

# A Python-based Software Tool for Power System Analysis

Federico Milano, *Senior Member, IEEE*

**Abstract**—This paper presents a power system analysis tool, called DOME, entirely based on Python scripting language as well as on public domain efficient C and Fortran libraries. The objects of the paper are twofold. First, the paper discusses the features that makes the Python language an adequate tool for research, massive numerical simulations and education. Then the paper describes the architecture of the developed software tool and provides a variety of examples to show the advanced features and the performance of the developed tool.

**Index Terms**—Python language, scripting, power flow analysis, eigenvalue analysis, time domain integration.

## I. INTRODUCTION

THE basic requirements that a programming language has to satisfy to be eligible for scientific studies and, in particular, for power system analysis, are the availability of efficient and easy-to-use libraries for:

- Basic mathematical functions (e.g., trigonometric functions and complex numbers).
- Multi-dimensional arrays (e.g., element by element operations and slicing).
- Sparse matrices and linear algebra (e.g., sparse complete LU factorization).
- Eigenvalue analysis of non-symmetrical matrices.
- Advanced and publishing-quality plots.

The requirements above reduce the choice to a only a handful of programming languages. This is clearly captured by the software tools for power system analysis that are currently actively developed. Neglecting proprietary software tools, that are out of the scope of this paper, open-source packages are mainly developed in scientific languages, e.g., Matlab, and in well-structured system languages, e.g., Java. An in-depth discussion about advantages and drawbacks of scientific and system programming languages is given in [1].

Among Matlab-based packages there are PST [2], PSAT [3], and MatPower [4]. Among packages based on system programming languages there are InterPSS [5], developed in Java, OpenDSS [6], developed in Delphi, and UWPFLOW [7], developed in C. In recent years, the Python language have been chosen by some small project such as PYPOWER [8], which is a port of MatPower to Python, and minpower [9]. A comprehensive description of open-source software packages for power system analysis can be found in [10].

The main object of this paper is to show that the Python language is mature enough for power system analysis. Moreover, the paper discusses how Python can be extended by means of open-source scientific libraries to provide a performance

comparable to proprietary solutions. Specific contributions of the paper are:

- 1) To show that the Python language is an adequate tool for power system analysis studies.
- 2) To describe the architecture of the developed Python-based software tool, namely DOME.
- 3) To provide a comparison through power system simulations of the performance of some open-source scientific libraries.

The paper is organized as follows. Section II provides the rationale for using Python as the main language for a power system analysis tool. Section III provides a description of the developed Python-based software tool. Section IV provides a variety of examples aimed to compare the performances of efficient open-source mathematical libraries. Conclusions and future work are indicated in Section V.

## II. PYTHON AS A SCRIPTING LANGUAGE FOR POWER SYSTEM ANALYSIS

Python is a dynamically and safely typed language. Polymorphism, meta-programming, introspection and lazy evaluation are easy to implement and to use. Parallel programming, such as multithreading, concurrency and multiprocessing, is also possible even if with some limitations. The interested reader can find an in-depth discussion on types in [11] while an introduction to parallel programming techniques is [12]. Other concepts cited above are defined and extensively discussed in [13]. Moreover, [14] provides a monograph on power system modeling and scripting with plenty of examples written in Python.

Other relevant features of Python are the following.

- Python is a modern language fully based on well-structured classes (unlike most scientific languages such as Matlab and R), which make easy creating, maintaining and reusing modular object-oriented code.
- As most recent scripting languages, Python inherits the best features and concepts of both system languages (such as C and Fortran) and structural languages (such as Haskell).
- Libraries such NumPy and CVXOPT provide a link to legacy libraries (e.g., BLAS, LAPACK, UMFPACK, etc.) for manipulating multidimensional arrays, linear algebra, eigenvalue analysis and sparse matrices. As the number crunching is done by efficient libraries, the slowness of the Python interpreter is not a bottle-neck.
- Thanks to graphical libraries such as Matplotlib, the ability of producing publication quality 2D figures in Python is as powerful as in Matlab.

