VFI Toolkit: Workshop, Part 1

vfitoolkit.com

Robert Kirkby

robertdkirkby.com

Victoria University of Wellington

March 11, 2025

VFI Toolkit

- The vision/goal:
  - Solving heterogenous agent, incomplete markets models.
  - You just write out the model.
  - You don't have to understand and write all the algorithms.
  - Making model modifications (change utility fn, add i.i.d. shock) is easy.
- VFI Toolkit vs writing your own code?
  - If you write naive code, toolkit runtimes are faster.
  - If you write smart code, toolkit runtimes are longer.
  - Of course, learning to write smart code, and writing it, both take time.

VFI Toolkit

- What you need: Matlab, NVIDIA GPU (graphics card).
  Parallel Computing Toolbox (but matlab is anyway fairly useless for most things without this).
  To use GPU, you must install CUDA (free; roughly, GPU drivers).
- Download VFI Toolkit.

  vfitoolkit.com
- Tell Matlab where to find VFI Toolkit ('add to path').
  front page of vfitoolkit.com explains these steps.

VFI Toolkit

- Why do I need a GPU?
- NVIDIA has a market capitalization of 2-3 trillion.
- They make essentially just one product: GPUs!
- GPUs are at the heart of essentially all modern computation.
- OpenAI/DeepSeek/Llama all get trained on tens of thousands of the best GPUs money can buy.
  Better GPUs than I can get even just one of ;)
- VFI Toolkit can do fairly brute force things thanks to GPU, and hence can use very flexible reliable algorithms.
- Without a GPU: you can solve some very basic models in VFI Toolkit, but everything is 100x slower, and only one endogenous state with one markov exogenous state is supported.
- You need little to no knowledge of GPUs to use VFI Toolkit.

VFI Toolkit

- Core Capabilities:
  - Solve infinite horizon value fn.
  - Solve Life-Cycle Models (finite horizon value fn).
  - Solve agents stationary distribution.
  - Calculate various moments/statistics. Simulate panel data.
  - Finding stationary general equilibrium, e.g. OLG models.
  - Finding general equilibrium transition path, e.g. OLG transitions.
  - Calibrate and GMM estimate Life-Cycle Models.
  - Tools to analyse all of the above.
- Note: Does not handle aggregate shocks.

VFI Toolkit: Plan for Workshop

- I don't plan to talk about the algorithms VFI Toolkit uses.
  Mostly pure discretization, a little linear interpolation. Lots of GPU!
- You can find pseudo-codes at: github.com/vfitoolkit/Pseudocodes
- We will see life-cycle models, OLG models, and OLG transition paths.
- I won't show, but you can do infinite horizon.
- In infinite horizon: one or two endogenous states, only markov exogenous states.
- Examples: Aiyagari (1994) (plus transition). Restuccia & Rogerson (2008). Gourio & Miao (2010).
  Incomplete markets models of infinitely lived households and firms.

Solving a Life-Cycle Model

- I am going to walk through solving a basic life-cycle model.
- Then we will make some changes/improvements/extensions.
- Then calibrate/GMM estimate.

Solving a Life-Cycle Model

- Seven core steps to solve a Life-Cycle Model with VFI Toolkit.
  1. Model action and state-spaces.
  2. Parameters
  3. Grids
  4. ReturnFn
  5. Solve for value fn and policy fn.
  6. Agents stationary distribution.
  7. Generate model moments/statistics.

Basic Life-Cycle Model 1

- Household problem

$$
\begin{aligned}
V(a,j) \quad = \quad & \max_{c,aprime} \frac{c^{1-\sigma}}{1-\sigma} + s_j \beta V(aprime, j+1) \\
& \text{if } j < Jr: \; c + aprime = (1+r)a + w\kappa_j \\
& \text{if } j >= Jr: \; c + aprime = (1+r)a + pension \\
& aprime \geq 0
\end{aligned}
$$

- Let's write the code to solve this.
  Code: *WorkshopModel1.m* (and *WorkshopModel1_ReturnFn.m*)
- I will explain this as the seven core steps.

