Open Source, Agent-based Energy Market Simulation with Python

Richard W. Lincoln, *Student Member, IEEE* Stuart Galloway, Graeme Burt, *Member, IEEE*

***Abstract*—Increasingly, the electric energy transmitted and distributed by national power systems is traded competitively in free markets. Long-term decisions must be made by authorities as to the structure of energy markets and the regulations that govern interactions between participants. It is not practical to experiment with real energy markets and in order to establish the potential effects of making these decisions there are few options but to simulate the markets computationally. This paper proposes that the complexity of power systems and the associated energy markets necessitates an open approach in their modelling and simulation. It presents an open source software package for simulating electric energy markets using the Python programming language. Power systems and their associated constraints are modelled using traditional steady-state analysis techniques. While market participants are represented by reactive agents that learn through reinforcement. The software and all of its dependencies are open and freely available to the scientific community.**

***Index Terms*—Steady-state simulation, Energy markets, Agent-based simulation, Open source software, Reinforcement learning, AC Optimal Power Flow**

I. Introduction

ELECTRIC energy use pervades almost every aspect of life in an industrialised society and as such a myriad of factors influence the balance of demand and supply. Both technically and operationally, no practical inroads have yet to be made in forecasting market prices. Nevertheless, electricity is traded competitively on a national scale on a day to day basis. Large sums of money change hands in the world's energy markets and the benefits of competitive trade to society can be great. Small improvements in market design or strategic participation can benefit social welfare and conversely so.

Relative to most other commodities, trade of electric energy is still in its infancy. When competition was first introduced, the unique nature of electricity as a commodity raised many doubts as the whether such a vital resource for modern societies could be reliably traded in the free market. Important decisions were made when the trading arrangements for early energy markets were first decided upon and had to be supported through research where possible[1]. Liberalisation and unbundling of electricity supply industries costs many millions of pounds to implement. Countries, having made this investment, continue to restructure and adjust their energy markets in the hope of further reducing costs to the consumer and promoting innovation and efficiency through competition.

Electric energy is key to the operation of industrialised societies. It is not practical to experiment with real energy markets and in order to establish the potential effects of particular changes to rules and regulations there are few options but to attempt to simulate them computationally.

Game theoretic models are commonly associated with economics and attempt to capture behaviour in strategic situations mathematically. They have been applied to electric energy problems of many forms, including but not limited to analysis of market structure, market liquidity, pricing methodologies, regulatory structure, plant positioning and network congestion. More recently, agent-based simulation has received a certain degree of attention from researchers and has been applied in some of these fields also.

While popular and seemingly promising, agent-based simulation is still centred around abstracted models. The assumptions made is this abstraction must be subjected to the same verification and validation as with equation-based models. Verification of assumptions and model validation are often overlooked in agent-based simulations of energy markets, yet they are possibly the most important steps in the model building process. Techniques used to develop, debug and maintain large computer programs can often be used to verify that a model does what it is intended to do.

Validation of an energy market model is more difficult. It can be accomplished using the intuition of experts or through comparison of simulation results with either historical market data or theoretical results from more abstract representations of the model. Finding verifyable trends in existing markets is a very large challenge. To then prove that a computational model replicates these characteristics with suitable fidelity is yet more challenging still. Only when a model is suitably verified and validated can any conclusions be drawn from results obtained through implementation and simulation of suitable scenarios.

To accelerate knowledge and understanding in this important and challenging field, the scientific community requires a more *open* approach. Possibly for commercial reasons, all but one[2] of the recently developed energy market simulation programs remains closed source. While the features of these simulators and the methods used in implementing them are documented to some extent in their associated publications, no access is provided to the source code for the software. As with all research related to closed source software projects, this prevents third parties from scrutinising the intricacies of the implementation and building directly upon previous effort.

This paper presents an agent-based energy market simulation package developed as an extension to an existing power systems analysis package. It is written in a general purpose, high-level programming language and has been released under version two of the GNU General Public License. It is intended

R. W. Lincoln, S. Galloway and G. Burt are with the Department of Electronic and Electrical Engineering, The University of Strathclyde, Glasgow, Scotland, G1 1XW UK e-mail: rlincoln@eee.strath.ac.uk.

to be a free and open research tool for the simulation of energy markets that pays due consideration to the technical constraints imposed by transmission and distribution networks. It was developed by members of the Engineering and Physical Research Council (EPSRC) funded SUPERGEN – Highly Distributed Power Systems consortium to study future mechanisms of energy trade.