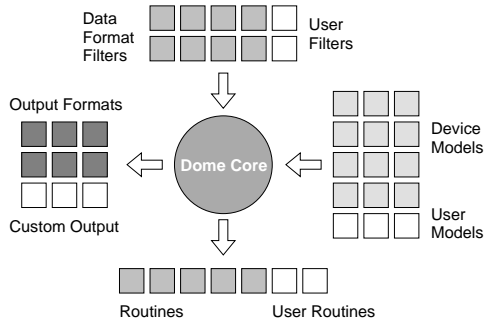


Fig. 1. Qualitative representation of the structure of DOME.

- The huge variety of free third-party libraries available for Python, allows easily and quickly extending the features of an application well beyond the scope of the original project (e.g., the Python profiler and the multiprocessing modules).
- Python is free and open source. Hence Python promotes the implementation and distribution of open projects.
- Python syntax is relatively simple, neat, compact and elegant. Hence, Python is particularly adequate for education and illustrative examples.

The features listed above can be likely found in other scripting languages and do not imply that Python is flawless. However, such features have proved to be flexible and reliable enough to successfully develop and maintain the software tool described in this paper.

### III. OUTLINES OF THE DEVELOPED PYTHON-BASED SOFTWARE TOOL

The developed software tool, called DOME [15] has its roots on PSAT [3], which is a Matlab-based tool for power system analysis. Based on the experience of PSAT, the architecture has been rethought from scratch in order to avoid structural issues and, whenever possible, to provide fully-fledged solutions.

The main idea on which the Python-based tool is founded is the modularity and reusability of the code. Basically no part of the code, except for a tiny kernel, is really necessary. Rather, the code is based on a reduced number of milestones that have to be available, but how such milestones solve their duty is not relevant for the main kernel. The milestones are only four:

- 1) Parsing the data file and initializing device models.
- 2) Solving the power flow analysis.
- 3) Solving other analyses, e.g., time domain simulation.
- 4) Dumping data to adequate output files.

Each milestone is composed by several independent Python modules. The user can also provided his own custom modules. Figure 1 illustrates the concepts discusses so far. The only requirement that each module has to satisfy is the *communication protocol* with DOME core functions. This is obtained by passing to each module a pointer to a core common class, called `system`. This modular structure is the key, along with the versatility of Python classes, for a quick and easy development of the code.

Currently, DOME provides about 50 parsers for input data, including most popular power system formats such as PSS/E

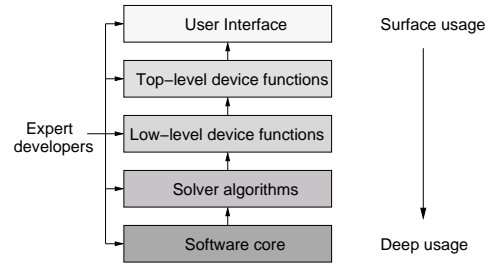


Fig. 2. Layer organization of DOME.

and GE and SIMPOW formats; about 300 devices ranging from standard power flow models, synchronous machines, AVRs and other basic controllers to a variety of wind turbines, energy storage devices and distributed energy sources; 10 analysis tools including standard power flow analysis as well as three-phase unbalanced power flow, continuation power flow, OPF, time domain simulation, electromagnetic transients, eigenvalue analysis, short circuit analysis, equivalencing procedures and load admission control strategies for smart grids; and 10 output formats, including  $\text{\LaTeX}$ , Excel, and 2D and 3D visualization tools.

Despite the vastness of the tools and models provided, DOME remains fundamentally a light tool. Thanks to laziness, only needed modules are loaded at run-time. These are generally less than 1% of available modules. Hence, the project can grow indefinitely without affecting performance. Modularity and laziness has also the advantage of allowing parallel development of new modules: if a beta-version of a new function is broken, all users that are not using such function can continue using DOME smoothly. Moreover, no forking is necessary as different versions of the same module can coexist.

Another important aspect is the possibility of using DOME as a didactic tool at different levels: undergraduate, master and Ph.D. In fact, only a very surface knowledge of the code is needed to develop a new device model. This allows using DOME for undergraduate final projects. To develop a new routine requires a deeper knowledge, which can be achieved by Master and Ph.D. students. Figure 2 illustrates the layer organization of DOME. A description of didactic aspects of DOME can be found in [16].

