Power System Dynamic Response Calculations

BRIAN STOTT, MEMBER IEEE

Invited Paper

Abstract—Engineers in the power industry face the problem that, while stability is increasingly a limiting factor in secure system operation, the simulation of system dynamic response is grossly overburdening on present-day digital computing resources. Each individual response case involves the step-by-step numerical solution in the time domain of perhaps thousands of nonlinear differential-algebraic equations, at a cost of up to several thousand dollars. A high premium is thus to be placed on the use of the most efficient and reliable modern calculation techniques.

This paper is a critical tutorial-review of the calculation methods used routinely or investigated for use by the industry. It concentrates on solution concepts and computational techniques rather than on the analysis of the numerical methods. Details of system modeling are only emphasized when they affect the choice of solution method. The paper concludes with a view of the state of the art and a prediction of future directions of development.

NOMENCLATURE

General

t	Time.
h	Integration step length.
$\mathscr{I}$	Identity (unit) matrix.
J	Jacobian matrix.

Vectors and matrices are denoted by boldface font, e.g., $\mathbf{x}$. First derivative with respect to time is denoted by dot, e.g., $dx/dt = \dot{x}$. Subscripts $n-1, n$ denote values at times t_{n-1}, t_n. Superscript c denotes that the vector or matrix is complex. Superscript e denotes a real vector or matrix resulting from the expansion of the original complex form into its real and imaginary parts. Subscripts d, q and re, im denote components expressed in the machine rotor and network (complex) reference frames, respectively.

Main Equation Sets

y	state variables in differential equations.
x	State variables in algebraic equations.
f	Functions defining differential equations.
g	Functions defining algebraic equations.
u	Subset of x that appears in f } interface variables.
E	Subset of y that appears in g } interface variables.

Network

V	Bus (nodal) voltages.
I	Bus (nodal) injected currents.
Y	Bus (nodal) admittance matrix.
Z	Bus (nodal) impedance matrix $= Y^{-1}$.

Machine

E'	Stator internal voltage.
I_s	Stator current.
V	Terminal voltage (also an element of bus voltage vector $\mathbf{V}$).
X'	Stator reactance.
δ	Rotor angle.
P_e	Air gap power.
P_m	Turbine power output.
E_f	Field voltage.
$\Delta\|V\|$	Deviation of $\|V\|$ from reference setting value.

I. INTRODUCTION

THE CONVENTIONAL power-system stability study computes the system response to a sequence of large disturbances, usually a network short circuit, followed by protective branch-switching operations. The process is a direct simulation in the time domain of duration varying between say 1 s and 20 min or more. Different components of the power system have their greatest influences on stability at different stages of the response, and the system modeling reflects this fact. It has become convenient to recognize three modes of simulation, called short, mid, and long term, covering the post-disturbance times of up to 8 s, 5 min, and 20 min, respectively. The short-term models emphasize the rapidly responding system electrical components, while the long-term models are more concerned with representing the slowly oscillatory system power balance, assuming that the rapid electrical transients have damped out. A different method of classification which is now rather blurred calls the short-term problem "transient stability," while anything longer is called "dynamic stability."

In the engineering applications, it is frequently desirable to make many response simulations to calculate, for example, the effects of different fault locations and types, automatic switching, initial power system operating states, and in design studies, different network, machine and control-system characteristics. However, the volume of computation imposes very severe constraints on such studies. For a large system, thousands of equations must be solved and each case can take an hour of CPU time on a large modern computer. Hence, there is always considerable incentive to find superior calculation methods.

Recent years have seen significant improvements in the application of numerical and computational methods to the problem. Also, hardware developments are continuing to reduce the cost of computation spectacularly. Unfortunately, the computational demands of stability studies are rising rapidly at the same time. As power and interconnection levels

Manuscript received May 1, 1978; revised October 6, 1978.
The author is with CEPEL, Ilha do Fundão, Rio de Janeiro, Brazil.

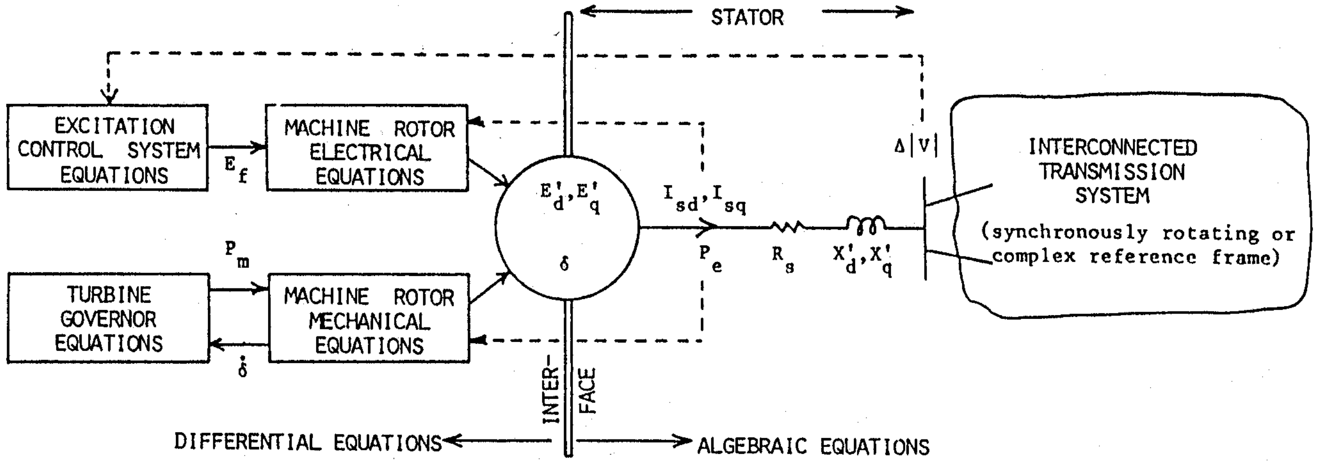
STATOR
EXCITATION CONTROL SYSTEM EQUATIONS
E_f
MACHINE ROTOR ELECTRICAL EQUATIONS
E'_d, E'_q
δ
I_{sd}, I_{sq}
P_e
R_s
X'_d, X'_q
$\Delta|V|$
INTERCONNECTED TRANSMISSION SYSTEM
(synchronously rotating or complex reference frame)
P_m
TURBINE GOVERNOR EQUATIONS
MACHINE ROTOR MECHANICAL EQUATIONS
$\dot{\delta}$
INTER-FACE
DIFFERENTIAL EQUATIONS
ALGEBRAIC EQUATIONS

Fig. 1. Schematic of transient model of synchronous generator connected to transmission network.

increase, the role of stability as a limiting design and operating factor increases. It becomes necessary to solve larger systems, with increased detail of modeling, over longer response times, more frequently. Therefore, the stability calculation remains a painful number-crunching exercise that is not getting any easier.

Stability is one of the three major routinely performed power-system computations, the other two being load-flow and fault analysis. The stability problem is much the most complex of the three in terms of modeling and solution methods. Power-system dynamic modeling is a very large topic in its own right. In relation to computational economy, it is always desirable to choose the computationally simplest models that will simulate the system with adequate accuracy. There is some degree of concensus, albeit incomplete, in the industry about what constitute acceptable models for various types of study. References [3]-[15] all contain very useful reviews of industry modeling practices.

For its own part, the present review concentrates on the different ways of solving the power-system equations, introducing modeling details only to the extent that they affect the choice of numerical methods. The reference section, while not intended to provide a complete bibliography, lists many papers that describe specific power-system dynamic response calculation methods. In addition to these, reference [1] is an excellent general review of solution principles that so far has dated very little with age. Reference [2] is a research report containing a lot of important and often detailed comparative material. It has been used as a primary source for the revision of [54] to produce this review paper.

Mid-term and long-term simulations are relatively recent developments. The traditional stability problem is in the short-term or transient mode, on which the vast majority of effort and literature has concentrated and with which the author has had most experience. Thus this review will be biased throughout towards the short-term problem. However, the structures of the various models have many features in common, and most of the numerical techniques are relevant to all the simulation modes. The solution principles to be exposed here should provide a guide to the treatment of special and/or non-conforming models in whichever mode.

The reader is assumed to have some acquaintance with the numerical solutions of both ordinary differential equations and large sparse algebraic equations. The power-system stability calculation is a differential–algebraic initial value problem. Large infrequent discontinuities in the form of faults and branch switching are introduced into the algebraic transmission-network equations, usually at predetermined times. Smaller random discontinuities occur due to limiting in the differential equations of the automatic control apparatus which gives the problem the characteristic of "roughness."

II. Analytical Structure of Stability Problem

The power-system equations as conventionally formulated for general large-scale stability studies have a well-defined structure, which remains valid over a wide range of specific modeling details. Fig. 1 outlines the structure of a very typical model. It shows one of the synchronous generators and its controls connected to the power-system transmission network.

A. The Synchronous Machine

Virtually all synchronous machine models used for routine large-scale studies are based on Park's transformations, in which the machine electrical behavior may be represented by equivalent circuits in the rotor direct and quadrature axes.

Except for special studies, stator transients are neglected, in which case the stator becomes represented as a simple impedance with reactance components in the d- and q-axes, as shown in Fig. 1. Thus the stator equations are algebraic in common with the network equations.

The machine rotor can be represented by varying numbers of d- and q-axis equivalent circuits, which are translated into a set of first-order differential equations. These equations are written directly in terms of the equivalent circuit resistances and inductances, or in terms of the "standard" machine test parameters (synchronous, transient, subtransient reactances, time constants). The state variables are the flux linkages, and/or some defined equivalent voltages. Rotor-circuit currents may be retained in the equations or eliminated. Rotor models of these types result in an equivalent internal stator voltage with d- and q-axis components.

The rotor mechanical equations represent the accelerating torque acting on the machine, and include the important state variable δ, the rotor angle between the d, q frame and the network-variable coordinate system which is known as the synchronously rotating or complex frame.

The machine also has the excitation-control and turbine-governor control-circuit equations, which are quasi- or entirely linear, with limits on some quantities. These controls depend upon various fed-back signals, of which only the most essential are shown in Fig. 1, for simplicity.

All the components to the left of the 'interface' line in Fig. 1 become represented by first-order differential equations in the state variables y. Included in set y are the stator internal voltage components, shown here as E_d' and E_q', and the rotor angle. These are the interface state variables that appear in both the differential and algebraic equations of the machine, to be denoted as subset E of y. The other interface variables that appear in both sets are $I_{sd}, I_{sq}, \Delta|V|$, and P_e (denoted as subset u of x). Possible additional feedbacks from the stator equations to the differential equations are saturation factors (to the rotor electrical equations) and P_e (to the excitation control).

It should be noted that in denoting the stator internal voltage as E', it is not implied that the structure applies only to a conventional "transient" model–E' may be substituted by E'' to imply a "subtransient" model, or by any similar equivalent voltage, with the stator impedance modified accordingly.

B. The Transmission Network

The network is described by a large sparse algebraic nodal admittance matrix equation. This matrix is usually complex and symmetrical, and constant in between infrequent branch-switching operations. Each bus load is modeled as an exponential or polynomial function of the bus voltage magnitude and occasionally of the frequency. Unless all loads have the simplest representation as fixed shunt admittances, the overall network/load equation set is nonlinear, with a similar structure to that of the standard load-flow problem.

C. General Overall Form of System Equations

The complete power-system model comprises a set of first-order differential equations

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

and an algebraic set

$$0 = g(y, x). \tag{2}$$

Set (1) comprises the differential equations of all machines. Since each machine is coupled to the other machines only through the network, set (1) is a collection of separate uncoupled subsets. In the model shown in Fig. 1, there are two such subsets per machine, but which become joined together whenever δ is fed back to the excitation control.

Set (2) comprises the stator equations of each machine, transformed into the complex network reference frame, coupled to the equations of the network and loads, plus the equations defining the fed-back stator quantities u.

D. Specific Form of System Equations

Equation (1) has a quasi-linear structure that can be shown as:

$$\dot{y} = f(y, u) = A \cdot y + B \cdot u. \tag{1a}$$

Matrix A is square, sparse, and block-diagonal. Matrix B is rectangular, sparse, and blocked. (Note that the matrix form is not necessarily retained in the programming.) When saturation is not represented, both A and B are constant in many of the most common specific models.

The algebraic set (2) can be subdivided into two parts:

$$I(E, V) = Y \cdot V \tag{2a}$$

and

$$u = u(E, V) \tag{2b}$$

where (2a) is the sparse bus admittance matrix equation of the loaded network. I is the vector of bus current injections. For a load, the injection is a function of the bus voltage, and for a generator, it is the stator current as a function of the stator internal and terminal voltages, transformed into the network frame. Equation (2b) simply serves to calculate u.

The model depicted in Fig. 1 is sometimes simplified by omitting one or both control blocks. Conversely, more advanced excitation controllers require supplementary feedback signals such as electrical power output or rotor speed. For mid-term and long-term stability studies, the turbine governor has a boiler model block attached to it (for thermal units). Also, where appropriate, centralized automatic generation control and economic dispatch are modeled, and signals from them are fed into the turbine governor blocks.

In the Appendix, the specific equations (1a) and (2b) are illustrated for a typical simple model, and an outline is given of the way in which the initial conditions are obtained.

III. Basic Considerations in the Numerical Solution Problem

A. Solution Requirements

The objective of the step-by-step calculation process is to compute the system dynamic response as rapidly as possible, consistent with the following requirements:

1) Sufficient Accuracy for Engineering Purposes: For general system stability studies, this is usually measured by the maximum error in machine rotor angles over the study period (and in other variables for special studies). Typically, errors of several percent are tolerable.

2) Reliability, in the Sense of Not Experiencing Numerical Breakdown on Practical Problems: The critical factors here are the mathematical stability of the integration method and the convergence of any iterative processes.

3) Economy of Computer Storage: This is less or more important according to the power-system size and computing environment.

4) Flexibility: Flexibility in the range of modeling detail that can be assigned to any part of the power system, and in the ease with which models can be altered to accommodate changing study requirements and new types of apparatus.

5) Ease of Maintenance and Enhancement of the Whole Program: This includes item (4) above. It implies simple and robust algorithms which need minimal tuning, and uncomplicated coding implementations.

Total computing time for a study depends upon the work at each integration step, and the step size(s) used. Requirements 1), 2), and 5), in particular, tend to conflict with the objective of fast solutions, making the design of the overall calculation scheme a very intricate exercise in tradeoffs.

B. Integration Methods

Integration methods fall into the main categories explicit or implicit, and single-step or multistep. In explicit methods, the integration formulas are applied directly to each of the individual differential equations being solved. In implicit integration, the differential equations are algebraized and the resulting equations are solved simultaneously as a set. This is more complicated, but it has the reward of greater numerical stability, as described in Section III-D.

