Modeling of Power System in Overcoming Inverse Problems

[1]**AJANTA PRIYADARSHINI,** *Gandhi Institute of Excellent Technocrats, Bhubaneswar, India*

[2]**SIBASUNDAR MIRDA,** VITS Engineering College, Khordha, Odisha, India

***Abstract*—Large disturbances in power systems often initiate complex interactions between continuous dynamics and discrete events. The paper develops a hybrid automaton that describes such behavior. Hybrid systems can be modeled in a systematic way by a set of differential-algebraic equations, modified to incorporate impulse (state reset) action and constraint switching. This differential-algebraic impulsive-switched (DAIS) model is a realization of the hybrid automaton. The paper presents a practical object-oriented approach to implementing the DAIS model. Each component of a system is modeled autonomously. Connections between components are established by simple algebraic equations. The systematic nature of the DAIS model enables efficient computation of trajectory sensitivities, which in turn facilitate algorithms for solving inverse problems. The paper outlines a number of inverse problems, including parameter uncertainty, parameter estimation, grazing bifurcations, boundary value problems, and dynamic embedded optimization.**

I. INTRODUCTION

Interactions between continuous dynamics and discrete events are an intrinsic part of power system dynamic behavior. Devices that obey physical laws typically exhibit continuous dynamics. Examples range from generators and their controllers at the system level, through to capacitors and inductors within power electronic circuits. On the other hand, event-driven discrete behavior is normally associated with rule-based components. Examples in this latter category include protection devices [1], tap-changing transformers [2], power electronic switches [3] and supervisory control [4]. Limits within physical devices also fall into this category; an event occurs when a controller signal saturates or a FACTS device encounters its maximum/minimum firing angle.

To illustrate continuous/discrete interactions in power systems, consider a disturbance consisting of an initiating event, such as a lightning strike on a transmission line, followed by protection action to remove the fault. The fault would disturb the steady-state balance between electrical and mechanical torques on generator shafts, causing angles and frequencies to respond dynamically. In parallel, protection relays should detect the fault and decide on the appropriate response. Trip signals would be sent to circuit breakers, which should disconnect the faulted feeder after a small (mechanical) time delay. Meanwhile, oscillations induced in inter machine angles may or may not be stable. Removal of the faulted line could lead to overloading of other feeders, and their subsequent tripping . The consequent increased demand for reactive power may activate generator over-excitation protection, causing a reduction in terminal voltage, increased system losses, further overloading of feeders and finally system disintegration. Whilst this scenario seems pessimistic, it has occurred, to the detriment (and annoyance) of many consumers! Similar continuous/discrete interactions exist across all layers of power systems. At the market layer, for example, system measurements and participant inputs are interpreted in terms of market rules to generate events that affect the physical system.

In all cases, discrete events influence continuous dynamics, which in turn trigger new events. Modeling and simulation must accurately capture these interactions. Power system simulation has generally evolved to the point where the continuous/discrete nature of dynamic behavior is fairly accurately replicated. However, it is common to find event handling treated as an *ad hoc* addition to continuous state simulation. The nature of inverse problems dictates a more systematic *hybrid systems* approach to capturing continuous/discrete interactions.

Power system analysis normally addresses *forward problems*. Given a system model and a set of parameters, system dynamic response can be determined. However, the disturbance scenario outlined above motivates analysis questions that are classed as *inverse problems* [5]. Such a disturbance would generate recordings from wide area measurement systems [6]. Those measurements could be used to improve estimates of parameters of system models [7], [8]. This is an inverse problem; the measured response is given, and a model is used to estimate parameters. It is easy to postulate other inverse problems. For example, what are the minimal changes in controllable parameters, e.g., generator real power and voltage setpoints, that would avoid cascading tripping of overloaded feeders or voltage dip problems or instability? How significantly do certain parameters, e.g., load voltage dependence, impact system behavior?

The paper has the following structure. Section II presents various systematic representations of hybrid systems. Implementation issues are discussed in Section III. Inverse problems are considered in Section IV, and conclusions are presented in Section V.

II. Hybrid System Representation

A. Background

Power systems are an important example of hybrid systems, which are characterized by:

- continuous and discrete states;
- continuous dynamics;
- discrete events, or triggers;
- mappings that define the evolution of discrete states at events.

Conceptually, such systems can be thought of as an indexed collection of continuous dynamical systems

$$F_q(\dot{x}, x, y) = 0 \tag{1}$$

along with a mechanism for "jumping" between those systems, i.e., for switching between the various F_q. Each system is indexed by the discrete state q, whilst x and y are the continuous dynamic and algebraic states, respectively. The jumping reflects the influence of the discrete event behavior, and is dependent upon both a trigger condition and a discrete state evolution mapping. Overall system behavior can be viewed as a sequential patching together of dynamical systems, with the final state of one dynamical system specifying the initial state for the next.

B. Hybrid Automaton

Interest in hybrid systems spans a broad range of scientific communities, from control to computer science; see, for example, [9]. A consequence is that numerous formal definitions of hybrid systems have been proposed. Whilst each representation has its own particular flavor, they all capture the fundamental aspects of hybrid systems identified above. The following definition of a hybrid system has been adapted from [10], [11].

A *hybrid automaton* is described by the triple

$$H = [Q, \Sigma, E] \tag{2}$$

where

- Q is the finite set of discrete states. They form the *vertices* of a graph.
- $\Sigma = \{\Sigma_q\}_{q \in Q}$ is the collection of dynamical systems $\Sigma_q = [X_q, F_q]$ where each X_q is a topological space forming the continuous state space of Σ_q, and F_q generates the continuous state dynamics according to (1).
- $E = \{E_q\}_{q \in Q}$ is the finite set of *events*. The events are described by the triple $E_q = [L_q, A_q, G_q]$, where
  - L_q is the set of symbols that label the events;
  - $A_q = \{A_\ell\}_{\ell \in L_q}$, $A_q \subset X_q$ is the collection of autonomous jump sets for each $q \in Q$, i.e., the conditions which trigger jumps from state q;
  - $G_q = \{G_\ell\}_{\ell \in L_q}$, where $G_\ell : (A_\ell \times q) \to S = \bigcup_{q \in Q} (X_q \times \{q\})$ is the autonomous jump transition map that describes the outcome of event ℓ originating in state q. The transitions form *edges* of a graph.

As indicated above, the hybrid state space of H is given by

$$S = \bigcup_{q \in Q} (X_q \times \{q\}). \tag{3}$$

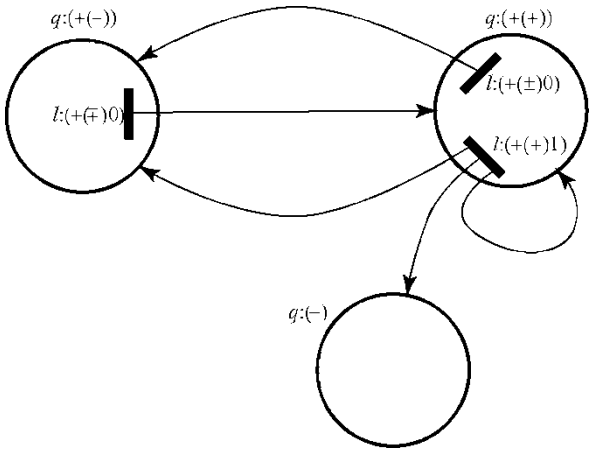

Fig. 1. Hybrid automaton.

The state of a hybrid automaton consists of a discrete part $q \in Q$ together with a continuous part $(x, y) \in X_q$. Fig. 1 provides an example of a graph defined by the hybrid automaton. (The symbols labeling this graph are described in Section II-E, and relate to the example of Section II-F.)

The dynamic behavior of a hybrid system can be described as follows. Given an initial state $q_0 \in Q$, $(x_0, y_0) \in X_{q_0}$, the system trajectory evolves continuously according to F_{q_0} until (possibly) the state enters A_{q_0} at (x^-, y^-). Encountering A_{q_0} will trigger jump $\ell_0 \in L_{q_0}$, with G_{ℓ_0} describing the transition to the new state $q_1 \in Q$, $(x^-, y^-) \in X_{q_1}$. The process continues from that new point.

C. Petri Nets

Systematic modeling of power systems requires an unambiguous methodology for describing discrete event activity. A number of formal languages exist, including Petri nets and finite state machines [12]. It is not clear that any particular language is best for power system applications, though Petri nets certainly capture the asynchronous and distributed nature of power system events [13]. Therefore, a Petri net representation has been adopted. As an example, Fig. 2 provides a (partial) Petri net model of the automatic voltage regulator (AVR) of a tap-changing transformer [2]. This example is considered further in Section II-F.

A Petri net is represented by a directed bipartite graph with two types of nodes: *places* (drawn as circles) and *transitions* (drawn as rectangles). Weighted directed arcs connect places to transitions and vice versa. Weights are denoted as b_{ij}^- for an arc from place p_i to transition t_j and b_{ij}^+ for an arc from transition t_j to place p_i. In the example of Fig. 2, weights with unit value have not been shown.

