu}$ are changed. However, if the procedure works, then despite having different parameters in the policy function, the outcomes should be similar. Therefore, it might be convenient to measure the convergence by comparing medians (and higher order moments) of various variables obtained from the simulated data rather than comparing the policy rule parameters directly.

4 Conclusion

The popular "Moore's law" suggests that the computing power doubles roughly every two years. Although impressive enough, the rate of real progress of quantitative methods enabling economists to solve ever more complicated models with higher precision is much higher. New developments in software and algorithms might be just as important. These developments are the focus of this chapter.

I started by reviewing some recent developments focusing on the solution of large macroeconomic models and suggested extensions which make these methods more robust, faster and usable in situations where kinks might play an important role.

This chapter has three contributions which can be used in a wide range of solution algorithms. First, I show how to effectively vectorise the code used in projection methods to speed-up the solution by over 50 times against implementation with loops only. Second, I show how to use truncated Gaussian quadrature to effectively evaluate expectations over shocks with truncated effects. Third, I suggest a way how to better characterise the state space when constructing the approximations of policy functions. This is an alternative or possibly a complement to the the robust regression methods introduced in Judd *et al.* (2011), however it might provide better insight in case of troubleshooting divergent cases.

In the context of ergodic grids, I show how stretching might be useful to obtain a solution which has better properties in the wider neighborhood of the simulated data. This is particularly useful for situations where the initial guess is not very good and hence the grid moves a lot before the algorithm converges as it increases stability and reduces the chances that the algorithm diverges. I also suggest a possibility of moving the grid using the information obtained from the saving function to reduce the number of times the model needs to be simulated. Finally, I compare the accuracy of iterative versus solver solutions on an ergodic and a product grid.

References

Aruoba, S. B. and Fernandez-Villaverde, J. (2014). *A Comparison of Programming Languages in Economics*. NBER Working Papers 20263, National Bureau of Economic Research, Inc.

den Haan, W. J. and Marcet, A. (1990). Solving the Stochastic Growth Model by Parameterizing Expectations. *Journal of Business & Economic Statistics*, **8** (1), 31–34.

Judd, K. L., Maliar, L. and Maliar, S. (2011). Numerically stable and accurate stochastic simulation approaches for solving dynamic economic models. *Quantitative Economics*, **2** (2), 173–210.

—, — and — (2012). *Merging Simulation and Projection Approaches to Solve High-Dimensional Problems*. NBER Working Papers 18501, National Bureau of Economic Research, Inc.

Maliar, L. and Maliar, S. (2014). Chapter 7 - numerical methods for large-scale dynamic economic models. In K. Schmedders and K. L. Judd (eds.), *Handbook of Computational Economics Vol. 3*, *Handbook of Computational Economics*, vol. 3, Elsevier, pp. 325 – 477.

Marcet, A. (1988). *Solution of nonlinear models by parameterizing expectations*. Tech. rep., Carnegie Mellon University Working paper.