The flow of electric charge has many associated phenomena and the means by which their fundamental influence on the energy trade has been accounted for begins in the following section. Use of the Python programming language is an important aspect of this work and the reasoning behind its selection is explained in Section III. The simulator makes use of a modular machine learning library, the structure and origins of which are explained in Section IV. Section V then shows how this framework has been built upon to model trade of electric energy.

II. Power System Simulation

High voltage transmission and distribution networks are the mechanisms by which traded electric energy is delivered. Limits to line and cable power flows and the availability of reactive power can constrain particular trades. As such, they are fundamental to energy market operation and must be duly accounted for in a model thereof. Steady-state analysis of power systems is a mature field of Power Engineering. Many tools are available for calculating flows of energy and voltage levels around balanced three-phase networks.

To maintain the open nature of Pyreto, the importance of which having been explained in Section I, it has been built as an extension to the power system simulator, Pylon. Pylon is a software project that takes an entirely open approach to power system analysis and has also been developed by members of the SUPERGEN Highly Distributed Power Systems consortium from The University of Strathclyde. It is largely a translation of MATPOWER[3] and PSAT[4] to the Python programming language and boasts routines for solving linearised and non-linear power flow and optimal power flow (OPF) problems.

Pyreto utilises the robust OPF routines of Pylon to optimise the despatch of generation according to their cost while ensuring that voltage and line flow constraints are met. Full details of the interface between the multi-agent energy market model and the Pylon network model are provided in Section V.

III. Python Programming Language

The C, C++ and Java programming languages have been selected for several recent power engineering software projects[5], [2], [6], [7] and were considered for Pyreto also. While an equivalent Java/C/C++ program will typically execute faster and use less memory than a Python program (particularly so with C/C++), this speed comes at the cost of greatly increased complexity.

Python is an important programming language for Scientific and Engineering computing. It, like MATLAB®, offers a layer of abstraction to the programmer that hides certain low-level details that, while very important in the theories of Computer Science, need not be of daily concern to Engineers.

By way of an example, Java and C are statically typed programming languages. When a new variable is declared the type of the variable must be specified and generally can not subsequently be changed. This allows specific areas of memory to be allocated and for type checking to be avoided each time the variable is used. For Engineers interested in solving problems and creating prototypes quickly there is value in the flexibility and lack of syntactic clutter offered by dynamically typed languages such as Python. The use of whitespace indentation, rather than parenthesis, to separate statement blocks, duck typing and operator overloading are other good examples of how Python simplifies program code for relatively inexperienced programmers.

Many other popular programming languages also use dynamic typing (Perl, Tcl) and exhibit relatively clean syntax (Ruby), but none have the universality of Python. Not only is there a well established set of Python libraries for mathematical computing (Numpy, Scipy, CVXOPT etc.), but also a broad set of non-numerical libraries for networking, GUI development, manipulation of graphs structures, rendering of three-dimensional plots etc. This provides Engineers writing software in Python many equivalents to the features offerred by proprietary applications such as MATLAB.

Pylon and Pyreto are part of an effort to make Power Engineering software tools free, open and extensible. The Python programming language itself follows an open, community-driven development model that is managed by a non-profit foundation. Few other programming languages exhibit this level of openness, offering users of the language the chance to influence the design of the language itself.

A. Graphical User Interface

Managing large amounts of data, typical to an energy market study, can be cumbersome and error prone through text file manipulation and command line use. Table editors can present data in familiar spreadsheet formats and tree editors allow hierarchical data structures to be visualised. Moreover, Engineering often involves processing structured data that may be represented with graphs. A Graphical User Interface (GUI) can offer the opportunity to visualise such data and the contined inter-relationships. Furthermore, two and three dimensional plotting utilities can provide a powerful means by which to interact with both input and output data for simulations.

1) GUI Toolkits: Python boasts bindings to many stable and highly popular cross-platform GUI toolkits (wxPython, PyQT, PyGTK etc.). The tookits provide large libraries of widgets (controls that display an information arrangement changeable by the user) that may be positioned, sized, arranged and styled according to the developers preference. However, maintaining a GUI built with such toolkits can require a lot of effort if frequent changes are made to the underlying data model. Effort that would likely be better focussed on solving Engineering problems.

Another issue related to GUIs for Engineering applications is the selection of the most appropriate toolkit. The function-

ality offered by the most popular toolkits is fairly generic. It is often not practical to maintain a GUI using more than one toolkit and a particular selection can impact upon integration with other software applications as development progresses.

