Open Source Computational Economics

The State of the Art

Sebastian Benthall

Econ-ARK

PyData NYC, Nov 2023

Table of Contents

DYNARE

DYNARE: Software tools for solving Representative Agent models.

- Began by Michel Juillard in 1994
- Dynamic Stochastic General Equilbrium (DSGE) models
    - representative agents
    - complete markets
- A domain specific language (DSL)
    - Pre-processor in C++
    - Models portable to MATLAB, Scilib, GNU Octave, now Julia
    - Became open source in the 2000s.

Changing timess

- Great Financial Crisis of 2008 throws doubt on DSGE and RA paradigm.
  - Heterogeneous agents
  - Incomplete Markets
  - Bounded rationality
- New software paradigms
  - New programming languages and optimizations: faster Python, Julia, GPU programming
  - New open source development workflows
  - New computational science tools: Jupyter, etc.

QuantEcon

- Thomas Sargent (Nobel-prize winner!) and John Stachurski, and others
- Lecture notes in computational notebooks
- Libraries of basic building-blocks used in repeated demos
- Both Python and Julia versions (common pattern)

From Representative to Heterogeneous Agent Modeling

- HA modeling is more expressive than RA modeling. E.g. Krusell and Smith, '98
- Effective modeling has been limited by the computational complexity of solving high-dimensional models.

Abstract problem notation

Definition: Optimization Problem (time-invariant) pt. 1

An exogenous state $m_{t+1} \in \mathbb{R}^{n_m}$ following Markov process driven by i.i.d. $\epsilon_t \in \mathbb{R}^m$

$$m_{t+1} = M(m_t, \epsilon_t)$$

An endogenous state s_{t+1} driven by the exogenous state m_t and controlled by choice $x_t \in \mathbb{R}^{n_x}$ according to

$$s_{t+1} = S(m_t, s_t, x_t, m_{t+1})$$

The choice x_t satisfies the constraint in the form $x_t \in X(m_t, s_t)$.

(Maliar, Maliar, and Winant, 2021)

Abstract problem notation

Definition: Optimization Problem (time-invariant) pt. 2

The state (m_t, s_t) and choice x_t determine the period reward $r(m_t, s_t, x_t)$.
The agent maximizes discounted lifetime reward

$$\max_{\{x_t, s_{t+1}\}_{t=0}^{\infty}} E_0 \left[\sum_{t=0}^{\infty} \beta_t r(m_t, s_t, x_t) \right]$$

where $\beta \in [0, 1)$ is the discount factor and $E_0[]$ is an expectation function across future shocks $(\epsilon_1, \epsilon_2, ...)$ conditional on the initial state $(m0, s0)$.

(Maliar, Maliar, and Winant, 2021)

Abstract problem notation

Definition: Optimal decision rule

An optimal decision rule is a function $\varphi : \mathbb{R}^{n_n} \times \mathbb{R}^{n_s} \rightarrow \mathbb{R}^{n_x}$ such that

$$\forall t, x_t = \varphi(m_t, s_t) \in X(m_t, s_t)$$

and the sequence $\{x_t, s_{t+1}\}_{t=0}^{\infty}$ maximizes the lifetime reward for any initial condition (m_0, s_0)

Abstract problem notation

Definition: Parametric decision rule

A parametric decision rule is a member of a family of functions $\varphi(;\theta)$ parameterized by a real vector $\theta \in \mathbb{R}^{n_\theta}$ such that for each θ, we have $\varphi : \mathbb{R}^{n_m} \times \mathbb{R}^{n_s} \rightarrow \mathbb{R}^{n_x}$ and

$$\forall t, x_t = \varphi(m_t, s_t) \in X(m_t, s_t)$$

The goal is to find the parameters $\theta \in \mathbb{R}^{n_\theta}$ under which the parametic decision rule $\varphi(\cdot, \theta)$ best approximates the optimal decision rule φ. (Maliar, Maliar, and Winant, 2021)

Grid-based Dynamic Programming Solutions Methods

Lifetime reward function implies recursive Bellman equation:

$$V(m_t, s_t) = \max_{x_t \in X(m_t, s_t)} r(m_t, s_t, x_t) + \beta_t E_{m_{t+1}} \left[V(m_{t+1}, S(m_t, s_t, x_t, m_{t+1}) \right]$$

This problem can be solved using a variety of techniques:

- Dynamic Programming
- Reinforcement Learning (not common in Economics)
- Backwards induction (when the problem is finite horizon)

Value Iteration Algorithm

Set V to arbitrary value function.