Tokens are drawn as black dots, and places act as *token holders*. The number of tokens in a place cannot be negative. At any time instant, the *marking* (state) of a Petri net is given by the number of tokens at its places. Transitions model events, and cause the manipulation, creation, or disappearance of tokens. Transition t_j is *enabled* only if each of its input places p_i has at least b_{ij}^- tokens. When transition t_j *fires*, it removes b_{ij}^- tokens from each of the input places p_i and adds b_{ij}^+ tokens to each output place p_i. In hybrid systems, transitions fire when

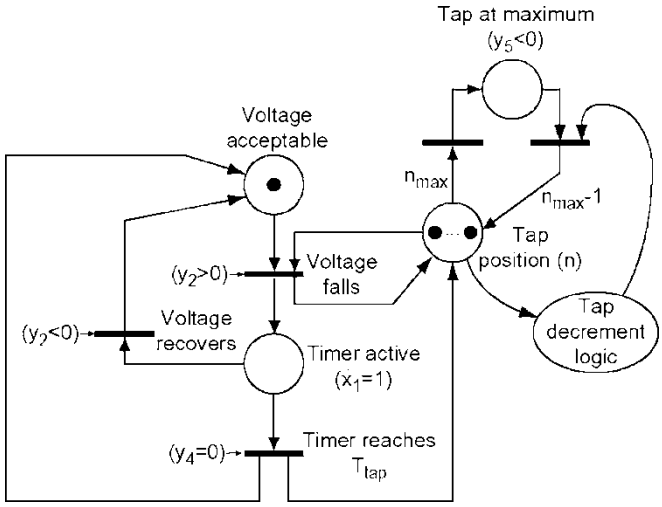

Fig. 2. Tap-changing transformer AVR logic for increasing tap.

the (evolving) continuous state satisfies the corresponding trigger condition.

D. Simulation Model

Petri nets and hybrid automata provide a framework for establishing rigorous mathematical representations of physical devices and systems. However, those representations are not immediately applicable to forward problems (via simulation), much less inverse problems. A model that captures the full richness of hybrid system behavior, yet has a form suitable for simulation, is required.

Simulation techniques and properties are well established for differential-algebraic-equation (DAE) systems [14]. Therefore, the proposed hybrid system model is adapted from that basic form by incorporating impulsive action and switching of algebraic equations, giving the DA impulsive switched (DAIS) model

$$\dot{x} = f(x, y) + \sum_{j=1}^{r} \delta(y_{r,j})(h_j(x, y) - x) \tag{4}$$

$$0 = g(x, y) = g^{(0)}(x, y) + \sum_{i=1}^{s} \left(g^{(i-1)}(x, y) + u(y_{s,i})\left(g^{(i)}(x, y) - g^{(i-1)}(x, y)\right)\right) \tag{5}$$

where

- $x \in \mathbb{R}^n$ are $x \in \mathbb{R}^n$;
- $\delta(\cdot)$ is the Dirac delta;
- $u(\cdot)$ is the unit-step function;
- $f, h_j : \mathbb{R}^{n+m} \to \mathbb{R}^n$;
- $g^{(0)}, g^{(i)} : \mathbb{R}^{n+m} \to \mathbb{R}^m$; some elements of each $g^{(\cdot)}$ will usually be identically zero, but no elements of the composite g should be identically zero; each $g^{(i)}$ may itself have a switched form, and is defined similarly to (5), leading to a nested structure for g
- y_r, y_s are selected elements of y that trigger state reset (impulsive) and algebraic switching events respectively; y_r and y_s may share common elements.

The impulse and unit-step terms of the DAIS model can be expressed in alternative forms:

- Each impulse term of the summation in (4) can be expressed in the state reset form

$$x^+ = h_j(x^-, y^-) \quad \text{when } y_{r,j} = 0 \tag{6}$$

where the notation x^+ denotes the value of x just after the reset event, whilst x^- and y^- refer to the values of x and y just prior to the event. This form motivates a generalization to an implicit mapping $h_j(x^+, x^-, y^-) = 0$.
- The contribution of each $g^{(i)}$ in (5) can be expressed as

$$g^{(i)}(x, y) = \begin{cases} g^{(i-1)}(x, y), & y_{s,i} < 0, \\ g^{(i)}(x, y), & y_{s,i} > 0, \end{cases} \qquad i = 1, \ldots, s \tag{7}$$

with (5) becoming

$$0 = g(x, y) = g^{(0)}(x, y) + \sum_{i=1}^{s} g^{(i)}(x, y). \tag{8}$$

This form is often more intuitive than (5).

Equations (4) and (5) are a reformulation (and slight generalization) of the model proposed in [15].

It can be convenient to establish the partitions

$$x = \begin{bmatrix} \underline{x} \\ z \\ \lambda \end{bmatrix} \qquad f = \begin{bmatrix} \underline{f} \\ 0 \\ 0 \end{bmatrix} \qquad h_j = \begin{bmatrix} \underline{x} \\ \underline{h}_j \\ \lambda \end{bmatrix} \tag{9}$$

where

- $\underline{x}$ continuous dynamic states, for example generator angles, velocities and fluxes;
- z discrete dynamic states, such as transformer tap positions and protection relay logic states;
- λ parameters such as generator reactances, controller gains and switching times.

This partitioning of the differential equations f ensures that away from events, $\underline{x}$ evolves according to $\dot{\underline{x}} = \underline{f}(x, y)$, whilst z and λ remain constant. Similarly, the partitioning of the reset equations h_j ensures that $\underline{x}$ and λ remain constant at reset events, but the discrete dynamic states z are reset to new values given by $z^+ = \underline{h}_j(x^-, y^-)$.

Remarks:

1) The DAIS model assumes constant state–space dimension $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ across events. This differs from some other hybrid system implementations, e.g., [16], where the state dimension is allowed to vary upon component switching. The DAIS formulation is not restrictive, though it may require carrying some "inactive" states following an event. Maintaining constant state dimension has a number of advantages though: 1) the variational equations describing trajectory sensitivities, presented in Appendix I, have a simpler form, and 2) switched states are more easily incorporated into objective functions of optimization-based inverse problems.
2) The simulation of DAE systems is prone to technical issues that do not arise with ordinary differential equation (ODE) systems [14]. The switched nature of the DAIS model introduces some extra complexities that require careful consideration. Component switching can result in

coupling of states, with a consequent increase in the DAE index. The higher index implies the system must operate on a submanifold of the state space. Therefore, at such a switching event, the states must be reinitialized to an appropriate point on the submanifold [17]. This difficulty is usually a legacy of modeling approximations. However, such modeling is ubiquitous, so the issue cannot be ignored.

Initial conditions for the model (4) and (5) are given by $x(t_0) = x_0$ and $y(t_0) = y_0$, where y_0 is a solution of $g(x_0, y_0) = 0$. Note that in solving for y_0, the constraint switching described by (5) must be taken into account. This establishes the initial discrete state q_0.

We define the *flows* of x and y as

$$\phi(x_0, t) = \begin{bmatrix} \phi_x(x_0, t) \\ \phi_y(x_0, t) \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} \tag{10}$$

where $x(t)$ and $y(t)$ satisfy (4) and (5), along with initial conditions

$$\phi_x(x_0, t_0) = x_0 \tag{11}$$
$$g(x_0, \phi_y(x_0, t_0)) = 0. \tag{12}$$

E. Hybrid Automaton Interpretation of DAIS Model

The DAIS model (4) and (5) captures the fundamental attributes of hybrid system behavior, and is a realization of the hybrid automaton model (2). Between events, system behavior is governed by the DAE dynamical system F_q given by

$$\dot{x} = f(x, y) \tag{13}$$
$$0 = g_q(x, y) \tag{14}$$

where g_q is composed of $g^{(0)}$, together with functions from (5) chosen depending on the signs of the elements of y_s. Each different composition of g_q is indexed by a unique $q \in Q$. One approach to indexing is to associate q with a string of length s, where each character of that string is the sign of the corresponding element of y_s i.e., either "+" or "−." If, however, $g^{(i)}$ has a nested definition, the ith symbol is replaced by a string of the form "$(\pm(+\cdots\pm))$," with each "+" symbol taking the sign of the corresponding element of y_s. This labeling scheme is illustrated in Fig. 1, where each discrete state q forms the vertex of a graph. (This figure relates to the example presented in Section II-F.)

An event is triggered by an element of y_r passing through zero and/or an element of y_s changing sign. Therefore, each condition of the form $y_{r,i} = 0$ or $y_{s,j} = 0$ contributes a jump set A_ℓ, with A_q formed from the union of those A_ℓ. The general nature of g_q ensures unrestricted specification of trigger conditions for any event ℓ. In particular, arbitrarily complicated logical propositions can be represented [18].

In the case of a switching event $y_{s,i} = 0$, the composition of g_q is forced to change. Subsequent solution of $g_q = 0$ may induce further switching. This is acceptable, provided the originating event does not attempt to reverse, i.e., switching events must satisfy $y_{s,i}^- \times y_{s,i}^+ < 0$. The outcome of a switching event always involves a transition to a new discrete state. The dynamic states x are fixed at a switching event, whilst the jump transition map G_ℓ for the algebraic states y is defined (implicitly) through the solution of the algebraic equations $g_q = 0$.

