INSTRUCTIONAL USE OF DECLARATIVE LANGUAGES FOR THE STUDY OF MACHINE TRANSIENTS

F. L. Alvarado, Senior Member C. A. Cañizares, Student Member
The University of Wisconsin, Madison, WI

A. Keyhani, Member B. Coates, Student Member
Ohio State University, Columbus, OH

ABSTRACT

This paper illustrates uses of high level and declarative programming environments for the teaching of machine transients. It demonstrates the clarity and flexibility afforded by this new generation of mathematical software environments. Three environments are compared: Speakeasy, PC-Matlab, and SOLVER-Q. Results obtained using these environments are compared with simulations obtained using the Electromagnetic Transients Program (EMTP). Numerical results of simulations are similar, with minor differences between constant and adaptive time step programs. Execution times differed, with environments that emphasize symbolic computation slower than those that are able to perform numeric simulations alone. The paper also compares the clarity of the expressions needed to produce a simulation in each environment. Keywords: Education, simulation, electric machines, declarative languages.

INTRODUCTION

Simulation of complex systems such as machines has often been thought of as requiring specialized tools. Many specialized programs for the simulation of machine problems have been developed over the years, such as the EMTP [1]. These programs are often cumbersome, large, lack flexibility, and are difficult to debug and maintain.

In several cases it has been possible to use or adapt relatively general purpose programs for the simulation of machine transients. An important example is the use of ACSL [2,3]. General purpose programs, however, often lack some of the necessary constructs to make dealing with machine equations as clear and efficient as it should be.

The trend recently is toward higher level languages and declarative environments using an object oriented programming style [4-6]. These environments permit a nearly direct transition from mathematical expressions to simulations. Preparation of data and programming at this level becomes nearly negligible; once the user understands the equations of interest, the environments are capable of performing all other necessary aspects of the simulation.

This paper introduces no new equations or new theory about machine modelling. Instead, it illustrates different ways in which generally accepted models of machines can be represented in various high level environments. Thus, we begin this paper with a description of the established equations for one type of machine that we intend to use as an example throughout this paper.

90 WM 071-1 PWRS A paper recommended and approved by the IEEE Power Engineering Education Committee of the IEEE Power Engineering Society for presentation at the IEEE/PES 1990 Winter Meeting, Atlanta, Georgia, February 4 - 8, 1990. Manuscript submitted August 21, 1989; made available for printing December 6, 1989.

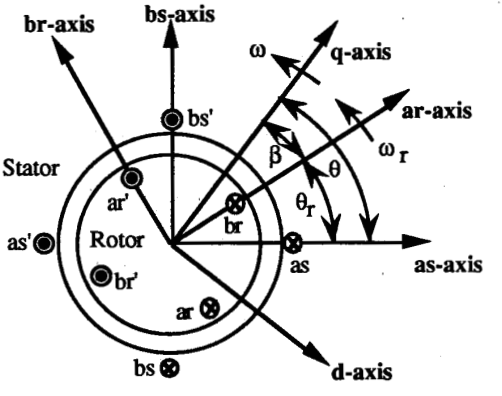
THE EQUATIONS FOR A TWO PHASE INDUCTION MOTOR

This paper demonstrates the simulation of transients produced by cold start-up of a two phase induction motor [9]. Equations (1) and (2) represent the electrical equations for the ideal two phase induction motor.

$$v_{as} = r_s\, i_{as} + \frac{d\lambda_{as}}{dt} \qquad v_{bs} = r_s\, i_{bs} + \frac{d\lambda_{bs}}{dt} \tag{1}$$

$$v_{ar} = r_r\, i_{ar} + \frac{d\lambda_{ar}}{dt} \qquad v_{br} = r_r\, i_{br} + \frac{d\lambda_{br}}{dt}$$

$$\begin{aligned}
\lambda_{as} &= L_s\, i_{as} + L_{sr}[\cos(\theta_r) i_{ar} - \sin(\theta_r) i_{br}] \\
\lambda_{bs} &= L_s\, i_{bs} + L_{sr}[\sin(\theta_r) i_{ar} + \cos(\theta_r) i_{br}] \\
\lambda_{ar} &= L_r\, i_{ar} + L_{sr}[\cos(\theta_r) i_{as} + \sin(\theta_r) i_{bs}] \\
\lambda_{br} &= L_r\, i_{br} + L_{sr}[-\sin(\theta_r) i_{as} + \cos(\theta_r) i_{bs}]
\end{aligned} \tag{2}$$



Where L_s and L_r stand for the self inductances of the stator and rotor circuits, and L_{sr} represents the mutual inductance between stator and rotor. The electrical angle θ_r is related to the actual mechanical displacement of the rotor θ_{mr} by the number of poles P of the machine.

$$\theta_r = \frac{P}{2}\,\theta_{mr} \qquad \omega_r = \frac{d\theta_r}{dt} \tag{3}$$

