What is Quantitative Macro?

Raül Santaeulàlia-Llopis

MOVE-UAB and Barcelona GSE

Fall 2018

1. Introduction
2. The computational experiment
3. Typical steps in the quant experiment
   - Pose the question
   - Choose the model
   - Calibrate your measurement device
   - Run the measurement experiment
4. More on mapping model to data
5. External model validation
6. Some computational basics

Introduction

- In quantitative macro theory we solve and simulate economic models computationally with the idea of mapping the model outcomes to the data. An econ model/theory is a structured story.

- Quantitative theory $>$ qualitative analysis.

  (a) More comprehensive characterization of equilibrium through simulation, one can exhaustively study behavior of the model.
  (b) Not only care about signs, but also about size/measurement. Typical question starts with "How much ... ?"

- Quantitative theory $>$ analytical examples:

  ▷ Use computation to solve for more complex and more relevant special cases: One picks the right point in the parameter space. This implies that we can be very precise in the moments of the data that we care about.

- Recurrent "con": The model can be a very complex animal. It is your job to make sure your audience understand what drives what in your model.

- Computers are changing economics: Dont stay behind the curve.

  "The era of closed-form solutions for their own sake should be over. Newer generations get similar intuitions from computer-generated examples than from functional expressions", Jose-Víctor Ríos-Rull, JME (2008).

The computational experiment

- Lucas' view:

  "Our task as I see it . . . is to write a FORTRAN program that will accept specific economic policy rules as 'input' and will generate as 'output' statistics describing the operating characteristics of time series we care about, which are predicted to result from these policies."

  And how are we to build this *FORTRAN* program?

  "Progress in economic thinking means getting better and better abstract analogue economic models, not better verbal observations about the world."

- Kydland and Prescott view:

  "In a computational experiment, the researcher starts by posing a well-defined quantitative question. Then the researcher uses both theory and measurement to construct a model economy that is a computer representation of a national economy. A model economy consists of households, firms and often a government. The people in the model economy make economic decisions that correspond to those of their counterparts in the real world. The researcher then calibrates the model economy so that it mimics the world along a carefully specified set of dimensions. Finally, the computer is used to run experiments that answer the question"

- Theory only provides restrictions on behavior, not a law of motion for the aggregate state (as in the natural sciences)

- Computational experiments in economics include the additional step of computing the equilibrium process of the aggregate state. Most of the course will focus on this step.

Typical steps in the quant experiment

1. Questions are about measurement, and answers are numbers
2. Key ingredient: a theory of the aggregate economy (an equilibrium model)
3. The model is the measurement device to derive the quantitative implications of the theory
4. The model is calibrated along some dimensions of the data and used to explain other dimensions of the data
5. The computer is used to solve for the equilibrium process and run the computational experiment that answers the question

Step 1: Pose the question

Types of questions:

1. How much can we explain of Y (e.g., fall in C in the Great Recession) with X (e.g., shocks that led to the fall in house prices)?
2. What are the welfare/redistributive implications of policy P? Or, what is the optimal policy P (e.g., degree of tax progressivity)
3. How does the answer to question Q (e.g., shift in the Beveridge curve) change if we introduce the new feature X (e.g., firms choice of recruiting intensity) into the well-known model M (e.g., DMP random matching model) that abstracts from X?

Step 2: Choose the model

1. Start from a well-tested theory. E.g., neoclassical growth model, Aiyagari-Bewley-Huggett-Imrohoroglu model, DMP matching model, NK Model etc.
   Strength: solid ground limit: extent of innovation
2. Less is better than more: do not add features that are irrelevant for the question you are asking. Never lose sight of the question.
3. Think hard about the trade-off between amount of detail and the feasibility of computing the equilibrium process
4. Think hard about whether you have enough data to discipline your model parameters: do not proliferate free parameters!
   Existence of free parameters weakens your conclusion

Step 3: Calibrate your measurement instrument

1. Empirical moments used to calibrate the model are different from the ones the model aims to account for
   - Different frequency: RBC calibrated based on "long-run" growth facts and evaluated on business cycle facts
   - Different moments of distribution: in Aiyagari model, β set to match average wealth (relative to income) and often one tries to account for wealth inequality (higher moments)
   - Different variables: in Aiyagari model, (exogenous) income process is estimated to replicate income dynamics, but objective is to account for wealth inequality
2. Do not justify parameter choices based on prior studies unless mapping between model and data is the same
3. Match measurement to model. Establishing this correspondence may require to re-organize data. E.g., durable C = I