Single-step methods do not use information about the solution prior to the beginning of each integration step. Therefore they are self-starting, which is convenient in the presence of discontinuities. Runge–Kutta is the most famous class of these methods. Multistep methods require the storage of previous values of the variables and/or their derivatives, and thus, in principle, are more efficient. However, the process has to be restarted whenever a discontinuity occurs. Most multistep methods use the open and/or closed difference formulas, of which the Adams family is best known.

There are several simple formulas which fit into the multistep category that are self-starting. These include explicit and implicit Euler, implicit trapezoidal and midpoint rules.

Not very many of the enormous number of integration methods in the literature (see [16] and [17]) have found useful application to the power-system stability problem.

C. Solution Errors

It is important to recognize the sources of numerical error in the solution over a given step. These sources are:

1) inexactness of the integration formulas used, i.e., truncation error;
2) computational inaccuracy due to arithmetic roundoff and, in methods that employ iteration, to nonexact convergence;
3) failure to achieve truly simultaneous solution in time of the sets (1) and (2), i.e., interface error;
4) approximation error, where certain assumptions about the behaviors of the variables or the linearity of the equations over a given step are made in return for computational economies;
5) failure to detect and apply limiting and limit backoff of variables at exactly the correct times, due to the use of finite time increments, and often to imprecise limit-handling techniques.

With any solution method, truncation, interface, approximation, and limit errors can be kept within acceptable bounds by using sufficiently small step lengths. This tends to be detrimental to overall computing speed.

D. Stability of Integration Methods [2], [16]–[19]

The error in the solution at the end of any integration step is some function of the errors incurred as above during the step, and the error inherited from the beginning of the step. Numerical stability of the method is concerned with the propagation of error over many successive steps. An unstable method is one in which the error tends to accumulate so that it eventually "blows up" and swamps the true solution.

Integration methods are classified stability-wise according to certain properties that basically relate to their performances on linear problems, giving rise to terms such as "symmetrically A-stable," "stiffly stable," "conditionally stable," etc. Here, it seems inappropriate to use this terminology without providing the analytical background. Thus we will typically refer to methods as "very stable," "not very stable," and so on, on a comparative basis. For instance, those methods that have been most widely used in the power-system problem (e.g., explicit Euler, Runge–Kutta) are not very stable ones, compared with the more recently used implicit methods.

The less stable an integration method is, the more it is necessary on a given problem to limit the generation and thus propagation of errors by using high-order (low truncation error) versions of the method, by converging iteration cycles accurately, by using high-precision arithmetic, and most of all, by using small step lengths. In the present power-system problem, interface error is a special hazard which has been given much attention in the design of overall solution schemes. All these measures tend to increase the overall computing time. The difficulties are exacerbated if the problem itself is mathematically "stiff," as described in the following section.

E. Problem Stiffness

The stiffness of a set of differential equations is a property analogous to ill-conditioning in algebraic equations. It is associated with the range of time constants (the rates of response of the different variables) in the system. The problem is stiff if the ratio between the largest and smallest time constants is high. More precisely, stiffness is measured by the ratio between the largest and smallest eigenvalues of the linearized system.

On a stiff problem, a relatively unstable integration method will need very small step lengths to track accurately the rapidly changing components in the system response in order to maintain truncation (and other) errors at safely low levels. This is the case even when these components are small-magnitude fluctuations (quiescent modes) superimposed on slower varying responses, and which have very little effect on the solutions of the main variables of interest. On the other hand, a more stable integration method can tolerate much larger errors per step, because they are not going to be propagated as much. Hence, it becomes possible to use larger step lengths and/or be less concerned to minimize other errors for the same overall accuracy of solution.

The advantages of highly stable methods over weakly stable ones tend to reduce as the problems to be solved become less stiff. The classical "fixed voltage behind reactance" power-system transient stability model is not stiff at all, unless machine inertias vary widely (as is the case if small synchronous motors or compensators, or induction motors are represented). Stiffness increases with the detail of synchronous machine modeling. For instance, subtransient time constants are an order smaller than transient time constants. Particular sources of stiffness are the small-time constants that can be found in excitation control transfer-function blocks. It is also important to recognize that stiffness is not simply identifiable from the physical time constants in the input data. There is hidden stiffness in the algebraic equations [6], especially with non-impedance loads.

IV. Classification of Overall Solution Approaches

A. General

Until recently [2], [54], there have been no attempts to locate all overall solution schemes within a classification system, and, therefore, there is no agreed terminology on the matter. The present review adopts the system suggested in [2], with small differences in the definitions.

The problem is to solve the differential set (1) simultaneously in time with the algebraic set (2). The many possible overall schemes are characterized mainly by: a) the way in which (1) and (2) are interfaced with each other, b) the integration method(s) used, and c) the technique(s) used for solving the algebraic equations.

Choosing a) as the main level of classification, interfacing approaches are identified as Partitioned or Simultaneous (the corresponding names given in [54] were Alternating and combined). The Partitioned approach may further be divided according to whether an Explicit or Implicit integration method is used. All practical Simultaneous-solution methods use implicit integration.

B. The Partitioned-Solution Approach

This is the traditional approach, used in nearly all present-day industrial programs. The differential-equation set (1) is solved by integration separately for y, and the algebraic set (2) is solved separately for x. These solutions are alternated with each other in some manner. The respective solutions may or may not be iterative, and true elimination of interface error may or may not be achieved, depending on the specific methods and system models.

The two salient defining features of the Partitioned approach are: a) the integration method and the network-solution method may in principle be chosen independently of each other, and b) it is always possible, through the use of extrapolation/interpolation techniques (see Section V-D) to solve the network only every few integration steps. This latter is generally the case in mid- and long-term stability calculations, although it is much less common in the short-term mode.

C. Simultaneous-Solution Approach

Implicit integration methods convert (1) into a set of algebraic equations in the unknowns y_n and x_n, i.e., the values at the end of the step. In the Simultaneous-solution approach, these algebraized equations are lumped together with (2) to form a single larger algebraic set, all of whose variables are then solved simultaneously. Inherently in this approach, equation (1) is solved with the same frequency as (2), and there is no interface error. The Simultaneous-solution approach has been adopted in at least one routinely used industrial program, and a number of prototype test programs. It has been attracting interest as possibly a superior scheme.

V. Partitioned-Solution Approach—Interfacing Principles

A. General

Consider a step in the numerical integration of (1a) from a given point (y_{n-1}, u_{n-1}) at time t_{n-1}. Except in the simplest methods (e.g., explicit Euler and predictors), the integration process generates at one or more points in the interval $t_{n-1} < t \leqslant t_n$ a new value of y for which the corresponding value of u is required. How to deal with this requirement for u is the interfacing problem.

B. Explicit Integration—The General Method

Explicit methods evaluate $\dot{y}$ directly from (1a), where the relevant value of y is available, and the value of u must be provided. Prior to each such evaluation, the subset E of y is inserted into (2a), which is solved for V. Then the required value of u is obtained from (2b), and $\dot{y}$ may now be computed. This general method eliminates the interface if the solution of (2a) each time is exact. It is nothing more than the conventional solution of a set of differential equations in which the evaluation of the derivatives happens to be fairly complicated. Unless the power-system model is a simplified one, each exact solution of (2a) must be iterative (see Section VII). This is likely to be computationally expensive, and alternative interfacing schemes are used in practice.

C. Rigorous Interfacing by Iteration

Any integration formula in which y_n is computed as a function of u_n permits exact interfacing by iteration. A starting value of u_n is provided, usually by extrapolation from stored previous values. Using this value, integration on (1a) is performed to obtain y_n. Subset E_n can then be inserted into (2a), which is solved for V_n and, hence, a new estimate of u_n is calculated from (2b). Now the integration step is repeated using the new u_n ("reintegration"), and so on until all variables have converged.

The convergence rate of this block-iterative process is a function of the coupling between the sets (1) and (2), in the form of E and u. Fortunately, the nature of this coupling is quite favorable. The iterative process has linear order, but its convergence rate is usually high.

At the same time, the accurate iterative solution of (2a) at each reintegration is not a computationally efficient strategy. It is generally much better to perform a single iteration in the solution of (2a) each cycle, so that the solution for V_n becomes "interlaced" with that of y_n. This idea can be taken a step further if the integration itself is an iterative process by performing also only a single iteration in the solution for y_n at each cycle.

D. Nonrigorous Interfacing by Extrapolation of u

The above iterative scheme is not possible with some integration methods (notably Runge-Kutta), and, moreover, may not be desirable if computational economy is served by solving the network less frequently than the differential equations. In such cases, any values of u required during the integration step(s) are provided by extrapolation. A new value of u is only properly solved for at the end of the integration step(s) when the network equation (2a) is solved, using the final value of E. This gives imperfect interfacing, which might nevertheless be tolerable if the extrapolated values of u are sufficiently accurate.

However, there is always a danger that too much error will be generated. If the stability of the integration method is weak, the whole solution could fail. More commonly, time drift in the system response is experienced. At least one short-term stability program has controlled (but not eliminated) the interface error as follows. The new value of u_n calculated at the end of the step is compared with its extrapolated value. If the difference is excessive, the new value is used to re-estimate the previously extrapolated intermediate values by interpolation, and the step is reintegrated.

E. Note on Extrapolation

The above exposition on extrapolation implies that the u variables are extrapolated individually. Divided differences are convenient, since they do not rely on equidistant points. It is enough to store one or two values previous to the beginning of the current step. Attempts at more accurate extrapolation either give marginal further improvements, or become less reliable due to solution roughness, and add to the problems of restarting after discontinuities.

Instead of extrapolating u itself, another approach is to extrapolate only the bus voltages. These have to be extrapolated anyway in most overall solution schemes. Then each value of u is computed when needed from (2b), using the current value of E and the appropriate extrapolation of V. It is difficult to say which approach is better. In principle, the latter benefits from the use of up-to-date E's. However, there is the question of whether the calculation of stator current and power from (2b), based on the vector difference between two imprecise quantities (E and V), might be prone to greater error.

This section provides an opportunity to discuss the extrapolation of the bus voltages in general, whether used only to get the u's or as starting values for the iterative solution of the network equations themselves. Reference [2] has made the important observation that V should be extrapolated in polar rather than rectangular coordinates, since angles and magnitudes tend to vary somewhat independently of each other, and it is found that accurate estimates of the angles are particularly valuable. However, the bus voltages are normally handled in the rectangular form in the programming. A neat approximation was achieved by geometric (logarithmic) extrapolation of the rectangular bus voltages, which gives weighting to the rotational component. Thus using one and two previous values respectively the extrapolation formulas become

$$V_n = V_{n-1}^2 / V_{n-2}$$

and

$$V_n = V_{n-1}^3 V_{n-3} / V_{n-2}^3 .$$

Where u is individually extrapolated, this technique could be applied to the stator currents.

For mid- and long-term stability studies, the extrapolation of u is routinely used. Here, the network conditions vary more slowly in relation to some of the differential variables, and the network is solved not at each integration step, but according to heuristic automatic controls in the programs. In several programs, it was elected to extrapolate from linear sensitivity relations at the last network solution instead of from previous values. References [23], [25], and [26] extrapolate P_e as a function of δ, and [24] extrapolates the rotor accelerating power in a similar way.

VI. Partitioned-Solution Approach—Specific Integration Methods

A. General

In this section, the interfacing principles already outlined are applied to integration methods that have most popularly been used for the power-system problem. To date, explicit methods have been much the more widely used, but several newer industrial programs have opted for implicit methods.

B. Explicit Euler Method

This least accurate low-stability method seems to have been fairly widely used in the past, presumably because of its simple implementation. The basic application is trivial. Starting at point (y_{n-1}, u_{n-1}), $\dot{y}_{n-1}$ is computed from (1a) and $y_n = y_{n-1} + h\dot{y}_{n-1}$ is obtained. Then (2) is solved to give V_n and u_n. Although there is no interface error, this scheme is uncompetitive. Euler's method demands very small step lengths unless the power-system model is very simple and nonstiff. Thus while only a single evaluation of $\dot{y}$ is made per step, the network has to be solved a very large number of times in total, consuming perhaps 80 percent or more of the total computation. There used to be an argument, when the choice of network-solution methods was between Gauss–Seidel and matrix inversion, that a small step length does not hurt, because then with good initial conditions Gauss–Seidel converges in only a few iterations at each step. This is no longer persuasive with the use of sparse matrix factorization and good extrapolation techniques.

Solving the network every few integration steps and/or using different step lengths for different machines is somewhat better; but again, Euler's poor performance is a limitation on computing economy and detail of system modeling.

C. Open Multistep Formulas

The methods referred to here are predictors in the predictor-corrector class of integration methods. They are exactly the same as explicit Euler in implementation, differing only in the use of previous values to gain improved accuracy (but then suffering from the problem of discontinuities). They are also weakly stable, and the same remarks offered about Euler's method may be applied to them. At least one major industrial program has used them.

D. Explicit Runge–Kutta Methods [2], [21], [27], [28], [34]

The explicit Runge–Kutta method was one of the earliest to be used for stability calculations, and is still retained in a number of major programs. Adapted to the differential-algebraic problem, a popular fourth-order version can be written:

$$k_1 = hf(y_{n-1}, u_{n-1}) \tag{3a}$$

$$k_2 = hf(y_\alpha, u_\alpha), \qquad \text{where } y_\alpha = y_{n-1} + k_1/2 \tag{3b}$$

$$k_3 = hf(y_\beta, u_\beta), \qquad \text{where } y_\beta = y_{n-1} + k_2/2 \tag{3c}$$

$$k_4 = hf(y_\gamma, u_\gamma), \qquad \text{where } y_\gamma = y_{n-1} + k_3 \tag{3d}$$

$$y_n = y_{n-1} + (k_1 + 2k_2 + 2k_3 + k_4)/6. \tag{3e}$$

In the (unattractive) rigorous interfacing scheme of Section V-B, the set (2) is solved exactly prior to stages (3b)–(3d), to provide the values of u corresponding to those of y. Reference [27] used virtually this approach, but carefully modeled the problem so that (2a) was linear, except for bus loads. By approximating the load currents as constant over the integration step, it was possible to make a noniterative solution of (2a) at each stage above (see also Section VII-D).

The extrapolation of u as described in Section V-D avoids these intermediate solutions of (2), and was used, for example, in a former BPA program. Having thus provided the values of u in (3b)–(3d), and then having obtained y_n from (3e), the network is solved to give V_n and u_n. At this point, reintegration with improved interpolated estimates of u in (3b) and (3c) and $u_\gamma = u_n$ is performed where dictated by the interface error control mechanism.