### IV. CASE STUDY

This section provides some examples of simulations that can be solved using the developed software tool. The goal is not simply to show the performance of the code but, rather, to illustrate the features that makes DOME an unique tool among currently available power system software packages.

This section illustrates the following features:

- 1) Ability of including a variety of efficient libraries for the factorization of sparse matrices. With this regard, DOME can be conveniently used to test mathematical libraries and define their suitability for power system analysis.
- 2) Ability of numerically integrating stochastic differential equations (SDAEs). A variety of stochastic device mod-

TABLE I  
COMPARISON OF THE PERFORMANCE OF SPARSE MATRIX FACTORIZATION  
LIBRARIES FOR THE SOLUTION OF THE POWER FLOW PROBLEM

Library	Total CPU time [s]	1 <sup>st</sup> fact. time [s]	Next fact. time [s]
KLU	0.0933	0.0044	0.0026
CXSPARSE	0.0936	0.0043	0.0027
UMFPACK	0.1750	0.0126	0.0095
SUPERLU	0.1927	0.0247	0.0082
LUSOL	0.3112	0.0360	0.0195

els are included in the proposed case study. Multiprocessing is also exploited to run these simulations.

- 3) Ability of solving the eigenvalue analysis of delayed differential equations (DDAEs). This analysis is particularly demanding as the size of the problem can be huge even for small systems. Also in this case, a variety of mathematical libraries are compared.

All simulations are solved on a server equipped with two processors Intel Xeon Six Core 2.66 GHz, an Intel SSD of 256 GB, and 64 GB of RAM, and mounting a Linux 64 bits operating system.

#### A. Comparison of Sparse Matrix Factorization Libraries

This subsection compares the performance of a variety of sparse matrix factorization libraries through power system analysis. Only open-source libraries are considered. These are: CXSPARSE, UMFPACK, KLU, LUSOL, and SUPERLU. The former three libraries are part of the SuiteSparse package [17], whereas SUPERLU and LUSOL are available at [18] and [19], respectively. A 1254-bus 1944-line network that models that UCTE 2002 Winter Off-peak is used as benchmark system. Details on this system can be found in [20].

Table I shows the results obtained for the considered libraries which are compared in terms of the CPU time required to solve the power flow analysis for the UCTE 2002 Winter Off-peak. Eight iterations are required to solve the power flow problem by means of a standard Newton-Raphson technique. This relatively high number of iterations is due to the fact that the voltages of some buses are higher than 1.1 pu, which leads PQ loads to switch to constant impedance models.

Table I indicates the total time required to solve the power flow analysis, which includes the Jacobian matrix factorization as well as the time to build the Jacobian matrix itself and by the overall power flow algorithm; the CPU time to factorize for the first time the Jacobian matrix, which includes both the symbolic and the numeric factorization steps; and the time for solving the factorization the second and following iterations, which implies only the numeric factorization step.

The two most efficient libraries are KLU and CXSPARSE. KLU is known to be particularly suited for factorizing sparse matrices that describe electrical circuits. As a matter of fact, KLU is the library used in the OpenDSS project [6]. On the other hand, the efficiency of CXSPARSE was not expected, as its main purpose is didactic. Other libraries, such as UMFPACK (used in Matlab) and SUPERLU (which is a fork of KLU) are not particularly efficient for solving the power flow

analysis problem. Finally, LUSOL, which is a legacy Fortran library used, for example, in GAMS [21], is the slowest of all the considered libraries.

The comparison given in Table I cannot be fully fair as each library stores sparse matrices in a slightly different way. The representation used in DOME is a C object that stores the matrix in the compact compressed column storage (CCS) format. This is the well-known Harwell-Boeing sparse matrix representation. In particular, DOME uses the CCS implementation provided by the CVXOPT package [22]. This C object is fully compatible only with KLU and UMFPACK libraries, whereas extra memory allocation is required to link the other libraries, so that their performance can be affected.