2) Application Frameworks: To formalise development of a software project and ease maintenance and extension it can be built on an application framework. Frameworks do little on their own, but can provide a basis for programmes of many types. They define how different elements of the application are linked and permit resuse of common components. Python boasts many such frameworks for web application development (Zope, Django, TurboGears, CherryPy, Pylons etc.). While web applications benefit from the ubiquity of clients (web browsers) and the ability to be updated without the need to distribute and install new code, they typically require more advanced programming skills to develop and do not offer the same visualisation functionality as desktop applications.

Fewer frameworks for desktop application development exist, but they allow feature and graphically rich applications to be built and maintained with relative ease. Envisage is a framework that is one of several open source projects intended to accelerate the development of scientific applications included in the Enthought Tool Suite (ETS). The core of Envisage defines the plug-in and extension mechanisms that form the basic application architecture. Plug-ins define extension points, make contributions to the extension points of other plug-ins and register services (shared application objects). Using these simple concepts Envisage can be used to build powerful and scalable programs of any type from separable components.

A selection of useful plug-ins are provided with the ETS, including a workbench plug-in. This provides an application window with areas for *views* and *editors* that are arranged according to *perspectives*. The concept of having a central editing area, in which attention is primarily focussed, with surrounding views, providing supplementary information, is similar the that employed by the Eclipse Integrated Development Environment. The workbench plug-in offers several extension points including one to which new views can be contributed. ETS plug-ins that contribute to this currently include an interactive Python interpreter, text editors and a logging plug-in.

IV. Machine Learning Library

Reinforcement learning is a sub-area of machine learning and can be applied to a wide variety of problems[8]. To allow the same learning algorithms developed for traditional, academic reinforcement learning problems (chess, backgammon, lift scheduling etc.) to be applied to models of energy markets (and vice versa) Pyreto utilises a modular machine learning library called PyBrain (Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library).

Central to PyBrain are artificial neural network models. The extensive libraries for reinforcement learning, for unsupervised learning and evolution, provided with PyBrain, accept these networks and may be configured to train them. The use of artificial neural networks in PyBrain as function approximators allows for the library to be applied to complex, noisy and partially observable problems with continuous state and action spaces.

PyBrain has been developed by researchers at the Dalle Molle Institute for Artificial Intelligence, Switzerland and the Technische Universität München, Germany. It has been released under a highly permissive BSD (Berkley Software Distribution) license and benefits from a friendly and supportive surrounding community.

A. Reinforcement Learning

Typical to reinforcement learning problems, PyBrain uses the notions of *Agent*, and *Environment*. An agent being defined, in simplest sense, as an object capable of producing actions according to previous observations. The environment is the agent's *universe*, in which it may choose to perform these actions.

In a reinforcement learning problem, PyBrain employs the concept of a *Task* to link an agent and its environment. An environment in PyBrain may return observations which are indicative of its state. It is the task that associates a purpose with the environment and may provide an agent with the reward signal required for reinforcement learning. The task may filter the observations from the environment and restrict that which is visible to the agent. Similarly, the task may make adjustments to actions before they are transmitted to the environment.

For learning problems, PyBrain generalises the base `Agent` class and defines a `HistoryAgent` that may stores actions, states and rewards in a `ReinforcementDataSet`. This class itself is generalised by the `LearningAgent` which has an associations with a controller/module and a learner. The controller maps the current state to an action and is typically a `Network` from PyBrain's `structure` package.

Simulations are coordinated by subclasses of `Experiment`, that associate agents with tasks and handle their interactions.

V. Pyreto

Pyreto is a subpackage of Pylon that defines some generalisations of PyBrain classes for simulating competitive trade of electric energy. A simulation may be performed by instantiating a `MarketExperiment` and calling the `doInteractions()` method. The experiment takes a list of agents, a list of tasks and a power system model on instantiation. The market experiment synchronises:

1) integrating observations of the environment into agents,
2) getting actions from the agents and applying them to their task,
3) getting rewards from the task and giving them to the agents, and
4) instructing the agents to learn from their actions.

Fig. 1 illustrates the hierarchy of agent classes and references to the power system model that the `ParticipantEnvironment` uses in getting samples and performing actions. The experiment class solves the optimal power flow problem after the actions of the agents

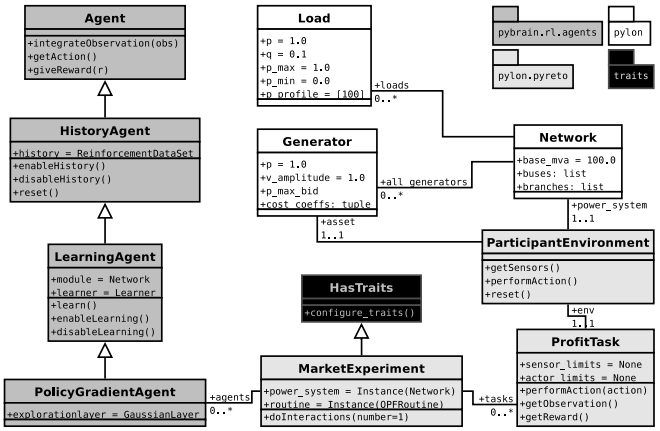

Fig. 1. Class diagram illustrating the link between agent and task, handled by the MarketExperiment.