Equations (1) and (2) have time-varying inductances which make them difficult to simulate. This problem can be avoided by using an orthogonal d-q reference frame represented by the transformation equations (4) and (5). In these equations f stands either for voltage v, current i, or flux linkage λ.

$$\begin{aligned}
f_{qs} &= f_{as}\cos(\theta) + f_{bs}\sin(\theta) \\
f_{ds} &= f_{as}\sin(\theta) - f_{bs}\cos(\theta)
\end{aligned} \tag{4}$$

$$\begin{aligned}
f_{qr} &= f_{ar}\cos(\beta) + f_{br}\sin(\beta) \\
f_{dr} &= f_{ar}\sin(\beta) - f_{br}\cos(\beta)
\end{aligned} \tag{5}$$

$$\beta = \theta - \theta_r \tag{6}$$

Using these transformations phase quantities are converted to d-q variables.

0885-8950/91/0200-0407$01.00 © 1991 IEEE

$$v_{qs} = r_s\, i_{qs} + \frac{d\lambda_{qs}}{dt} + \lambda_{ds}\frac{d\theta}{dt} \qquad (7)$$

$$v_{ds} = r_s\, i_{ds} + \frac{d\lambda_{ds}}{dt} - \lambda_{qs}\frac{d\theta}{dt}$$

$$v'_{qr} = r'_r\, i'_{qr} + \frac{d\lambda'_{qr}}{dt} + \lambda'_{dr}\frac{d\beta}{dt}$$

$$v'_{dr} = r'_r\, i'_{dr} + \frac{d\lambda'_{dr}}{dt} - \lambda'_{qr}\frac{d\beta}{dt}$$

$$\lambda_{qs} = L_{\ell s}\, i_{qs} + L_{ms}\,(i_{qs} + i'_{qr}) \qquad (8)$$

$$\lambda_{ds} = L_{\ell s}\, i_{ds} + L_{ms}\,(i_{ds} + i'_{dr})$$

$$\lambda'_{qr} = L'_{\ell r}\, i'_{qr} + L_{ms}\,(i_{qs} + i'_{qr})$$

$$\lambda'_{dr} = L'_{\ell r}\, i'_{dr} + L_{ms}\,(i_{ds} + i'_{dr})$$

$$L_{\ell s} = L_s - L_{ms} \qquad L'_{\ell r} = L'_r - L_{ms} \qquad (9)$$

$$L_{ms} = \frac{N_s}{N_r} L_{sr}$$

The mechanical equations for this machine are illustrated in (10), where T_e is the electromagnetic torque, J is the inertia, D is the damping, and T_L is the load torque. For our discussion, we consider T_L constant.

$$T_e = \frac{P}{2} L_{ms}\,(i_{qs}\, i'_{dr} - i_{ds}\, i') \qquad (10)$$

$$T_e = \frac{2}{P} J \frac{d\omega_r}{dt} + \frac{2}{P} D\, \omega_r + T_L$$

For our discussion, we consider that the machine is connected to an infinite bus (an ideal voltage source). The rotor is assumed to be short circuited.

$$v_{as} = \sqrt{2}\ \mathrm{VRMS}_{as} \cos(\omega_e t) \qquad (11)$$

$$v_{bs} = \sqrt{2}\ \mathrm{VRMS}_{bs} \sin(\omega_e t) \qquad \omega_e = 2\pi f$$

$$v_{ar} = v_{br} = 0 \qquad (12)$$

DECLARATIVE STYLE PROGRAMMING

Declarative style programming refers to environments where the emphasis is not on the *solution steps* to a problem, but on the *formulation* of the problem. The objective is to express problems as much as possible not in the language of computers but in the natural language of the discipline. The declarative programming era in engineering can be traced at least in part to Konopasek et al. [4]. For more recent descriptions of the declarative style of programming see [5,6]. The ideas of declarative style programming can be extended to incorporate graphics as part of fully graphic declarative languages [7,8].

In a declarative environment there exist constructs that permit the mathematical description of a problem at the level of abstraction a user is used to in a discipline. For induction motors, this means that the equations above are all that should be necessary. Actions to be performed upon these equations are not part of the fundamental problem description. For example, a request to perform a numeric simulation using a given numerical method is irrelevant to the *problem specification*. Declarative environments must also allow for *commands* that permit users to perform actions on models. The richer the command set, and the more it is separated from problem specification, the better we can judge the declarative environment to be. In some environments, commands are embedded as special language constructs, while in others all commands are interactive; no commands are permitted within the problem specification.

The purpose of mathematical models is to enhance student understanding of the behavior of physical systems. An important aid in understanding the behavior of physical systems is to gain insight into the relationship between parameters and the effect of changes to these parameters. This understanding can be obtained by inspection of symbolic expressions, or it can be obtained by numerical experimentation with parameter values. Two common ways of depicting parameter-variable relationships are tabular displays and plots. Good declarative environments should have the ability of producing both of these.