#### B. Time Domain Integration of SDAEs

This subsection illustrates the ability of DOME to simulate SDAE systems. DOME includes implicit A-stable time integration schemes for stiff DAE systems. These are the backward Euler, the trapezoidal method and the backward differentiation formula of order 2. Moreover, DOME includes the possibility to model and numerically integrate stochastic processes, such as the Wiener and the Orstein-Uhlenbeck processes, which have been proposed in the literature [23], [24]. Continuous SDAE wind models based on the Weibull distribution are also included [25]. Stochastic processes are integrated using Maryuama-Euler and Milstein schemes [26].

As previously discussed, DOME allows including new devices with the minimal effort. Then, Python classes and metaprogramming allow easily implementing stochastic models. These features allows defining a stochastic version of any device by simply merging together the original device class and the class implementing the stochastic process. Moreover, a special meta-device that adds stochastic processes to an arbitrary state or algebraic variable allows to modify the behavior of any implemented device.

Figure 3 shows 1000 trajectories (strong solutions of the SDAE) as well as the mean trajectory (weak solution of the SDAE) for the IEEE 14-bus system that includes stochastic processes in the bus voltage phasors, machine rotor speeds as well as load power consumption. All standard dynamic data of this system can be found in [14]. The simulations consist in applying the outage of line 2-4 at  $t = 1$  s. All devices are perturbed through Orstein-Uhlenbeck processes, which allows bounding the standard deviation thanks to the mean-reverting feature of this process [23]. The diffusion term of all stochastic processes is considered constant and is assumed to be 1% of the mean value. These processes account for harmonics, vibrations, and load randomness.

Integrating SDAEs requires is a demanding task. In general, one has to solve a few thousands of time domain integrations with different generations of the normal distributions used to define the stochastic processes. These high number of solutions is required to properly define the statistical properties of the resulting trajectories (e.g., standard deviation, autocorrelation, etc.). However, since each simulation is fully decoupled from all others, parallelization can be easily exploited. DOME includes some basic mechanism to parallelize simulations by distributing multiple tasks over available processors. As a matter

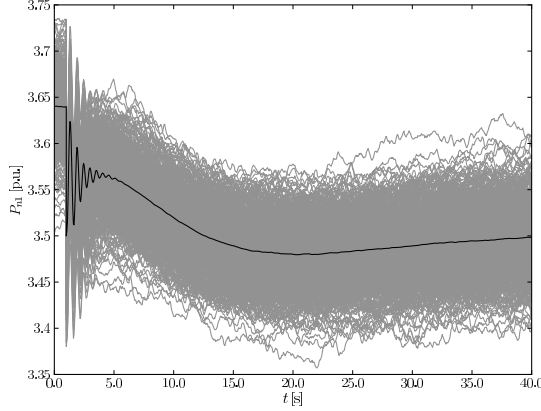


Fig. 3. Active power injected at bus 1 for the IEEE 14-bus system. Gray lines are the 1000 computed trajectories; the black line is the mean value.

of fact, multiprocessing is particularly easy in Python (thanks to the built-in `multiprocessing` module), especially if processes do not share memory. As a result, solving the 1000 simulations with a fixed time step of 0.05 s and a standard dishonest Newton-Raphson solver takes about 8.5 seconds on the 24 processors (12 processors are virtual) available on the server used in this case study.

### C. Eigenvalue Analysis of DDAEs

This subsection describes the small-signal stability analysis of a power system modeled as a delayed DAE. As it is well-known, the number of eigenvalues  $\lambda$  of a DDAEs with constant delays is infinite, as it is the solution of the following characteristic equation:

$$\Delta(\lambda) = \lambda \mathbf{I}_n - \mathbf{A}_0 - \sum_{i=1}^{\nu} \mathbf{A}_i e^{-\lambda \tau_i} \quad (1)$$

where  $\mathbf{A}_0$  is the standard state matrix,  $\mathbf{I}_n$  is the identity matrix of order  $n$ ,  $\mathbf{A}_i$  are the characteristic matrix for each constant delay  $\tau_i$  [27].