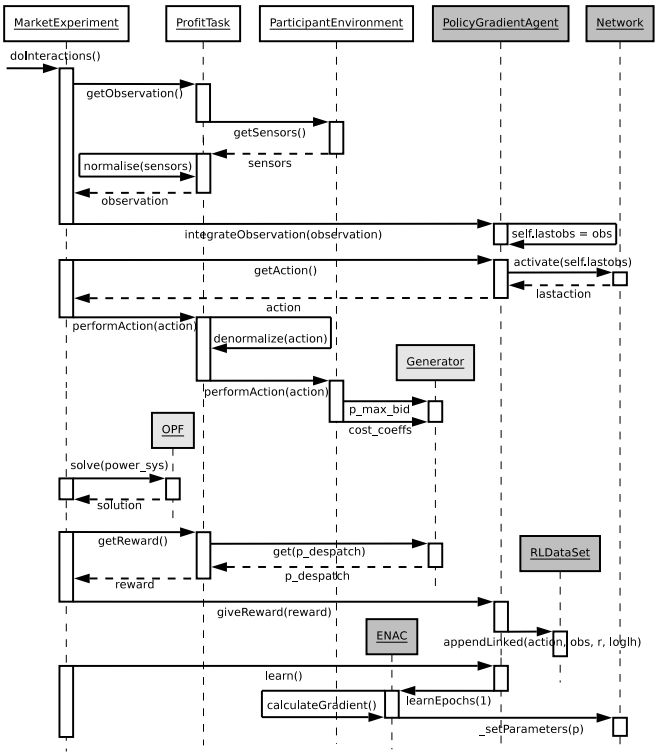

Fig. 2. Sequence diagram for one interaction of a MarketExperiment.

have been performed on the assets in their environment. The power at which the generator has been despatched and the clearing price for the market are then used to produce the reward value, as illustrated in Fig. 2.

For convenience, a `simulate_trade()` function is supplied with Pyreto that accepts a `pylon.Network` and a integer number of interactions. For each `Generator` in the network this function will instantiate a `ParticipantEnvironment` and set the appropriate reference (See Section V-A). It will handle instantiation of a `ProfitTask` and an agent for each environment. Providing the agent with appropriate controller and learner instances. A `MarketExperiment` instance is finally activated, performing any specified number of interactions.

A. Environment

For the sake of simplicity, each agent is currently responsible for one `Generator` and a reference to one is required to instantiate a `ParticipantEnvironment`.

The `.getSensors()` method of the environment returns the state as a NumPy array of doubles. The state is currently defined using the demand for the current period, the clearing price for the current period and (forecast) demand for the successive period. This state is returned to the task when the method is called and filtered appropriately before being use by the agent.

The `.performAction()` method of the environment accepts an action as a numpy array of doubles. This argument currently consists of two scalar values that are assigned to the `p_max_bid` and the `price_coeffs` attributes of the generator. `p_max_bid` represents the maximum real power output that should be used by the optimal power flow routine. Not to be confused with the maximum rated output of the generator (`p_max`). The second value is assigned to the second coefficient for the quadratic price curve, that specifies the linear relationship between output and price.

B. Task

The task of each agent is to maximise the profit that it receives for operating its asset. The `ProfitTask` class defines this by overriding the `.getReward()` method from `Task` such that it returns the product of the real power output, at which the asset has been despatched, and the market clearing price from the optimal power flow routine.

It also handles denormalizing the action values such that they scale from being between -1 and 1 to between the maximum (`p_max`) and minimum (`p_min`) real power ratings of the asset.

C. Agent & Learner

Pyreto has no specific agent implementation, but makes use of those included in `pybrain.rl.agents` package. The `ParticipantEnvironment` defines *continuous* state and action spaces and thus requires a policy gradient agent. The `PolicyGradientAgent` adds a Gaussian layer to the top of its controller using an `IdentityConnection` and adds a field to its dataset for storage of *log likelihoods*.