Symbolic expressions are useful toward the goal of better understanding of model behavior only as long as they are simple enough that insight is possible. An explicit symbolic expression is one where variables of interest are explicit in terms of parameters and other variables, as in y=f(x,p). An implicit expression is one where the variables of interest are not necessarily separable from the parameters and remaining variables, as in f(x,y,p)=0. Little insight is gained from complicated symbolic expressions. Thus, a simple *implicit* model is generally better than a complicated *explicit* model. In the simulation of machines, it has often been considered necessary for users to develop explicit models. Good declarative environments should make explicit models unnecessary. This paper illustrates both explicit and implicit versions of the machine model.

SPEAKEASY

Speakeasy [10,11] is a user-oriented programming language more than a declarative environment. It is designed for interactive computations. It permits the use of free-format data input, and provides the means for quick formulation of engineering problems and their solutions. Many mathematical operations routinely used in engineering problems are built-in. The language can be quickly learned, since its form is similar to ordinary mathematical notation. Speakeasy is a step up from general purpose simulation software such as ACSL. It permits the user to work with matrices. However, it is not entirely declarative: statements must be numbered, their order is significant, and the solution method must be coded by the user. To demonstrate the capabilities of Speakeasy, consider the matrix form of the induction machine current equations:

$$L\,\dot{I} = R\,I + D\,U \qquad (13)$$

In Speakeasy we must make the state variable vector *explicit*. Thus, we solve for $\dot{I}$:

$$\dot{I} = A\,I + B\,U \qquad (14)$$

where:

$$A = L^{-1} R \qquad B = L^{-1} R \qquad (15)$$

In the case of unsaturable machine inductances, the inductance matrix needs to be inverted once. This can done by Speakeasy, as shown in figure 1(a), line 31 of the Speakeasy program. Since the R matrix is a function of ω_r, the matrices A and U are not constant and are updated each time step using the subroutine designated as AU. At this point the matrices A, B, and U have been obtained and can be used by the explicit predictor-corrector trapezoidal algorithm. The predicted value of currents, I_{n+1}, are found in figure 1(b), line 45, and are left in matrix form, IB. The corrected values of current, I_{n+1}, are found by using the previously predicted value of current, and are shown in figure 1(b), lines 56-60. The corrected values of current are then used to calculate the rotor speed and the electromagnetic torque.

Figure 1(a) provides the machine parameters. Figure 1(c) computes the matrix A and the vector U for use in the trapezoidal integration. The sampling time, equation (16), was calculated based on the largest real part of the matrix A eigenvalues.

$$\text{Largest eigenvalue} = \alpha + j\omega \qquad (16)$$

$$\text{Smallest settling time (SST)} = \frac{1}{\alpha}$$

$$\text{Sampling time } (\Delta T) = \frac{1}{K}\,\text{SST}$$

In this study K was chosen to be equal to 4.3. Three different cases simulating a cold start-up of a two phase induction motor were run for the following machine:

$r_s = 0.295$ $\quad L_{ms} = 35.14$ mH $\quad r'_r = 0.144\ \Omega$

$L_{\ell s} = 1.33$ mH $\quad J = 0.026$ Kg-m^2 $\quad L'_{\ell r} = 0.554$ mH

$D = 0 \qquad T_L = 0$

In all the simulations the d-q frame was fixed to the rotor ($\omega = \omega_r$), although in principle the speed of this arbitrary frame can be anything the user wants it to be.

```
12 $  INITIAL CONDITIONS
14 WE=377; DT=0.001; K=200; ALPHA=0; DWR(1)=0; WR(1)=0; WWR(1)=0;
15 IQS(1)=0; IDS(1)=0; IQR(1)=0; IDR(1)=0
19 $  INITIALIZATION OF CONSTANT MATRICES
21 ID1=MATRIX(1,4:1,0,0,0);    ID2=MATRIX(1,4:0,1,0,0)
22 ID3=MATRIX(1,4:0,0,1,0);    ID4=MATRIX(1,4:0,0,0,1)
23 IA=MATRIX(4,4:)
24 DM=MATRIX(4,4:DT,0,0,0,0,DT,0,0,0,0,DT,0,0,0,0,DT)
25 DPLUS=MATRIX(4,4:1+ALPHA,0,0,0,0,1+ALPHA,0,0,0,0,1+ALPHA,
26 &0,0,0,0,1+ALPHA)
27 DMINUS=MATRIX(4,4:1-ALPHA,0,0,0,0,1-ALPHA,0,0,0,0,1-ALPHA,
28 &0,0,0,0,1-ALPHA)
29 S=DM/DT
30 L=MATRIX(4,4:LSS,0,IMS,0,0,LSS,0,IMS,IMS,0,IRR,0,0,IMS,0,IRR)
31 LINV=1/L
32 B=LINV*S
```