Reset events, on the other hand, map x to a new value, but may leave the discrete state q unchanged. However, it is also possible that through the solution of $g_q = 0$, the reset value of x induces changes in the signs of some elements of y_s, and hence, a consequent transition in the discrete state q. In this case, jump transition maps G_ℓ are defined (explicitly) in terms of reset equations h_j and (implicitly) through the solution of $g_q = 0$.

Note that in power system applications, there is usually no guarantee of a unique solution for $g_q = 0$. In such cases, G_ℓ is nonunique.

Event labels L_q can be derived from the discrete state labeling scheme. For the switching event $y_{s,i} = 0$, the ith symbol in the q index would be changed to "∓" if the transition was from the "+" state to the "−" state, i.e., $y_{s,i}$ was changing from positive to negative, or "±" for the opposite sign change. This form extends naturally to nested indexing structures. Labeling for reset events can be achieved by augmenting the discrete state label by a sequence of symbols, e.g., the symbol "0." For a reset event $y_{r,j} = 0$, the jth symbol would be replaced by a different symbol. Fig. 1 illustrates this scheme for labeling events. The figure also indicates that an event may generate a variety of transitions, depending on the value of the state when the event is triggered. The transitions form the edges of the graph.

Remark:

The graph defined by the hybrid automaton describes all possible discrete state trajectories. As a hybrid system evolves, it will generate a path that traverses the graph. (Though generally not all vertices will be visited.) The structure of the graph therefore provides insights into system behavior. In the context of power systems, such a representation could assist in understanding the nature of cascading disturbances. The graph structure may serve to highlight unanticipated events, and identify (discrete) control actions.

The partitioning of x, f, and h_j given by (9) ensures that elements of x are piecewise constant variables that only change value at reset events. Therefore, the dynamical system representation of (1) can be rewritten as

$$F_{(q,z)}(\dot{x}, x, y) = 0 \tag{15}$$

and the discrete state redefined as $(q, z) \in Q'$, where Q' is now countable rather than finite. Reset events cause a change in z, and a possible consequent change in q. Switching events only change q.

F. Example

In order to demonstrate the ability of the DAIS structure (4) and (5) to model logic-based systems, this example considers a relatively detailed representation of the AVR of a tap-changing transformer [2]. The model incorporates a voltage deadband and timer. Deviation of the regulated bus voltage beyond the deadband initiates the timer. If the timer times out, i.e., reaches its maximum, a tap change occurs and the timer is reset. However,

should the voltage recover (return to within the deadband) before the timer reaches its maximum, the timer is reset and the AVR returns to the wait state. The Petri net model of this AVR logic for low voltages, i.e., for increasing tap ratio, appeared earlier in Fig. 2. The model can be represented in the DAIS form as

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ z_1 \\ n \end{bmatrix} = \begin{bmatrix} y_1 \\ 0 \\ 0 \end{bmatrix} + \delta(y_4)\begin{bmatrix} 0 \\ x_1 - y_3 - z_1 \\ n_{\text{step}} \end{bmatrix} \tag{16}$$

and (17), shown at the bottom of the page, where y_3^- is the value of y_3 just prior to the switching event. Expressing the model using the alternative forms (6)–(8) perhaps provides clearer insights into model behavior,[1] as

$$\left.\begin{array}{l|l} \dot{x}_1 = y_2 & x_1^+ = x_1^- \\ \dot{z}_1 = 0 & z_1^+ = x_1^- - y_3^- \\ \dot{n} = 0 & n^+ = n^- + n_{\text{step}} \end{array}\right\} \text{ when } y_4 = 0$$

$$0 = nV_1 - V_2$$

$$0 = y_2 - V_{\text{low}} + V_2$$

$$0 = y_1 - x_1 + z_1 + y_3 + T_{\text{tap}}$$

$$0 = y_5 + n - n_{\max} + \frac{n_{\text{step}}}{2}$$

$$\left.\begin{array}{l} 0 = y_1 \\ 0 = y_3 - x_1 + z_1 \end{array}\right\} y_2 < 0 \quad q:(\ )$$

$$\left.\begin{array}{l} 0 = y_1 \\ 0 = y_3 - x_1 + z_1 \end{array}\right\} y_2 < 0 \qquad \left.\begin{array}{l} 0 = y_1 - 1 \\ 0 = y_3 - y_3^-(\text{const}) \end{array}\right\} y_2 > 0 \qquad \begin{array}{l} y_2 > 0, \\ \\ \end{array} \quad \begin{array}{l} q:(+(\ )) \\ q:(+(+)) \end{array}$$

(18)

To assist in connecting AVR logic with the model, Fig. 2 indicates variables that are related to particular functions.

The hybrid automaton of Fig. 1 provides a representation of this model. The labeling scheme of Section II-E has been adopted in that figure. States are labeled according to the signs of switching triggers y_3 and y_2, and are indicated in (18). Event labeling is based on zero crossings of y_5, y_2, and y_4.

The behavior of the timer is central to the dynamics of this device. The timer value $t_{tim} = x_1 - z_1 - y_3$ ramps up ($\dot{x}_1 = y_1 = 1$) when the voltage deviates outside the deadband ($y_2 > 0$). While the timer is active, both x_1 and y_3 remain constant. Their values change only at events that reset the timer.

1) If the timer reaches the threshold T_{tap}, i.e., $t_{tim}^- = T_{\text{tap}}$ or equivalently $y_4 = 0$, a tap change occurs via the reset event. This event forces $z^+ = x_1^- - y_3^-$ and hence $t_{tim}^- = 0$. The timer is not directly disabled by this event, so will continue ramping if the voltage remains outside the deadband ($y_2 > 0$).
2) If the voltage returns to within the deadband ($y_2 < 0$), y_3 is set equal to $x_1 - z_1$, ensuring $t_{tim}^- = 0$. In this case, the timer is disabled ($\dot{x}_1 = 0$). Note that y_3 remains constant whenever the voltage deviates outside the deadband, i.e., when y_2 changes from positive to negative. It only changes value when the voltage is restored.

Remarks:

1) It is clear from the example that translating a Petri net model into its equivalent DAIS model is not always an intuitive process. However, the approach adopted in [19] provides the basis for an automated procedure.
2) An event is always triggered by a zero-crossing somewhere in the system. The actual trigger is only directly observable to its local component. However, the consequences of the event are generally more widely observable. The example illustrates this difference between local and remote events. Consider the events that trigger timer resetting. A tap change is a local (to the transformer) event that is triggered by the local variable y_4 crossing zero. Voltage recovery, on the other hand, may occur as a consequence of a remote event, such as feeder restoration. This consequent event is observed by the tap-changer via a sign change (but not a zero-crossing) of the local variable y_2. In the former case, the zero-crossing can be used to trigger a reset event. In the latter case, switching will occur as a consequence of the remote event. This distinction necessitates the parallel roles of x_1 and y_3 in the example. It is an important consideration in object-oriented system modeling, as discussed in Section III-A.

G. Trajectory Sensitivities

Trajectory sensitivities provide a way of quantifying the variation of a trajectory resulting from (small) changes to parameters and/or initial conditions [20]. To obtain the sensitivity of the flows ϕ_x and ϕ_y to initial conditions x_0, the Taylor series expansion of (10) is formed. Neglecting higher order terms gives

$$\Delta x(t) = \frac{\partial x(t)}{\partial x_0}\Delta x_0 = \Phi_x(t)\Delta x_0 \tag{19}$$

$$\Delta y(t) = \frac{\partial y(t)}{\partial x_0}\Delta x_0 = \Phi_y(t)\Delta x_0. \tag{20}$$

In accordance with the partitioning (9), x_0 incorporates parameters λ, so that sensitivity to initial conditions x_0 includes parameter sensitivity. Equations (19) and (20) describe the changes $\Delta x(t)$ and $\Delta y(t)$ in a trajectory, at time t, along the trajectory, for a given (small) change in initial conditions $\Delta x_0 =$

$$0 = g(x,y) = \begin{bmatrix} nV_1 - V_2 \\ y_2 - V_{\text{low}} + V_2 \\ y_1 - x_1 + z_1 + y_3 + T_{\text{tap}} \\ y_5 + n - n_{\max} + \frac{n_{\text{step}}}{2} \\ \begin{bmatrix} y_1 \\ y_3 - x_1 + z_1 \end{bmatrix} + m(y_2)\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} + m(y_2)\begin{bmatrix} -1 \\ x_1 - z_1 - y_3^- \end{bmatrix}\right) \end{bmatrix} \tag{17}$$

[1]Actual implementation, however, simply checks whether ($y_2 > 0$ and $y_4 > 0$) is true or not, corresponding to $u(y_2)u(y_4) = 1$ or 0, respectively.

$[\Delta x_0^t \quad \Delta \dot{x}_0^t \quad \Delta \lambda^t]^t$. The time-varying partial derivatives Φ_x and $\Phi_{\dot{x}}$ are known as *trajectory sensitivities*. An overview of the variational equations describing the evolution of these sensitivities is provided in Appendix I.