The second time around, the network solution is expected to converge very rapidly since it has improved starting values. If necessary, reintegration can be performed more than once. There seems to be little difference in total computing time or accuracy between a method with one reintegration every step (systematic reintegration) and a method without reintegration using half the step size [2]. The latter has an advantage in coping with fast machine components and roughness. Nevertheless, provided that the step lengths and tolerances are tuned so that it is not applied too often, nonsystematic reintegration does guard against excessive interface errors at critical stages in the power-system response.

There is not too much agreement on the question of the best Runge–Kutta version. The studies in [21] found that a fifth-order method was best on a stiff problem (but extrapolation of u and reintegration were not tried). A fourth-order version has been used in most industrial programs. Studies in [2] concluded that a third-order version is most efficient, bearing in mind solution roughness.

E. Predictor–Corrector Methods [2], [21], [22], [28]–[30]

A predictor–corrector pair consists of an open formula and a closed formula in the multistep category. The predictor is applied once per step to provide good initial conditions for the corrector. The corrector itself may be applied with a fixed number of iterations per step (frequently, once only), or it may be iterated to convergence.

There is wide agreement that the best general-purpose formulas are those in the Adams family. A simple one-corrector-iteration version for the power-system problem is illustrated here with a typical Adams pair:

(i) compute $\dot{y}_{n-1} = f(y_{n-1}, u_{n-1})$ from (1a)

(ii) predict y_n from

$$y_n = y_{n-1} + h(23\dot{y}_{n-1} - 16\dot{y}_{n-2} + 5\dot{y}_{n-3})/12 \quad (4a)$$

(iii) insert subset E_n of y_n into (2a), solve for V_n and, hence, compute u_n from (2b)

(iv) compute $\dot{y}_n = f(y_n, u_n)$ from (1a)

(v) correct y_n from

$$y_n = y_{n-1} + h(5f(y_n, u_n) + 8\dot{y}_{n-1} - \dot{y}_{n-2})/12 \quad (4b)$$

(vi) insert subset E_n of y_n into (2a), solve for V_n and, hence, compute u_n from (2b).

The above implementation eliminates interface error if (2a) is solved exactly in stages (iii) and (vi), this being the general method of Section V-B. Even in the one-corrector-iteration version, several variants are possible. At the end, an additional calculation of $\dot{y}_n$ as in stage (iv) could be made, for use in the next integration step. Or stage (vi) could be omitted, accepting the predicted value of u_n as final. Or stage (iii) could be omitted, supplying an extrapolated value of u_n for use in stage (iv) [22]. The last two options do not achieve interface elimination.

An extrapolated value of V_n should be provided to start the first solution of (2a). Subsequent solutions of this equation will converge very rapidly, since their starting values become successively better. When corrector iteration is considered, the number of variants multiplies. Rather than catalog them all, it may be better to make a few general remarks. The corrector may be iterated in a loop between stages (iv) and (v), with u_n constant at the predicted (or extrapolated) value, i.e., giving emphasis to the accuracy of integration rather than the interface. Alternatively, stage (vi) may be included in the loop, at the expense of extra network solutions. If in this case the corrector is converged accurately, interface error is eliminated even if stage (iii) is omitted. (See Section V-C.)

The main point about corrector iteration is that the corrector formula is generally more stable and accurate than the predictor. When iterated to convergence, the solution is independent of the performance of the predictor, which has simply provided initial values for the iterative process, and could even be replaced by a more conventional extrapolation formula. In a mathematical sense it would be always desirable to converge the corrector. However, this may not be computationally attractive in terms of the work per integration step. Also (and this is highly relevant to the methods of the next section), the corrector convergence becomes increasingly slow and unreliable as the stiffness of the problem increases.

The formulas in the Adams and other families all have the same structure and virtually the same computational effort. They differ only in the coefficients and the numbers of previous values required. The lowest order version is Euler, which is self-starting. Therefore it becomes possible to program a variable-order code where, to restart after a discontinuity, successive integration steps use successively increasing orders of formula, building up the necessary previous values as they go. As with all methods that use previous values, predictor–corrector methods suffer from solution roughness—restarting is inefficient and/or inaccurate. On the other hand, failure to restart after a discontinuity can introduce gross errors.

This class of methods is competitive for problems with very limited stiffness and few discontinuities. Reference [21] demonstrates excellent results on such a problem using fifth-order Adams formulas and several interfacing schemes (step lengths up to 0.12 s and lower total computing times than other methods). The studies in [2], using a variable-order code on a range of more practical power system problems, were less encouraging, and it was concluded that solution roughness makes Runge–Kutta more attractive on the whole. For stiff problems, neither approach is satisfactory.

F. Implicit Multistep Integration [2], [18]–[21], [29], [40], [41]

These more modern methods were designed to overcome the deficiencies of the predictor–corrector approach on stiff problems. It was recognized that: a) some very stable closed corrector-type formulas are available, b) to get the maximum advantage from these formulas, it is necessary to iterate the corrector to convergence, thereby freeing the solution from the influence of the prediction (or extrapolation), and c) the forward-substitution mode of iterating the corrector in the previous section must be replaced by a much more powerfully convergent technique.

It is noted that (4b) is a set of simultaneous equations in y_n, with u_n the only other unknown. In fact, all closed multistep formulas of this type can be expressed in the general form

$$y_n = khf(y_n, u_n) + C \quad (5)$$

where C is the sum of weighted y and $\dot{y}$ terms backwards from time t_{n-1}, and k is a constant coefficient. The set (5) can be solved for y_n by a method such as generalized Newton(–Raphson). This solution directly replaces the corrector iterations of the last section. A predictor of the type (4a) is now only a somewhat arbitrary means of providing good starting values for the iterations, and may be replaced by any other convenient method of extrapolating y. As usual, starting values for V_n and u_n are obtained by extrapolation.

In the absence of saturation, equation (5) is usually linear in y_n and a direct solution, equivalent to a single Newton iteration, can be made. Substituting for f from (1a) and rearranging, equation (5) becomes:

$$[\mathfrak{I} - khA] \cdot y_n = khB \cdot u_n + C. \quad (6)$$

With A and B constant, the left-hand matrix is constant over the step, and a matrix solution of (6) gives y_n directly in terms of u_n. In the original and still perhaps largely definitive scheme [18], equation (6) is thus solved alternately with an iteration in the solution of (2) for V_n and hence u_n, until the process has converged and interface error is eliminated. (An interesting feature of (6) is that it handles even zero-valued machine time constants with no difficulty [18].)

Compared with the forward-substitution scheme for iterating the corrector in the previous section, the direct solution of (6) costs a little more work per step, and is more complicated to program. However, much larger steps can be taken on stiff problems, at a great overall saving in computation.

Saturation modeling in the machine and/or the excitation control introduces nonlinearities into matrices A and B, and an exact direct solution of (5) is not possible. In principle we could revert to a rigorous Newton solution approach for (5), preferably keeping the Jacobian matrix constant over the step. Usually, the effect of saturation is not too large, and excellent convergence is expected. A more convenient scheme is to include saturation in an outer loop, simply modifying the reactance elements in A and B every time (6) is solved. This scheme is not quite as strongly convergent as the Newton approach, but in the author's experience it very rarely worsens overall convergence. The exception was where a heavily saturated generator was marginally stable with saturation represented, and unstable without. Then at critical periods in the response, overall convergence was slowed down, and failed in some runs using otherwise-acceptable step lengths. Often, saturation is assumed constant over a step.

G. *Implicit Multistep Formulas*

There remains the question of which closed integration formula(s) to use. As an alternative to the not-very-stable Adams(-Moulton) family, Gear has offered a corresponding family which, while a little less accurate, is extremely ("stiffly") stable. At the low-order end, the self-starting implicit Euler and Trapezoidal methods with so-called A-stability are available, and as in the predictor–corrector approach, variable-order and variable-step codes have been developed. In any such schemes, the second-order Trapezoidal rule will always be preferred to the first-order Euler. Another highly stable self-starting second-order formula, slightly different in form from (5), is the implicit Midpoint rule.

Reference [18] exposed clearly the benefits of stable implicit integration in the power-system application. On the basis of excellent comparative results the Trapezoidal rule was chosen to replace the Runge–Kutta Method in the BPA program. Shortly after, the same method was incorporated by the author and colleagues into another production program, also with considerable savings over previous methods. As expected from the theory, there is no need to use very small step lengths, however stiff the problem. At the same time, on nonstiff problems, the method appears to be at least competitive with higher order explicit methods.

At first sight, this last statement seems to contradict the theory, and an explanation will give some insight into the special character of the power-system problem. When comparing integration methods with each other, numerical analysts nearly always tacitly assume that accuracies to several decimal places are required, and on general nonstiff problems, the comparisons then depend heavily on the relative truncation errors. In contrast, the accuracy requirements in the power-system case are very relaxed (a few percent error tolerable), permitting the design of an unusually "sloppy" overall solution scheme. Over an integration step, truncation error is only a part of the relatively large total error generated (see Section II-C), which diminishes the importance of the order of the integration method. Nevertheless, the method has to cope with the propagation of the total error over many steps, and, therefore, numerical stability is a great advantage, and tends to compensate for low order.

Higher order stable implicit methods have been used in slightly different, but for the purposes of this discussion, equivalent implementations. References [20], [40], and [41] used the Gear formulas, and [19] used hybrid integration, in variable-order codes. Promising results were obtained in each case. However, these and other stable methods (e.g., the Midpoint rule) can be unreliable for fast nonquiescent variables, such as those in some of the excitation-control circuits. The integration formulas have spurious roots that can under certain conditions lead to spurious solutions. From the engineering point of view, the worst case is if a stable solution to an unstable problem is produced. Because of this danger, [2] endorsed the choice of the Trapezoidal rule on its own, since this method is not prone to the above phenomenon and gives unique solutions to problems. The extent and likelihood of the danger in power-system studies is not yet well quantified and further investigation would be justified. It is unfortunate that there are no equally well-behaved very stable multistep methods of higher order than the Trapezoidal rule. On the other hand, such hypothetical methods would not be self-starting, and would suffer during periods of solution roughness.

The comparative studies in [2] concluded that the Trapezoidal rule is "significantly more efficient than any explicit method." The fifth-order implicit Adams–Moulton formula was tried in [21] on a stiff problem, showing that this also is not nearly as economical in overall computation as the Trapezoidal rule.

H. *Partitioned Solutions of the Differential Equations*

In any integration method, the differential equations of a machine and its controls may be partitioned. The more natural form of partitioning is to divide the equations into subsets according to the component blocks as shown in Fig. 1. From the viewpoint of flexible program structure, it is then easier to build the complete model of each machine by assembling the desired combination of blocks from a series of different standard component models that are available in the program. Each of the constituent blocks for a machine has very limited coupling with the machine's other blocks. This is illustrated in (A.5), where in matrix A there is a single coupling variable between the excitation-control and rotor electrics blocks, and two such variables between the rotor mechanics and turbine-governor blocks. There is little more coupling in advanced models.

In some schemes, partitioning of the differential equations is only a device for program organization and does not affect numerical performance. In others, it is essential to the solution methodology. For instance, the different blocks may be solved with different integration step lengths, as described further in Section IX-A. Here, we will restrict our attention to partitioning in the interesting implicit multistep methods assuming that the same h is used for all equations.

Referring to (6), the structure of $[\mathcal{I} - khA]$ is almost identical with that of matrix A itself. It is not essential to solve it directly as a single matrix equation per machine. Each component block may be solved separately, and then linked with the block(s) coupled to it.

One such scheme takes advantage of the fact that the whole solution process over the integration step is iterative. For a given machine, a certain component block, say Block A, solved as per (6) for the subset of variables y_A. Then the value of the coupling variable from y_A is substituted into some Block B, which is solved for y_B, and so on. This is a block-successive substitution process that hopefully converges in unison with the interfacing iterations between the network and the differential equations as a whole. The structure of matrix A suggests the most convenient specific order for the substitutions. For example, from (A.5), the excitation-control equations will be solved first, and the new value of E_f will then be inserted into the rotor electrical equation block. In general, care must be taken to define substitution orderings that produce the most rapid and reliable convergence.

Rather than solve the individual equation blocks by numerical matrix methods, it is more efficient to perform manual algebraic manipulation on them and, hence, produce the analytical expressions that give the solution directly. Different analytical solutions for different modeling options are then coded directly into the program. It is then particularly easy to link a combination of blocks together for a given machine. The step length h, integration-formula coefficients, and machine reactances (for saturation) may be changed at will with no extra overheads, and storage is saved.

The above advantages can also be gained without resorting to a block-successive iteration scheme. Taking specific advantage of the fact that matrix A is already almost upper triangular, even in more detailed models, the analytical solution of all the machine equations simultaneously may be obtained. The identities of the coupling variables are retained; therefore, the interconnection of different model blocks remains easy. This approach eliminates the previous successive interblock substitution process as a possible source of convergence problems.

It should be noted that the degenerate case of partitioning, where each equation is solved individually, is the Gauss–Seidel solution of (6). This is structurally as flexible as explicit integration, but would not normally be used due to its inferior covergence properties.

I. Matrix Exponential Method [23], [25], [26]

This is an implicit single-step method that is based on the fact that (1a) is in classical state-space form.

Assuming that the "forcing function" u varies linearly over the step, the solution of (1a) at the end of the step is

$$y_n = \phi \cdot y_{n-1} + W_1 \cdot u_{n-1} + W_2 \cdot u_n \qquad (7)$$

where ϕ, W_1 and W_2 are functions of the matrix exponential e^{Ah}. This exponential can be expressed as an infinite matrix power series, and can be evaluated numerically by truncating the series after a small number of terms. The matrix functions are computed as follows:

(i) compute $F = \sum_{k=0}^{m} A^k \cdot h^{k+1}/(k+2)!$

(ii) $W_2 = F \cdot B$

(iii) $G = \mathcal{I} + A \cdot F$

(iv) $W_1 = G \cdot h - F \cdot B$

(v) $\phi = \mathcal{I} + A \cdot G \cdot h$

The integration formula (7) has excellent stability and is of order $m + 1$. It has exactly the same general form as (6), and can be implemented in the same way, i.e., by solving for y_n at each iteration in the solution of (2). The matrices in (7) are functions of A, B, and h. Provided that variation of machine reactance with saturation is not represented, they are recalculated only when the step length h changes, and of course (7) constitutes a set of smaller separate matrix equations, one or two per machine. Thus the matrix exponential method can be considered as a direct alternative to the implicit multistep method, particularly to the also-self-starting Trapezoidal rule. Some comparative comments are:

a) the formula (7) is self-starting for all orders; b) the assumption of linear variation of u over the step introduces a small approximation in the use of (7), which may nullify the advantage of high order; c) the calculation of the matrix functions in (7) is more complicated and time-consuming than the equivalent work in (6)—this disadvantage is pronounced if the functions have to be evaluated repeatedly due to saturation and step length changes, especially with high-order versions and detailed models; d) matrix powering sacrifices sparsity, incurring a storage and computational penalty, again especially for high orders and detailed models.

