Applying Symbolic Mathematics in Stata using Python

Kye Lippold

2020 Stata Conference

7/31/2020

Introduction

- Stata 16 includes integration with Python through the Stata Function Interface (SFI).
- This opens up opportunities to use Stata as a computer algebra system.
- I will demonstrate basic usage through an application substituting empirical elasticities into a dynamic labor supply model.

Computer Algebra Systems

- Commonly used via software like *Mathematica*.
- Represent mathematical expressions in an abstract symbolic (rather than numeric) form.
  - Allows exact evaluation of expressions like π or $\sqrt{2}$.
- Perform operations like expression evaluation, differentiation, integration, etc.
- Stata's Python integration allows performing symbolic computations in Stata via the *SymPy* library.

SymPy

SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible in order to be comprehensible and easily extensible.

Info: https://www.sympy.org/

Figure 1: Sympy Logo

SymPy Installation

- Part of many Python package managers (Anaconda, Pip, etc)

```
! pip install sympy
```

SymPy Usage

- Enter python environment, load module, and perform symbolic calculations:

```
. python
---------------------------------------------- python (type
>  end to exit) ---------------------------------------------
>>> import sympy
>>> x, y = sympy.symbols('x y')
>>> expr = x + (y**2 / 2)
>>> print(expr)
x + y**2/2
>>>
>>>
>>>
>>>
```

SymPy Usage

```
>>> # prettier printing:
... sympy.init_printing(use_unicode=True)
>>> expr
     2
    y
x + ──
    2
>>> expr * x**2
   ⎛     2⎞
 2 ⎜    y ⎟
x ⋅⎜x + ──⎟
   ⎝    2 ⎠
>>>
>>>
>>>
```

SymPy Usage

```
>>> # solver
... from sympy import solve, diff, sin
>>> solve(x**2 - 2,x)
[-√2, √2]
>>> diff(sin(x)+x,x)
cos(x) + 1
>>> end
--------------------------------------------------------------------
```

Empirical Application

- In Lippold (2019), I develop a dynamic labor supply model that compares changes in work decisions after a temporary versus permanent tax change.
  - Agents decide each period whether to work based on wages, income, tax rates, etc.
  - My study uses a temporary tax change for identification, so want to estimate the response if the change was permanent.
- Formally, I relate the compensated steady-state elasticity of extensive margin labor supply ϵ_s to the intertemporal substitution elasticity ϵ_I.

Model

The model equation is

$$\varepsilon_I \approx \left(\frac{1 - \frac{\gamma W_t}{1-s_t}\left(1 - \frac{2\alpha}{1+r_t} + \frac{(2+r_t)\alpha^2}{(1+r_t)^2}\right)}{1 - \frac{\gamma W_t}{1-s_t}} \right) \epsilon_s$$

where the relationship varies based on

- The coefficient of relative risk aversion γ
- The marginal propensity to save α (equal to $1-\mu$, where μ is the marginal propensity to consume)
- The interest rate on assets r_t
- The savings rate s_t
- The percent change in post-tax income when working W_t

Empirical Estimates

- Using variation in tax rates from the Child Tax Credit, I compute ε_I with a regression discontinuity design in Stata.
- I then want to plug my results into my formula. The usual methods:
  - Enter into a calculator or Excel by hand. (Not programmatic, prone to error).
  - Solve an expression written using macros. (Hard to modify expression in future).
- The SFI creates a direct link from the empirical estimate to the symbolic formula.

Import LaTeX Formula

```
. python:
---------------------------------------------- python (type
>  end to exit) ---------------------------------------------
>>> import sympy as sp
>>> gamma, alpha, w, s, r = sp.symbols(r'\gamma \alpha W_{t}
>  s_{t} r_{t}')
>>> formula = r"\frac{\left(1-\frac{\gamma W_{t}}{1-s_{t}}\l
> eft(1-\frac{2\alpha}{1+r_{t}}+\frac{\left(2+r_{t}\right)\a
> lpha^{2}}{\left(1+r_{t}\right)^{2}}\right)\right)}{\left(1
> -\frac{\gamma W_{t}}{1-s_{t}}\right)}"
>>> # clean up for parsing
... formula = formula.replace(r"\right","").replace(r"\left"
> ,"")
>>>
>>>
```

Import LaTeX Formula

```
>>> # parse
... from sympy.parsing.latex import parse_latex
>>> multiplier = parse_latex(formula)
>>> multiplier
                ⎛ 2                                              ⎞
                ⎜α ⋅(r_{t} + 2)           2⋅α                    ⎟
  W_{t}⋅γ⋅⎜──────────────── + - ─────────── + 1⎟
                ⎜              2        r_{t} + 1                ⎟
                ⎝ (r_{t} + 1)                                    ⎠
- ──────────────────────────────────────────────── + 1
                          1 - s_{t}
─────────────────────────────────────────────────────
                          W_{t}⋅γ
                      - ─────────── + 1
                          1 - s_{t}
```