The technique implemented in DOME attempts to compute a reduced number of eigenvalues of (1) using the approximation proposed in [28]–[30]. These methods are based on a discretization of the partial differential equation (PDE) representation of the DDAE. The implementation of such discretization is surprisingly simple while results proved to be accurate. The idea is to transform the original DDAE problem into an equivalent PDE system of infinite dimensions. Then, instead of computing the roots of retarded functional differential equations, one has to solve a finite, though possibly large, matrix eigenvalue problem of the discretized PDE system.

Without entering into mathematical details (the interested reader can find an exhaustive discussion in [31]), let simply say that the resulting problem to be solved is a standard eigenvalue analysis of a highly sparse square matrix whose size is  $N \times n$ , where  $n$  is the number of state variable of the system and  $N$  is the number of nodes of the Chebyshev grid used to approximate the PDE representation of the DDAE [29]. As

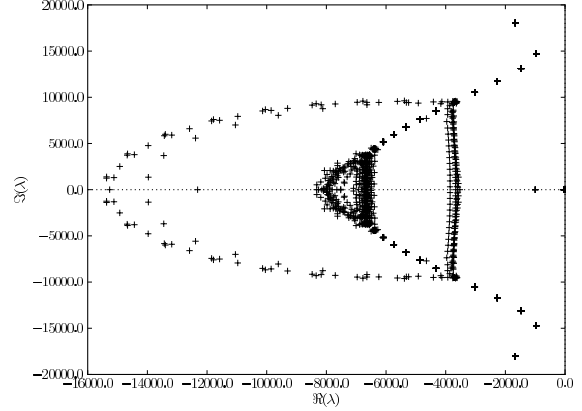


Fig. 4. Full spectrum for the IEEE 14-bus system with 5 ms time delay in the AVR measured voltage signals.

it can be easily noted, even for small  $N$  the computational burden of the solution of the eigenvalue problem dramatically increases. Fortunately, only a reduced set of eigenvalue is needed for the purpose of stability analysis, i.e., those whose real part is close to the imaginary axis. Hence, efficient algorithms able to compute a reduced number of eigenvalues with a given property are to be used.

The case study considered in this subsection is again the IEEE 14-bus system as described in [14]. The dynamic order of this system is  $n = 49$ . Let assume that the terminal voltage signal of the 5 AVRs of synchronous machines have a measurement delay of  $\tau = 5$  ms. One can debate whether considering the same delay for all AVRs is reasonable or not. However, how to set-up the technique proposed in [29] to include multiple time delays is currently an open question and is out of the scope of this paper.

Figure 4 shows the full spectrum in the  $S$ -domain obtained with  $N = 40$ . Hence, the total number of eigenvalues are 1960. The eigenvalues have been computed using the routine for generalized matrix provided by the well-known LAPACK library. The total simulation time is 28.3 s. Further increasing  $N$  provides more details on the spectrum but do not alter the values of eigenvalues close to the imaginary axis, which are the only of interest for the stability analysis.

While the  $S$ -domain is widely used in power system analysis, there are alternative domain that can be used to solve the eigenvalue analysis. In particular, DOME implements the  $Z$ -domain bilinear transformation, as follows:

$$\mathbf{A}_Z = (\mathbf{A}_S + \chi \mathbf{I})(\mathbf{A}_S - \chi \mathbf{I}_{n_x})^{-1} \quad (2)$$

where  $\mathbf{A}_S$  is the original state matrix,  $\mathbf{I}$  the identity matrix of the same size as  $\mathbf{A}_S$ , and  $\chi$  is a weighting factor that, based on heuristic considerations, can be set to  $\chi = 8$  [32].

Computing  $\mathbf{A}_Z$  is more expensive than  $\mathbf{A}_S$  but using  $\mathbf{A}_Z$  can be useful to better visualize stiff systems (as the eigenvalues falls within or close to the unitary circle) and for fastening the determination of the maximum amplitude eigenvalue (e.g., by means of the Arnoldi iteration), especially in case of unstable equilibrium points with only one eigenvalue

TABLE II  
COMPARISON OF THE PERFORMANCE OF LIBRARIES FOR EIGENVALUE ANALYSIS OF LARGE SPARSE MATRICES.