The use of partitioning and successive block substitution as described in the last section can alleviate considerably the disadvantages of the Matrix Exponential method described in points c) and d) above. There have not been any large-scale applications of the method in the Partitioned-solution interfacing approach. However, the method has been receiving attention due to very successful results obtained in a hybrid integration scheme which has been classified in the Simultaneous-solution category (see Section VIII-E).

VII. The Modeling and Solution of the Network

A. The Network Model

The network model comprises the loaded transmission system plus the machine stators. In order to construct and solve the network equation (2a), the d, q-axis stator equation (A.6) of each machine has to be expressed in the form (A.8), i.e., transformed into the network complex reference frame. From (A.8), the stator internal voltage is now $E'_{re} + jE'_{im}$ and the stator impedance is Z_s.

In (2a) as originally stated, the nodal injection at a machine terminal bus is the machine stator current, obtained by solving (A.8). There is some advantage in taking the Norton equivalent of each machine stator. Then a shunt impedance Z_s is inserted at the machine terminal bus, and the injected current becomes $(E'_{re} + jE'_{im})/Z_s$. The network equation (2a), restated here for convenience, now becomes:

$$I(E, V) = Y \cdot V \qquad (8a)$$

where Y includes the machine-terminal Norton shunts. Vector I comprises the machine-terminal Norton injections that are functions of E and the load-bus currents that are functions of voltage magnitude and perhaps frequency.[1] This form of the network equation will be assumed henceforth, unless otherwise stated.

Whenever a machine has no dynamic saliency, X'_d is equal to X'_q, and the T's in (A.8) cancel each other out. Z_s is then a constant complex impedance.

However, when there is saliency, Z_s is nonbilateral and also changes with rotor angle δ. Nonbilateralism means that Z_s and therefore equation (8a), cannot be written in complex form or solved by complex arithmetic.

[1] The local frequency at a bus is measured by the rate of change of its bus voltage angle. It is obtained as a filtered extrapolation from stored previous values, and is assumed constant over an integration step.

We must consider two cases. One is when *all* Z_s's are nonsalient, so that (8a) may be written as a complex equation:

$$I^c = Y^c \cdot V^c \tag{8b}$$

where superscript c stands for complex. The other case is when *any* of the Z_s's are salient, so that (8a) must be written as a real equation, expanded into its real and imaginary parts:

$$I^e = Y^e \cdot V^e \tag{8c}$$

where superscript e stands for expanded, and it will henceforth be assumed that structurally, each complex admittance element in (8b) has been replaced by a 2×2 real block in (8c).

Equation (8b) is half the order of (8c). Matrix Y^c is usually symmetrical (if there are no phase shifters represented in it), while Y^e is unsymmetrical. Then the latter requires 3–4 times the storage of the former.

B. The Network Solution Problem

The problem is to solve either (8b) or (8c) for V. For a given value E, obtained from the solution of the differential equations (1), the machine-bus Norton injections and shunts are constant.[2] The nonlinearity of (8) is then due entirely or mainly to load currents that are functions of V. Unless all loads are represented as fixed shunt impedances (injected current always zero), an iterative solution of (8) not unlike a standard load-flow solution [31] needs to be performed.

For mid-term and long-term dynamic studies, excitation control is assumed to hold the machine terminal (or other bus) voltage magnitude constant [24], [25], which introduces a conventional constant-V load-flow constraint into (2a). When automatic transformer tap changing is represented in longer term studies, the relevant admittances in Y can change frequently. Very occasionally, network branch admittance variation with frequency is represented, in which case the elements of Y change continually. Such network changes can be dealt with by bus-injection techniques to avoid continual matrix alterations.

C. Network Solution Techniques

In this subsection, we consider four alternative methods for solving (8). Only the last two are now regarded as of interest for efficient modern large-scale industrial applications, but programs employing the first two are still in practical use.

1) Gauss–Seidel: This method has the merits of low storage, ease of programming, and of being able to accommodate any changes in the matrix elements with ease because the algorithm operates directly on the branch admittances.

The economical complex-symmetrical storage scheme can be used in the programming, even if nonbilateral elements are present. Since the Norton admittances are usually large, Y is better conditioned than in the standard load-flow case. Except at fault and switching times, each iterative solution has good starting values of V from the previous solution(s), preferably extrapolated.

Usually, the "load-flow" problem has no voltage-controlled (PV) buses, in which case the best convergent version seems to be the secondary correction method [31]. Nevertheless, convergence to acceptable accuracy can vary a great deal from problem to problem, from 2–3 iterations to hundreds (or no convergence) in difficult cases. The Gauss–Seidel method is best suited to approaches using small integration steps, so that the starting point is close to the solution each time.

Unwanted passive buses (buses with zero injected current) can be eliminated [34] from (8), giving the reduced system

$$I_r = Y_r \cdot V_r. \tag{9}$$

This may increase the total number of nonzero elements in Y and thus the computation per iteration, but usually improves convergence. Reduction prevents immediate access to the eliminated buses and associated branch flows, but efficient techniques for recovering these are available [34].

2) Z-Matrix: The direct solution of (8) is obtained by inverting Y

$$V = Z \cdot I \tag{10}$$

where Z is nonsparse. This method has only been applied to the complex version (8b) in which Z remains unchanged in between switching operations [35]. Any nonconstant elements are incorporated into I, and (10) is solved iteratively for V, updating I every iteration. Extrapolation provides good starting values for I. Convergence is usually fairly rapid (2–6 iterations).

For large systems, even using the best inverse-matrix assembly and modification techniques, the computation for obtaining and updating Z is unattractive. Much worse, the computation per iteration and the storage are excessive. To keep the order of the nonsparse equation down, all unnecessary passive buses should be eliminated. Z-matrix methods are now quite obsolete in this and most other large-scale network applications.

3) Factored Y: Sparsity-programmed ordered triangular factorization [36] provides a modern approach to the direct solution of (8) for V. The sparse LDU factors of Y are obtained, enabling (8) to be expressed as:

$$I = L \cdot D \cdot U \cdot V. \tag{11}$$

When Y is symmetrical (the complex version), L is the transpose of U and need not be computed or stored. Storage is then typically around 50 percent more than for Gauss–Seidel. If asymmetry is due only to a few terms, it is possible to store only the affected columns of L.

Equation (11) is solved for V in terms of I by forward and backward substitution, so that the iterative scheme is the same as for (10), except that (10) may use a successive-displacements mode, while the former is necessarily simultaneous displacements. This advantage of the Z-matrix approach is completely outweighed by the much-greater speed of the sparse method.

The factorization of (11) should use a good bus ordering scheme to minimize the number of nonzeros in the factors, since the solution is to be repeated many times. In power system networks, the nonzeros, and therefore the solution time and storage increase almost linearly with size. Factorization takes between 3 and 6 times as long as a repeat solution. A repeat solution takes only about 50 percent longer than a single Gauss–Seidel iteration; as a result, it is extremely difficult for the latter to be competitive.

Note that the number of arithmetic operations required for sparsity solutions of the complex and real versions (8b) and (8c) are very similar to each other. The former will be a little faster due to fewer operational overheads.

[2] Not strictly true if machine saturation is being represented rigorously.

Sparsity-preserving network reduction can be performed [37]. Unwanted passive buses are eliminated in an optimal order only as long as the total number of nonzeros in the matrix factors continues to reduce.

4) Newton Method: The Newton method cannot be applied to most power network equations in complex form; therefore the expanded version (8c) is used. This equation can be written as

$$F^e = I^e - Y^e \cdot V^e \tag{12}$$

where F^e is zero at the solution. Each iteration of the Newton solution requires the construction of the Jacobian-matrix equation:

$$F^e = -J^e \cdot \Delta V^e \tag{13}$$

and its direct solution by sparse triangulation for the correction vector ΔV^e. This solution corresponds to the "rectangular current mismatch" Newton load-flow method, which is the natural version for the stability application although it is less so for conventional load flow [18], [31], [38]. When, as is usual, the series branches of the network have constant admittances, the Jacobian matrix J^e differs from Y^e only in the bus "self" terms—those in the 2×2 diagonal blocks.

A strict implementation of Newton's method, with quadratic and reliable convergence, updates J^e in (13) every iteration. This demands an expensive triangulation of J^e each time, and is too high a price to pay, as confirmed by comparative studies in [2]. Therefore, a compromise is adopted, using the same triangular factors of J^e for several or many consecutive iterations. A practical criterion for reforming and factoring J^e is when the last solution took more than say 5 iterations, or when the present solution has diverged or failed to converge in a maximum specified number of iterations. Naturally, convergence is no longer quadratic, but it is still very fast for practical accuracies.

It should be noted that the decoupled-Jacobian techniques that have achieved popularity in conventional load flow [31] are not appropriate in the network solution, because MW and MVAR flows are highly interactive during dynamic system conditions.

D. Network Solution in Relation to Modeling

Modeling plays an important part in determining which solution method to choose for the network equations, and how to interface them with the differential equations. The critical modeling aspects are machine dynamic saliency, nonimpedance bus loads, and machine saturation. Only the factored-Y^c and Newton methods are considered here.

Although a modern general-purpose stability program is likely to cater for the whole range of modeling details, it may be useful to give an item-by-item account of the computational consequences of the different individual model options.

1) The Simplest Model: Let us first consider the simplest case where there is no saliency, saturation, or nonimpedance loads. Then the Norton impedance of each machine, and the representation of each bus load, are fixed complex shunt branches. The network equations are most economically handled in the complex form (8b), where Y^c and its factors are (usually) symmetric, and constant in between switching operations. I^c is a function of E only; so, for a given E, an exact noniterative solution for V^c is obtained rapidly from a repeat solution using (11).

2) Machine Saliency: Now suppose that machine dynamic saliency is introduced into the above simple model. The Norton shunts are nonbilateral, and, in theory, the expanded form (8c) must be used. A noniterative exact solution of (8c) for V^e is then possible, but since the shunts change every time E changes, continual refactorization is necessary. This is prohibitively expensive, just as it is in the Newton method.

A practical alternative [18] is to insert into Y^c a constant complex approximation to each Norton shunt, thus retaining the complex equation form (8b). To compensate for this approximation, a correction term as a function of V^c is added to I^c; so an iterative solution becomes necessary, performing repeat solutions of (11) for V^c and updating I^c each time until converged.

Whether or not the need for iteration in this manner is a drawback depends on the integration/interfacing method. If the interfacing scheme itself is noniterative (e.g., with explicit Euler, open multistep formulas and Runge-Kutta), then iteration for dynamic saliency on its own is a great nuisance, to say the least. Reference [27] circumvented the problem by standardizing on a subtransient machine model (even if more detailed than required by the studies). Subtransient d- and q-axis reactances are almost equal to each other, and can be approximated as nonsalient. (A dummy rotor winding can be added in the differential equations to compensate for the error thus introduced.) Subtransient time constants are smaller than transient ones, but these may still not be a limitation on the performance of the integration method if some of the excitation-control time constants are even smaller.

When iteration is imperative for interfacing or any other reason, then the factored-Y^c approach usually introduces no computational penalty. Used in the block-successive iterative scheme with the Trapezoidal rule, reference [18] quoted 2–3 iterations as typical per integration step, which includes elimination of the interface. These figures were confirmed using a similar program (reference [45] and author's experience). The method runs into very occasional convergence problems which [45] counteracted by a form of acceleration.

In the Newton network solution method, the Jacobian matrix is constructed from Y^e by adding to the 2×2 diagonal block for each machine the partial derivatives of I_{re} and I_{im} with respect to V_{re} and V_{im}, respectively, as in (13). With nonsaliency, equation (13) is equivalent to the factored-Y^c method. In fact, the latter could be regarded almost as a Newton method with a constant approximation to the Jacobian matrix. However, there is one important difference—Newton's method inserts into J^e an incremental model of the machine stator, formed about the last point of linearization, whereas Y^c contains a nonincremental and, therefore, a less accurate approximation. Thus Newton's method, with J^e updated at intervals, is somewhat stronger in convergence, and may require fewer iterations. In the interlacing scheme, this will depend on the overall convergence rate. Any savings will usually be offset by the extra factorizations of J^e. In cases of difficult convergence, however, when the network is ill-conditioned and some machines have high saliency and large rapid rotor angle swings, Newton's method is at a definite advantage.

3) Nonimpedance Loads: Nonimpedance loads are dealt with in a similar manner to machine saliency. In the factored-Y^c approach, a proportion of each load is represented as a fixed complex shunt in Y^c, and the residue of the load enters

I^c as a nonlinear function of V^c to be iterated to convergence. Low-voltage cutoff must be provided so that for instance constant-power loads do not demand infinite current during a solid fault. In standard Z-matrix load flow, this "fringing current" technique was found to aid convergence considerably. Here, it is valuable though not equally successful because of the much greater bus voltage variation. Both [18] and the author have found that typical numbers of iterations are 2–6. In other words, nonimpedance loads cause more trouble than dynamic saliency.

In Newton's method, partial derivative terms are added to the 2×2 diagonal blocks in J^e to represent the loads incrementally, and this is better than the nonincremental fringing-current modeling. Newton's method is now noticeably superior, and the above-mentioned numbers of iterations reduce to 2–3.

How accurate the solution for nonimpedance loads must be is a matter for some conjecture, since the load characteristics are rarely well known. On the other hand, it is widely agreed that some improvement over the classical fixed-impedance model is necessary [12]. Reference [27] investigated the effect on accuracy of keeping the load current constant over the step (which was essential in that Runge–Kutta method with a noniterative network solution.) The results and the discussion of the paper suggest that the errors only become important for marginally stable longer duration studies. Using voltage extrapolations to estimate the required intermediate load currents would be more reliably accurate.

4) Machine Saturation: Papers describing specific stability-calculation methods most often ignore saturation. As already seen, it introduces unpleasant complications in otherwise-elegant integration algorithms such as (6) and (7). The inference is that it is not always represented. Where this is the case, justifications have been based on the proposition that saturation has a small effect on stability, and that neglecting it is in any case conservative, since it effectively reduces machine reactance. However, this is an arguable point, and there are many cases where the representation of saturation makes a major difference to the calculated system response.

In the network solution, its treatment is again similar to that of dynamic saliency. The stator impedances X_d' and X_q' are now modified by saturation factors that depend on the Potier voltage (a function of V and E). In the factored-Y^c method, to preserve the Norton shunts constant, an extra correction term is added to the bus injected current to be included in the iterations. Usually, this does not affect convergence, but counter examples have been found. If the network solution is in all other respects noniterative, it becomes computationally desirable to approximate saturation as constant over the step (with extrapolation where needed).