Along smooth sections of the trajectory, the trajectory sensitivities evolve according to a linear time-varying DAE system (31) and (32). For large systems, these equations have high dimension. However, the computational burden is minimal when an implicit numerical integration technique such as trapezoidal integration is used to generate the trajectory. An overview of this result is provided in Appendix II. Details can be found in [21]–[23].

More complete details of both appendices can be found in [15].

III. Implementation

A. Flexible Component Interconnection

Models of large systems are most effectively constructed using a hierarchical or modular approach. With such an approach, components are grouped together as subsystems, and the subsystems are combined to form the complete system. This allows component and subsystem models to be developed and tested independently. It also allows flexibility in interchanging models.

The interactions inherent in hybrid systems are counter to this decomposition into subsystems and components. The discussion following the tap-changer example reflects this difficulty. However, the algebraic equations of the DAIS model can be exploited to achieve the desired modularity. Each component or subsystem can be modeled autonomously in the DAIS structure, with "interface" quantities, e.g., inputs and outputs, established as algebraic variables. The components are then interconnected by introducing simple algebraic equations that "link" the interface variables. This is similar to the connections concept of [24]. Note that all interconnections are noncausal [25], i.e., no rigid input–output arrangement of components is assumed.

To illustrate this linking concept, consider a case where the mth algebraic state of component j, denoted $y_{j,m}$, is required by component k. In the model of component k, the corresponding quantity would appear as an algebraic variable $y_{k,m}$. The connection is made via the simple algebraic equation $y_{j,m} - y_{k,m} = 0$. In general, all linking can be achieved by summations of the form

$$\sum c_k y_{i,j} = 0 \tag{21}$$

where c_k is ± 1 or 0. Note that all connections are external to the component models.

The linking strategy results in an interesting structure for the complete system Jacobian

$$J = \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix} \tag{22}$$

(This Jacobian has the same structure as the matrix F_w given by (48), that is required for implicit numerical integration and for computing trajectory sensitivities.) Components contribute square blocks down the diagonal of f_x and flattened rectangular blocks along the diagonal of the upper section of g_y. The lower section of g_y is an incidence matrix, with ± 1's given by the external connections (21). The corresponding lower section of g_x is zero. Fig. 3 illustrates this structure[2]. A Jacobian structure like that of J was identified in [26], where a similar arrangement of components and connections was used in the development of an optimal power flow.

Remarks:

- Because the linking process yields the full system Jacobian J trajectory sensitivities are well defined, and may be efficiently computed, for the full system.
- The Jacobian J effectively defines a graph of the system topology. Vertices (nodes) are established by the physical network and communication networks. Edges (interconnections) are described by the connections within multi-node (network) components, together with the incidence matrix defined by the external connections.
- The structure and values of the lower connection submatrix of J, and hence F_w, are fixed for all time. This can be exploited in the factorization of F_w to improve the efficiency of solving (46) and (47). The efficiency improvement can be significant, as these equations are solved at every time step.
- In general, components and subsystems of any form can be modeled, provided they are structured with interfacing algebraic variables that can be linked to other components. Noise and/or random disturbances can be added to the model by linking components that generate random signals.

B. Matlab Implementation

The proposed modular approach to constructing hybrid systems has been implemented in Matlab [27]. In this implementation, the system is described by a data file that contains `model_data` information and (separate) `connections` details. Each component of the system contributes an entry to `model_data` that consists of the component name, initial values for x_0 and y_0, and background parameters. Links between components are fully described in `connections` using the form given by (21).

Every component is described by a `function` file that calculates values for f, g and h, and sparsely stores elements of the partial derivative matrices f_x, f_y, g_x, g_y, h_x and h_y. These component files are reusable, i.e., case independent, and reside in a component library. Relative indexing is used within the component files, as each component model is autonomous. (All connection information is externally defined.) Hence, within a model, the indexing of Jacobian elements uses only local equation and variable numbering. The simulation kernel uses these relative indices, along with knowledge of equation and variable dimensions across all models, to generate the location of each element in the full matrices, i.e., the absolute indices. The actual matrices are never built explicitly though, but rather are stored sparsely. Full details can be found in [27].

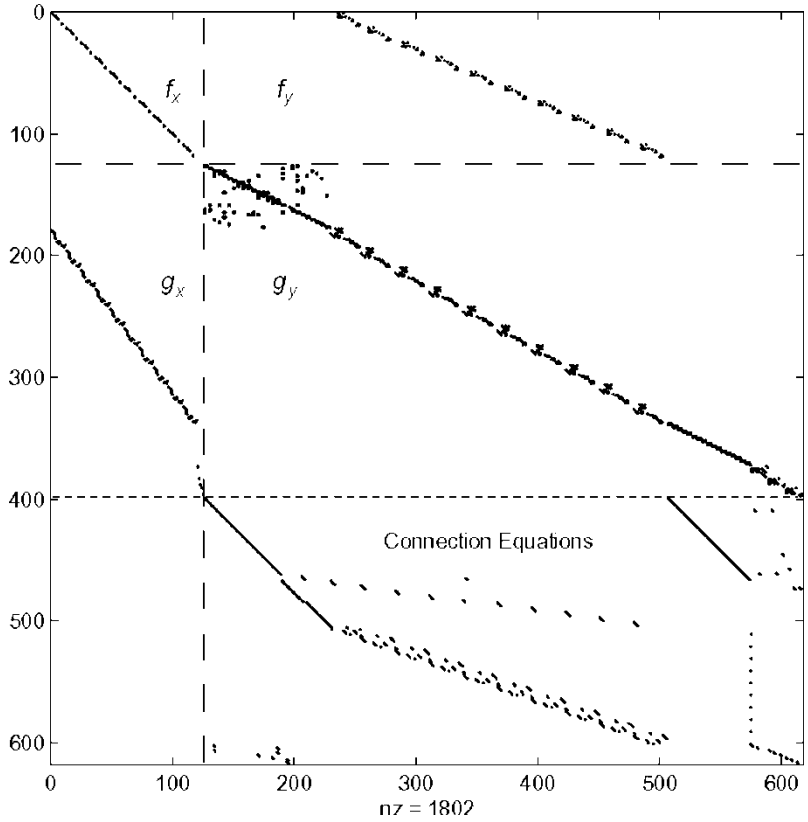

Fig. 3. Sparsity structure of the complete system Jacobian J.

C. Symbolic Differentiation

As indicated above, partial derivative matrices are calculated and stored sparsely within component files. Hand derivation of these partial derivatives can be tedious for large complicated models. Therefore, the process has been automated through the use of symbolic differentiation [27]. Symbolic manipulation has been utilized in power system simulation previously [28], though the implementation was quite different.

The generation of a component file begins with an analytical model in the DAIS form. The analytical model must be unambiguously mapped into a character representation that can be manipulated symbolically. It is also important that this mapping does not restrict the implementation of the DAIS form. Fortunately, the DAIS model structure is well suited to such translation. All elements of the model can be clearly and uniquely identified.

A Matlab function has been developed for translating the input model representation into a component file that can interact with the simulation kernel. Building the $\dot{x}$, and 0 equations involves relatively straightforward character string manipulation. Generating the partial derivatives is more challenging. Firstly, equations and variable strings are converted to symbols. Symbolic differentiation produces partial derivatives that must be simplified and converted back to strings. If the final expression is zero, the derivative is discarded, as the matrices are stored sparsely.

Component files are generated off-line and stored in the component library. Therefore, symbolic manipulation does not slow simulation speed.

D. Computation of Junction Points

Switching and reset events generically do not coincide with the time instants of the numerical integration process. However, simulation accuracy depends upon correct location, between integration time steps, of events [18].

A simple check of sign changes in trigger variables y_r and y_s at each integration step will reveal most events [15]. However, this check fails to detect events where the associated trigger variables change sign an even number of times over a time step. A more thorough search for events is required, though a tradeoff must be made between search accuracy and computational cost. An efficient approach proposed in [18] uses interpolation polynomials generated by a backward differentiation formula (BDF) integration method [14].

IV. Inverse Problems

System analysis is often tantamount to understanding the influence of parameters on system behavior, and applying that knowledge to achieve a desired outcome. The "known" information is the desired outcome. The parameters that achieve that outcome must be deduced. Because of the inverse nature of the problem, the process has traditionally involved repeated simulation of the model. This can be time consuming and frustrating, as the relationship between parameters and behavior is often not intuitively obvious.

Systematic modeling, as presented in Sections II and III, allows the development of new tools that can solve inverse problems directly, albeit via iterative techniques. The DAIS model

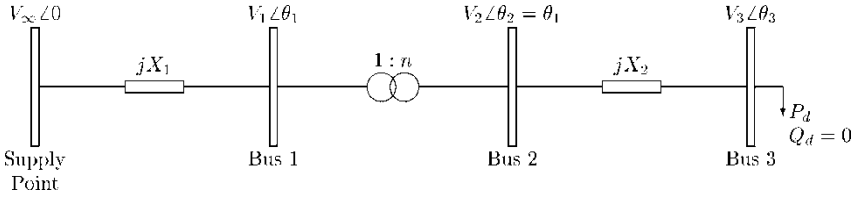

Fig. 4. Tap-changer dynamic load system.

is conducive to the efficient generation of trajectory sensitivities. Those sensitivities quantify, to the first order, the effects of parameters on dynamic behavior. They therefore underlie the development of gradient-based algorithms.