(a) Initial conditions.

```
37 FOR  N=1,K
38 $  LINES 39-47 PREDICT NEXT STEP VALUE OF CURRENTS AND ROTOR SPEED
39 M=N
40 GOTO AU
41 FIRST:
43 UA=U
44 DIA=A*IA+B*UA
45 IB=IA+DT*DIA
46 WR(M+1)=WR(M)+DT*DWR(M);       WWR(M+1)=WWR(M)+DT*WR(M)
48 $  LINES 50-64 CORRECT THE PREDICTED VALUE OF CURRENTS AND ROTOR
49 $  SPEED AND CALCULATE THE TORQUE
50 M=N+1
51 GOTO AU
52 SECOND:
54 UB=U;       DIB=A*IB+B*UB
56 IB=IA+(DT/2)*(DPLUS*DIB+DMINUS*DIA)
57 IQS(M)=ID1*IB;       IDS(M)=ID2*IB
59 IQR(M)=ID3*IB;       IDR(M)=ID4*IB
61 DWR(M)=(P/(2*J))*((P*IMS/2)*(IQS(M)*IDR(M)-IDS(M)*IQR(M))-TL)
62 WR(M)=WR(M-1)+(DT/2)*(DWR(M)+DWR(M-1))
63 WWR(M)=WWR(M-1)+(DT/2)*(WR(M)+WR(M-1))
64 TE(M)=(2*J/P)*DWR(M)+(2/P)*D*WR(M)+TL
65 IA=IB
66 N
67 ENDLOOP N
68 GOTO FINISH
```

(b) Integration loop.

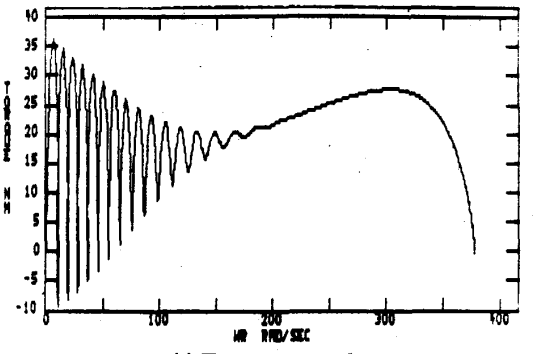
```
74 AU:
75 W=WR(M)
76 WIMS=(W-WR(M))*IMS
77 WIRR=(W-WR(M))*IRR
78 R=MATRIX(4,4:-RS,-W*LSS,0,-W*IMS,W*LSS,-RS,W*LS,0,0,-WLMS,-RQR,
79 &-WIRR,WLMS,0,WIRR,-RDR)
80 A=LINV*R
81 VQS=1.41421*(VAS*COS(WE*M*DT)*COS(WWR(M))
82 &+VBS*SIN(WE*M*DT)*SIN(WWR(M)))
83 VDS=1.41421*(VAS*COS(WE*M*DT)*SIN(WWR(M))
84 &-VBS*SIN(WE*M*DT)*COS(WWR(M)))
85 U=MATRIX(4,1:VQS(M),VDS(M),0,0)
86 IF(M.EQ.N)  GOTO FIRST
87 GOTO SECOND
```

(c) Matrix update subroutine.

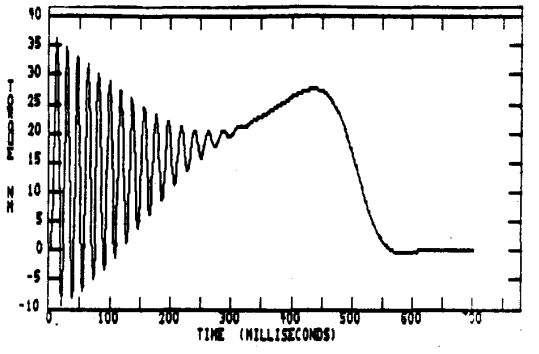
Figure 1: Speakeasy input files.

The first case is a cold start-up with balanced phase voltages ($VRMS_{as} = VRMS_{bs} = 110$ V). Results are depicted in figure 2. The torque and speed characteristics are what can be expected from a balanced start-up of an induction motor, with relatively large oscillations of the torque at the beginning, and a peak torque before the machine reaches steady state.The second case is a start-up of the induction motor but with unbalanced voltages ($VRMS_{as} = 100$ V, $VRMS_{bs} = 88$ V). Figure 3 depicts the results for this simulation. Notice the sustained oscillations of the torque around the steady state condition.

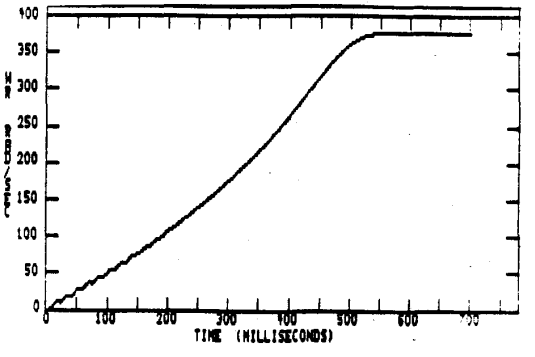
For the last simulation the resistors in each axis of the rotor are given different values ($r'_{r_q} = 0.144\ \Omega$ $r'_{r_d} = 0.288\ \Omega$). The results in figure 4 show a large initial torque, and uneven oscillations that last until the machine reaches steady state.