Basic Life-Cycle Model

1. Model action and state-spaces.

- VFI Toolkit thinks of a model in terms of:
- Decision variables, d, none here, we will see what this is later.
- Endogenous states, a, we have one, assets.
- Exogenous markovs states, z, none here.
- Finite-horizon, J.

Basic Life-Cycle Model 1

1. Model action and state-spaces.
   - So we write the code,

```
%% Model action and state-space
n_d=0;
n_a=201; % number of grid points for our endogenous
    state
n_z=0;
N_j=81; % periods, represent ages 20 to 100
```

   - We are telling toolkit, how many grid points for each.

Basic Life-Cycle Model 1

2. Parameters
   - Create structure with all the parameters in it, by name

```
%% Parameters

% Age and Retirement
Params.J=N_j; % final period
Params.agej=1:1:N_j; % model period
Params.Jr=65-19; % retire at age 65, which is period 46


% Preferences
Params.beta=0.98; % discount factr
Params.sigma=2; % CRRA utilty fn

% Deterministic earnings
Params.kappa_j=[linspace(0.5,2,Params.Jr-15),linspace
    (2,1,14),zeros(1,Params.J-Params.Jr+1)];
% hump-shaped, then zero in retirement
```

For parameters that depend on age, like *agej* and κ_j, just create vectors of length *N_j*. Not showing all the parameters, just enough to give the idea.

3. Grids
   - Create grids as column vectors.

```
%% Grids

a_grid=10*linspace(0,1,n_a)'.^3; % Column vector of
    length n_a
% ^3 will put more points near 0 than 1, model has more
    curvature here

% We are not using them so,
d_grid=[];
z_grid=[];
pi_z=[];
```

Must be a column vector and have same number of points we specified in step 1.

Basic Life-Cycle Model 1

4. ReturnFn

- Can write Bellman equation like
$$V(a) = \max_{a'} F(a', a) + s_j \beta V(a')$$
- ReturnFn will be a function that plays role of $F(a', a)$
- In most models, is essentially utility function and constraints, combined.
- First inputs are $(aprime, a)$, anything after these is interpreted as parameters.

4. ReturnFn

```
function F=WorkshopModel1_ReturnFn(aprime,a,sigma,w,
    r,kappa_j,agej, Jr)
% first two entries are the action space

F=-Inf;

% budget constraint
if agej<Jr % working
    c=(1+r)*a +w*kappa_j - aprime;
else % retired
    c=(1+r)*a -aprime;
end

if c>0
    % utility fn
    F=(c^(1-sigma))/(1-sigma);
end
```

Basic Life-Cycle Model 1

5. Solve for Value fn and Policy fn.

```
%% Solve for value function and policy function
vfoptions=struct(); % Just using the defaults.
tic;
[V, Policy]=ValueFnIter_Case1_FHorz(n_d,n_a,n_z,N_j,
    d_grid, a_grid, z_grid, pi_z, ReturnFn, Params,
    DiscountFactorParamNames, [], vfoptions);
toc
```

- V is the value fn, evaluated on (a, j) space at our grids.
- *Policy* is the policy fn, it contains the index for *aprime*, over the (a, j) space.
- Runtime on my laptop, < 1 second.

Just does pure discretization value function iteration. (pure=next period values are on grid)

I skipped over where we have the code telling it the name of discount factor: $'DiscountFactorParamNames = \{'sj','beta'\}'$. I always do this just before I do the ReturnFn.