Sections IV-A–F present a range of inverse problems. Algorithms that address those problems are outlined. This list is not exhaustive, but seeks to provide an overview of the possibilities that follow from systematic modeling.

A. Parameter Uncertainty

System parameters can never be known exactly. In fact, uncertainty in some parameters, e.g., load models, can be quite high. Quantifying the effects of parameter uncertainty is not strictly an inverse problem, but illustrates the value of the extra trajectory sensitivity information available from systematic models such as the DAIS representation.

Because of the uncertainty in parameters, investigation of system behavior should (ideally) include multiple studies over a range of parameter values. However, simulation of large systems is computationally intensive. Such an investigation would be extremely time-consuming. A common practical approach is to assume that a *nominal* set of parameters provides an adequate representation of behavior over the full range of values. This may not always be a good assumption though.

A computationally feasible (though approximate) approach to repeated simulation is to generate a first-order approximation of the trajectory for each set of perturbed parameters. The first-order approximation is obtained by truncating the Taylor series expansion of the flow ϕ. Using (19) and (20) gives

$$\phi(\bar{x}_0, t) = \phi(x_0, t) + \begin{bmatrix} \Phi_x(t) \\ \Phi_y(t) \end{bmatrix} (x_0 - x_0) \qquad (23)$$

where $\Phi_x(t)$, $\Phi_y(t)$ are computed along the nominal trajectory $\phi(x_0, t)$. Therefore, if the trajectory sensitivities $\Phi_x(t)$, $\Phi_y(t)$ are available for a nominal trajectory, then (23) can be used to provide a good estimate of trajectories $\phi(\bar{x}_0, t)$ corresponding to other (nearby) parameter sets $\bar{x}_0$. (Recall the parameters λ are embedded in x_0.)

The computational burden involved in generating the approximate trajectories is negligible. Given the nominal trajectory and associated trajectory sensitivities, new (approximate) trajectories can be obtained for many parameter sets. Therefore, a Monte Carlo technique can be employed to quantify the uncertainty in a trajectory:

- parameter sets are randomly generated;
- first-order approximations are obtained using (23).

The simple system of Fig. 4 can be used to illustrate the Monte Carlo process. This system includes the tap changing transformer AVR of Section II-F, and a dynamic load. The dark line of Fig. 5 shows the nominal trajectory corresponding to the tripping of a supply feeder (simulated by an increase in impedance X_1). In response to that event, voltage drops instantaneously, causing an initial reduction in load. Load tries to recover, further stressing the system and driving the voltage lower. Eventually, after a time delay T_{tap}, the transformer taps. This process continues until the transformer reaches its maximum tap position.

Fig. 5 also shows the bound obtained from the Monte Carlo process. The parameters T_{tap} and the load time constant were uniformly distributed over a range of $\pm 10\%$ around their nominal values. The bound was obtained using 200 randomly chosen sets of parameters. Further details can be found in [29].

Statistics quantifying the uncertainty in system behavior due to parameter uncertainty can be obtained from the Monte Carlo simulation. For example, it is possible to estimate the probability that a disturbance would initiate protection operation or that the voltage would fall below some predetermined threshold.

Another approach to assessing the significance of parameter uncertainty is via worst case analysis [30]. This involves finding the values of parameters (within specified bounds) that induce the greatest deviation in system variables, for example voltages. The algorithm can be formulated as a constrained optimization, and is truly an inverse problem. Such optimization problems are discussed as part of later inverse problems.

B. Parameter Estimation

System-wide measurements of power system disturbances are frequently used in post-mortem analysis to gain a better understanding of system behavior [6]–[8], [31]. In undertaking such studies, measurements are compared with the behavior predicted by a model. Differences are used to tune the model, i.e., adjust parameters, to obtain the best match between the model and the measurements. This process requires a systematic approach to

1) identifying *well-conditioned* parameters that can be estimated reliably from the available measurements;
2) obtaining the best estimate for those parameters.

It is shown in [8] that trajectory sensitivities can be used to guide the search for well-conditioned parameters, i.e., parameters that are good candidates for reliable estimation. Large trajectory sensitivities imply the corresponding parameters have leverage in altering the model trajectory to better match the measured response. Small trajectory sensitivities, on the other hand, imply that large changes in parameter values would be required to significantly alter the trajectory. Parameters in the former category are well-conditioned, whereas the latter parameters are ill-conditioned. Only parameters that influence measured states can be identified. A parameter may have a significant influence

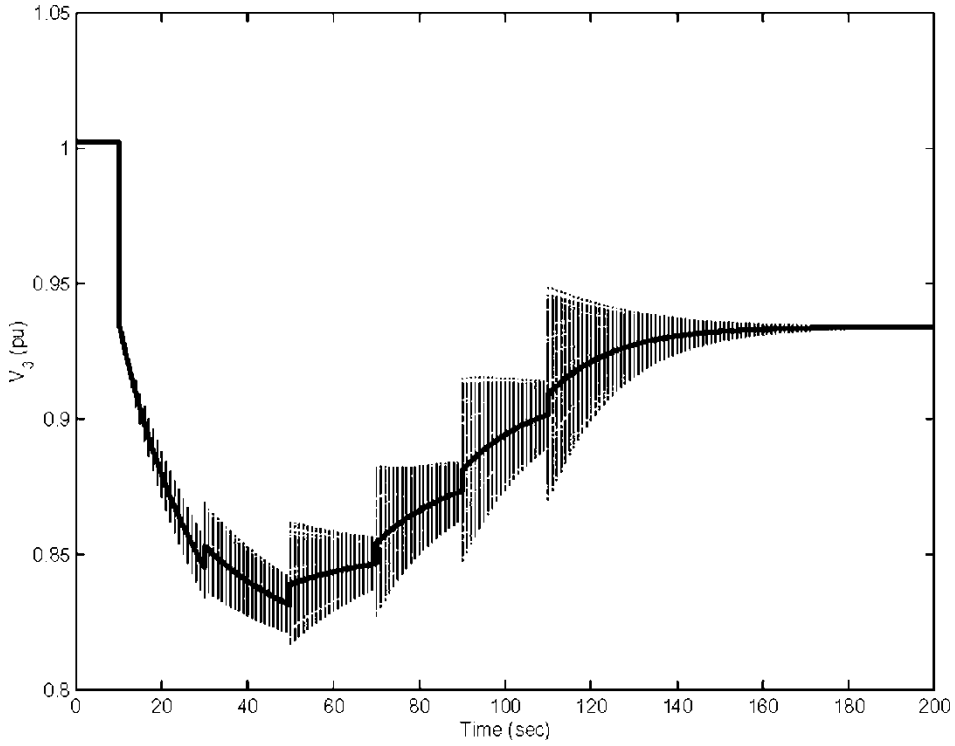

Fig. 5. Trajectory bounds.

on system behavior, but if that influence is not observable in the measured states, then the parameter is not identifiable. The concept of identifiability is explained more formally in [32].

A parameter estimation algorithm that is based on a Gauss–Newton iterative procedure is presented in [8]. The algorithm minimizes a nonlinear least-squares cost

$$\mathcal{V}(\theta) = \frac{1}{2}\|\breve{y}(\theta) - ms\|_2^2 \tag{24}$$

where ms are the sampled measurements of the disturbance, $\breve{y}(\theta)$ are the flows provided by the model that correspond to the measured quantities, and θ are the unknown parameters. This minimization can be achieved (locally at least) by the iterative scheme

$$\begin{aligned} S(\theta^j)^t S(\theta^j)\Delta\theta^j &= -S(\theta^j)^t(\breve{y}(\theta^j) - ms) \\ \theta^{j+1} &= \theta^j - \epsilon^j \Delta\theta^j \end{aligned} \tag{25}$$

where ϵ^j is a scalar that determines the parameter update step size.[3] The matrix S is built from the trajectory sensitivities $\breve{y}_\theta$, i.e., sensitivity of model flows $\breve{y}$ to parameters θ. The invertibility of $S^t S$ relates directly to identifiability [32].

Remarks:

1) Parameter estimation via (25) is not restricted to smooth systems. In fact, it is possible to estimate parameters that underlie event descriptions (provided measurements capture an occurrence of the event.)
2) For large systems, feasibility of the Gauss–Newton algorithm is dependent upon efficient computation of trajectory sensitivities. This underlines the importance of systematic modeling, as provided by the DAIS model.