(a) Torque vs speed.

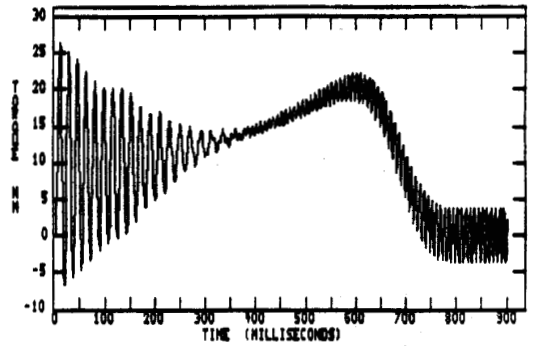


(b) Torque characteristic.

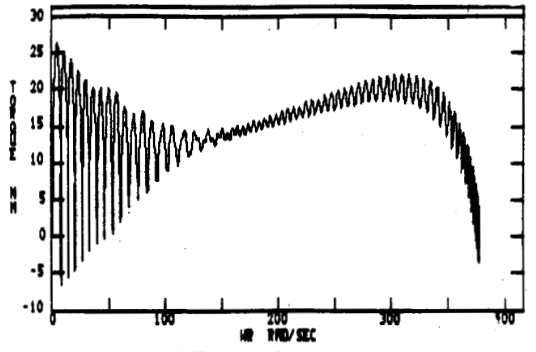


(c) Speed characteristic.

Figure 2: Balanced case. Speakeasy results.

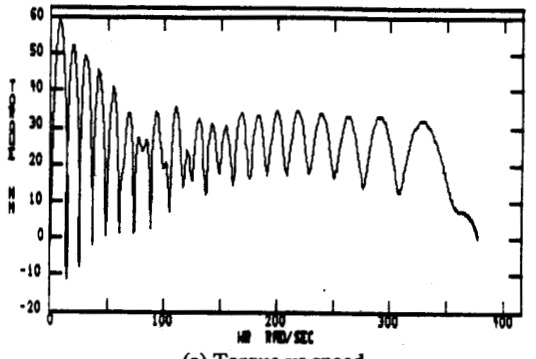


(a) Torque vs speed.

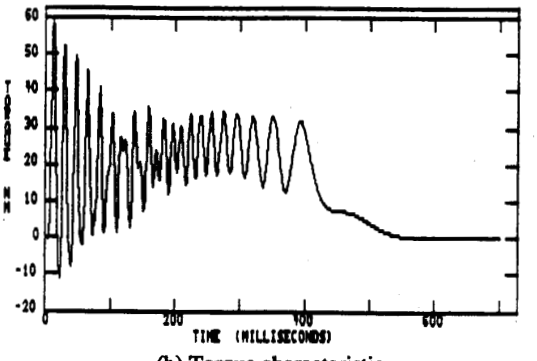


(b) Torque characteristic.

Figure 3: Unbalanced stator voltages. Speakeasy results.

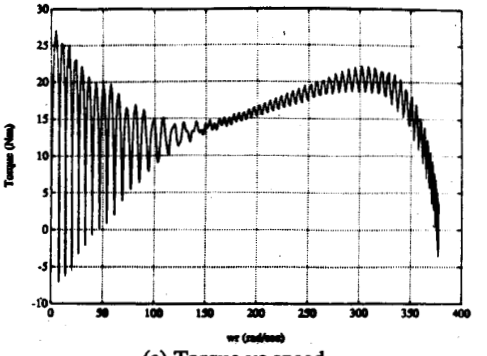


(a) Torque vs speed.

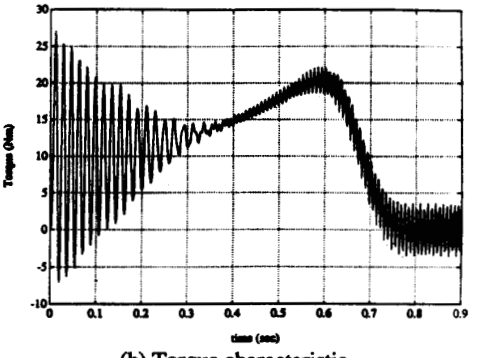


(b) Torque characteristic.

Figure 4: Unbalanced rotor resistances. Speakeasy results.