Basic Life-Cycle Model 1

6. Agents stationary distribution.

```
%% Initial distribution of agents at birth (j=1)
jequaloneDist=zeros(n_a,1,'gpuArray'); % Put no households anywhere on grid
jequaloneDist(1)=1; % start with 0 assets

% Mass of agents of each age
Params.mewj=ones(N_j,1)/N_j; % equal mass of each age (must some to one)
AgeWeightsParamNames={'mewj'}; % So VFI Toolkit knows which parameter is the mass of agents of each age

% Solve Stationart Distribution
simoptions=struct(); % Use the default options
StationaryDist=StationaryDist_FHorz_Case1(jequaloneDist,AgeWeightsParamNames,Policy,n_d,n_a,n_z,N_j,pi_z,Params,simoptions);
```

Basic Life-Cycle Model 1

7. Generate model moments/statistics. (1/2)

```
FnsToEvaluate.earnings=@(aprime,a,w,kappa_j) w*
    kappa_j; % w*kappa_j is the labor earnings
FnsToEvaluate.assets=@(aprime,a) a; % a is the
    current asset holdings
```

- FnsToEvaluate, create names and equations.
- Note: first inputs are action space, same as ReturnFn. Everything after is understood as parameters.

7. Generate model moments/statistics. (2/2)

```
%% Calculate various stats
AllStats=EvalFnOnAgentDist_AllStats_
%% Calculate the life-cycle profiles
AgeConditionalStats=LifeCycleProfiles_FHorz_Case1(
    StationaryDist,Policy,FnsToEvaluate,Params,[],n_d
    ,n_a,n_z,N_j,d_grid,a_grid,z_grid,simoptions);
```

- AllStats: computes things like mean, variance, Lorenz curve, etc.
- LifeCycleProfiles: computes the same, but conditional on age.
- Results are all by names of FnsToEvaluate: e.g., *AllStats.earnings.Mean*, and *AgeConditionalStats.assets.Gini*.

Basic Life-Cycle Model

- Done!
- Code: WorkshopModel1.m (and WorkshopModel1_ReturnFn.m)
- Let's add endogenous labor.

Basic Life-Cycle Model 2

- Household problem

$$
\begin{aligned}
V(a,j) \quad = \quad & \max_{h,c,aprime} \frac{c^{1-\sigma}}{1-\sigma} - \psi \frac{h^{1+\eta}}{1+\eta} + s_j \beta V(aprime, j+1) \\
& \text{if } j < Jr: \;\; c + aprime = (1+r)a + w\kappa_j h \\
& \text{if } j >= Jr: \;\; c + aprime = (1+r)a + pension \\
& 0 \leq h \leq 1, aprime \geq 0
\end{aligned}
$$

- Let's write the code to solve this.
  Code: *WorkshopModel2.m* (and *WorkshopModel2_ReturnFn.m*)
- Will only explain which of our seven core steps we change.

Basic Life-Cycle Model 2

1. Model action and state-spaces.
   - Decision variable: a variable that is in ReturnFn, but does not determine next period state.

```
%% Model action and state-space
n_d=21;
n_a=201; % number of grid points for our endogenous
    state
n_z=0;
N_j=81; % periods, represent ages 20 to 100
```

- Add $n\_d = 21$.
- Explain d vars: d vars concept

does not determine next period state=after we choose *aprime* directly, obviously *h* has an influence, but not after we account for *aprime*

Basic Life-Cycle Model 2

3. Grids
   - Add $d\_grid = linspace(0, 1, n_d)$.

```
%% Grids
d_grid=linspace(0,1,n_d)'; % note, 0<h<1 was a model eqn

a_grid=10*linspace(0,1,n_a)'.^3; % Column vector of length n_a
% ^3 will put more points near 0 than 1, model has more curvature here

% We are not using them so,
z_grid=[];
pi_z=[];
```

Step 2 was parameters, we need to add some, but changes are obvious.