C. Boundary Value Problems

It is interesting to consider boundary value problems of the form

$$r(x_0, x(t_f)) = 0 \tag{26}$$

where t_f is the final time, and $x(t)$ is the trajectory that starts from x_0 and is generated by (4) and (5). The initial values x_0 are variables that must be adjusted to satisfy r. (Though r may directly constrain some elements of x_0.) To establish the solution process, (26) may be rewritten

$$r(x_0, \phi_x(x_0, t_f)) = 0 \tag{27}$$

which has the form $\tilde{r}(x_0) = 0$. Boundary value problems are solved by shooting methods [34], [35], which are a combination of Newton's method for solving (27) along with numerical integration for obtaining the flow ϕ_x. Newton's method requires the Jacobian

$$J = \frac{\partial r}{\partial x_0} + \frac{\partial r}{\partial \phi_x}\Phi_x(t_f) \tag{28}$$

which is dependent upon the trajectory sensitivities evaluated at t_f. Boundary value problems *per se* are uncommon in power systems. However an application of increasing importance is the calculation of limit cycles (sustained oscillations). Oscillations have been observed in a variety of power systems, from generation [36] to distribution [37]. In this latter case, oscillations were driven by interactions between transformer tapping and capacitor switching. Systematic modeling, as provided by the DAIS representation, is vital for capturing such hybrid system phenomena.

To solve for limit cycles, (27) can be written as

$$x_0 - \phi_x(x_0, T) = 0$$

where x_0 lies on the limit cycle and T is its period. [4] The solution of this boundary value problem via a shooting method requires $\Phi_x(T)$, which is exactly the Monodromy matrix [34], [38]. The eigenvalues of this matrix determine the stability of the limit cycle.

D. Grazing Bifurcations

When a system trajectory encounters the operating characteristic of a protection device, a trip signal is sent to circuit breakers. If the trajectory almost touches the operating characteristic but just misses, no trip signal is issued. The bounding (separating) case corresponds to the trajectory grazing, i.e., just touching, the operating characteristic, but not crossing it. This is a form of global bifurcation; it separates two cases that have significantly different outcomes. Numerous names exist for this phenomenon, including switching-time bifurcation and grazing bifurcation. There is a close relationship to border-collision bifurcations.

Examples of such bifurcations can be found in many application areas. They are particularly important in power electronic circuits, where zero-crossings are fundamental to control strategies, and to the switching of self-commutating devices [3], [39]. In fact it has been shown that grazing bifurcations can provide a path to chaos in simple dc–dc converters [40].

Identifying the critical values of parameters that correspond to a grazing bifurcation is an inverse problem. Let the switching characteristic (border) be described by $b(x, y) = 0$. A trajectory will be tangential to that characteristic at the point $[x^* y^*]^t = \phi(x_0^*, t^*)$ given by

$$\begin{matrix} b(x^*, y^*) = 0 \\ \left(b_x - b_y g_y^{-1} g_x\right)\big|_{(x^*, y^*)} f(x^*, y^*) = 0. \end{matrix}$$

The critical values of parameters are given by x_0^*. This is a special form of boundary value problem. Shooting methods provide the basis for gradient-based algorithms. Further details can be found in [41].

E. Dynamic Embedded Optimization

Optimization problems arise frequently in the analysis of power system dynamics. Examples range from tuning generator AVR/PSSs to determining the optimal location, amount and switching times for load shedding [42]. Most problems can be formulated using a Bolza form of objective function

$$\min_{\theta, t_f} \mathcal{J}(x, y, \theta, t_f) \tag{29}$$

where

$$\mathcal{J} = \varphi(x(t_f), y(t_f), \theta, t_f) + \int_{t_0}^{t_f} \psi(x(t), y(t), \theta, t) dt \tag{30}$$

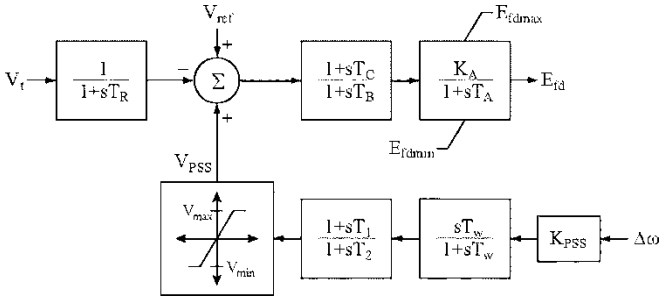

Fig. 6. AVR/PSS block representation.

θ are the design parameters, i.e., the parameters adjusted to achieve the objective, and t_f is the final time.

The solution of (29) for hybrid systems is complicated by discontinuous behavior at events. However, those complications largely disappear under the assumption that the order of events does not change as θ and t_f vary, i.e., no grazing bifurcations occur. This assumption is common throughout the literature, though it is expressed in various ways: transversal crossings of triggering hypersurfaces are assumed in [10], existence of trajectory sensitivities is assumed in [43], and [44] assumes all flows have the same *history*. All statements are equivalent.

Under that assumption, and other mild assumptions, it is concluded in [44] that if $\mathcal{J}$ is continuous in its arguments then a solution to (29) exists. Further, [43] shows that if $\mathcal{J}$ is a smooth function of its arguments, then it is continuously differentiable with respect to θ and t_f. The minimization can therefore be solved using gradient-based methods. Trajectory sensitivities, as provided by the DAIS model, underlie the gradient information.

If the event ordering assumption is not satisfied, $\mathcal{J}$ may be discontinuous. The optimization problem then takes on a combinatorial nature, as each continuous section of $\mathcal{J}$ must be searched for a local minimum.

Nontraditional design capabilities arise from embedding the DAIS model within the optimization framework (29) and (30). To illustrate, consider the generator AVR/PSS shown in Fig. 6. The clipping limits on the PSS output V_{PSS} and the anti-windup limits on the field voltage E_{fd} introduce events that can be captured by the DAIS model. Typically, PSS output limits are assigned on an *ad hoc* basis. However, [45] determines optimal limit values by establishing a cost function (30) that maximize damping whilst minimizing deviations in the generator terminal voltage. Fig. 7 compares optimal performance with that obtained using standard limit values. (Note that only the limit values differ between these two cases. All other parameters are fixed.)

Other optimization problems do not naturally fit the Bolza form of objective function (30). Cascaded tap-changing transformers provide an interesting example [46]. Minimizing the number of tap change operations is equivalent to minimizing the number of crossings of triggering hypersurfaces. Such a problem, by definition, does not satisfy the earlier assumption requiring constant ordering of events. This minimization is best addressed using switching control design techniques [47], though the solution process is not yet well established.

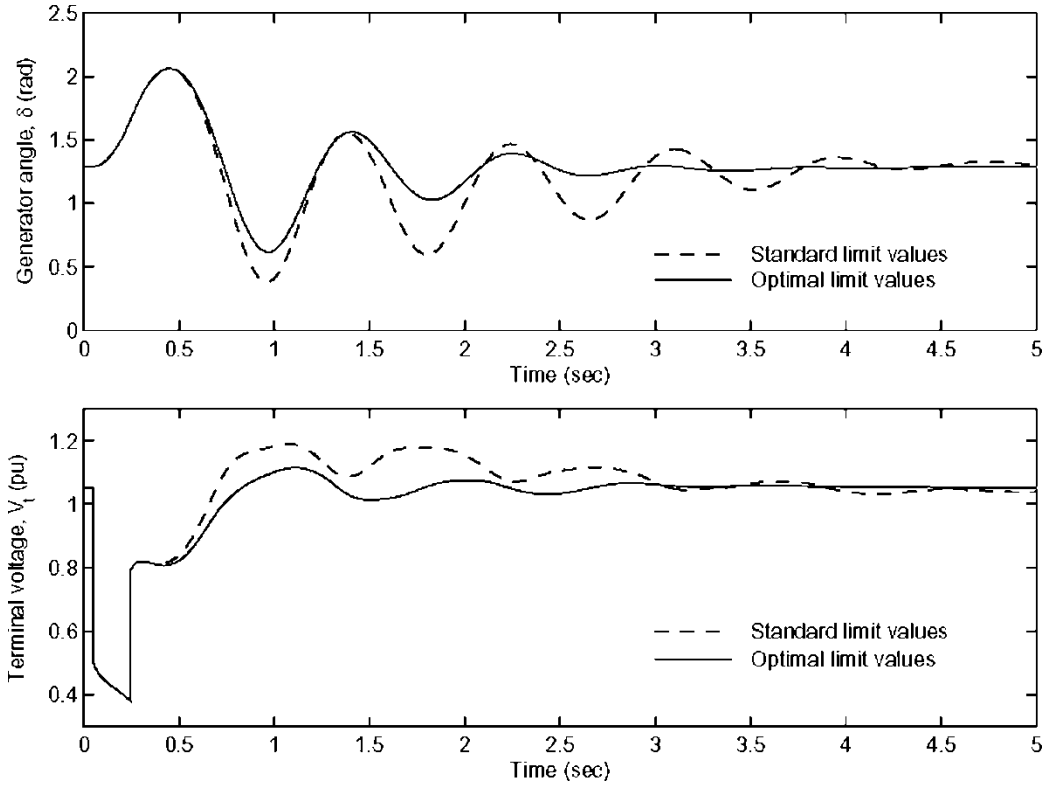

Fig. 7. Generator angle response.