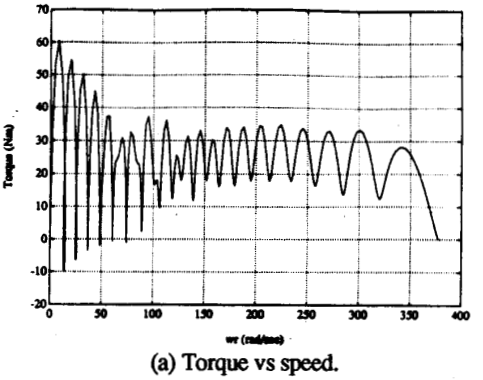


(a) Torque vs speed.

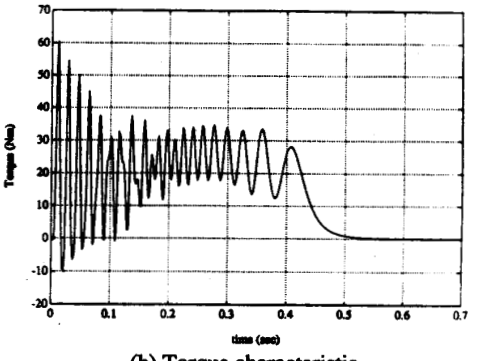


(b) Torque characteristic.

Figure 5: Unbalanced stator voltages. Matlab results.



(a) Torque vs speed.



(b) Torque characteristic.

Figure 6: Unbalanced rotor resistances. Matlab results.

MATLAB

This environment attains a higher level of abstraction than Speakeasy by using the concept of a matrix as a fundamental programming block, and hiding from the user the details of the underlying numerical method. This allows a more clear problem specification. Another advantage of Matlab is that the user can add commands to it via so-called M-files to increase its functionality.

The same three cases were simulated using PC-Matlab [13]. The results obtained are very close to those produced by Speakeasy, although the method of integration is completely different. Matlab uses a fourth-fifth order Runge-Kutta-Fehlberg method with variable time step. There is a small difference in the last two cases for the machine characteristics that can probably be attributed to the difference in the time steps used by each integration method. The results obtained with the EMTP agree with those of PC-Matlab. Figures 5 and 6 depict the Matlab results for the unbalanced machine.

In our simulation, the explicit method of integration used by PC-Matlab does not result in numerical instabilities. However, if control devices were added to the machine this method could have problems due to the possible stiffness of the equations.

The machine equations for this program are handled in matrix form, as shown in figure 8. The equations in this figure resemble those previously defined for the induction motor. The linkage fluxes have been scaled by the electric speed ω_e to avoid numerical problems. Figure 7 presents the additional variables and commands needed to define the initial conditions, method of integration, and plot variables of interest. Although the matrices are sparse, zero values must be represented explicitly in PC-Matlab.

```
%  Simulation of a 2-phase Induction Motor  (Cold Start-Up)
  t0 = 0;      tf = 0.7;       % Initial and final time
  tol = 0.0001;        % Tolerance for Integration method
  x0 = [0 0 0 0 0 0]';            % Initial conditions
  file = 'imurflux';              % Machine equations

%  4th-5th order Runge-Kutta-Fehlberg Integration.  Wait...
  [t,x] = ode45(file,t0,tf,x0,tol,0);

%  Calculate currents from fluxes
   l = [ x(:,1) x(:,2) x(:,3) x(:,4)]';
  Lls = 0.00133;  Llrp = 0.000554;
  Lms = 0.03514;
  L = [ Lls+Lms      0         Lms         0     ;
           0      Lls+Lms       0         Lms    ;
          Lms        0       Llrp+Lms      0     ;
           0         Lms         0      Llrp+Lms ];
  x = [(inv(L)*(l./377))' x(:,5) x(:,6)];

% Calculate Electromagnetic Torque (Te)
  P = 2;
 Te = (P/2)*Lms*(x(:,1)*x(:,4) - x(:,2)*x(:,3));

% Plot results
  plot(x(:,6),Te);
  title('Torque vs rotor speed');
  xlabel('wr (rad/sec)');
  ylabel('Torque (Nm)');
  grid
```

Figure 7: PC-Matlab definition of initial conditions, method of integration, current and torque calculations, and plotting commands.

SOLVER-Q

This environment was developed by one of the authors (Alvarado) to provide a completely general environment for the manipulation of mathematical models of all kinds, including not only numeric but also symbolic computational capabilities. This environment represents an attempt at ultimate declarative clarity. Only declarative constructs are permitted.

Although explicit matrix notation is not permitted in the current version of SOLVER-Q [14], all kinds of matrix operations can in fact be performed with ease. By using explicit algebraic notation, however, this environment stays closer to the computational realities of simulation, such as the need for sparsity preservation.