In principle, Newton's method is again at an advantage because extra partial derivative terms can be added to the 2×2 diagonal blocks of the Jacobian matrix, giving good convergence properties. However, these terms can be quite analytically complicated, and some approximation might be preferred, even handling saturation in an outer loop as in the factored-Y^c approach.

Saturation is the least standard feature of machine modeling. A much more convenient though less rigorous representation is not to modify the stator reactances at all, but to modify the stator internal voltage instead. Another scheme is to modify the differential equations describing the machine.

With respect to exciter saturation, neglecting it implies far better regulation response than actual, which will generally lead to an optimistic solution.

VIII. The Simultaneous-Solution Approach

A. General

The Simultaneous-solution approach was classified as one in which the variables y and x in (1) and (2) are by definition solved simultaneously with each other. All the integration methods used are implicit, and interface error is eliminated. Many of the issues relating to the integration methods and the network have already been covered in Sections VI and VII, and will not be repeated here except where there are important differences.

B. Multistep Integration

In this approach to the differential–algebraic problem, possibly attributable to Gear [39], the differential equations are algebraized using an implicit formula as in Section VI-F and (5). We can then write (5) and (2) together as

$$F_1 = y_n - khf(y_n, x_n) - C \tag{14a}$$

$$F_2 = g(y_n, x_n) \tag{14b}$$

where the F's are zero at the solution. Newton's method is used to solve this set, requiring the construction and solution at each iteration of the sparse Jacobian matrix equation

$$\begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = - \begin{bmatrix} J_1 & J_2 \\ J_3 & J_4 \end{bmatrix} \cdot \begin{bmatrix} \Delta y_n \\ \Delta x_n \end{bmatrix} \tag{15}$$

where $J_1 = \partial F_1/\partial y$, $J_2 = \partial F_1/\partial x$, $J_3 = \partial F_2/\partial y$ and $J_4 = \partial F_2/\partial x$. Good starting values of y_n and x_n are established by extrapolation. Being a Newton method, all the modeling details such as saliency, nonimpedance loads, and saturation can be incorporated without difficulty.

The above approach was first used by IBM [20], [40] designating the set x as the bus voltages V^e and bus currents I^e. Vorley [41] and Boeing [2] obtained a smaller Jacobian matrix equation by designating x as V^e only. This more compact formulation means inserting the analytical functions for u from (2b) directly into (1a). Equation (14) then becomes

$$F_1 = [\mathcal{I} - khA] \cdot y_n - khB \cdot u(y_n, V_n^e) - C \tag{16a}$$

$$F_2 = I_b^e(y_n, V_n^e) - Y_b^e \cdot V_n^e \tag{16b}$$

References [40] and [41] both used Gear's stiffly stable formulas in variable-order variable-step codes in comparative tests against other programs using detailed models. The IBM comparisons against a production Runge–Kutta program using $h = 1/30$ s. showed speed improvements of 2–6 times over stable system responses of duration 5–30 s. For unstable system responses, the timings of the two programs were similar. Vorley ran tests against a production program that uses the implicit Trapezoidal rule with factored-Y^c network solutions (i.e., the method of [18]) and found the Trapezoidal program generally a little faster, though not as reliably convergence at large step lengths in marginal cases. It was also found that the higher order formulas did not show to advantage during the first couple of seconds of the response, because of restarting due to switching and limiting. However, it is likely that they would become more efficient over longer stable response.

In their tests, comparing a variety of methods and techniques, fears about falsely stable solutions and difficulties with limits and fast components using the stiffly stable formulas caused Boeing to consider a version using the Trapezoidal rule only. The tests concluded that this is significantly more efficient than any explicit Partitioned-solution method.

C. Matrix Solutions in Multistep Integration

Each integration step consists of the iterative solution of (14) using the algorithm (15). As in the Newton solution of the network on its own (Section VII), it is too expensive to construct and triangulate the Jacobian matrix at each iteration. True quadratic convergence is sacrificed by using the same LU factors over successive iterations and steps, with a criterion of maximum number of iterations and/or divergence for updating the factors.

Submatrices J_1, J_2, and J_3 are composed of separate machine blocks, and J_4 is exactly the same as J^e in the Newton network solution (13). The machine blocks have no interaction with each other during the triangular factorization, except through the network portion J_4 of the Jacobian matrix. To illustrate, suppose that we have performed Gaussian elimination on all columns to the left of J_4; then all that has happended to J_4 at this stage is that each of its 2×2 diagonal blocks has acquired some additional terms from the machines connected to its bus. Thus it is sensible not to construct and solve (15) as shown, but to treat J_1, J_2, and J_3 separately, machine by machine, and then handle only the network portion as a whole.

Reference [2] offers a scheme to achieve this, described as follows,[3] on the understanding that the relevant submatrix operations are performed on a per-machine basis.

1) When updating the Jacobian matrix: Compute the new matrix elements. Compute a 2×2 block diagonal matrix $-J_3 \cdot J_1^{-1} \cdot J_2$ and add this to J_4 to give $\hat{J}_4$. Factorize $\hat{J}_4$.
2) When performing an iteration by solving (15): Compute a vector $-J_3 \cdot J_1^{-1} \cdot F_1$ and add it to F_2 to give $\hat{F}_2$. Solve the network equation $\hat{J}_4 \cdot \Delta V^e = \hat{F}_2$. Compute $\Delta y = J_1^{-1} F_1 - J_2 \cdot \Delta V^e$.

As in Section VI-H, analytical solutions involving the machine component blocks can be derived and coded directly into the program. For example, an analytical expression for $-J_3 \cdot J_1^{-1} \cdot J_2$ can be obtained and then used to compute the numerical values to be added to J_4 in stage (a).

D. Additional Comments on Implicit Multistep Integration

When examining the detailed implementation of these methods in the Simultaneous-solution interfacing approach, some subtle similarities with the corresponding Partitioned-solution versions emerge. Let us start by noting that it is by no means essential to use the strict Newton algorithm (15) for the simultaneous solution of (14). For instance, we have already seen that in the interests of computational economy the Jacobian matrix in (15) may be approximated as constant over a number of iterations. Other approximations that serve the purposes of economy and/or reliability and/or simplicity are permissible. Suppose that in (15) we neglect the coupling submatrices J_2 and J_3. If we continue to iterate in the same simultaneous-displacements manner, convergence will have been noticeably weakened. But the differential and algebraic components have been algorithmically decoupled, and we can change to a block-successive iteration scheme. That is, a Newton iteration using J_1 and F_1 is performed to calculate Δy_n and, hence, to update y_n. The new value of y_n is used in constructing F_2 to perform a Newton iteration with J_4 to reevaluate Δx_n, update x_n, and so on. This scheme is none other than a straightforward Partitioned-solution version using Newton network solution as described in Sections VI and VII. Overall convergence is good, and generally even faster than (15), at least for the first couple of iterations (typically 2–3 are needed in total). The methodology is precisely the same as that used to develop the decoupled Newton load flow [31].

The above Partitioned-solution version has the advantages over (15) of lower program complexity and somewhat reduced work per iteration. It is just possible that (15) could be more reliably convergent in extreme cases, primarily when the interface variables are changing rapidly and the step length is big. On the other hand, any form of accuracy control will not permit large steps in any case during such periods. This convergence comparison is one of the main questions to be answered at the present state of the art in the short-term stability calculation. If it turns out that (15) is not fundamentally more reliable, there is no reason to use it. The Partitioned interfacing scheme retains algorithmic flexibility. For instance, the network may not be solved at each integration step, or the machines or machine components may be solved with different integration step lengths when and if any such measures are seen to be attractive.

There are also various hybrid possibilities of unknown or dubious value. Simplifying approximations to J_2 and J_3 in (15) could be made. The differential equations in (15) could be partitioned as in Section VI-H, and iterated block successively.

E. The APS Method

The APS method has something in common with the one in Sections VIII-B and -C. It likewise achieves the Simultaneous solution of all variables during an integration step, using a separate Newton network solution where each 2×2 diagonal block of the Jacobian matrix receives additional terms obtained by reducing the equations of the machines connected at its bus. However, the big difference is that instead of successively substituting incremental algebraized machine equations into the network equation, which is relatively easy and general, the APS method succeeds in substituting the algebraized machine equations themselves. This involves some considerable heuristic algebraic manipulation, which is not reproduced here.

Implicit integration formulas of the types (6) and (7) can both be written, inserting the expression for u from (2b) into them (see (A.9)–(A.11)), in the general form:

$$y_n = M \cdot u(E_n, V_n^e) + K \tag{17}$$

where matrix M and vector K are constant over the step and y_n contains E_n as a subset.

The machine stator currents can be expressed in the network complex frame by premultiplying (A.9) by the trans-

[3]The reader may derive this scheme by writing (15) in partitioned form, expressing Δy as a function of Δx, eliminating Δy, solving for Δx, and finally solving for Δy.

formation operator T^{-1} as some function

$$I_s^e = f_s^e(E_n, V_n^e). \tag{18}$$

The nonlinear functions in (17) and (18) are fairly complicated. However, for standard machine models it is possible to solve (17) analytically for E_n in terms of V_n^e, and substitute this expression into (18) to give the stator currents as a function of V_n^e only. These currents are the machine contributions to the network bus injections (in this approach we need not think in terms of a Norton equivalent of the machine stator).

Then the network equation (12) loses its dependence on E and becomes

$$F^e = I^e(V_n^e) - Y^e \cdot V_n^e \tag{19}$$

where F^e is zero at the solution. We can now solve (19) iteratively by Newton's method for the bus voltages, from which each stator current in the complex frame may be calculated. These are already the final values at the end of the integration step. Now the machine air gap power P_e can be calculated directly. Happily, it turns out that the machine rotor angle δ_n is a function of P_e only, and can therefore be obtained. With it, the stator currents are transformed back into the machine d, q frame, and using (A.9), E_n is found. Finally, the variables y_n are calculated directly from (17). The only iterative process in the above integration step has been the Newton network solution, yet interface error is eliminated. Convergence problems are minimal.

Basically, the integration method used in (17) is the Matrix Exponential method of Section VI-I; but, in order to make certain manipulations tractable, the Trapezoidal rule is used for a couple of differential equations. This partitioning weakens slightly the stability properties of the method as a whole, and [2] shows that falsely stable solutions could be obtained with sufficiently stiff problems. Nevertheless, the method is found to perform very well in the APS production program.

Efficiency tests comparing this method with the Trapezoidal rule as used in Section VIII-B were inconclusive about which is better [2], and the choice between them may rest on other factors, such as flexibility and simplicity for program maintenance. The algebraic manipulations mentioned above depend on the particular structure of the machine model. It is not clear if the method can handle saturation efficiently, or special models, or certain fairly standard features such as where the excitation-control signal is taken from a remote bus.

The general APS methodology can accomodate certain variants. It would be possible to replace the Matrix Exponential method throughout by an implicit multistep method. The obvious candidate would be the Trapezoidal rule, in which case it is speculated that the performance would be very similar, with the advantage of greater simplicity. The network could be solved by the factored-Y^c method, not that this would in general be as good as Newton's method. In the extension of the APS method to cover longer term dynamics, the network is not solved at every integration step.

F. Laplace Inversion

There is an interesting class of highly stable implicit methods based on Laplace inversion [42]-[44]. Apart from its stability, the approach has the attractions of being self-starting and completely noniterative, and very-high-order versions cost commensurately little computation. The method was tried on detailed multimachine power-system problems in [41] and [45]. The following is a brief outline, paraphrasing the description given in [54].

Let us linearize (1) and (2) at the beginning of the current integration step, obtaining

$$\begin{bmatrix} J_1 - \mathcal{I}p & J_2 \\ J_3 & J_4 \end{bmatrix} \cdot \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} M \end{bmatrix} \tag{20}$$

where p is the differential operator, vector M is constant, and the Jacobian matrix elements are very similar to those in (15); in fact, J_3 and J_4 are the same.

The inverse of (20) involves the exponential of its square matrix. To solve for y_n and x_n it is necessary to evaluate this exponential numerically, for which the stable Padé rational polynomial approximants are used. Choosing computationally favorable approximants, the integration over a step becomes expressed by the formula

$$\begin{bmatrix} y_n \\ x_n \end{bmatrix} = \sum_{i=1}^{N} \mathrm{Re}\,\{K_i\}, \quad \text{where} \quad L_i = H_i \cdot K_i. \tag{21}$$

Here, H_i is the matrix in (20) with the operator p replaced by a given complex constant α_i/h, and L_i is a given complex vector containing y_{n-1}. The order of the method is $4N - 1$. Thus choosing $N = 1$ for instance, a third order method is obtained at the cost of solving one large sparse equation to obtain re $\{K_1\}$, each integration step. For a nineteenth-order method, five such equations have to be solved each step.

Reference [41] applies versions up to nineteenth order to the power-system problem. In general the linearization error is dominant, and discourages the versions of higher order than 3 from showing to advantage. This third-order version was found to match the accuracy and maximum step lengths obtainable with the implicit Trapezoidal rule. Reference [45] introduced several extra sophistications into the method, with mild improvements in performance.

As usual, the matrix equation in (21) is solved by first reducing the equations separately for each machine, then adding the resulting terms to the diagonals of J_4, and then performing a network solution. This noniterative scheme might be competitive with the Trapezoidal rule, were it not for the fact that the equation is complex (and not much saving can be made by exploiting the simplicity of the imaginary part of H). As it is, the Laplace inversion approach may possibly be more profitable in other applications, such as small-perturbation dynamic analysis where enormous steps can be made with numerical stability and accuracy.

IX. Further Aspects of the Problem

A. Special Techniques for Stiffness

Problem stiffness arises from the fact that some system variables vary slowly, while others respond very rapidly. Stiffness is thus very much dependent on detail of modeling, particularly in the machine control circuits. Some stiff components are quiescent, meaning that they produce smallamplitude ripples on the main response. Stable integration methods can handle these very well, while the less stable methods can avoid difficulties only by using very small step lengths. Other stiff components undergo large-magnitude oscillation, usually only for a short time following a major disturbance, after which they damp out and become quiescent. Stable methods are also better at handling the rapid large-amplitude responses. Nevertheless, unless the integration is accurate enough (e.g., small step lengths) even stable methods cannot reproduce accurately these responses, but instead have a filtering effect on them. Whether this is dangerous or not depends on the influence of the relevant variables on overall machine and system stability. In programs using explicit methods, or in other words nearly all existing industrial stability programs, it is mandatory to protect against the need for extremely small step lengths to cater for the stiff components. The following measures have been taken in various programs.

1) Model Modification: Machine and control-circuit time constants are checked at data input. Any that are below certain rule-of-thumb values, found by experience with the program, are rejected. Then either the model is reduced by omitting the representation of the relevant fast-varying components, or the short-time constants involved are artificially increased to some computationally acceptable levels. This analytically primitive approach has often been found to be adequate for engineering purposes. However, considerable experience is needed to strike an intelligent balance between the engineering/modeling and analytical/computational aspects of the simulation problem. Even where this experience is available, it is easy to foresee cases where fast components which have a significant effect on the dynamic response are omitted or modified erroneously.

