estimagic

Janoś Gabler

May 23, 2019

Outline

What is Estimagic?

What is Estimagic?

- **Estimagic** is a Python package for the estimation of (structural) econometric models and inference on the estimated parameters
- Model ≈ Criterion function for extremum-estimator
- It is a meta package!
- **Today**: estimagic's tools for numerical optimization
- **Next Time**: tools for standard errors

look at 1_scipy_minimize.ipynb

Traditional Optimizers

- Take an array of start parameters
  - Encourage position based parameter handling
- Run for hours or days without feedback
  - User writes/prints logs for monitoring
- Lose all information when they crash
  - User saves parameters after every step
- Don’t support constraints typical in economics
  - User implements them by reparametrizations
- Only do minimization
  - User flips sign manually

Tutorial: Implement Ordered Logit

Goal

```
def ordered_logit(formula, data, dashboard=False):
    """Estimate ordered logit model described by *formula* on *data*.

    Args:
        forumla (str): A valid patsy formula
        data (DataFrame)
        dashboard (bool): Switch on the magic in estimagic!

    """
    pass

# usage
formula = 'apply ~ pared + public + gpa'
data = pd.read_stata('ologit.dta')
ordered_logit(formula, data)
```

look at 2_ordered_logit_example.ipynb

Design Philosophy

```
>>> import this

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one -- and preferably only one -- obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Deep Functions an Modules

- Terms coined by John Ousterhout
- **Deep Modules**: Each module has only one or two public functions that serve as table of contents for the rest. All other functions are private and not imported anywhere else
- **Deep Functions**: Functions have few mandatory arguments and sensible defaults but many optional arguments to configure every detail

look at 3_respy_example.ipynb and

4_constraint_example.ipynb

There should be one way ...

```
There should be one - and preferably
only one - obvious way to do it.
Although that way may not seem obvious
at first unless you’re Dutch
```

There should be one way ...

```
There should be one - and preferably
only one - obvious way to do it.
Although that way may not seem obvious
at first unless you’re Dutch
```

- Don’t implement things just for completeness
- We need good defaults for everything!

Good Defaults

- Convergence checks will be enabled by default
- (Pseudo) global optimizers are default
- Things that should not be changed, cannot be changed
- Safe methods for numerical differentiation

Minimize State

- No classes
- No functions with side-effects
- Avoid mutable objects

Code Reviews

- No PR is merged without review
- Iterate over code two or three times to:
  - Improve readability
  - Reduce complexity
  - Ensure consistency

Unit Tests for Everything!

Build on Existing Code

- Wrap code, don’t copy-paste it!
- That’s why we don’t build on mystic
  - A library with similar goals as ours
  - Achieves some things we cannot!
  - Copies and modifies code of optimizers
  - So far supports 6 optimizers
  - Adding C or Fortran Optimizers would be a pain!

Some Implementations

Implement Constraints

```
def _create_internal_criterion(criterion, params, internal_params,
    constraints, criterion_args, criterion_kwargs):

    def internal_criterion(x):
        params_sr = _params_sr_from_x(x, internal_params, constraints, params)
        return  [criterion(params_sr, *criterion_args, **criterion_kwargs)]

    return internal_criterion
```

Maximize

```
def maximize(criterion, params, algorithm, criterion_args=None,
    criterion_kwargs=None, constraints=None):

    def neg_criterion(*criterion_args, **criterion_kwargs):
        return -criterion(*criterion_args, **criterion_kwargs)

    res_dict, params = minimize(
        neg_criterion, params=params, algorithm=algorithm,
        criterion_args=criterion_args, criterion_kwargs=criterion_kwargs,
        constraints=constraints)

    res_dict["f"] = -res_dict["f"]

    return res_dict, params
```

Estimagic in Action

look at 5_skillmodel_example.ipynb

Roadmap

Roadmap

- Add interface to pygmo's parallelization capabilities
- Save state of optimizers at KeyboardInterrupt or shutdown
- Add profiling to dashboard
- Use numerical gradients for nlopt algorithms
- ...