Library	Method	CPU time [s]
ARPACK	Arnoldi Iteration	2.15
SLEPC	Arnoldi Iteration	2.16
SLEPC	Krylov-Shur method	1.50
SLEPC	Lanczos method	1.48

outside the unit circle. In fact, the eigenvalues in the  $Z$ -plane are stable if their magnitude is lower than 1, and unstable if greater than 1. Bifurcation points are on the unitary circumference.

Since the critical eigenvalues in the  $Z$ -domain have magnitude close to 1, it is possible to take advantage of algorithms that compute a reduced number of eigenvalues with a given property. DOME provides C extensions to some efficient open-source libraries such as ARPACK [33] and SLEPC [34]. The latter library is based on PETSc, which is a C++ based suite of data structures and routines for the scalable solution of scientific applications [35].

Table II shows a comparison of different algorithms and libraries to compute the 50 eigenvalues with largest magnitude in the  $Z$ -domain for the IEEE 14-bus system with time delays and  $N = 40$ . In particular, the Arnoldi iteration, the Krylov-Shur method and the Lanczos method are compared. As it can be observed, the Arnoldi iteration of ARPACK and SLEPC implementations shows similar performance. On the other hand, the Krylov-Shur and the Lanczos methods provide best results.

## V. CONCLUSIONS

This paper shows that Python is a modern, mature, complete and versatile scripting language that is fully prepared for scientific research and education on power system analysis. The case studies discussed in the paper demonstrate that properly linking Python to efficient general purpose mathematical libraries allows obtaining excellent performance.

Future work will focus on further developing DOME in various directions, such as smart grid modeling, parallel computing including GPUs and heterogeneous architectures, as well as testing novel mathematical tools for power system analysis.

## REFERENCES

- [1] H. P. Langtangen, *Python Scripting for Computational Science*. Heidelberg: Springer-Verlag, 2002, third edition.
- [2] J. H. Chow and K. W. Cheung, "A Toolbox for Power System Dynamics and Control Engineering Education and Research," *IEEE Trans. on Power Systems*, vol. 7, no. 4, pp. 1559–1564, Nov. 1992.
- [3] F. Milano, "An Open Source Power System Analysis Toolbox," *IEEE Trans. on Power Systems*, vol. 20, no. 3, pp. 1199–1206, Aug. 2005.
- [4] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education," *IEEE Trans. on Power Systems*, vol. 26, no. 1, pp. 12–19, Feb. 2011.
- [5] M. Zhou and S. Zhou, "Internet, Open-source and Power System Simulation," in *IEEE PES Gen. Meeting*, Montreal, Quebec, Jun. 2007.
- [6] R. C. Dugan and T. E. McDermott, "An Open Source Platform for Collaborating on Smart Grid Research," in *IEEE PES Gen. Meeting*, July 2011, pp. 1–7.

