VFI Toolkit Workshop, pt1: Basic Life-Cycle Models

vfitoolkit.com/2025-workshop-lse

Robert Kirkby

robertdkirkby.com
Victoria University of Wellington

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 almost 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 afford 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. function problems
  - 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: How do the codes work?

- I don't plan to talk about the algorithms VFI Toolkit uses.
  Mostly pure discretization, preferably with 'divide-and-conquer' or 'refine', sometimes linear interpolation. Lots of GPU!
- You can find pseudo-codes at: github.com/vfitoolkit/Pseudocodes
  I add commands to this pseudo-codes pdf 'on demeand', so if the command you want to see is not there yet, ask me via the forum: discourse.vfitoolkit.com

- I will however shout-out to three algorithms that are important to VFI Toolkit
- Tan (2020): Tan improvement, used for all agent distribution codes. Revolutionary improvement.
- Gordon and Qiu (2018): Divide-and-conquer, slashes both memory and runtimes for value fn iteration (GPU adaptation).
- Tanaka and Toda (2013): basis of the most powerful quadrature methods (discretizing shocks).

VFI Toolkit: Plan for Workshop

- Today we will see Life-Cycle models.
- Part 1: Start with basic life-cycle model.
- Part 2: markov shocks, i.i.d. shocks, semi-exogenous shocks, permanent types.
  And some more model analysis: simulate panel data, conditional model stats.
- Part 3A: Alternative Preferences (Epstein-Zin, Gul-Pesendorfer, Loss Aversion)
- Part 3B: Other Endogenous States (housing, portfolio-choice, human capital)
- Part 3C: Calibration and Estimation.
- Tomorrow: OLG models (general eqm) and transition paths.
- Part 4: OLG Models
- Part 5: OLG transition paths.
- (Part 6: Infinite Horizon models)

Caveat:

- This workshop is not going to do any actual Economic Science.
- For Economic Science, you *must* care about the numbers in the model, and make sure they are 'realistic'.
- Here I just use made up parameter values.
- This workshop is about how to solve models that are very useful and powerful for doing Economics.
  Innumerable papers use these models to do Economic Science.
- But we won't do any actual Economics, in the sense that none of the model results is based on empirically relevant numbers.

We will talk about how to put empirically relevant numbers into models. Just want to emphasise that none of the models we will use has had the numbers chosen to be empirically relevant.

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.

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

people are born in period 1,
j is their age,
they live for a finite number of periods: J

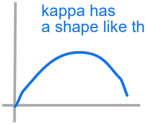
- Household problem

Poeople have exegoenous income k
The decision is: consume or save it in a given period?

$$V(a,j) = \max_{c,aprime} \frac{c^{1-\sigma}}{1-\sigma} + s_j \beta V(aprime, j+1)$$

earnings

kappa has
a shape like this



$$\begin{aligned}
&\text{if } j < Jr: \; c + aprime = (1+r)a + w\kappa_j \\
&\text{if } j >= Jr: \; c + aprime = (1+r)a + pension \\
&aprime \geq 0
\end{aligned}$$

budget constraint

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

aprime is greater than zero, so savings cannot
be negaive - this is a no Ponzi condition

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.

It is a state that appears in V(a,j)
people will make decisions based on that
state

How is it endogenous?

Because it is a state that we can
influence the evolution of

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.

Basic Life-Cycle Model 1

A prime makes it a column vector because in matlab they are row vectors by default

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
$$V_j(a) = \max_{a'} F(a', a) + s_j \beta V_{j+1}(a')$$
- 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 the **action space**, $(aprime, a)$, anything after these is interpreted as parameters.
  'action space'=things you can choose, and the state-space (not counting j).

Basic Life-Cycle Model 1

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
```

Action space parameters: aprime and a need to be enetered first in that order
the rest of params can be entered in any given way

Basic Life-Cycle Model 1

5. Solve for Value fn and Policy fn.

Solving of life cycle models relies on backward iteration

```
%% 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
```

Policy prints out the index of optimal policies not the values of that policy!

- 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.

6. Agents stationary distribution. Setup.

Agent distribution is obtained with forward iteration. To do that we need to define where each agent starts and define how many people they are at different ages

```
%% 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
```

- Have to say what households look like in period 1 (here, zero assets).
- Mass for initial dist is 1.

Basic Life-Cycle Model 1

Policy is the solution to the problem,
it is the decisions that agents take in every period

6. Agents stationary distribution. Solve.

```
% 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);
```

- Runtime is very fast.

Iterate on agent distribution, use Tan improvment (when there are shocks, so not actually used here).

Basic Life-Cycle Model 1

By this point we have solved the model, now we can calculate statistics

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
```

best to enter params in the same order as in the objective function

- 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 change 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.

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 four 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)
- Mass for initial dist is 1.
- 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 $(d, aprime, a, z, \ldots)$ [inputs to ReturnFn and FnsToEvaluate]
  decision parameters, next period endogenous states, this period endogenous states, markov shocks
- Can do two of each variable, then $(d1, d2, a1prime, a2prime, aa1, a2, z1, z2, \ldots)$
  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.
- Lastly: adding a single line $vfoptions.divideandconquer = 1$...
  and remove $vfoptions = struct()$, or 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!
- Or $vfoptions.lowmemory = 1$ solves slower but uses less gpu memory.
  Loops over z (or if have e, loop over e, and $vfoptions.lowmemory = 2$ loops over e and z)

Basic Life-Cycle Model

- Enough of basic Life-Cycle Models, let's look at 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

Grey Gordon and Shi Qiu. A divide and conquer algorithm for exploiting policy function monotonicity. *Quantitative Economics*, 9(2), 2018. doi: https://doi.org/10.3982/QE640.

Robert Kirkby. VFI toolkit, v2. *Zenodo*, 2022. doi: https://doi.org/10.5281/zenodo.8136790.

Eugene Tan. A fast and low computational memory algorithm for non-stochastic simulations in heterogeneous agent models. *Economics Letters*, 193, 2020. doi: https://doi.org/10.1016/j.econlet.2020.109285.

Kenichiro Tanaka and Alexis Akira Toda. Discrete approximations of continuous distributions by maximum entropy. *Economics Letters*, 118(3):445–450, 2013.

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 1A: $FnToEvaluate.laborsupply = \texttt{@}(h, aprime, a)\ h$
- Answer 1B: $FnToEvaluate.effectivelaborsupply = \texttt{@}(h, aprime, a, kappa\_j)\ kappa\_j * h$

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