Supplemental Appendixes, Agent-Based Modeling in Economics and Finance: Past, Present, and Future by Robert L. Axtell and J. Doyne Farmer

Appendix 1: Meanings of Acronyms Mentioned

ABE: agent-based economics
ABM: agent-based modeling or agent-based model
ACE: agent-based computational economics
AI: artificial intelligence
ALife: artificial life
BR: bounded rationality
CA: cellular automaton or automata
CAS: complex adaptive system
CDA: continuous double auction
CES: constant elasticity of substitution
CGE: computable general equilibrium model
CRISIS: European Union funded project to model the Financial Crisis with *ABM*
DAI: distributed artificial intelligence
DSGE: dynamic stochastic general equilibrium model of macroeconomics
EU: European Union
EWA: experience-weighted attraction, an empirically-grounded learning algorithm
FFRDC: Federally-funded research and development corporation
GARCH: generalized autoregressive conditional heteroskedasticity
GFDL: Geophysical Fluid Dynamics Laboratory at Princeton University
GIS: geographic information systems
GSIA: Graduate School of Industrial Administration at the Carnegie Institute of Technology; today: Tepper School of Business at Carnegie-Mellon University
IBM: individual-based model
LANL: Los Alamos National Laboratory
LFN: labor flow network
MAS: multi-agent systems
MERS: Middle East Respiratory Sickness
MIDAS: Models of Infectious Disease Agent Study at *NIH*
MLS: Multiple listing service, a real estate firm and data aggregator
MRS: marginal rate of substitution of one good for another
NASDAQ: National Association of Securities Dealers Automated Quotations
NCAR: National Center for Atmospheric Research
NIH: National Institutes of Health
NOAA: National Oceanic and Atmospheric Administration
NSF: National Science Foundation
NWS: National Weather Service
OFR: Office of Financial Research within the Department of Treasury
OR: operations research
REE: rational expectations equilibrium/equilibria
SARS: severe acute respiratory syndrome
SBE: Social, Behavioral & Economic Sciences Directorate at *NSF*
SD: system dynamics, modeling approach pioneered by Jay Forrester at MIT

SEC: U.S. Securities and Exchange Commission
SES: Social and Economic Sciences Division at *NSF*
SNA: social network analysis
SOES: Small Order Execution System on the *NASDAQ*
UCAR: University Consortium for Atmospheric Research
V&V: verification and validation
VaR: value at risk
WMAD: Walras-McKenzie-Arrow-Debreu model of general equilibrium
ZI: zero-intelligence, trading agents who act purposively but without an internal model
ZIP: zero-intelligence plus trading agents

Appendix 2: Computer Terms, Languages, and Systems Discussed

ABM: agent-based model or agent-based modeling
ACT-R: computational cognitive architecture
AgentSheets: simple, user-friendly *ABM* software environment
ASCII: American Standard Code for Information Interchange, for character encoding
BASIC: early programming language, little used today
BDI: belief-desires-intentions representation of agent behavior, popular in *MAS*
C: early low-level programming language, still in wide use today
C++: object-oriented version of *C*
C#: object-oriented programming language from Microsoft
CLARION: computational cognitive architecture
CMIP: Coupled Model Intercomparison Project
CORMAS: *ABM* software commonly used for natural resource models
CPU: central processing unit
DP: dynamic programming, pioneered by Richard Bellman in the 1950scu
DSGE: dynamic stochastic general equilibrium model of macroeconomics
EINSTEIN: combat modeling toolkit
EPISIMS: epidemic simulation code derived from TRANSIMS at Los Alamos
EurACE: agent-based macroeconomic model in use in Europe for research and policy
FLAME: *ABM* software environment for running models on *GPU*s
FLOPS: floating point operations per second
FORTRAN: early programming language, still in use today for scientific computing
GAMS: General Algebraic Modeling Systems
GEMS: General Electric modeling and simulation language
GPSS: general purpose simulation system
GPU: graphics processing unit
HPC: high-performance computing
ISAAC: Irreducible, Semi-Autonomous Adaptive Combat model, early military *ABM*
JABOWA: early forest simulation system in *IBM* ecology
Java: *OOP* language originally created by Sun Microsystems, currently owned by Oracle
MABM: macroeconomic *ABM*
MASON: *ABM* software framework in Java from George Mason University
Mathematica: commercial mathematics software and programming package
MATLAB: commercial software package
MESA: *ABM* software framework in Python
NetLogo: popular *ABM* environment requiring modest programming background
NP: complexity class of problems solvable nondeterministically in polynomial time
Objective-C: early object-oriented programming language, still in use at Apple
ODD: protocol for reporting *ABM*s
OOP: object-oriented programming
P: complexity class of problems solvable in polynomial time
PAC: probably approximately correct learning, a learning algorithm
Pascal: programming language created at ETH Zurich in the 1970s, litte used today
PPA/*PPAD*: complexity classes between *P* and *NP*; polynomial parity argument on either undirected or directed graphs, respectively
RAM: random access memory