Basic Life-Cycle Model 2

4. ReturnFn

```
function F=WorkshopModel2_ReturnFn(h,aprime,a,sigma,psi,eta,w,r,kappa_j,agej, Jr)
% first three entries are the action space

F=-Inf;

% budget constraint
if agej<Jr % working
    c=(1+r)*a +w*kappa_j*h - aprime;
else % retired
    c=(1+r)*a -aprime;
end

if c>0
    % utility fn
    F=(c^(1-sigma))/(1-sigma)-psi*(h^(1+eta))/(1+eta);
end
```

Basic Life-Cycle Model 2

7. Generate model moments/statistics. (1/2)

```
FnsToEvaluate.earnings=@(h,aprime,a,w,kappa_j) w*kappa_j*h; % w*kappa_j*h is the labor earnings
FnsToEvaluate.assets=@(h,aprime,a) a; % a is the current asset holdings
```

- FnsToEvaluate, create names and equations.
- Note: first inputs are action space, same as ReturnFn. Everything after is understood as parameters.
- Question: how to set up FnsToEvaluate for labor supply? Answer

Solve V and Policy unchanged. Solve stationary dist unchanged (note that d is in 'action space' but not 'state space', hence does not chane dimensions of stationary dist and V). Shape of Policy is slightly different as now contains optimal (index) d and *aprime*.

Basic Life-Cycle Model

- Done!
- Code: WorkshopModel2.m (and WorkshopModel2_ReturnFn.m)
- Let's add a markov exogenous state.

Basic Life-Cycle Model 3

- Household problem

$$
\begin{aligned}
V(a,z,j) \quad = \quad & \max_{h,c,aprime} \frac{c^{1-\sigma}}{1-\sigma} - \psi \frac{h^{1+\eta}}{1+\eta} + s_j \beta E[V(aprime, zprime, j+1)|z] \\
& \text{if } j < Jr: \; c + aprime = (1+r)a + w\kappa_j h exp(z) \\
& \text{if } j >= Jr: \; c + aprime = (1+r)a + pension \\
& 0 \leq h \leq 1, aprime \geq 0 \\
& z' = \rho_z z + \epsilon', \quad \epsilon \sim N(0, \sigma_{z,\epsilon})
\end{aligned}
$$

- Let's write the code to solve this.
  Code: *WorkshopModel3.m* (and *WorkshopModel3_ReturnFn.m*)
- Will only explain which of our seven core steps we change.

Basic Life-Cycle Model 3

1. Model action and state-spaces.
   - Exogenous markov variable: z

```
%% Model action and state-space
n_d=21;
n_a=201; % number of grid points for our endogenous
    state
n_z=9;
N_j=81; % periods, represent ages 20 to 100
```

- Add $n\_z = 9$.
- Explain z vars: z vars concept

Basic Life-Cycle Model 3

3. Grids
   - Add *z_grid* and *pi_z* (grid and the markov transition matrix).
   - Farmer-Toda is a standard quadrature method to discretize AR(1).
     Alternatives are Tauchen, Rouwenhorst, etc.

```
%% Grids
d_grid=linspace(0,1,n_d)'; % note, 0<h<1 was a model eqn

a_grid=10*linspace(0,1,n_a)'.^3; % Column vector of
    length n_a
% ^3 will put more points near 0 than 1, model has more
    curvature here

% Discretize AR(1) using Farmer-Toda method
[z_grid,pi_z]=discretizeAR1_FarmerToda(0,Params.rho_z,
    Params.sigma_zepsilon,n_z);
```

Step 2 was parameters, we need to add some, but changes are obvious.

4. ReturnFn

```
function F=WorkshopModel2_ReturnFn(h,aprime,a,z,
    sigma,psi,eta,w,r,kappa_j,agej, Jr)
% first three entries are the action space

F=-Inf;

% budget constraint
if agej<Jr % working
    c=(1+r)*a +w*kappa_j*h*exp(z) - aprime;
else % retired
    c=(1+r)*a -aprime;
end

if c>0
    % utility fn
    F=(c^(1-sigma))/(1-sigma)-psi*(h^(1+eta))/(1+eta
        );
end
```

Basic Life-Cycle Model 3

6. Agents stationary distribution.

```
%% Initial distribution of agents at birth (j=1)
jequaloneDist=zeros([n_a,n_z],'gpuArray'); % Put no
    households anywhere on grid
jequaloneDist(1,ceil(n_z/2))=1; % start with 0
    assets, median z shock
```

- Have to say what households look like in period 1 (here, zero assets).
- Just change initial dist to have the right 'state space', which is (a, z)
- Rest is unchanged.