- [7] C. A. Cañizares, F. L. Alvarado, and S. Zhang, "UWPFLOW Program," 2006, university of Waterloo, available at <http://www.power.uwaterloo.ca>.
- [8] "PYPOWER," available at [www.pypower.org](http://www.pypower.org).
- [9] "minpower," available at [minpowertoolkit.com](http://minpowertoolkit.com).
- [10] F. Milano and L. Vanfretti, "State of the Art and Future of OSS for Power Systems," in *IEEE PES Gen. Meeting*, Calgary, Canada, Jul. 2009.
- [11] B. Pierce, *Types and Programming Languages*. Cambridge, MA: MIT Press, 2002.
- [12] P. S. Pacheco, *An Introduction to Parallel Programming*. Burlington, MA: Elsevier, 2011.
- [13] B. O'Sullivan, J. Goerzen, and D. Stewart, *Real World Haskell*. Sebastopol, CA: O'Reilly, 2008.
- [14] F. Milano, *Power System Modelling and Scripting*. London: Springer, 2010.
- [15] "Dome Project," available at [www3.uclm.es/profesorado/federico.milano](http://www3.uclm.es/profesorado/federico.milano).
- [16] L. Vanfretti and F. Milano, "Facilitating Constructive Alignment in Power Systems Engineering Education using Free and Open Source Software," *IEEE Trans. on Education*, vol. 55, no. 3, pp. 309–318, 2012.
- [17] "SuiteSparse version 4.0.2," available at [www.cise.ufl.edu/research/sparse/SuiteSparse](http://www.cise.ufl.edu/research/sparse/SuiteSparse).
- [18] "SuperLU version 4.3," available at [crd-legacy.lbl.gov/xiaoye/SuperLU](http://crd-legacy.lbl.gov/xiaoye/SuperLU).
- [19] "LUSOL," available at [www.stanford.edu/group/SOL/software/lusol.html](http://www.stanford.edu/group/SOL/software/lusol.html).
- [20] Q. Zhou and J. W. Bialek, "Approximate Model of European Interconnected System as a Benchmark System to Study Effects of Cross-Border Trades," *IEEE Trans. on Power Systems*, vol. 20, no. 2, pp. 782–787, May 2005.
- [21] "General Algebraic Modeling System," available at [www.gams.com](http://www.gams.com).
- [22] "CVXOPT – Python Software for Convex Optimization," available at [abel.ee.ucla.edu/cvxopt](http://abel.ee.ucla.edu/cvxopt).
- [23] M. Perninge, V. Knazkins, M. Amelin, and L. Söder, "Risk Estimation of Critical Time to Voltage Instability Induced by Saddle-Node Bifurcation," *IEEE Trans. on Power Systems*, vol. 25, no. 3, pp. 1600–1610, Aug. 2011.
- [24] Z. Y. Dong, J. H. Zhao, and D. J. Hill, "Numerical Simulation for Stochastic Transient Stability Assessment," *IEEE Trans. on Power Systems*, vol. 27, no. 4, pp. 1741–1749, Nov. 2012.
- [25] R. Zárate-Miñano, M. Anghel, and F. Milano, "Continuous Wind Speed Models based on Stochastic Differential Equations," accepted for publication on *Applied Energy*, 2012.
- [26] E. Kloeden, E. Platen, and H. Schurz, *Numerical Solution of SDE Through Computer Experiments*. New York, NY, third edition: Springer, 2003.
- [27] W. Michiels and S. Niculescu, *Stability and Stabilization of Time-Delay Systems*. Philadelphia: SIAM, 2007.
- [28] A. Bellen and M. Zennaro, *Numerical Methods for Delay Differential Equations*. Oxford: Oxford Science Publications, 2003.
- [29] D. Breda, "Solution Operator Approximations for Characteristic Roots of Delay Differential Equations," *Applied Numerical Mathematics*, vol. 56, pp. 305–317, 2006.
- [30] D. Breda, S. Maset, and R. Vermiglio, "Pseudospectral Approximation of Eigenvalues of Derivative Operators with Non-local Boundary Conditions," *Applied Numerical Mathematics*, vol. 56, pp. 318–331, 2006.
- [31] F. Milano and M. Anghel, "Impact of Time Delays on Power System Stability," *IEEE Trans. on Circuits and Systems - I: Regular Papers*, vol. 59, no. 4, pp. 889–900, 2012.
- [32] M. Ilić and J. Zaborszky, *Dynamic and Control of Large Electric Power Systems*. New York: Wiley-Interscience Publication, 2000.
- [33] "ARPACK – Arnoldi Package," available at [www.caam.rice.edu/software/ARPACK](http://www.caam.rice.edu/software/ARPACK).
- [34] "SLEPC – Scalable Library for Eigenvalue Problem Computations," available at [www.grycap.upv.es/slepc](http://www.grycap.upv.es/slepc).
- [35] "PETSc – Portable, Extensible Toolkit for Scientific Computation," available at [www.mcs.anl.gov/petsc](http://www.mcs.anl.gov/petsc).



**Federico Milano** (S'09) received from the University of Genoa, Italy, the Electrical Engineering degree and the Ph.D. degree in 1999 and 2003, respectively. From 2001 to 2002 he was with the University of Waterloo, Canada, as a Visiting Scholar. From 2003 to 2013, he was with the University of Castilla-La Mancha, Spain. He joined the University College Dublin, Ireland, in 2013, where is currently Associate Professor. His research interests include power system modeling, stability and control.