2) Mixed Integration Step Lengths: The stiff parts of the problem may be solved with different step lengths than the nonstiff parts. This has already been noted in the context of not solving the network equations every integration step. The method is applicable only to the Partitioned-solution approach.

Given some nominal value of h, the program can choose at data input the step length h/m (m integer) required for each individual machine, based on the smallest time constant in that machine's equations. Thus while the network is solved at intervals of h, different machines are integrated using step lengths $h/2$, $h/3$, or $h/4$, etc. Such a scheme relies on the extrapolation of u and/or V as described in Sections V-D and E. Interface error cannot be eliminated, but can be controlled by integration.

A variant on the scheme is to choose different step lengths for individual differential equations or the individual partitioned machine component blocks (see Section VI-H). In this case, the small-h integration steps require extrapolated values of certain linking variables in y that are being integrated at larger h.

3) Mixed Integration Methods: It is possible to apply different integration methods to the stiff and nonstiff equations of a machine, the rationale being that the stiff components need a stable method, whereas the nonstiff components can be solved with comparable accuracy by a perhaps less-stable method that is simpler to implement [19]. A satisfactory implementation of this idea depends very much on the precise formulas chosen, and the details of interfacing. In general, it is easiest to apply with explicit integration methods, but certain combinations of formulas permit an explicit/implicit mix, or an all-implicit mix used either in the Partitioned or Simultaneous-solution modes. Viewed from the basis of stability alone, the argument for this scheme does not appear to be too strong. A stable method that can handle the stiff components also tends to perform well on the nonstiff parts, even better than a more-accurate less-stable method (see the comments in Section VI-G).

4) Formal Dynamic Reduction: Rather than remove the stiff components of the problem heuristically by modifying time constants, eliminating transfer-function blocks, or changing from subtransient to transient machine models, etc, systematic model reduction can be carried out. The unwanted eigenvalues are removed from the linearized model. This may change the structure of the equations. The method is most acceptable when applied to the truely linear nonlimited equations of the machine, especially some control circuits. Otherwise the approach is reserved for studies requiring less detailed modeling in the relevant machines or parts of the system.

B. Limits on Variables

Limits impose important constraints, at least during the short-term simulation, on the design and performance of the overall solution method. Upper and lower limits on certain variables in the machine control systems are violated at random times. The most critical are usually those in the excitation control, where modern equipment can force the field voltage and stabilizer signals against limits very rapidly.

The computational situation is worst when the machine is going unstable and the variables bang up and down. In a study where a large number of machines is represented in detail, limiting activity is often widespread, although it will typically be most concentrated in the first second of the response period.

Large integration steps cannot be used to track these events accurately. When very small step lengths have to be used because the problem is stiff and the integration method is unstable, limits can be enforced easily and accurately enough. Once a variable has violated its limit, it is simply clamped at the limit. The effect is not properly transmitted to the rest of the equations until the next step. With methods using larger values of h, simple clamping is not satisfactory (although it still appears to have been rather widely used). Having clamped a variable, the whole system should in theory be resolved. In a method such as implicit Trapezoidal, for instance, this means taking an extra iteration or so during the integration step. In practice, for the sake of economy, only the relevant control circuit or the whole machine may be resolved. Integration methods that use previous values need to be restarted after limit enforcement. Again, it may be enough to restart only for the control circuit or the machine. Limit backoff should be treated the same as limit enforcement. A clamped variable remains on its limit until backoff is detected by testing the sign of its derivative [18].

How accurately the limit on/off points must be simulated depends on the influences of the relevant variables on overall stability. It appears that in many cases there is a lot of margin for error without affecting the machine rotor angle response noticeably. However, the error could make an enormous difference in critically stable systems if a large value of h is used. Consider for instance such a system being solved with a step length of 0.1 s (which several workers have found to be usable in various cases). Then the exciter output or turbine power could stay on or off limits almost 0.1 s too short or too long. which would distort the calculated response considerably.

A result of the above problem is that there is no possibility of using very large step lengths during periods of solution roughness. Thus to no little extent, the presence of limits sabotages research effort to develop more powerful integration methods that can take large steps economically during the short-term response period.

In one sense, there is a very good case for solving the limiting control circuits with smaller step lengths than the other equations. This is compatible with the previously mentioned use of mixed step lengths to cater for stiffness since the fastest variables are often the ones with limits. However, mixed step lengths rely on the use of extrapolation techniques, which themselves introduce approximation. A self-starting integration method is particularly desirable when dealing with limits, and there may be some advantage in using mixed methods as well as mixed step lengths for this purpose. Such schemes have been used with simple explicit integration methods, at the expense of additional program complexity. They would be more difficult to implement with implicit methods, and it is not known how much the net benefit would be, if any.

As a final note, care should be taken when interpreting the model and programming the enforcement of limits. It is quite easy to put the limit in the wrong place relative to the transfer-function block, and thus alter its effect.

C. Solution Accuracy/Speed Considerations

Ensuring a reliably accurate solution while minimizing solution time is the essential problem facing the developer of a stability program, since the two objectives are mutually opposed. The total error at any stage in the solution is an unknown function (some cancellation, some accumulation) of the errors generated at all previous integration steps (see Section II-C).

The only practical way of trying to keep the global error within acceptable bounds is to discover roughly how much to constrain the generation of local errors by a process of experimentation with the program on different problems. The most important means available for influencing the local error is step length selection. As h is reduced, the errors due to truncation, approximation (e.g., extrapolation, linearization), interfacing (if any), and limit-simulation all diminish. Other sources of error not directly related to step length are arithmetic precision and inexact convergence of iterative processes.

Most short-term stability programs still use fixed step lengths. To tune such a program for accuracy/speed, many cases have to be run with different step lengths, making intelligent adjustments to convergence tolerances, extrapolation techniques and orders, matrix refactorization criteria, etc., as appropriate. To check the global accuracy of a given solution, all that can be done is to compare the response with supposedly more accurate ones (e.g., ones obtained with smaller h). It seems that program tuning is more of an art than a science. The optimum tuning is different for different power system problems; therefore some tuning policy is required. Typically, the program will be "optimized" to give economical solutions on the most common range of problems, rather than completely reliable solutions in all cases. Then there should be some device in the program to warn the user if excessive local error at any integration step is suspected. He then has the option to rerun the study at a smaller h.

As regards the values of the step lengths used in practice, there is not a wealth of concrete information in the literature. Research papers usually present results of tests based on incomplete or unrealistic system models. Competitive industry is cautious about publishing too many performance figures about their programs, for obvious reasons. Step lengths are often quoted in cycles, which is not without special significance. With the less-stable integration methods, it is advantageous where possible to use integral multiples of cycles, since the errors due to supply-frequency components (often quiescent) then tend to cancel out over successive steps.

With explicit methods such as Runge-Kutta, 2-4 cycles seems typical for nonstiff problems, but much less for stiff problems. Some programs choose h automatically at data input on the basis of the smallest time constant (say 5-10 times less).

Using implicit integration, the same range 2-4 cycles can accomodate stiffness. It would frequently be possible to use higher values of h, except for the problems of roughness and where fast nonquiescent components need to be solved more accurately.

It should be noted however that in terms of total computing time, the largest steps commensurate with accuracy are not necessarily the best. Increasing h too much reaches a point where even with good extrapolation or prediction methods, the numbers of iterations per step increase. A plot of computing time against h rises sharply at both ends, terminating very soon with solution failure at high h. Fortunately, the curve is usually fairly shallow around the minimum, which helps when tuning.

Even a nominally constant step may have to vary occasionally during the solution to conform with switching times, print intervals, and perhaps restarts after limiting. It is common to reduce h for a few steps after a large disturbance.

As regards midterm and long-term stability programs, few are yet in production use. In them, the network is solved at relatively infrequent intervals, which can be ten or more times the integration step length (4-120 cycles?).

D. Automatic Error Control

In principle, the reliability/accuracy/speed compromise can be resolved much better if a program has automatic error control. In practice, the industry has not developed this area of the stability calculation as much as it might have. The main form of error control is step-length changing, to keep the local error within limits. The general idea is that when any of the variables are changing rapidly, a small value of h is needed; but, when they are changing slowly, h can be increased to more computationally economical values.

Several rule-of-thumb criteria for adjusting h have been used, and are based on the following:

1) rate of change of the variables;
2) comparison between extrapolated values and integrated values of the variables;
3) number of iterations during a step (especially with implicit Trapezoidal rule) on the theory that the number of iterations indicates the rate at which the important variables are changing;
4) comparison between k_2 and k_3 in the Runge-Kutta method of (3) [17].

Such criteria can be tuned to work quite well, provided that not too much sophistication is attempted. It is usually not profitable to try to adjust h every step, nor to allow h to deviate too much from some nominal value.

Step-length changing causes difficulties with the previous values in multistep integration methods, unless Nordsieck versions are used [16], [40]. With extrapolation, nonequidistant formulas can accomodate the step changes without trouble. Using implicit integration, extra work might be needed whenever h changes to refactorize the Jacobian or reevaluate the matrix exponential, depending on the precise method and its implementation. A more mathematically sound criterion for step changing is local truncation error, although in the power-system problem it is not necessarily the predominant error generated over a step. The ease and accuracy with which the truncation error can be estimated varies from method to method. The estimation can be embedded in Runge-Kutta formulas at a little extra cost per step. In the predictor-corrector methods, it can be obtained from the difference between the predicted and corrected values. With many common multistep formulas, the "polynomial" method can be used [2] as follows. The dominant truncation error term of order p has the simple form $kh^p y^{(p)}$, where k is a known constant particular to the formula, and $y^{(p)}$ is the pth derivative of the variable y. If we want to test the truncation error at the end of an integration step, a pth-order polynomial is fitted through the $p + 1$ points comprising the beginning and end of the step plus $p - 1$ stored previous values. Differentiating this polynomial p times gives $y^{(p)}$ and, hence, the truncation error.

A very general method for truncation error estimation is step doubling. Two normal steps of length h are taken, at the end of which any variable is say y_N. Then the integration is repeated using a single step of double length $2h$, giving a new value y_D. If the order of the dominant error term is known to be p, then the truncation error over a single-length step is estimated to be $(y_D - y_N)/(2^p - 2)$. Compared with no error estimation, this scheme costs an extra step in two. However, in practice, the error is not estimated after every pair of steps, and any iterative solutions in the $2h$-step are aided by good starting values already available from the normal-step integration.

Where usable, the polynomial method is more efficient than the step-doubling method. The former relies on previous values, and, in fact, more of them than are required by the multistep formula itself. In the specific case of the Trapezoidal rule, the required two previous values are in any case likely to be available if quadratic extrapolation is used to provide good starting values for the iterative solution.

Explicit integration methods have a much stronger need for error control than do implicit methods, which can tolerate and even recover from temporary periods of high-error generation during the course of a solution. However, automatic step-length adjustment seems to have been used very little in traditional programs. Instead, the emphasis has been on protecting the process from fast components prior to the solution, as described in Section IX-A.

Truncation error control has been used in several implicit integration applications. Reference [20] reported that h can vary between one and 40 cycles (the latter obviously when the power system is substantially returning to the steady state), using the Gear formulas in the Nordsieck mode. Reference [41] used a similar approach, but ran into trouble trying to maintain a reasonable level of accuracy for all variables at all times. At certain stages of the solution, particularly after large disturbances, fast nonquiescent variables in the excitation control and elsewhere will not solve within the given truncation-error tolerance without uneconomically small values of h. In order to avoid excessive solution costs, it is necessary to prohibit h from shrinking to very low values, and, therefore, to suffer the attendant errors for limited periods. Usually, this makes remarkably little difference to the computed system response. Reference [2] performed some very instructive tests, comparing the cost effectiveness of solutions with and without automatic step-length control. In well-behaved stable system responses, it was found that a constant-h approach can maintain the local truncation error more or less within the required range throughout the solution. In such cases, the overheads of automatic control would not be worthwhile. However, in one marginally stable test problem, there was considerable variation in the value of h needed at different stages of the solution to keep the truncation error within bounds.

With automatic step-length control, the computing time for a study on a given power system is very much a function of the severity of the disturbance to the system. Well-implemented control of h should give more reliable solutions, and, while little computational saving over fixed-step methods is expected during perhaps the first second of a stable response when there is rapid movement and limiting is active, step expansion as the system damps down could give significant savings.

In longer term simulation programs, there is the double problem of controlling the integration step length and the frequency with which the network is solved. Reference [25] applies automatic control for the latter only, based on comparing the extrapolated and calculated values of P_e.

E. Arithmetic Precision

Any discussion on solution accuracy should not ignore the question of arithmetic precision. An adequate computer word length for floating-point variables is the important first prerequisite in the whole calculation process. There is some scope for optimization in terms of storage and perhaps in speed. The main solution variables can be stored with high precision, while basic data (of which there is a lot) may not need this. Care must be taken to find out exactly what the computer does with mixed-precision arithmetic statements.

In general, a mixed-precision scheme should only be undertaken with considerable caution. The following remarks, assuming that a uniform word length is used, are based on the author's experiences in implementing short-term stability programs with implicit integration and a factored-Y^c network solution. For large problems, 32-bit words proved to be quite inadequate. The accumulation of error contaminates the solution significantly. This contrasts with other calculations such as load flow, where 32 bits are just sufficient, although high-accuracy convergence (nor normally needed) is often not possible. 36-bit words were dangerously marginal, while 48 bits were reliable. Thus, for example, there is no problem on CDC 60-bit machines, and IBM 360/70 series (32 bits), UNIVAC 1100 series (36 bits) and 24-bit machines working in double precision. Machines based on 16-bit words, mostly minis, must and usually do have a triple-precision facility. However, extended precision often slows the computation down significantly (not IBM). The above factors must also be taken into account when contemplating the use of the emerging generation of powerful microprocessors.

X. Discussion

A. General

This section attempts to translate the information and ideas thus far presented into a view of the state of the art. The direct evidence on which to base such a view is not too great. In this subject, it is much more difficult to draw general conclusions about methods from the results of small-scale algorithmic testing than it is in, say, load-flow analysis. The development of a stability program requires that extensive effort (and cost) be devoted to modeling detail, realistic testing, and algorithmic tuning. Probably there are no two stability programs in industrial use that employ the same numerical methods, especially the smaller details that can affect performance dramatically—extrapolation being an obvious example. There has been little circulation of standard test data to use on different programs, nor any clear basis for evaluating their constituent methods (hence, the comparative study reported in Reference [2]). Naturally, also, the type of power system and its modeling will affect the comparisons between methods.