1. $\Delta \leftarrow 0$.
2. For each $s \in S$:
   1. $V'(m_t, s_t) \leftarrow \max_{x_t \in X(m_t, s_t)} r(m_t, s_t, x_t) + \beta_t E_{m_{t+1}} [V(m_{t+1}, S(m_t, s_t, x_t, m_{t+1})]$
   2. $\Delta \leftarrow \max(\Delta, V'(s) - V(s))$
3. $V \leftarrow V'$
4. If $\Delta < \varepsilon$ halt, otherwise return to step 1.

Continuous state spaces must be discretized first.
The problem becomes exponentially complex in the dimensionality of the state space! Curse of dimensionality.

Dolo

- YAML-based configuration language for models (Dolang)
- Compiles to Python and Julia
- GPU and Numba optimizations
- Many solution algorithms
- Simulates Markov Chains using transtion matrices
- Modeling limitations:
  1. time invariance
  2. no market aggregration

```yaml
name: Real Business Cycle

symbols:

   exogenous: [e_z]
   states: [z, k]
   controls: [n, i]
   parameters: [beta, sigma, eta, chi, delta, alpha, rho, zbar, sig_z]

definitions: |
    y[t] = exp(z[t])*k[t]^alpha*n[t]^(1-alpha)
    c[t] = y[t] - i[t]
    rk[t] = alpha*y[t]/k[t]
    w[t] = (1-alpha)*y[t]/n[t]

equations:

    arbitrage: |
        chi*n[t]^eta*c[t]^sigma - w[t]                     ⊥ 0.0 <= n[t] <= inf
        1 - beta*(c[t]/c[t+1])^(sigma)*(1-delta+rk[t+1])   ⊥ -inf <= i[t] <= inf

    transition: |
        z[t] = rho*z[t-1] + e_z
        k[t] = (1-delta)*k[t-1] + i[t-1]
```

HARK

- Part of Econ-ARK, NF sponsored project.
- Python-only
- Endogenous Gridpoints Method (EGM) and policy iteration
- Lifecycle problems (time varying optimization)
- Monte Carlo simulation
- Population mortality dynamics
- Market aggregation (e.g. Krusell-Smith models)

Deep Learning for HA Models (!)

New uses of deep learning in Economics shaking things up:

- Encoding model conditions in objective function: Maliar Maliar and Winant (2021), Azinovic, Gaegauf, Scheidegger (2022)
- Estimating models: Chen, Didisheim, and Scheidegger (2023), Chassot and Creel (2023)

Deep Learning for HA Models (!)

The overall strategy is to construct a function

$$\Xi(\theta) = E_\omega(\xi, \theta)$$

where $\omega = (m, s, \epsilon)$ and:

- $\Xi(\theta)$ contains all model equations by construction
- Minimizing $\Xi(\theta)$ solves the entire model, including $\varphi(\theta)$
- Ξ is an "All-in-One" (AiO) operator, meaning it takes expectation over both the sequence of future shocks $(\epsilon_0, ..., \epsilon_T)$ and initial state (m_0, s_0)

Deep Learning for HA Models (!)

Recall that dynamic programming solutions involved a grid over states, leading to the curse of dimensionality and exponential computational costs.
Instead of using a grid, MMW'21:

- Sample $\omega_{i=1}^{n} = (m_i, s_i, \epsilon_i)_{i=1}^{n}$ from the model during training.
- Compute empirical loss $\Xi^n(\theta) = \frac{1}{n}\sum_{i=1}^{n}\xi(\omega_i;\theta)$

The Monte Carlo simulation of $\omega_{i=1}^{n}$ converges on the ergodic distribution of the simulation.

This makes high-dimension problems tractable!

Algorithm

1. Initialize the algorithm:
    1. construct theoretical risk $\Xi(\theta) = E_\omega[\xi(\omega;\theta)]$ (lifetime reward, Euler/Bellmanequations);
    2. define empirical risk $\Xi^n(\theta) = \frac{1}{n}\sum_{i=1}^{n}\xi(\omega_i,\theta)$;
    3. define a topology of neural network $\varphi(\cdot,\theta)$;
    4. fix initial vector of the coefficients θ.
2. Train the machine, i.e., find θ that minimizes the empirical risk $\Xi^n(\theta)$:
    1. simulate the model to produce data $\{\omega\}_{i=1}^{n}$ by using the decision rule $\varphi(\cdot,\theta)$;
    2. construct the gradient $\nabla\Xi^n(\theta) = \frac{1}{n}\sum_{i=1}^{n}\nabla\xi(\omega_i,\theta)$;
    3. update the coefficients $\hat{\theta} = \theta - \lambda_k\nabla\Xi^n(\theta)$ and go to step 2.i;

    End Step 2 if the convergence criterion $||\hat{\theta} - \theta|| < \varepsilon$
3. Assess the accuracy of constructed approximation $\varphi(\cdot,\theta)$ on a new sample.