Several learners are supplied with PyBrain that when used with the `PolicyGradientAgent` change the weights of the Gaussian and other layers according to a certain gradient descent. The Episodic Natural Actor-Critic (ENAC) learner estimates a natural gradient with regression of log likelihoods to rewards[9] and is used by default.

D. User Interface

The `ExperimentPlot` class provides a view of a vertical plot container that houses plots for state, action and reward. These are interactive plots with panning, zooming and data point selection. They may be saved to PNG or PDF format independently. The experiment plot view may be used with an `ExperimentViewModel` that handles a view of the

experiment while providing generic persistence actions (Open, Save, Save As), undo and redo actions plus logging views.

A plug-in for the Envisage framework is included in the `pyreto` package. It depends on a plug-in for Pylon and either the DC-OPF or the AC-OPF routine. The plug-in may optionally contribute to the resource plug-in an editor and a new resource wizard. The resource plug-in is used with the workbench plug-in to provide an adaptable and extensible application interface centred around a workspace of objects persisted (pickled) to the file system.

VI. Conclusion

The need for greater understanding of the operation of our energy markets has been established and an open, collaborative simulation project proposed. The project addresses technical aspects of the problem through use of linearised or non-linear OPF routines. Competition between market participants is modeled using well established reinforcement learning techniques with artificial neural networks being employed to allow agents to handle continuous state and action spaces. The structure and the sequence of operation of a market experiment that coordinates actions on power system components and the results of a custom optimal power flow routine have been described and illustrated in detail. The project is being used by members of the EPSRC funded SUPERGEN – Highly Distributed Power Systems consortium to investigate congestion management techniques in future power systems.

Acknowledgment

The authors would like to thank the United Kingdom Engineering and Physical Research Council for funding this work under grant GR/T28836/01.

References

[1] D. W. Bunn and F. S. Oliveira, "Agent-based simulation–an application to the new electricity trading arrangements of england and wales," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 5, pp. 493–503, October 2001.

[2] J. Sun and L. Tesfatsion, "Dynamic testing of wholesale power market designs: An open-source agent-based framework," *Computational Economics*, vol. 30, no. 3, pp. 291–327, 2007.

[3] H. Wang, C. Murillo-Sanchez, R. Zimmerman, and R. Thomas, "On computational issues of market-based optimal power flow," *Power Systems, IEEE Transactions on*, vol. 22, no. 3, pp. 1185–1193, Aug. 2007.

[4] F. Milano, "An open source power system analysis toolbox," *Power Systems, IEEE Transactions on*, vol. 20, no. 3, pp. 1199–1206, Aug. 2005.

[5] E. Zhou, "Object-oriented programming, C++ and power system simulation," *Power Systems, IEEE Transactions on*, vol. 11, no. 1, pp. 206–215, Feb 1996.

[6] J. Sun and L. Tesfatsion, "Open-source software for power industry research, teaching, and training: A DC-OPF illustration," in *IEEE Power Engineering Systems General Meeting Proceedings*, Tampa, Florida, June 2007.

[7] C. A. Cañizares and F. L. Alvarado, "Uwpflow, continuation and direct methods to locate fold bifurcations in ac/dc/facts power systems," online, 1999, available at http://www.power.uwaterloo.ca.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning, An Introduction*. The MIT Press, 1998.

[9] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7-9, pp. 1180–1190, 2008.

Richard W. Lincoln Richard W. Lincoln was born in Edinburgh in 1980. He received the M.Eng. degree in Electrical and Mechanical Engineering from The University of Edinburgh in 2005. Presently, he is a final year postgraduate at The University of Strathclyde in Glasgow. His areas of interest include simulation and visualisation of highly distributed electric power systems and energy markets.

Stuart Galloway Stuart Galloway is currently a lecturer in the Institute for Energy and Environment at the University of Strathclyde. He was initially appointed as a Rolls-Royce Senior Research Fellow focusing on novel distributed generation control and electricity market trading problems. Prior to this he undertook doctoral research in applied mathematics at the University of Edinburgh. His current research interests include the application of optimisation techniques to power engineering problems, the modelling of novel electrical power systems and market simulation.

Graeme Burt Graeme Burt received his BEng in Electrical and Electronic Engineering from the University of Strathclyde in 1988. He received his PhD also from the University of Strathclyde in 1992. He is currently a Professor at the University of Strathclyde, and is Director of the University Technology Centre in electrical power systems, sponsored by Rolls Royce. His research interests lie predominantly in the areas of power system modelling and simulation, power system protection, and network integration of distributed generation.