An ABM in the box

Ben Klemens

18 July 2013

[Part of a series of posts that started here (entry #146)]

I promised you agent-based models, so here's a model of the demand-side of an economy:

- There are 1,000 agents.
- Each agent i has a budget allocation b_i and preference coefficient α_i, each drawn from independent Normals. We fix $\sigma = 1$ for both distributions, leaving us with two parameters from this part of the model: μ_b and μ_α.
- Two goods are available for purchase; the first has price p and the second price one (this is without loss of generality from a setup with two prices p_1 and p_2).
- Each agent's utility from purchasing the good is $U(q_1, q_2, \alpha) = q_1^\alpha + q_2$. Agents are utility-maximizing.
- Of course, agents can't overspend, so they maximize subject to the constraint that $pq_1 + q_2 \leq b_i$.
- We observe the mean consumption Q_1 and Q_2 (total consumption divided by the number of agents).

We could readily add to this simple framework agent interactions, a more interesting utility function, agent irrationality, or other bits of added realism.

Is this mechanical sequence of events a statistical model? Yes. All of the entries in this series to this point have been aimed at allaying any concerns about this question.

Does it fit the basic form of a model (entry #147)? Yes: given a set of parameters μ_b, μ_α, p_1, and a observed mean consumption Q_1, Q_2, the model specifies the likelihood $L_m(\mu_b, \mu_\alpha, \sigma_{b\alpha}, p, Q_1, Q_2)$.

Can it be wrapped in the same black box as a Normal distribution or regression? Yes: we can use this the same way we use all of the models to this point, making random draws, estimating the parameters given data, transforming (entry #148) or chaining it with other models (entry #150), et cetera. The only restant technical issue is that a likelihood with a built-in random number generator is stochastic, so we have to be careful about consistency (i.e., as draws $\to \infty$, parameter estimates converge), and there are practical difficulties of using a stochastic function (entry #153).

Can this model be tested? Yes, but only in the same way that OLS and its parameters can be tested, meaning that we use the likelihood function asserted by the model (entry #152) to test the odds that statistics are significant.

OLS is much easier to state: as in earlier entries, we could write down the entire model in six symbols, $L_{\mathcal{N}}(Y, X\beta, \sigma)$, versus the text exposition for the model above. But that means that the difficult subtleties are implications, not assumptions. I'd go over them, but there are entire textbooks[1] devoted to the implications of the OLS model. The random-demand model above is more explicit about its moving parts, but it doesn't have bookshelves devoted to probing further implications.

The modeling world tends to break down into social subgroups based on which assumptions are most plausible to whom. Agent-based modelers often find the linearity assumptions underlying OLS to be entirely implausible; people accustomed to simple linear models often suspect that ABMs have hidden assumptions or undesirable and unanticipated characteristics.

I'll have a few more points on this in a few entries.

Meanwhile, here's today's sample code to implement the above model, delivering on the promise that we can make draws, get likelihoods, and estimate parameters. As usual, it's easiest to read the code from the bottom up.

- As is increasingly the pattern, the bulk of the file is fleshing out the elements of the model, and `main` does simple model applications. The application is mostly the same sort of round-trip as past entries, generating a small data set, then estimating the optimal parameters given that data set. The numeric accuracy for this round-trip is really not great; Apophenia 1.0 should improve on it. I use the EM-style strategy of dimension-by-dimension optimization.
- Looking at the `p` function, we see that a single likelihood for a given data/parameter set is found by making 500 draws, then smoothing it using a kernel density estimate in which a Multivariate Normal is placed over every drawn point; a lot of lines of code are about setting up the MVN and wiring up the KDE.
- The core of the model—the part I described above—is a single draw of a population and its decisions. That's the `draw` function. The first several lines shunt parameters, then we draw the agents, then the next loop does the optimization for each agent. There are many ways to make this more streamlined, but I wanted to keep it ABM-like. For example, we might want to have an interaction step between generating the population and their consumption decision, so I left those as separate loops.
- Given the problem

  $$\max u = q_1^{\alpha} + q_2 \ni b = p_1 q_1 + q_2$$

  the optimum, where $\partial u / \partial q_1 = 0$ and the budget constraint is satisfied, is

  $$q_1 = (p_1/\alpha)^{1/(1-\alpha)} q_2 = \max(b - p_1 q_1, 0)$$

[1] `http://www.amazon.com/exec/obidos/tg/detail/-/1405182571/qid=1120157199/sr=8-1/ref=pd_bbs_ur_1?v=glance&s=books&n=507846`

```
#include <apop.h>

typedef struct {
    double b, alpha, q1, q2;
} an_agent;

void draw(double *qs, gsl_rng *r, apop_model *m){
    double m1 = apop_data_get(m->parameters, 0);
    double m2 = apop_data_get(m->parameters, 1);
    double p1 = apop_data_get(m->parameters, 2);

    apop_model *ba_model = apop_model_stack(
            apop_model_set_parameters(apop_normal, m1, 1),
            apop_model_set_parameters(apop_normal, m2, 1)
            ); //leaks; don't care.

    //set up agents
    int agent_count=1000;
    an_agent a_list[agent_count];
    for(int i=0; i< agent_count; i++){
        double out[2];
        do {
            apop_draw(out, r, ba_model);
            a_list[i] = (an_agent){.b=out[0], .alpha=out[1]};
        } while (a_list[i].alpha <=0);
    }

    //agents decide
    qs[0]=qs[1]=0;
    for (int i=0; i< agent_count; i++){
        qs[0] += a_list[i].q1 = GSL_MIN(
                        pow(p1/a_list[i].alpha, 1./(1-a_list[i].alpha)),
                        a_list[i].b/p1);
        qs[1] += a_list[i].q2 = (a_list[i].b - p1*a_list[i].q1);
        //printf("%g %g\n", a_list[i].q1,a_list[i].q2);
    }
    qs[0] /= agent_count;
    qs[1] /= agent_count;
    apop_model_free(ba_model);
}

//for the Kernel density: center a Uniform distribution around a datum
void set_fn(apop_data *d, apop_model *m){
    Apop_matrix_row(d->matrix, 0, onerow);
    gsl_vector_memcpy(m->parameters->vector, onerow);
}

long double p(apop_data *d, apop_model *m){
    apop_multivariate_normal.vsize =
    apop_multivariate_normal.msize1=
    apop_multivariate_normal.msize2= 2;
    apop_model *kernel = apop_model_set_parameters(apop_multivariate_normal,
            0, 1, 0,
            0, 0, 1);

    apop_model *smoothed = apop_model_copy_set(apop_kernel_density, apop_kernel_density
            ,
            .base_data = apop_model_draws(m, 500), .kernel=kernel, .set_fn=set_fn);
    double out = apop_p(d, smoothed);
    apop_data_free(smoothed->data);
```