F. Technical Issues

Changes in event ordering, as discussed in the Section IV-E, influence all gradient-based algorithms for solving inverse problems. The effect is similar to power flow solution when reactive power limits change status. Algorithms usually converge, though with a slower convergence rate and a reduced region of convergence.

Another interesting aspect of hybrid systems is that trajectories may not be unique in reverse time even though they are unique in forward time. In other words, the same final value $\phi(x_0, t_f)$ can be reached from different initial values x_0. In such cases, the trajectory sensitivity matrix $\Phi_x(t_f)$ is singular. This matrix underlies solution algorithms for numerous inverse problems, for example (28). An approach to addressing this issue is to decompose x_0 into components that influence $\phi(x_0, t_f)$ and those that do not. Attention is then restricted to the former group. This is an area of on-going research.

V. Conclusion

The response of power systems to large disturbances often involves interactions between continuous dynamics and discrete events. Such behavior can be captured by a hybrid automaton. The automaton has a graph representation, with vertices corresponding to modes (discrete states) of system operation, and edges describing transitions induced by events. The system responds smoothly within each mode. Cascading failure of a power system results in a path traversing the graph.

Hybrid systems can be modeled by a set of DAE equations, modified to incorporate impulse (state reset) action and constraint switching. This DAIS model is a realization of the hybrid automaton.

Models of large systems are most effectively constructed using a modular or object-oriented approach. However the interactions inherent in hybrid systems make that difficult to achieve. The desired modularity can be achieved in a practical way with the DAIS model though. Components and/or subsystems are modeled autonomously, with connections established via simple algebraic equations. The Jacobian of the DAIS model effectively defines a graph of system topology. Furthermore, the object-oriented model structure is amenable to symbolic manipulation.

Systematic modeling allows the development of tools for solving inverse problems, including parameter uncertainty and estimation, boundary value problems, grazing bifurcation analysis and dynamic embedded optimization. The DAIS model is conducive to the efficient computation of trajectory sensitivities. Those sensitivities underlie gradient-based algorithms for addressing inverse problems.

Many power system simulators now use implicit numerical integration techniques, such as trapezoidal integration. Modification of such simulators to compute trajectory sensitivities along smooth sections of a trajectory is therefore quite straightforward. However, mapping trajectory sensitivities through events requires cleanly defined event triggering conditions, as provided by the DAIS model. Incorporating that feature into existing simulators may be a challenging, though certainly surmountable, problem.

APPENDIX I
TRAJECTORY SENSITIVITY EQUATIONS

Away from events, where system dynamics evolve smoothly, the sensitivities Φ_x and Φ_y are obtained by differentiating (13) and (14) with respect to x_0. This gives

$$\dot{\Phi}_x = f_x(t)\Phi_x + f_y(t)\Phi_y \tag{31}$$

$$0 = g_x(t)\Phi_x + g_y(t)\Phi_y \tag{32}$$

where $f_x = \partial f/\partial x$, and likewise for the other Jacobian matrices. Note that f_x, f_y, g_x, g_y are evaluated along the trajectory, and hence are time varying matrices. It is shown in Appendix II that the solution of this (potentially high order) linear time-varying DAE system can be obtained as a by-product of solving the original DAE system (13) and (14).

Initial conditions for Φ_x are obtained from (11) as

$$\Phi_x(t_0) = I \tag{33}$$

where I is the identity matrix. Initial conditions for Φ_y follow directly from (32)

$$0 = g_x(t_0) + g_y(t_0)\Phi_y(t_0). \tag{34}$$

Equations (31) and (32) describe the evolution of the sensitivities Φ_x and Φ_y between events. However, at an event, the sensitivities are often discontinuous. It is necessary to calculate *jump conditions* describing the step change in Φ_x and Φ_y. For clarity, consider a single switching/reset event, so the model (4)–(8) reduces to the form [5]

$$\dot{x} = f(x, y) \tag{35}$$

$$0 = \begin{cases} g^-(x,y) & s(x,y) < 0 \\ g^+(x,y) & s(x,y) > 0 \end{cases} \tag{36}$$

$$x^+ = h(x^-, y^-) \quad s(x,y) = 0. \tag{37}$$

Let $(x(\tau), y(\tau))$ be the point where the trajectory encounters the hypersurface $s(x,y) = 0$, i.e., the point where an event is triggered. This point is called the *junction point* and τ is the *junction time*. Assume that the trajectory encounters the triggering hypersurface transversally.

Just prior to event triggering, at time τ^-, we have

$$x^- = x(\tau^-) = \phi_x(x_0, \tau^-) \tag{38}$$

$$y^- = y^-(\tau^-) = \phi_y(x_0, \tau^-) \tag{39}$$

where $g^-(x^-, y^-) = 0$. Similarly, x^+, y^+ are defined for time τ^+, just after the event has occurred. It is shown in [15] that the jump conditions for the sensitivities Φ_x are given by

$$\Phi_x(\tau^+) = h_x^* \Phi_x(\tau^-) - (f^+ - h_x^* f^-)\tau_{x_0} \tag{40}$$

where

$$h_x^* = \left(h_x - h_y (g_y^-)^{-1} g_x^-\right)\Big|_{\tau^-} \tag{41}$$

$$\tau_{x_0} = -\frac{\left(s_x - s_y (g_y^-)^{-1} g_x^-\right)\Big|_{\tau^-} \Phi_x(\tau^-)}{\left(s_x - s_y (g_y^-)^{-1} g_x^-\right)\Big|_{\tau^-} f^-} \tag{42}$$

$$f^- = f(x(\tau^-), y^-(\tau^-)) \tag{43}$$

$$f^+ = f(x(\tau^-), y^-(\tau^-)). \tag{44}$$

The sensitivities Φ_y immediately after the event are given by

$$\Phi_y(\tau^+) = -\left(g_y^+(\tau^+)\right)^{-1} g_x^+(\tau^+)\Phi_x(\tau^-). \tag{45}$$

Following the event, i.e., for $t > \tau^+$, calculation of the sensitivities proceeds according to (31) and (32), until the next event is encountered. The jump conditions provide the initial conditions for the post-event calculations.

Actual power systems involve many discrete events. The more general case follows naturally, and is presented in [15].

APPENDIX II
EFFICIENT TRAJECTORY SENSITIVITY COMPUTATION

Consider the DAE system (13) and (14) which describes the behavior over the periods between events. The trapezoidal approach to numerical integration approximates the differential equation (13) by a set of algebraic difference equations. These algebraic equations are coupled with the original algebraic equations (14) giving

$$x^{k+1} = x^k + \frac{\eta}{2}\left(f(x^k, y^k) + f(x^{k+1}, y^{k+1})\right) \tag{46}$$

$$0 = g(x^{k+1}, y^{k+1}) \tag{47}$$

where the superscripts k and $k+1$ index the time instants t_k and t_{k+1}, respectively, and $\eta = t_{k+1} - t_k$ is the integration time step. (The subscript η has been dropped from g_η for clarity.) Equations (46) and (47) describe the evolution of the states x, y from time instant t_k to the next time instant t_{k+1}.

Notice that (46) and (47) form a set of implicit nonlinear algebraic equations. To solve for x^{k+1}, y^{k+1} given x^k, y^k requires the use of a nonlinear equation solver. Newton-based iterative techniques are commonly used. The solution process involves forming and factorizing the Jacobian

$$F_x = \begin{bmatrix} \frac{\eta}{2} f_x - I & \frac{\eta}{2} f_y \\ g_x & g_y \end{bmatrix} \tag{48}$$

Now consider the sensitivity equations (31) and (32). Using trapezoidal integration, they are approximated by

$$\Phi_x^{k+1} - \Phi_x^k + \frac{\eta}{2}\left(f_x^k \Phi_x^k + f_y^k \Phi_y^k + f_x^{k+1}\Phi_x^{k+1} + f_y^{k+1}\Phi_y^{k+1}\right) \tag{49}$$

$$0 = g_x^{k+1}\Phi_x^{k+1} + g_y^{k+1}\Phi_y^{k+1}. \tag{50}$$

Rearranging gives

$$\begin{bmatrix} \frac{\eta}{2} f_x^{k+1} - I & \frac{\eta}{2} f_y^{k+1} \\ g_x^{k+1} & g_y^{k+1} \end{bmatrix} \begin{bmatrix} \Phi_x^{k+1} \\ \Phi_y^{k+1} \end{bmatrix} = \begin{bmatrix} -\frac{\eta}{2}\left(f_x^k \Phi_x^k + f_y^k \Phi_y^k\right) - \Phi_x^k \\ 0 \end{bmatrix} \tag{51}$$

Therefore, Φ_x^{k+1} and Φ_y^{k+1} are obtained as the solution of a linear matrix equation. But notice that the matrix to be inverted in solving (51) is exactly the Jacobian (48) used in solving for x^{k+1}, y^{k+1}. Because that matrix has already been built and factorized to calculate x^{k+1}, y^{k+1}, the solution of (51) involves little extra computation.