Step 4: Run the measurement experiment

- Lay out the computational algorithm, step by step
- Do not rewrite numerical routines if already available and well tested (but understand how they work, you may need to modify them)
- Test your code as you go along: e.g., solve for special cases you understand well or replicate findings from previous work
- Flesh out your main result as clearly as possible: do not underestimate the importance of well presented tables and graphs
- Sensitivity analysis is crucial when closed form are unavailable.
- If your model is quantitatively unsuccessful, call it a puzzle :-)

More on mapping model to data

1. Calibration = casual empiricism. Often true, but it should not be.
   - Use econometric techniques, e.g., Min Distance Estimators
   - Discuss (even better, prove) parameter identification
2. In the evaluation stage, no formal statistical approach is used
   - True, the logic is: all models are false, with enough data you always reject them. You can still learn from the model though.
3. Why not using all moments instead of a subset?
   - By design, parameter values selected are not the ones that provide the best fit to the data you wish to explain
   - Estimation: what does it take to match the data?
   - Calibration: how much of the data can we explain by restricting the model in a reasonable way?
4. In general, the talk about calibration approach vs. estimation approach is largely fruitless. Instead, time and effort should be devoted to discussing the identification of the chosen parameters, in particular, those that make a difference in the model outcomes (see Ríos-Rull et al., JME 2010).

What if the data are mismeasured? NIPA and intangibles.

- McGrattan and Prescott AER 2010
- It has consequences for the secular behavior of the labor share (see Koh, Santaeulàlia-Llopis and Zheng, 2018)

External model validation (Todd-Wolpin, AER 06)

- In macro we do not run natural experiments to identify effects. However, plenty of (quasi) randomized experiments run on a micro scale.
- Paladins of the experimental approach argue structural models are stylized and parameterized in arbitrary ways, so cannot give reliable policy lessons.
- Todd and Wolpin: use quantitative findings from randomized experiments to validate parameterizations of structural models
- Example: schooling subsidy
- Validated model can be more credibly used for counterfactual or large-scale (e.g, GE) policy analysis
- Complementarity btw structural and experimental approaches
- Experiments are not free of trouble: Lack of GE Effects, Scalability (see Rosenzweig's critique)

Some Computational Basics

1. Having a finite number of bits means we can't represent all possible real numbers.
2. Three types of errors will occur (rounding, approximation, and human). Examples: 0.10, catastrophic cancelation...
3. Routines

If you want more, dig into the lecture notes by Jun Yang at Pittsburg.

The IEEE 754 standard

The IEEE 754 standard defines number representations and operations for floating-point arithmetic.[1] Numbers are stored using a kind of scientific notation.

$$\pm \text{mantissa} * 2^{\text{exponent}}$$

We can represent floating-point numbers with three binary fields:

1. a sign bit s,
2. an exponent field e, and
3. a fraction field f.

[1] The IEEE standard gives an algorithm for addition, subtraction, multiplication, division and square root, and requires that implementations produce the same result as that algorithm. Thus, when a program is moved from one machine to another, the results of the basic operations will be the same in every bit if both machines support the IEEE standard. This greatly simplifies the porting of programs.

The IEEE 754 standard defines several different precisions (i.e., formats).

- Single precision numbers include an 8-bit exponent field and a 23-bit fraction, for a total of 32 bits.
- Double precision numbers have an 11-bit exponent field and a 52-bit fraction, for a total of 64 bits.

That is, in double precision the binary format occupies 64 bits in the computer. There is also quad precision...

Sign

- The sign bit is 0 for positive numbers and 1 for negative numbers.
- But unlike integers, IEEE values are stored in signed magnitude format.

Mantissa

- There are many ways to write a number in scientific notation, but there is always a unique normalized representation, with exactly one non-zero digit to the left of the point.

  Note the diffence:

$$0.232 \times 10^3 = 23.2 \times 10^1 = 2.32 \times 10^2 = ...$$

$$01001 = 1.001 \times 2^3 = ...$$

- What is the normalized representation of 00101101.101 ?

$$00101101.101 = 1.01101101 \times 2^5$$

- Whats the normalized representation of 0.0001101001110 ?

$$0.0001101001110 = 1.110100111 \times 2^{-4}$$

- The field f contains a binary fraction.
- The actual mantissa of the floating-point value is $(1 + f)$.
  - In other words, there is an implicit 1 to the left of the binary point.
  - For example, if f is 01101..., the mantissa would be 1.01101
- A side effect is that we get a little more precision: there are 24 bits in the mantissa, but we only need to store 23 of them.
- But, what about value 0?