The following comments are based on the author's critical interpretation of the literature, and on his own experience in the field.

B. Numerical Stability of Integration Methods

In the power-system application, most highly stable (implicit) integration methods cost little or no more computation per step than their explicit counterparts. However, when not limited by solution roughness they permit larger step lengths, and this performance difference increases as the problem becomes more stiff. A number of separate studies have concluded that on typical modern system models, a program using an implicit method will on average be several times faster than a comparable program using an equivalent explicit method.

An industrial stability program is an evolutionary tool, which must accept new modeling features as apparatus changes. New devices tend to have more and more rapid responses. Examples are: excitation controls including power-system stabilizers, electronic-turbine governors with fast valving, controlled static compensation, and *hvdc* links [46]. The traditional nonstiff model is becoming a thing of the past. With an implicit method, protecting the program against fast components, and tuning, are much less critical. Obviously, indiscriminately detailed modeling is not to be encouraged, but it is apparent that to base a program on explicit integration forces severer compromises between accuracy of modeling and computing efficiency.

An implicit method has one drawback—the program structure is more complicated because the algebraized differential equations of each machine must be solved simultaneously (see Section VI-H, however). Where a power utility uses its own program routinely in studies on its own and adjoining systems, this disadvantage is not too great. Whenever a new device is to be studied, it can be added as a new modeling option in the program. On the other hand, some programs that are used as general design tools have very flexible element-by-element model-building facilities in the data input. In these cases, there will be a significant increase in the organizational work in the program to connect the components together for Simultaneous solution.

C. Choice of Integration Method

Restricting attention at this stage to the implicit methods, the Trapezoidal rule seems to be the clearest choice, as concluded also in [2]. It is numerically stable, but does not threaten to produce falsely stable system responses in the presence of fast nonquiescent components, as do most other methods, and it is self-starting. These qualities are of particular importance during the rapid electrical response of the power system after a large discontinuity, when there is typically a lot of excitation-control limiting activity. They can also remain important throughout the solution in the case of marginally stable or unstable systems.

It seems logical to provide automatic step length control during the solution. A well-designed control mechanism is needed, avoiding very frequent and wide changes in h. Truncation error is the safest control criterion to use, but implementationally simpler more heuristic criteria might work equally well in practice. The theory and techniques in this area are still rather weak, and the onus for devising a good mechanism falls largely on the developer of each individual program.

A clever solution control scheme might be able to distinguish when fast components and limiting activity have subsided enough to justify switching from the Trapezoidal rule to higher order multistep formulas. The only difference in implementation is the need for previous values. Usually two of these will anyway be available, since they will be used in the Trapezoidal rule itself to provide good iteration starting values by extrapolation (the extrapolation method can be adapted to use $\dot{y}$ instead of y if necessary). The advantages of higher order Gear formulas will be seen after the first couple of seconds, say, in a stable system response, by permitting step length expansion.

D. Network Solution Methods

This topic is the easiest to comment upon. The factored-Y^c approach may be considered in the absence of nonimpedance loads or if storage is critical. Otherwise, Newton's method with periodic Jacobian matrix refactorization should be used from the viewpoints of reliability of convergence and of overall speed. It may not be convenient for all quantities to include their partial derivatives within the Jacobian matrix, in the strict Newton manner. This applies to special devices with complicated equations. Provided that the power system is not very sensitive to the state of the device, in an incremental sense, these special equations can be iterated in an outer loop without much loss of computational performance.

E. Partitioned versus Simultaneous Interfacing Approaches

The choice between these approaches applies mainly to the implicit multistep methods, including Trapezoidal. As explained in Section VIII-D, the Simultaneous approach would be attractive only if it has superior convergence to the Partitioned scheme. At present, there is no data to establish this superiority. The block-successive Partitioned solution is organizationally and algorithmically more flexible. This flexibility may be exploited in various ways, such as in connecting model blocks together, enforcing limits, or even solving some of the differential equations with different step lengths (see Section VI-H). Comparison studies are needed to evaluate some of these options.

F. Future Algorithmic Developments

At the time of writing, there are no signs of impending algorithmic breakthroughs for the power-system time-domain dynamic simulation. Broadly speaking, the more modern solution processes seem to be utilizing existing knowledge in the numerical methods field practically and effectively, albeit at times a little crudely. In the absence of major advances in the general numerical area, or some revolutionizing transform for the power-system equations, further developments will probably consist mostly of refinements to existing algorithms, especially in the automatic solution control features (see the next section, however).

The more
creased com
problems to
by factors
some of the
creates some
it should be
fast control
stability.

G. Future C

If algorith
hold great p
the computa
most is the
multiple pro
using distrib
sors. Curren
processors, w
There are re
stability calc
twenty times
The distrib
area, althoug
is not yet cl
study, since
Assuming th
bility calcula
ing a few doll
and algebraic
During an in
chines in para
normal serial
be performe
single (larger
and limits th
machine equa
on tearing th
in parallel on
solution of la
tearing than
for roundoff
can be decrea
sors is negligi
put of stabili
portance in re
several differ
including spa
is to make
since much o
handle the eq
Thus in this
being raised as
One possibl
work is that
cated and tir
software, and
and unecono
computing ta
tion. A con
studies might
microprocesso
major cost be
eased through

The more extensive use of dynamic equivalents promises increased computational economy. By reducing the sizes of the problems to be solved, overall computing times can be reduced by factors of several times [47]. Equivalencing eliminates some of the response modes of the original model, and usually creates some degree of shift in the remaining ones. Therefore, it should be applied with care, especially in the presence of fast control apparatus that has a significant effect on system stability.

D. *Future Computational Developments*

If algorithmic development as mentioned above does not hold great promise for much faster and cheaper simulations, the computational area is considerably more hopeful. Foremost is the parallel computation field, either through using multiple processing in large general purpose computers or using distributed processing via many cheap fast microprocessors. Currently, the front runner appears to be the use of array processors, which can be attached to ordinary large computers. There are reports of experiments in which the power-system stability calculation has been speeded up by an impressive twenty times.

The distributed use of microprocessors is also an intriguing area, although its ultimate compatibility with array processing is not yet clear. It offers great potential savings in cost per study, since the main hardware should be very inexpensive. Assuming that the existing numerical approaches to the stability calculation are to be used, an individual processor (costing a few dollars?) is assigned to the solution of the differential and algebraic equations of each machine or machine group. During an integration step, it is now possible to solve all machines in parallel, in negligible lapsed time compared with the normal serial approach. After this, a network iteration has to be performed. Solving the network in the usual way on a single (larger) microprocessor does not exploit parallelism, and limits the overall time saving to that achieved for the machine equations. Therefore, effort is being concentrated on tearing the network into smaller parts that can be solved in parallel on separate processors [48]-[50]. Generally, the solution of large sparse networks takes more operations with tearing than without, but this is not so important (other than for roundoff accumulation) if the absolute computing time can be decreased significantly. Where the cost of extra processors is negligible, the main challenge is to maximize the throughput of stability cases per unit time—this is of particular importance in real-time dynamic security assessment. There are several different ways of approaching the network tearing, including sparse diakoptics [51]-[53]. The biggest problem is to make optimal automatic sectioning of the network, since much of the existing tearing theory tells only how to handle the equations after the torn sections have been chosen. Thus in this and other respects, new algorithmic aspects are being raised as a result of parallel computation.

One possible eventual outcome of the distributed processing work is that the general-purpose computer with its complicated and time-consuming operating systems, vast support software, and high built-in versatility will become unnecessary and uneconomical for the routine performance of specific computing tasks such as the power-system stability calculation. A company that is heavily committed to stability studies might be able to maintain a special-purpose multi-microprocessor configuration solely for these studies, the major cost being in the software. Implementation will be based through standard hardware and operating-software modular architectures. The main problems will relate to the sharing of central data bases and peripherals.

With the availability of sufficient core storage, the stability calculation is also a natural application for code generation. This involves the generation by software of an object code for the execution of the program on each specific power-system problem, eliminating indirect addressing and other overheads. In conventional runs, these overheads are wastefully repeated during a calculation, every iteration and every integration step. The computational saving, estimated at no less than 50 percent, could justify the preparation of a code generation routine for a company that performs many large stability runs on a given computer.

Finally, it is evident that in many cases, significant economies can be made by giving greater attention to the time-consuming input–output aspects of stability programs.

In the face of the above developments, it seems unlikely that today's concepts of hybrid computation will survive very long.

XI. Conclusions

This paper has attempted to provide a general picture of how the conventional power-system dynamic response calculation is carried out. The subject is large and complex, and it is not easy for one paper to do justice to the whole area. For instance, the algorithmic variants needed to handle special models with slightly different structures have not been covered. It is hoped that the solution methodologies and techniques described here will serve as a guide to the appropriate schemes in such cases.

In the spirit of a critical review, the paper has recommended certain specific methods as being very suitable for large-scale short-term simulations using conventional models. Briefly, an excellent general scheme seems to be the implicit Trapezoidal rule, interfaced in the Partitioned manner with a Newton network solution. Extrapolation (prediction) techniques are essential, and automatic accuracy control is very desirable. The possibility of switching to higher order implicit multistep formulas should not be ignored. There is still scope for the refinement of a number of algorithmic aspects.

Although the technical and economic need for improvements in the stability calculation is greater than ever, it seems that there is little real development in the conventional algorithmic sense. The state of the art as indicated in the paper has not changed essentially for some years. Even so, the inertia against writing new even more complicated software is such that many industrial programs throughout the world are still using outdated methods and suffering excessive computing time and/or study restrictions as a consequence.

Most of the development potential appears to lie, not in the conventional algorithmic sphere, but in the computation itself, much of which is hardware-related. Parallel computation, in various different forms, is destined to reduce solution times and costs considerably, but is generating new calculation problems of both program-organization and algorithmic natures. The most spectacular improvements will be achieved if the various methods can be used in combination with each other.

This paper has dealt only with the response calculation process itself, and not the many related issues. One of the biggest single potentials for computational economy will always be in the engineering choice of study cases and the interpretation of results. A thorough rationalization of modeling practices versus simulation accuracy, and how they affect computing effort with the modern calculation methods, would be extremely helpful. These are still very intuitive areas. The industry has been and still is looking hard at ways of avoiding or mitigating the need for large-scale brute-force simulations, through approaches such as dynamic equivalencing and reduction, pattern recognition, hybrid computation, and Liapunov.

Fast, reliable, and inexpensive stability diagnoses are needed in critical phases of power-system planning, operation, and now real-time security monitoring and control. The volume of the calculation process continues to offer a great challenge to analytical and computational specialists from the power industry and other disciplines.

XII. Appendix—Typical Simple Machine Model

This Appendix illustrates the types of equations that are being treated throughout the review by showing a simple but commonly used machine model.

A. Differential Equations

Referring to Fig. 1, the rotor dynamic equations are

$$\dot{\delta} = \omega \qquad \text{(A.1)}$$

$$\dot{\omega} = (P_m - P_e - D\omega)/H \qquad \text{(A.2)}$$

where D is a damping factor and H is the inertia.

The rotor electrical equations for a model with transient effects only are:

$$\dot{E}'_q = -(E'_q - (X_d - X'_d)I_{sd} - E_f)/T'_{do} \qquad \text{(A.3)}$$

$$\dot{E}'_d = -(E'_d + (X_q - X'_q)I_{sq})/T'_{qo} \qquad \text{(A.4)}$$

where the X's are synchronous and transient reactances, and the T's are open-circuit time constants.

In (A.2), P_m is the mechanical power output of the turbine governor system, which is represented by a (usually) linear set of differential equations with at least one limiter. Similarly, E_f in (A.3) is the field voltage output of the excitation-control system, which again is represented by a set of differential equations. These equations are linear, except perhaps for saturation in one or two circuits. The time constants in the excitation control may be relatively low, and at least one circuit has a limiter.

$$\begin{array}{c}\text{excitation control}\\ \\ \text{rotor electrics}\\ \\ \text{rotor mechanics}\\ \\ \text{turbine governor}\end{array}
\underbrace{\begin{bmatrix}\dot{E}_s\\ \dot{E}_f\\ \dot{E}'_q\\ \dot{E}'_d\\ \dot{\delta}\\ \dot{\omega}\\ \dot{P}_m\\ \dot{P}_g\end{bmatrix}}_{y}
=
\underbrace{\begin{bmatrix}
\frac{T_e+K_s}{-T_eT_s} & \frac{-K_s}{T_eT_s} & & & & & & \\
\frac{-K_e}{T_e} & \frac{-1}{T_e} & & & & 0 & & \\
 & \frac{1}{T'_{do}} & \frac{-1}{T'_{do}} & 0 & & & & \\
 & & 0 & \frac{-1}{T'_{qo}} & & & & \\
 & & & & 0 & 1 & & \\
 & & & & 0 & \frac{-D}{H} & \frac{1}{H} & \\
 & & 0 & & & & \frac{-1}{T_t} & \frac{-1}{T_t} \\
 & & & & & \frac{-K_g}{T_g} & 0 & \frac{-1}{T_g}
\end{bmatrix}}_{A}
\underbrace{\begin{bmatrix}E_s\\ E_f\\ E'_q\\ E'_d\\ \delta\\ \omega\\ P_m\\ P_g\end{bmatrix}}_{y}$$

$$+\underbrace{\begin{bmatrix}
\frac{K_eK_s}{T_eT_s} & & & & & \\
\frac{K_e}{T_e} & & & 0 & & \\
 & \frac{X_d-X'_d}{T'_{do}} & & & & \\
 & & \frac{X_q-X'_q}{-T'_{qo}} & & & \\
 & & & & & \\
 & & & \frac{-1}{H} & & \\
 & & & & \frac{1}{T_t} & \\
 & & & & & \frac{K_g}{T_g}
\end{bmatrix}}_{B}
\underbrace{\begin{bmatrix}\Delta|V|\\ I_{sd}\\ I_{sq}\\ P_e\\ \omega_r\\ P_r\end{bmatrix}}_{u} \qquad \text{(A.5)}$$

Here, we shall introduce the simplest possible turbine and citation control models, merely to demonstrate the equan forms. Combining these equations with (A.1)–(A.4) gives .5), which corresponds exactly to (1a). In (A.5) the K's gains, and subscripts e, s, t and g refer to the exciter, citation-control stabilizer, turbine, and governor transfernction blocks, respectively. The governor speed reference and the power set point P_r have been added to vector u to e compatibility with the form of (1a). These quantities are nstant, however, and thus have not received special atten n in the main text.

