PyShop Session 3

Advanced Numerical Methods

Tyler Abbot

Department of Economics
Sciences Po

Fall 2015

Outline

Learning by Doing

Today we are going to discuss NumPy and SciPy through an example: Finite Elements. We will...

- ...discuss the background of finite elements in economics.
- ...outline a simple problem.
- ...build a naive program to solve that problem.
- ...vectorize the program for max speed!

What is Finite Elements?

- Used mainly in Physics/Fluid Dynamics/Engineering problems.
- Developed in the 1950's/1960's.
- Break down the problem to smaller ones (like everything we do).
- Relies on the weak formulation of a functional problem.
- Most often uses Galerkin weighted residual method.
- We won't talk about the theory, just an applciation.

FEM in Econ

- Seminole paper: McGratten 1993.
- Most often referred to as "projection methods".
- Based on parameterizing a decision rule to reduce dimensionality.
- Modern applications are much faster and offer a broader solution than linearization/perturbation.
- That doesn't mean it's better! Just different.
- Main drawback: high computing time.

Set Up

Based on McGratten, 1993.

Take a basic RBC model of capital accumulation in discrete time, where productivity follows a finite state markov chain. The agent's maximization is thus

$$
\begin{aligned}
\max_{\{c_t, k_{t+1}\}_{t=0}^{\infty}} \quad & \mathbb{E} \sum_{t=0}^{\infty} \beta \frac{c_t^{1-\gamma}}{1-\gamma} \\
\text{s.t.} \quad & y_t = a_t k_t^{\alpha} \\
& k_{t+1} = (1-\delta) k_t + i_t \\
& a_t \in \{a_1, ..., a_I\} \\
& \mathbb{P}[a_{t+1} = a_j | a_t = a_i] = \Pi_{i,j}
\end{aligned}
$$

Solution

By standard steps we arrive at the euler equation and equation of motion for capital:

$$c_t^{-\gamma} = \beta \sum_{i=1}^{I} \Pi_{ij} \left[c_{t+1}^{-\gamma} \left(1 - \delta + \alpha a_i k_{t+1}^{\alpha-1} \right) \right]$$
$$k_{t+1} = a_i k_t^{\alpha} + (1-\delta) k_t - c_t$$

for all i, j.

Parameterization

To reduce dimensionality, assume a parameterized form for capital choice:

$$\hat{k}_{t+1} = \sum_{l=1}^{L} N_l(k_t)\kappa_l$$

where $N_l(\cdot)$ is a set of basis functions for the state space and κ_l are constants.

In this case, this addition is sufficient to solve the problem! We look to determine the set of constants κ_l.

Basis

Must specify a set of basis functions. For simplicity (but this is actually much more difficult as a programming problem), we'll use the linear interpolator.

Partition the state space such that $k_l \in \{k_1, k_2, ..., k_L\}$ is a partition (evenly spaced or not). This, coupled with the discret state, implies a set of $(L-1) \times I$ finite elements.

Define the linear interpolator as the basis function

$$N_l(k) = \begin{cases} \frac{k-k_{l-1}}{k_l-k_{l-1}} & k_{l_1} \leq k \leq k_l \\ \frac{k_{l+1}-k}{k_{l-1}-k_l} & k_l \leq k \leq k_{l+1} \\ 0 & else \end{cases}$$

Basis II

Thus, the parameterized decision rule for capital becomes

$$\hat{k}(k) = \frac{k_{l+1} - k}{k_{l-1} - k_l}\kappa_l^i + \frac{k - k_l}{k_{l+1} - k_l}\kappa_{l+1}^i \qquad k \in [k_l, k_{l+1}]$$

Given this, our residual equations will be imprecise. Thus, we use the weak form and Galerkin weights to rewrite our problem as an integral.

Galerkin Weak Form

$$\sum_{i=1}^{I} \int_{k_0}^{k^L} w(k,i) R(k,i;\kappa) dk = 0$$

We define the weighting function to be the same linear interpolator:

$$\sum_{i=1}^{I} \sum_{a=1}^{L} w_a^i \left\{ \sum_{l=1}^{L-1} \int_{k_l}^{k_{l+1}} N_a(k) R(k,i;\kappa) dk \right\} = 0$$

Under the weak form Galerkin method, we assume this holds for all w, so the stuff inside the braces must be zero.

System

Thus, we arrive at our system:

$$\sum_{l=1}^{L-1} \int_{k_l}^{k_{l+1}} N_a(k) R(k, i; \kappa) dk \qquad\qquad \forall i, l$$

This defines a system of $L \times I$ equations in as many unknowns, which we can solve rather (key word: RATHER) easily.

We'll use gaussian quadrature to approximate the integral and build a simple implimentation, which we'll try to modify to be more Pythonic.

Brainstorm

What are the problems we need to solve? What steps will our program take?

Brainstorm

What are the problems we need to solve? What steps will our program take?

- Define parameters
- Define a state space
- Generate gaussian quadrature weights and abcissas
- Guess the coefficients
- Solve the system of equations

Brainstorm

What are the problems we need to solve? What steps will our program take?

Brainstorm

What are the problems we need to solve? What steps will our program take?

- For every initial state point...
- Calculate c_t, k_t, y_t...
- Use $\hat{k}$ to calculte k_{t+1}...
- Given k_{t+1}, calculate $k_{t+2}, c_{t+1}, y_{t+1}$...
- Calculate conditional expections...
- Calculate the weighted residual

OK! Let's program!

The Looped Version

Check out the notes for the program.

Pythonic/Vectorized

Check out the notes for the program.

Conclusion

Let's recap what we've seen:

- FEM is tough!
- Easily computable in loops
- We can (relatively) easily vectorize looped functions
- For large loops we can get a substantial speed up using NumPy

Lists II

List as Stack

```
import pandas as pd

DF = pd.read_csv('http://people.stern.nyu.edu/wgreene/Econometrics/dai
```