Exponent

- The e field represents the exponent as a biased number.
  - It contains the actual exponent plus 127 for single precision, or the actual exponent plus 1023 in double precision.
  - This converts all single-precision exponents from -126 to +127 into unsigned numbers from 1 to 254, and all double-precision exponents from -1022 to +1023 into unsigned numbers from 1 to 2046.
- Two examples:
  - If the exponent is 4, the e field will be 4+127 =131 (10000011_2).
  - If the e contains 01011101 (93_{10}) the actual exponent is 93-127=34
- Storing a biased exponent means we can compare IEEE values as if they were signed integers.

- There are special cases that require encodings (e.g., for infinity and zero)
  - Infinities (overflow)
  - NAN (divide by zero)
- For example:
  - Single-precision: 8 bits in e 256 codes; 11111111 reserved for special cases 255 codes; one code (00000000) for zero 254 codes; need both positive and negative exponents half positives (127), and half negatives (127)
  - Double-precision: 11 bits in e 2048 codes; 1111 reserved for special cases 2047 codes; one code for zero 2046 codes; need both positive and negative exponents half positives (1023), and half negatives (1023)

Three Types of Errors

- Rounding error: depends on the degree of precision in which numbers are stored. A property of hardware and software.
  - Avoid propagation!
  - Judd 1998:
    - avoid subtractions of numbers of similar magnitude;
    - when adding, add first small numbers and then add result to large numbers;
    - avoid multiplying very large with very small numbers (both poorly approximated)
- Approximation error: operations involving an infinite or long finite series which must be approximated by truncating the sequence
- Human error: the most likely one (there is usually a denial phase associated with it, get over this one quickly).

Rounding error

Primarily, rounding errors come from the fact that the infinity of all real numbers cannot possibly be represented by the finite memory of a computer, let alone a tiny slice of memory such as a single floating point variable, so many numbers stored are just approximations of the number they are meant to represent

Given any fixed number of bits, most calculations with real numbers will produce quantities that cannot be exactly represented using that many bits. Therefore the result of a floating-point calculation must often be rounded in order to fit back into its finite representation.

Catastrophic cancellation

Catastrophic cancellation occurs when the operands are subject to rounding errors.

For example in the quadratic formula, the expression $b^2 - 4ac$ occurs. The quantities b^2 and $4ac$ are subject to rounding errors since they are the results of floating-point multiplications. Suppose that they are rounded to the nearest floating-point number, and so are accurate to within .5 ulp. When they are subtracted, cancellation can cause many of the accurate digits to disappear, leaving behind mainly digits contaminated by rounding error. Hence the difference might have an error of many ulps.

For example, consider $b = 3.34$, $a = 1.22$, and $c = 2.28$. The exact value of $b^2 - 4ac$ is .0292. But b^2 rounds to 11.2 and $4ac$ rounds to 11.1, hence the final answer is .1 which is an error by 70 ulps, even though 11.2 - 11.1 is exactly equal to .16. The subtraction did not introduce any error, but rather exposed the error introduced in the earlier multiplications.

0.10

- During the Gulf War in 1991, a U.S. Patriot missile failed to intercept an Iraqi Scud missile, and 28 Americans were killed.
- A later study determined that the problem was caused by the inaccuracy of the binary representation of 0.10.
    - The Patriot incremented a counter once every 0.10 seconds.
    - It multiplied the counter value by 0.10 to compute the actual time.
- However, the (24-bit) binary representation of 0.10 actually corresponds to 0.099999904632568359375, which is off by 0.000000095367431640625.
- This doesnt seem like much, but after 100 hours the time ends up being off by 0.34 secondsenough time for a Scud to travel 500 meters!
- See "Roundoff Error and the Patriot Missile. SIAM News, 25(4):11, July 1992."

A word of caution on pre-exisitng routines

- In practice, we sometimes use existing programs to deal with our computations. This is useful, there are computer scientists and engineers that are definitely better at programming than what we are. But watch out, they do not solve econ models.

- "I cannot emphasize enough how important it is to know what is inside any canned routine that you might use. All manner of mistakes can arise that you would not even be aware of if you do not know how your routine of choice works. It can also be necessary to rewrite (parts of) existing routines, because, e.g., their tolerance criteria are often not sensitive enough." Irina Telyukova

- "Numerical Recipes" has a chapter devoted to optimization. For Fortran, the F77 book has all the details, while the F90 book just updates the code. Many of the methods we discussed, or some variates thereof, are presented in the form of routines there.