The same three cases of cold start-up of the two phase induction motor were simulated in a PC using this program. The results, as expected, were in close agreement with those obtained with the two previous environments, particularly with PC-Matlab.

```
% Equations of a 2-phase Induction Motor (fluxes)
function xprime = imurflux(t,x)

% Applied Voltages:
    f = 60.;          we = 2.*pi*f;
   vas = sqrt(2)*110*cos(we*t);
   vbs = sqrt(2)*110*sin(we*t);

% Park's Transformation
wr = x(6);  thetar = x(5);    % Mechanical variables
w = wr;   theta = thetar;     % Rotor reference frame
  vqs = cos(theta)*vas + sin(theta)*vbs;
  vds = sin(theta)*vas - cos(theta)*vbs;

% Machine data:
  Rs = 0.295;    Rqrp = 0.144;    Rdrp = 0.288;
  Lls = 0.00133;  Llrp = 0.000554;
  Lms = 0.03514;
    P = 2;            J = 0.026;

% Machine equations in matrix form:
  lqs = x(1);  lds = x(2);
  lqrp = x(3);  ldrp = x(4);
  l = [ lqs;  lds;  lqrp;  ldrp];
  v = [ vqs; vds; 0; 0];
  R = [ Rs  0    0     0;
        0  Rs    0     0;
        0   0  Rqrp    0;
        0   0    0   Rdrp];
  L = [ Lls+Lms     0         Lms        0     ;
           0      Lls+Lms      0         Lms    ;
          Lms        0      Llrp+Lms     0     ;
           0         Lms         0     Llrp+Lms ];
  W = [ 0   w   0    0   ;
       -w   0   0    0   ;
        0   0   0   w-wr ;
        0   0 -w+wr  0   ];
  i = inv(L)*(l./we);
  iqs = i(1);   ids = i(2);
  iqrp = i(3);   idrp = i(4);
  lp = -we.*R*i - W*l + we.*v;

% Mechanical equations:
   Te = (P/2)*Lms*(iqs*idrp - ids*iqrp);
   Tl = 0;
   thetarp = wr;
   wrp = (P/2)*(1/J)*(Te - Tl);

% Solution:
  xprime = [ lp; thetarp; wrp];
```

Figure 8: PC-Matlab machine equations. Linkage fluxes λ have been scaled by ω_e ($l = \lambda/\omega_e$), to avoid numerical problem.

The integration method used in this case was implicit trapezoidal integration. The simulation of the machine with unbalanced rotor resistances was run with a smaller time step ($\Delta t = 0.0005$) than the first two cases ($\Delta t = 0.002$). This reduction in time step increases the CPU time needed for the simulation. The SOLVER-Q environment also allows the use of variable time steps if desired.

Figure 9 shows the set of equations given to SOLVER-Q. These equations are in declarative style. They are identical to equations (7-12), with two exceptions. First, the equations have been set up to compute initial values for all the variables automatically by initially setting time to a negative value. Second, the fluxes have been scaled by $2\pi f$ to avoid numerical problems during the simulation. SOLVER-Q transforms these equations to symbolic algebraic form by using the trapezoidal rule of integration. The resulting algebraic equations are then solved numerically.

```
{ Applied voltage transformation: }
  w0 = wr;      dw = dwr;   {Reference frame fixed
  α = αr;                        to the Rotor}
              { Park's equations }
    vqsp = vas * COS(α) + vbs * SIN(α);
    vdsp = vas * SIN(α) - vbs * COS(α);
    { Phase stator voltage }
      vas = SQRT(2)*110*COS(we*t);
      vbs = SQRT(2)*110*SIN(we*t) ;
       we = 2*π*f;            f = 60;
  vqr' = 0;     vdr' = 0;       {Short-circuited rotor}

{ Connect the machine to the power supply at t = 0 }
      vqs = IF t>0 THEN vqsp ELSE 0;
      vds = IF t>0 THEN vdsp ELSE 0;

{ d-q Machine equations:  (link. fluxes l have been scaled) }
  vqs = Rs*iqs + D(γqs,t)/we + (w0 + dw)*γds/we;
  vds = Rs*ids + D(γds,t)/we - (w0 + dw)*γqs/we;
  vqr' = Rr'*iqr' + D(γqr',t)/we
         + (w0 - wr0 + dw - dwr)*γdr'/we;
  vdr' = Rr'*idr' + D(γdr',t)/we
         - (w0 - wr0 + dw - dwr)*γqr'/we;

     γqs = Xls*iqs + Xms*(iqs + iqr');
     γds = Xls*ids + Xms*(ids + idr');
    γqr' = Xlr'*iqr' + Xms*(iqs + iqr');
    γdr' = Xlr'*idr' + Xms*(ids + idr');

{ Electromagnetic torque: }
      Te = P/2*Lms*(iqs*idr' - ids*iqr');

{ Mechanical equations: }
   D(αr,t) = wr0 + dwr;
        Te = (2/P)*J*D(dwr,t) + Tl;
        Tl = 0;

{ Machine Data: 2-pole, 2-phase, 5 HP, 110 V, 60 Hz }
  Xls = we*Lls;   Xlr' = we*Llr';   Xms = we*Lms;
                        P = 2;
          { R's in Ohms, L's in H, J in Kg-m2 }
   Rs = 0.295;      Rr' = 0.144;    Lms = 0.03514;
  Lls = 0.00133;    Llr' = 0.000554;    J = 0.026;

{ Initial conditions for cold start-up: }
   wr0 = 0;
   t = -0.004;       {Connection at t=0}
   dwr = 0;       αr = 0;
   γds = 0;  γqs = 0;  γdr' = 0;  γqr' = 0;
```

Figure 9: SOLVER-Q format for the machine equations. The linkage fluxes λ have been scaled by ω_e ($\gamma = \lambda/\omega_e$), to avoid numerical problems.