REFERENCES

[1] L. G. Perez, A. J. Flechsig, and V. Venkatasubramanian, "Modeling the protective system for power system dynamic analysis," *IEEE Trans. Power Syst.*, vol. 9, pp. 1963–1973, Nov. 1994.
[2] M. S. Ćalović, "Modeling and analysis of under-load tap-changing transformer control systems," *IEEE Trans. Power App. Syst.*, vol. PAS-103, pp. 1909–1915, July 1984.
[3] I. Dobson, "Stability of ideal thyristor and diode switching circuits," *IEEE Trans. Circuits Syst. I*, vol. 42, pp. 517–529, Sept. 1995.
[4] M. D. Lemmon, K. X. He, and I. Markovsky, "Supervisory hybrid systems," *IEEE Control Syst. Mag.*, vol. 19, pp. 42–55, Aug. 1999.
[5] A. Tarantola, *Inverse Problem Theory*. Amsterdam, The Netherlands: Elsevier Science, 1987.
[6] J. F. Hauer, W. A. Mittelstadt, W. H. Litzenberger, C. Clemens, D. Hamai, and P. N. Overholt, "Wide area measurements for real-time control and operation of large electric power systems," DOE, Washington, DC, DOE Fin. Rep., 1999.
[7] D. N. Kosterev, C. W. Taylor, and W. A. Mittelstadt, "Model validation for the August 10,1996 WSCC system outage," *IEEE Trans. Power Syst.*, vol. 14, pp. 967–979, Aug. 1999.
[8] I. A. Hiskens, "Nonlinear dynamic model evaluation from disturbance measurements," *IEEE Trans. Power Syst.*, vol. 16, pp. 702–710, Nov. 2001.
[9] C. J. Tomlin and M. R. Greenstreet, Eds., "Hybrid systems: Computation and control. Proceedings 5th international workshop," in *HSCC 2002* New York, 2002, vol. 2289.
[10] M. S. Branicky, V. S. Borkar, and S. K. Mitter, "A unified framework for hybrid control: Model and optimal control theory," *IEEE Trans. Automat. Contr.*, vol. 43, pp. 31–45, Jan. 1998.
[11] A. van der Schaft and H. Schumacher, *An Introduction to Hybrid Dynamical Systems*. London, U.K.: Springer-Verlag, 2000.
[12] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Norwell, MA: Kluwer, 1999.
[13] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, pp. 541–580, Apr. 1989.
[14] K. E. Brenan, S. L. Campbell, and L. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Philadelphia, PA: SIAM, 1995.
[15] I. A. Hiskens and M. A. Pai, "Trajectory sensitivity analysis of hybrid systems," *IEEE Trans. Circuits Syst. I*, vol. 47, pp. 204–220, Feb. 2000.
[16] S. Galán, W. F. Feehery, and P. I. Barton, "Parametric sensitivity functions for hybrid discrete/continuous systems," *Appl. Numer. Math.*, vol. 31, pp. 17–47, 1999.
[17] P. J. Mosterman, "Implicit modeling and simulation of discontinuities in physical system models," in *Proc. 4th Int. Conf. Automation of Mixed Processes: Hybrid Dynamic Systems, ADPM'00*, Dortmund, Germany, Sept. 2000, pp. 35–40.
[18] T. Park and P. I. Barton, "State event location in differential-algebraic models," *ACM Trans. Model. Comput. Simul.*, vol. 6, no. 2, pp. 137–165, 1996.
[19] P. J. Mosterman, M. Otter, and H. Elmqvist, "Modeling Petri nets as local constraint equations for hybrid systems using modelica," in *Proc. Summer Computer Simulation Conf.*, Reno, NV, July 1998.
[20] P. M. Frank, *Introduction to System Sensitivity Theory*. New York: Academic, 1978.
[21] W. F. Feehery, J. E. Tolsma, and P. I. Barton, "Efficient sensitivity analysis of large-scale differential-algebraic systems," *Appl. Numer. Math.*, vol. 25, pp. 41–54, 1997.
[22] S. Li , L. Petzold, and W. Zhu , "Sensitivity analysis of differential-algebraic equations: A comparison of methods on a special problem," *Appl. Numer. Math.*, vol. 32, pp. 161–174, 2000.
[23] D. Chaniotis, M. A. Pai, and I. A. Hiskens, "Sensitivity analysis of differential-algebraic systems using the GMRES method—Application to power systems," in *Proc. IEEE Int. Symp. Circuits Systems*, vol. 2, Sydney, Australia, May 2001, pp. 117–120.
[24] H. Elmqvist, "A structured model language for large continuous systems," Ph.D. dissertation, Dept. . Automatic Control, Lund Inst. Technol. , Lund, Sweden, 1978.
[25] H. Elmqvist, S. E. Mattsson, and M. Otter, "Modelica—A language for physical system modeling, visualization and interaction," in *Proc. IEEE Symp. Computer-Aided Control System Design*, Maui, HI, Aug. 1999, pp. 630–639.
[26] R. Bacher, "Symbolically assisted numeric computations for power system software development," in *Proc. 13th Power Syst. Comput. Conf.*, Trondheim, Norway, June 1999, pp. 5–16.
[27] I. A. Hiskens and P. J. Sokolowski, "Systematic modeling and symbolically assisted simulation of power systems," *IEEE Trans. Power Syst.*, vol. 16, pp. 229–234, May 2001.
[28] F. L. Alvarado and Y. Liu, "General purpose symbolic simulation tools for electric networks," *IEEE Trans. Power Syst.*, vol. 3, pp. 689–697, May 1988.
[29] I. A. Hiskens, M. A. Pai, and T. B. Nguyen, "Bounding uncertainty in power system dynamic simulations," in *Proc. IEEE PES Winter Meeting*, Singapore, Jan. 2000.
[30] M. W. Tian and C.-J. R. Shi, "Worst case tolerance analysis of linear analog circuits using sensitivity bands," *IEEE Trans. Circuits Syst. I*, vol. 47, pp. 1138–1145, Aug. 2000.
[31] R. H. Craven, T. George, G. B. Price, P. O. Wright, and I. A. Hiskens, "Validation of dynamic modeling methods against power system response to small and large disturbances," in *Proc. CIGRÉ General Session*, Paris, France, Aug. 1994.
[32] I. A. Hiskens, "Identifiability of hybrid system models," in *Proc. 9th IEEE Conf. Control Applications*, Anchorage, AK, Sept. 2000.
[33] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed. Baltimore, MD: John Hopkins Univ. Press, 1989.
[34] R. Seydel, *Practical Bifurcation and Stability Analysis*, 2nd ed. New York: Springer-Verlag, 1994.
[35] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*. New York: Springer, 1980.
[36] K. Kim, H. Schättler, V. Venkatasubramanian, J. Zaborszky, and P. Hirsch, "Methods for calculating oscillations in large power systems," *IEEE Trans. Power Syst.*, vol. 12, pp. 1639–1648, Nov. 1997.
[37] P. Gavey, "Voltage stability in the Mid North Coast. investigation of voltage oscillation occurrence on 26th March 1993," Dept. Elect. Comput. Engi., Univ. Newcastle, Newcastle, Australia, Final Year Project Rep., 1995.
[38] I. A. Hiskens, "Stability of limit cycles in hybrid systems," in *Proc. 34th Hawaii Int. Conf. System Sciences*, Maui, HI, Jan. 2001.
[39] S. Jalali, I. Dobson, R. H. Lasseter, and G. Venkataramanan, "Switching time bifurcations in a thyristor controlled reactor," *IEEE Trans. Circuits Syst. I*, vol. 43, pp. 209–218, Mar. 1996.
[40] M. di Bernardo, "The complex behavior of switching devices," *IEEE Circuits Syst. Newslett.*, vol. 10, pp. 1–13, Dec. 1999.
[41] V. Donde and I. A. Hiskens, "Shooting for border collision bifurcations in hybrid systems," in *Proc. 42nd IEEE Conf. Decision Control*, Maui, HI, Dec. 2003.
[42] C. Moors and T. Van Cutsem, "Determination of optimal load shedding against voltage instability," in *Proc. 13th Power Systems Computation Conf.*, vol. 2, Trondheim, Norway, June 1999, pp. 993–1000.
[43] S. Galán and P. I. Barton, "Dynamic optimization of hybrid systems," *Comput. Chem. Eng.*, vol. 22, no. Suppl., pp. S183–S190, 1998.
[44] B. Piccoli, "Hybrid systems and optimal control," in *Proc. 37th IEEE Conf. Decision Control*, vol. 1, Tampa, FL, Dec. 1998, pp. 13–18.
[45] I. A. Hiskens, "Systematic tuning of nonlinear power system controllers," in *Proc. 2002 IEEE Int. Conf. Control Applications*, vol. 1, Glasgow, Scotland, Sept. 2002, pp. 19–24.
[46] M. Larsson, "Coordinated voltage control in electric power systems," Ph.D. dissertation, Dept. Ind. Elect. Eng. Automat., Lund Inst. Technol., Lund, Sweden, 2000.
[47] R. W. Brockett, "Minimum attention control," in *Proc. 36th IEEE Conf. Decision Control*, 1997, pp. 2628–2632.