Machine Stator Algebraic Equations

n the machine rotor d, q reference frame, the stator voltageop equation is

$$\begin{bmatrix} E_d' - V_d \\ E_d' - V_q \end{bmatrix} = \begin{bmatrix} R_s & -X_q' \\ X_d' & R_s \end{bmatrix} \cdot \begin{bmatrix} I_{sd} \\ I_{sq} \end{bmatrix} = Z_{dq} \begin{bmatrix} I_{sd} \\ I_{sq} \end{bmatrix}. \qquad (A.6)$$

e transmission network is described in the complex (real d imaginary) or "synchronously rotating" reference frame. order to solve the complete system, each stator equation st be transformed into the network frame. This transmation is simply a rotation of (A.6) through the rotor le δ. This difference in coordinate systems arises because machine saliency (dependence on rotor position δ of magic circuit reluctant and hence machine impedance) for ich the d, q axis approach provides an elegant analytical plification.

he transformation can be expressed symbolically in either the forms

$$q + j \cdot d = \exp(-j\delta\,(\text{re} + j \cdot \text{im}))$$

$$\begin{bmatrix} d \\ q \end{bmatrix} = \begin{bmatrix} -\sin\delta & \cos\delta \\ \cos\delta & \sin\delta \end{bmatrix} \cdot \begin{bmatrix} \text{re} \\ \text{im} \end{bmatrix} = T. \begin{bmatrix} \text{re} \\ \text{im} \end{bmatrix}. \qquad (A.7)$$

plying this transformation to (A.6), we obtain the stator ations in the network coordinate system:

$$\begin{bmatrix} E_{re}' - V_{re} \\ E_{im}' - V_{im} \end{bmatrix} = T^{-1}. \begin{bmatrix} R_s & -X_q' \\ X_d' & R_s \end{bmatrix} .T. \begin{bmatrix} I_{sre} \\ I_{sim} \end{bmatrix} = Z_s. \begin{bmatrix} I_{sre} \\ I_{sim} \end{bmatrix}. \qquad (A.8)$$

As defined by (A.8), Z_s is the stator impedance viewed from the synchronously rotating network frame.

After these preliminaries, we can now consider the computation of the variables u in (1a) or (A.5), as functions of the machine terminal voltage $V = V_{re} + jV_{im}$ and of the variables E_d', E_q' and δ, which comprise the subset E of y. An expression for the stator d, q currents is obtained by transforming V_d and V_q into the network frame, and solving (A.6). Thus

$$\begin{bmatrix} I_{sd} \\ I_{sq} \end{bmatrix} = Z_{dq}^{-1}. \begin{bmatrix} E_d' \\ E_q' \end{bmatrix} - Z_{dq}^{-1}.T. \begin{bmatrix} V_{re} \\ V_{im} \end{bmatrix}. \qquad (A.9)$$

Here, matrix T is a function of δ. The air gap power is now given by:

$$P_e = E_d' \cdot I_{sd} + E_q' \cdot I_{sq} \qquad (A.10)$$

where I_{sd} and I_{sq} are given by (A.9). The terminal voltage magnitude error is simply

$$\Delta|V| = V_r - (V_{re}^2 + V_{im}^2)^{1/2} \qquad (A.11)$$

where V_r is the voltage regulator setting. Equations (A.9)–(A.11) now correspond to (2b).

C. Calculation of Initial Conditions

The initial network operating state for a stability study is obtained by a load-flow solution. It is now necessary to calculate the initial steady-state operating point for each machine. Where there are parallel units at a bus, the power output of each machine or, especially in the case of reactive power the proportion of the total bus power supplied by each machine, must be specified.

Knowing the terminal voltage and complex power of a machine, the complex stator current is also known. At this point, the general method would be to solve simultaneously the nonlinear algebraic set comprising the differential equations, with their left-hand sides set to zero, plus a version of (A.8) with the E' terms transformed into the d, q frame. However, a much easier approach is used. From the machine analysis on which the equations were based, and in a nonobvious form from the equations themselves, the phasor $V + (R_s + jX_q)I_s$ lies on the q axis. Hence, it is easy to find the rotor angle δ between the real axis in the network frame and the machine q axis. With this, I_s and V can be transformed into their d, q components. E_d' is given directly from (A.4) and E_q' is found from (A.6). The steady-state value of E_f now follows immediately from (A.3).

Saturation as usual introduces complications. The machine reactances are functions of the operating point. The easiest way of handling this is to start assuming no saturation, and then repeat the above initialization process iteratively, successively updating the reactances from the machine's open-circuit saturation curve until converged. Whether or not saturation subsequently has an appreciable effect on the machine's response, apparently it is important to represent it in the initial-condition calculation. This is because it affects the initial value of E_f and, hence, the distance that E_f has to travel before it hits the excitation ceiling.

After this stage in the initialization, it is relatively easy to calculate the remaining variables. All the machine state variables are of course zero. P_e comes from (A.10), and is equal to P_m. It may possibly be necessary to perform an iterative initialization in the excitation control block because of saturation, but this can be avoided if the saturation characteristic is approximated simply enough.

ACKNOWLEDGMENT

The author is grateful to CEPEL (Centro de Pesquisas de Energia Elétrica, the Brazilian electric power industry research center) for the facilities to write this paper, and particularly for the encouragement of J. Lepecki, A. Mossé, and J. Barros. He acknowledges the very beneficial former research collaboration with C. P. Arnold, V. Zakian, D. H. Vorley, and many others. He thanks F. F. Wu for suggesting that this paper be prepared and J. M. Dembart for his helpful correspondence. The comments made by O. Alsaç on the draft of the paper have been very valuable.

REFERENCES

[1] W. F. Tinney, "Evaluation of concepts for studying transient stability," in *IEEE Tutorial Course: Modern Concepts of Power Systems Dynamics.* New York: IEEE, 1970.

[2] "Power system dynamic analysis phase I," Boeing Computer Services, EPRI EL-484 Project 670-1, Final Report, July 1977.

[3] "Symposium on adequacy and philosophy of modeling," prepared by IEEE Working Group on Dynamic System Performance and presented at IEEE PES 1975 Winter Meeting (IEEE Publ. no. 75-CH0970-4-PWR).

[4] J. M. Undrill, "Structure in the computation of power system nonlinear dynamical response," *IEEE Trans. Power App. Syst.*, vol. PAS-88, pp. 1-5, Jan. 1969.

[5] C. C. Young, "Equipment and system modeling for large-scale stability studies," *IEEE Trans. Power App. Syst.*, vol. PAS-91, pp. 99-109, Jan/Feb, 1972.

[6] J. M. Undrill, "Equipment and load modeling in power system dynamic simulation," in *Proc. Conf. Systems Engineering for Power: Status and Prospects* (Henniker, NH), pp. 394-418, Aug. 1975 (ERDA Publ. no. CONF-750-867).

[7] "Specifications for tomorrow's power flow and dynamic stability programs," in *Western Systems Coordinating Council Committee Report, IEEE PICA Conf.*, pp. 22-27, 1969.

[8] P. L. Dandeno, R. L. Hauth, and R. P. Schulz, "Effects of synchronous machine modeling in large scale system studies," *IEEE Trans. Power App. Syst.*, vol. PAS-92, pp. 574-582, Mar./Apr. 1973.

[9] P. L. Dandeno, P. Kundur, and R. P. Schulz, "Recent trends and progress on synchronous machine modeling in the electric utility industry," *Proc. IEEE*, vol. 62, pp. 941-950, July 1974.

[10] IEEE Committee Report, "Computer representation of excitation systems," *IEEE Trans. Power App. Syst.*, vol. PAS-87, pp. 1460-1468, June 1968.

[11] IEEE Committee Report, "Dynamic models for steam and hydro turbines in power system studies," *IEEE Trans. Power App. Syst.*, Vol. PAS-92, pp. 1904-1915, Dec. 1973.

[12] IEEE Committee Report, "System load dynamics—simulation effects and determination of constants," *IEEE Trans. Power App. Syst.*, vol. PAS-92, pp. 600-610, Mar./Apr. 1973.

[13] C. Concordia, D. R. Davidson, D. N. Ewart, L. K. Kirchmayer, and R. P. Schultz, "Long term power system dynamics—A new planning dimension," CIGRÉ, Paris, France, 1976.

[14] R. D. Dunlop, D. N. Ewart, and R. P. Schulz, "Use of digital computer simulations to assess long-term power system dynamic response," *IEEE Trans. Power App. Syst.*, vol. PAS-94, pp. 850-857, May/June, 1975.

[15] R. T. Byerly, "Large-scale dynamic analysis facilities," presented at Power Syst. Dynamics Symp., UMIST, Manchester, England, Sept. 1973.

[16] G. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations.* Englewood Cliffs, NJ: Prentice-Hall, 1971.

[17] L. Lapidus and J. H. Seinfeld, *Numerical Solution of Ordinary Differential Equations.* New York: Academic Press, 1971.

[18] H. W. Dommel and N. Sato, "Fast transient stability solutions," *IEEE Trans. Power App. Syst.*, vol. PAS-91, pp. 1643-1650, July/Aug. 1972.

[19] G. Gross and A. R. Bergen, "A class of new multistep integration algorithms for the computation of power system dynamic response," presented at IEEE PES Winter Meeting, New York, 1976, Paper F76-132-1.

[20] M. M. Adibi, P. M. Hirsch, and J. A. Jordan, "Solution methods for transient and dynamic stability," *Proc. IEEE*, vol. 62, pp. 951-958, July 1974.

[21] W. D. Humpage, K. P. Wong, and Y. W. Lee, "Numerical integration algorithms in power-system dynamic analysis," *Proc. Inst. Elec. Eng.*, vol. 121, pp. 467-473, June 1974.

[22] W. D. Humpage and B. Stott, "Predictor-corrector methods of numerical integration in digital computer analysis of power sys tem transient stability," *Proc. Inst. Elec. Eng.*, vol. 112, pp. 1557-1565, Aug. 1965.

[23] G. Steinbrenner and D. P. Gelopulos, "An algorithm for tran sient and dynamic stability solutions without interface err" presented at *IEEE PICA Conf.*, pp. 113-117, 1973.

[24] J. F. Luini, R. P. Schulz, and A. E. Turner, "A digital comput program for analyzing long term dynamic response of power sys tems," presented at *IEEE PICA Conf.*, pp. 136-143, 1975.

[25] V. Converti, D. P. Gelopulos, M. Housley, and G. Steinbren "Long-term stability solution of interconnected power system" *IEEE Trans. Power App. Syst.*, vol. PAS -95, Jan./Feb. 197 Also PSCC, Cambridge, Sept. 1975.

[26] J. E. Van Ness and F. B. Kern, "Use of the exponential of the system matrix to solve the transient stability problem," *IEEE Trans. Power App. Syst.*, vol. PAS-89, pp. 83-88, Jan. 1970.

[27] P. L. Dandeno and P. Kundur, "A non-iterative transient stabil program including the effects of variable load-voltage characte istics," *IEEE Trans. Power App. Syst.*, vol. PAS-92, pp. 14 1484, 1973.

[28] N. Stanton and S. N. Talukdar, "New integration algorithms fo transient stability studies," *IEEE Trans. Power App. Syst.*, vo PAS-89, pp. 985-991, May/June 1970.

[29] S. N. Talukdar, "Iterative multistep methods for transient sta bility studies," *IEEE Trans. Power App. Syst.*, vol. PAS 90, pp 96-101, Jan./Feb. 1971.

[30] F. Plitman and M. Iza, "Transient stability problem solution using the Hamming predictor-corrector method," presented at IEEE PES Summer Meeting, Portland, OR, July 1971, paper TP 592-PWR.

[31] B. Stott, "Review of load flow calculation methods," *Proc. IEEE*, vol. 62, pp. 916-929, July 1974.

[32] D. W. Olive, "New techniques for the calculation of dynamic sta bility," *IEEE Trans. Power App. Syst.*, vol. PAS-85, pp. 767-7 July 1966.

[33] F. P. de Mello and D. N. Ewart, "Face—a digital dynamic anal program," *IEEE PICA Conf.*, pp. 83-94, 1967.

[34] W. Janischewskyj and P. Kundur/P. L. Dandeno and P. Kund "Simulation of the non-linear dynamic response of interconne synchronous machines, Parts I and II," *IEEE Trans. Power A Syst.*, vol. PAS-91, pp. 2064-2075, Sept./Oct. 1972.

[35] H. E. Brown, H. H. Happ, C. E. Person, and C. C. Young, "Tran sient stability solution by an impedance matrix method," *IE Trans. Power App. Syst.*, vol. PAS-84, pp. 1204-1212, De 1965.

[36] W. F. Tinney and J. W. Walker, "Direct solutions of sparse ne work equations by optimally ordered triangular factorization *Proc. IEEE*, vol. 55, pp. 1801-1809, Nov. 1969.

[37] W. F. Tinney, W. L. Powell, and N. M. Peterson, "Spa oriented network reduction," in *IEEE PICA Conf.*, 1973.

[38] H. W. Dommel, W. F. Tinney, and W. L. Powell, "Further velopments in Newton's method for power system ap tions," presented at the PES Winter Meeting, New York, 19 IEEE Paper no. 70, CP 161-PWR.

[39] G. W. Gear, "Simultaneous numerical solution of differen algebraic equations," *IEEE Trans. Circuit Theory*, vol. CT pp. 89-95, Jan. 1971.

[40] H. L. Fuller, P. M. Hirsch, and M. B. Lambie, "Variable in tion step transient analysis—VISTA," in *IEEE PICA Conf.*, 156-161, 1973.

[41] D. H. Vorley, "Numerical techniques for analysing the sta of large power systems," Ph.D. dissertation, Univ. Manch England, 1974.

[42] V. Zakian, "Rational approximants to the matrix exponent *Electron. Lett.*, vol. 6, no. 25, pp. 814-815, 1970.

[43] V. Zakian, "Properties of I_{MN} and J_{MN} approximants and plications to numerical inversion of Laplace transform initial-value problems," *J. Math. Analysis Appl.*, vol. 50, 191-222, 1975.

[44] K. Singhal, J. Vlach, and M. Nakla, "Absolutely stable, order method for time domain solution of networks," *Elek. Ubertragung.*, pp. 157-166, 1976.

[45] C. P. Arnold, "Solutions of the multi-machine power stability problem," Ph.D. dissertation, Univ. Manchester, gland, 1976.

[46] "A description of discrete supplementary controls for sta IEEE Task Force Report, *IEEE Trans. Power App. Syst.*, PAS-97, pp. 149-165, Jan./Feb. 1978.

[47] "Development of dynamic equivalents for transient sta studies," EPRI EL-456 Project 763, Final Rep. by Sy Control Inc., May 1977.

[48] W. L. Hatcher, F. M. Brasch, and J. E. Van Ness, "A fea study for the solution of transient stability problems by processor structures," *IEEE Trans. Power App. Syst.*, vol 96, pp.

[49] J. E. Van Ness and F. M. Brasch, "Applications of multip sors to power system stability problems," presented at