(Maliar, Maliar, and Winant, 2021)

Example Problem

A simple consumption-saving problem with borrowing constraint:

$$\max E\left[\sum_{t=0}^{\infty} \beta_t U(c_t)\right]$$

$$w_{t+1} = r(w_t - c_t) + e^{y_t}$$

$$c_t \leq w_t$$

$$y_t = \sigma \epsilon_t; \epsilon \sim \mathcal{N}(0, 1)$$

One possible objective function:

$$\Xi(\theta) = E[\xi(\omega; \theta)] = E_{(y_0, w_0, \epsilon_0, \ldots, \epsilon_T)}\left[\sum_{t=0}^{T} \beta_t u(c_t)\right]$$

Others involve Euler equations, Bellman equations, etc. (Maliar, Maliar, and Winant, 2021)

Estimation

Consider a model F such that

$$(x_t, m_{t+1}, s_{t+1}) = F(m_t, s_t, \varepsilon | p)$$

via optimal decision rule φ_t, which is a solution of the model.
$p \in \mathbb{R}^{n_p}$ is a parameterization of the model.

This implies a joint distribution $\mathcal{P}_\epsilon(m_t, s_t, x_t | F, p)$.

Estimation involves, given empirical distribution $\mathbf{d} = (\mathbf{x}, \mathbf{m}, \mathbf{s})$, computing the likelihood $P(p|F, \mathbf{d})$, or

$$\hat{\theta} = \arg\max_{p \in \mathbb{R}^{n_p}} \mathcal{P}_\epsilon(\mathbf{d}|F, p)$$

This can involve running and solving F many times under different parameterizations. This can be expensive.

Estimation: Deep Surrogates

Trick: treat parameters p as part of a pseudo-state q, such that $(x_t, p) = q \in Q : \mathbb{R}^{n_s} \times \mathbb{R}^{n_p}$

Now

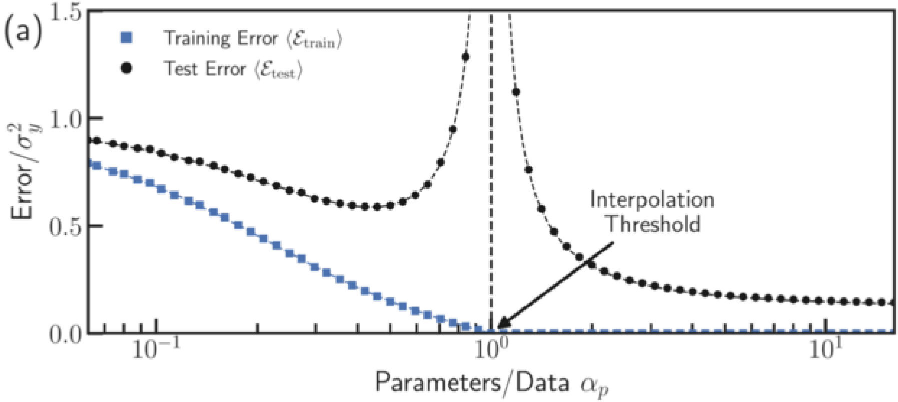
$$(x_t, p, m_{t+1}, s_{t+1}) = f(m_t, q_t) = F(m_t, s_t|p)$$

Now, given f, sample a large number of samples $(\mathbf{m_t}, \mathbf{q_t})$ and train a *deep surrogate* neural network $\hat{f}$ of the model based on the prediction loss with respect to the sample.

This training can take advantage of *double descent* and parallelized sampling easily because there is no noise!

Chen, Didisheim, and Scheidegger (2023)

Double Descent



Source: Wikipedia

Estimation: Deep Surrogates

Then use $\hat{f}$ when searching for optimal $\hat{\theta}$.

The derivative $\frac{\partial q}{\partial p}$ is given by $\hat{f}$ because it's a function of the model weights. So this optimization is easy.

Chen, Didisheim, and Scheidegger (2023)

Estimating in step with solving

There is another use of the pseudo-state trick.

If the parameters θ are included in the pseudo-states, and the loss function with respect to the empirical data $\mathbf{d}$ is encoded into the loss function Ξ, then the solution and estimation can be achieved simultaneously by training a neural network!

The Future: Modeling language and APIs

- Deep learning made solving and estimating a much larger range of models suddenly feasible.
- However, modeling and methods have been developed ad hoc
- Better standardizing around modeling configurations and APIs is the future!
- As is research on encoding models into NNs.

Conclusion

Feel free to reach out with any questions.
Email: spb413@nyu.edu