Tenth exercise class

Class 5

Introduction to numerical programming and analysis

Asker Nygaard Christensen

Spring 2021

Plan

1. Data project

2. Problem set 6

3. P. 3, Solow model and root-finders, 16:15

Data project

Problem set 6

My notes:

General tip

When doing *scipy.linalg*, Google is your friend.
For any standard matrix-operation you want to do, there will (likely) be a corresponding *scipy.linalg*-function.

2.1, A: You want to find the <u>det</u>erminant of the <u>inv</u>erse of the dot-product of two matrices. For dot-product you can use *np.dot()* but also the *@*-operator.

2.1, B: It should maybe say using *scipy.linalg*. Note that each equation can be solved seperately.

2.2: For the *gauss_jordan()*-function to work, you need to add e as a 4th column in F.
np.column_stack((F,e)) is the easiest in this case, but there are many others, I'll show you.

Problem 2.3, Symbolic

- Remember to initiate unknown variables in sympy using *sm.symbols()*
- And also notice that you need to use the sympy sine-function (*sm.sin()*)
- For the remaining sympy operators you need, I'd use google. Notice that their tutorial/documenation loads sympy as:
  *from sympy import **
- You can also look through Monday's notebook, which uses all the relevant sympy functions.

P. 3, Solow model and root-finders, 16:15

Problem 3, Solow model

A: Use the answer from above with solving equations symbolically. Notice that the default return of the solver is a list, even when there is only one solution, so you need to extract the first element of this list to get pretty printing.

B: *sm.lambdify((args),f)* is like *lambda args: f*, for symbolic arguments. In this setting you can use your answer to A as *f*.

C: There are multiple ways of using *root_scalar*. 'Brentq'-method, which requires an bounds (called *brackets* for *root_scalar*), 'bisect' is also possible with the same needs.
'newton' method is doesn't need bounds, but does need a first derivative and many more.

D: Using CES-production function is (relatively) easy because we're using Python instead of maths

Solving Solow, an alternative

An alternative to using *root_scalar*, would be to simply simulate the model, I find this to be the most intuitive way. Inserting a $\tilde{k_0} > 0$ in the transition equation:

$$\tilde{k}_{t+1} = \frac{1}{(1+n)(1+g)}[sf(\tilde{k}_t) + (1-\delta)\tilde{k}_t]$$

$$\tilde{k}_1 = \frac{1}{(1+n)(1+g)}[sf(\tilde{k}_0) + (1-\delta)\tilde{k}_0]$$

and iterating from that point to find $\tilde{k_1} \rightarrow \tilde{k_2} \rightarrow \ldots$, will eventually lead (approximately) to the steady state $\tilde{k}^*$. (where $\tilde{k}_t = \tilde{k}_{t+1}$)