Solve V and Policy unchanged. Except now the state space is (a, z, j), so they are different shapes.

Basic Life-Cycle Model 3

7. Generate model moments/statistics. (1/2)

```
FnsToEvaluate.earnings=@(h,aprime,a,z,w,kappa_j) w*kappa_j*h*exp(z); % w*kappa_j*h*exp(z) is the labor earnings
FnsToEvaluate.assets=@(h,aprime,a,z) a; % a is the current asset holdings
```

- FnsToEvaluate, create names and equations.
- Note: first inputs are action space, same as ReturnFn. Everything after is understood as parameters.

Basic Life-Cycle Model

- Done!
- Code: WorkshopModel3.m (and WorkshopModel3_ReturnFn.m)
- Easy to add/remove decisions/states.
- Even easier to change, e.g., utility function.

Basic Life-Cycle Model

- Value fn and Policy fn runtimes are all <2 seconds (on my laptop).
  Runtime for everything else combined is negligible.
- Examples have less grid points than you probably want.
  They are just to give you the idea. But with enough points, still likely just 2-3 seconds.
- Our action space is $(d, aprime, a, z, ...)$ [inputs to ReturnFn and FnsToEvaluate]
- Can do two of each variable, then $(d1, d2, a1prime, a2prime, aa1, a2, z1, a2, ...)$
  See 'Intro to Life-Cylce Models'. Two d and two z are easy, two a (endogenous states) works fine if you can divide-and-conquer and second state is say 51 points (e.g., housing). Two a is pushing the limits if both are full states.

Solving a Life-Cycle Model (repeated)

- Seven core steps to solve a Life-Cycle Model with VFI Toolkit.
  1. Model action and state-spaces.
  2. Parameters
  3. Grids
  4. ReturnFn
  5. Solve for value fn and policy fn.
  6. Agents stationary distribution.
  7. Generate model moments/statistics.

Basic Life-Cycle Model

- VFI Toolkit makes solving basic Life-Cycle Models easy.
- One last thing: adding a single line $vfoptions.divideandconquer = 1$...
  and remove $vfoptions = struct()$, are at least add the line after this
- ... means VFI Toolkit uses divide-and-conquer, exploiting monotonicity...
  $aprime(a)$ is an increasing function (monotone) in almost all models
- ... and making all the codes faster.
  Actually, on tiny models it can be slower, GPUs are so good at parallelization that in small models, brute force is actually fastest!

Basic Life-Cycle Model

- Enough of basic Life-Cycle Models, let's look some better ones :)
- Intro to Life-Cycle Models: pdf of 50 example Life-Cycle models, adding features one at a time. Covers everything we did here, plus much more.

References I

Basic Life-Cycle Model 2

- Can write Bellman equation like
$$V(a) = \max_{d,a'} F(d,a',a) + s_j \beta V(a')$$
- ReturnFn will be a function that plays role of $F(d, a', a)$
- Note that d matters for ReturnFn, but does not determine next period states.

Back

7. Generate model moments/statistics. (1/2)

```
FnsToEvaluate.earnings=@(h,aprime,a,w,kappa_j) w*
    kappa_j*h; % w*kappa_j*h is the labor earnings
FnsToEvaluate.assets=@(h,aprime,a) a; % a is the
    current asset holdings
```

- Question: how to set up FnsToEvaluate for labor supply?
- Answer: $FnToEvaluate.laborsupply = \texttt{@}(h, aprime, a)\ h$

Back

Basic Life-Cycle Model 3

- Can write Bellman equation like
$$V(a,z) = \max_{d,a'} F(d,a',a,z) + s_j \beta E[V(a',z')|z]$$
- ReturnFn will be a function that plays role of $F(d,a',a,z)$
- z is a markov exogenous state (so will have transition matrix pi_z).

Back