Import LaTeX Formula

```
>>> m = multiplier.subs([('gamma',1),(s,-0.02), ('alpha',0.7
> 5), (r,0.073)])
>>> m
1 - 0.602791447544363⋅W_{t}
───────────────────────────
1 - 0.980392156862745⋅W_{t}
>>> end
------------------------------------------------------------
```

Compute Empirical Values

After running my main analysis code, I have computed the following empirical values:

```
. scalar list
       W_t =  .80264228
 epsilon_I =  1.0401141
```

I can then plug these values into the previous formula to get the desired statistic.

```
. python
----------------------------------------------- python (type
>  end to exit) -------------------------------------------
>>> import sfi
>>>
>>>
```

Compute Empirical Values

```
>>> # empirical elasticity
... epsilon_I = sfi.Scalar.getValue("epsilon_I")
>>> # empirical return to work
... W_t = sfi.Scalar.getValue("W_t")
>>> m.subs([(w,W_t)])
2.42226308973109
>>> epsilon_s = epsilon_I / m.subs([(w,W_t)])
>>> print(epsilon_s)
0.429397657197176
>>> end
------------------------------------------------------------
```

Standard Errors via Bootstrapping

get_elasticity.ado:

```
prog def get_elasticity, rclass
    // analysis code...
    return scalar epsilon_I  = //...
    return scalar W_t  = //...
    python script py_compute.py
end
```

py_compute.py:

```
# repeat earlier code to get multiplier 'm'...
epsilon_I = sfi.Scalar.getValue("return(epsilon_I)")
W_t = sfi.Scalar.getValue("return(W_t)")
epsilon_s = epsilon_I / m.subs([(w,W_t)])
result = sfi.Scalar.setValue('return(epsilon_s)',epsilon_s)
```

Run Bootstrap

```
. set seed 77984

. bs elasticity = r(epsilon_s), reps(50): get_elasticity
(running get_elasticity on estimation sample)

Bootstrap replications (50)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5
..................................................    50

Bootstrap results                               Number of obs     =      9,443
                                                Replications      =         50

      command:  get_elasticity
   elasticity:  r(epsilon_s)

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
             |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
  elasticity |   .4293977    .205351     2.09   0.037      .026917    .8318783
------------------------------------------------------------------------------
```

Conclusion

- Using SymPy with Stata 16 opens up exciting possibilities to incorporate symbolic mathematics into Stata computations.
  - Solve equations with computer algebra, then substitute returned results.
  - Close correspondence between LaTeX output and code
- New `pystata` features announced yesterday would allow using these methods in Jupyter notebooks.
- Code will be available at https://www.kyelippold.com/data

References

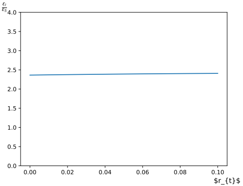
Lippold, Kye. 2019. "The Effects of the Child Tax Credit on Labor Supply." *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3543751.

Sensitivity plots

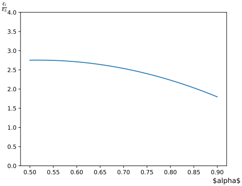
```
from numpy import linspace
import matplotlib.pyplot as plt
substitutions = [('gamma',1,0,2), (w,W_t,0,1), \
    (s,-0.02,-.05,.1), ('alpha',0.75,.5,.9), (r,0.073,0,.1)]
for param in substitutions:
    name = param[0]
    others = substitutions.copy()
    others.remove(param)
    sub = [(vals[0],vals[1]) for vals in others]
    expr = multiplier.subs(sub)
    lam_x = sym.lambdify(name, expr, modules=['numpy'])
    x_vals = linspace(param[2],param[3],100)
    y_vals = lam_x(x_vals)
```

Sensitivity plots

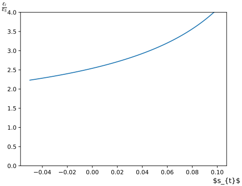
```
plt.figure()
plt.plot(x_vals, y_vals)
plt.ylabel(r'$\frac{\epsilon_I}{\epsilon_S}$',\
    rotation=0,fontsize=12, y=1)
plt.xlabel(r'\${}\$'.format(name),fontsize=12, x=1)
plt.ylim(0,4)
#plt.show()  # to see in session
disp_name = str(name).replace("\\","").replace("_{t}","")
plt.savefig('fig_{}.pdf'.format(disp_name))
plt.close()
```



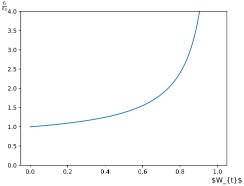
(a) r



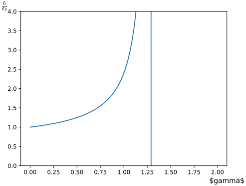
(b) α



(c) s_t



(d) W_t



(e) γ

Figure 2: Sensitivity of Results to Parameter Values