RePast and RePast HPC: open source *ABM* software framework in Java, C++, and C# from Argonne National Laboratory
RePastPy: RePast based on Python
RNG: random number generator
SimScript: early simulation language, still in use today
SIMULA: the first OOP language and a family of simulation languages
SmallTalk: early object-oriented programming language, in little use today
SOAR: early computational cognitive architecture
StarLogo: early programming language for beginners from MIT
Sugarscape: early *ABM* in which agents forage for resources and engage in exchange
SWARM: early agent-based modeling language
TRANSIMS: transportation simulation code created at Los Alamos

Appendix 3: Implementation of ABMs

Creating an *ABM* involves some amount of computer programming, so a researcher's ability to effectively utilize this new approach is often proportional to one's computing skills. But no very specific computational background is required, since *ABM*s can be created in a wide variety of ways. While courses in algorithms and data structures are helpful, the most important skill to possess for creating an *ABM* is strong command of some specific programming language, such as Java, Python, C/C++, C#, and so on. By far the most common question people have who are new to *ABM* is 'What software should I use to build my model?' This question has many facets and picking the wrong software for a project can be disastrous. Here we provide some guidelines based on current technology. Happily, there are good comparisons of existing software packages—Kravari and Bassiliades (2015), supplementing older ones of Gilbert and Bankes (2002) and Dibble (2006)—meaning we can be brief, editorializing a bit based on our experience.

There are essentially four distinct ways to create an *ABM*, (1) code in a native programming language like *Java* or *Python*, (2) write your model in a mathematical or statistical environment like *MatLab*, *R*, or *Mathematica*, (3) code your model using a software framework for *ABM* like *RePast*, *MASON*, *AnyLogic, FLAME*, or *MESA*, or (4) create your model in a high-level, *ABM*-specific software environment like *NetLogo* or *HashAI*. Each of these systems has advantages and disadvantages, so picking one involves trade-offs. Specifically, the *lower* the number on our list the faster your model will probably run, eventually, once it is successfully coded and debugged. However, the coding and debugging time typically declines as the number on our list gets *higher*. For example, native *Java* code is going to run much faster than *NetLogo* code but it might take you significantly longer (2-10x) to get a non-trivial model up and running in *Java* than in *NetLogo*. Empirically, many *ABM*s used for research in economics are built in *NetLogo*, *MASON*, *RePast*, or *Python*. These are each mature systems with sizable user bases, reasonable documentation, and performance good enough to use for research. In finance it is probably the case that more than half of all *ABMs* are created in *MatLab*. This is because that system is designed for high-performance numerical computation and is especially suitable for solving equations—agents in finance *ABM*s often have to solve portfolio optimization, arbitrage, and *oher* mathematical problems in determining how to behave. We summarize the characteristics and performance of several of these systems in table A1. Software systems less often used for *ABM* research these days include SWARM (Minar *et al*., 1996, Terna, 1998, Luna and Stefansson, 2000, Stefansson, 2000), CORMAS (LePage *et al*., 2000), and AgentSheets (Repenning, Ioannidou and Zola, 2000), and we will not say more about these here. For the entries in the table we provide a few points of description and perhaps one or more references to the literature. *NetLogo* (Wilensky and Rand, 2015) combines a programming language (having hybrid *OOP* and functional features) with an highly configurable development and analysis environment. It is excellent for rapid-prototyping but too slow for very large models. *MASON* (Luke *et al*., 2005) is based on Java and requires users to code in that language. It has easy-to-use analysis and visualization interfaces. *RePast* (North, Collier and Vos, 2006) users typically code their model in *Java*, *C#* or *C++*. It has many features in common with *MASON*. *RepastPy* is a version based on Python. *MatLab* has good performance. In some of these software frameworks *ABM*s can be written *very* compactly. For instance, Gaylord and D'Andria (1998) have programmed the Schelling model in 5 lines of *Mathematica* code!

Software	***OOP?***	***Programming***	***Compiled?***	***Animations?***	***Speed***	***Max agents***
NetLogo	yes	own language	no	yes	slow	100K
MASON	yes	*Java*	byte code	yes	good	millions
RePast	yes	*Java*, *C#*, *C++*	byte code	yes	good	millions
AnyLogic	yes	*Java*	byte code	yes	good	millions
Mesa	yes	Python	yes	yes	good	millions
MatLab	optional	own language	can be	yes	good	millions

Table A1*: Comparison of several software environments for creating* ABM*s*

A newer approach to *ABM* deserving of brief mention is through programming the video boards that are part of all modern microcomputers. Graphics processing units (*GPUs*) have greatly improved their performance. D'Souza, Lysenko and Rahmani (2007) programmed the *Sugarscape* model to run 1,000,000 agents at 25 frames/second while only a few hundred agents could run at that speed when that model was first created (Epstein and Axtell, 1996). *FLAME* is an *ABM* programming environment designed specifically for *GPUs* (Kiran *et al*., 2010).