COMPARISONS AND CONCLUSIONS

All the results shown above were verified against simulations done with the EMTP [1]. The results obtained with the EMTP agree with those obtained with PC-Matlab and SOLVER-Q for all cases. With Speakeasy there is a small difference for the unbalanced cases, which we attribute to the time step and numeric errors embedded in the different integration methods.

Explicit methods of integration, such as the one used by PC-Matlab, can in theory produce numerical instabilities in stiff systems. That could be the case, for example, if control circuits were added to the machine. The automatic step control in PC-Matlab reduces these problems at the expense of time step size.

In terms of clarity, Speakeasy is closely related to conventional programming languages, and therefore perhaps the one that would afford the greatest flexibility if non-declarative constructs were essential. Disadvantages of Speakeasy include the need for explicit equation forms, the need to program the numerical method by the user, the lack of matrix construct support, and the lack of symbolic constructs. Matlab permits a more clear specification by accepting matrix notation, user definable commands, and the use of built-in explicit numerical integration methods. However, it does not support symbolic constructs or sparsity. SOLVER-Q provided both symbolic constructs and sparse matrix methodology, implicit problem specification and solution, built-in implicit methods of integration, a fully declarative environment and very high level interactive commands separate from the problem specification. However, its heavy use of symbolic computation makes it the slowest. SOLVER-Q also lacks matrix notation.

Speakeasy was used in a time sharing environment, while PC-Matlab and SOLVER-Q were run in an IBM-PS/2 model 50. Because of the different computer environments used, it is not possible to compare CPU times of Speakeasy and the other two environments. For our example 1, PC-Matlab took 7 min 10 sec, while SOLVER-Q took 9 min 21 sec. That is, PC-Matlab was about 20% faster. For example 2, PC-Matlab was also about 20% faster than SOLVER-Q, while for example 3 PC-Matlab was about five times faster than SOLVER-Q because of its need for a very small constant step size. Nevertheless, because of its emphasis on sparsity preservation, we expect a much better relative performance of SOLVER-Q for models requiring larger sets of equations.

In closing, all the environments investigated provide excellent means for illustrating the behavior of machines directly from the mathematical models of the machines without the need for programming. This is of great value in the teaching of machine models. These environments, although computationally slow, also are useful for prototyping and testing the behavior of any new machine models a user may wish to experiment with. At the expense of some CPU performance, the user gains significant clarity, modifiability, and a close association between equations and the simulation environment. New equation models can be added and tested trivially. Furthermore, as the implementation of these environments continues to improve, we expect that they will ultimately become competitive for production applications and replace traditional programming and simulation environments as well.

REFERENCES

[1] "The Electromagnetic Transients Program," EPRI Report EL-4541-CCMP, Volume 1, April 1986.

[2] Mitchell and Gauthier Associates, *Advanced Continuous Simulation Language (ACSL) - User Guide/Reference Manual*, Third Edition, MGA Associates, Concord, Mass. 1981.

[3] T. A. Lipo and D. W. Novotny, "Dynamics and Control of AC drives," ECE 711, Chapter 8, pp. 8.2-23 to 8.2-61, Course Notes, Department of Electrical and Computer Engineering, The University of Wisconsin, Madison, 1986.

[4] M. Konopasek and S. Jayaraman, "Constraint and Declarative Languages for Engineering Applications: The TK!Solver Contribution," *Proceedings of the IEEE*, Vol. 73, No. 12, December 1985.

[5] F. L. Alvarado, "New Ideas in System Analysis," *Proceedings of the 1988 Grainger Lecture Series*, University of Illinois, Jumer's Lodge, Urbana, Illinois, May 24-25, 1988.

[6] F. L. Alvarado and D. J. Ray, "Symbolically Assisted Numeric Computation in Education," *The International Journal of Applied Engineering Education*, Vol 4, No. 6, November, 1988.

[7] F.L. Alvarado and Y. Liu, "General Purpose Symbolic Simulation Tools for Electric Networks," *IEEE Transactions on Power Systems*, Vol. 3, No. 2, pp. 689-697, May, 1988.

[8] C.A. Cañizares and F.L. Alvarado, "Graphic and Symbolic Analysis of Power Systems," *Proceedings of NAPS*, Purdue University, pp. 1-10, September 26-27, 1988.

[9] T. Sustersic and A. Keyhani, "A New Approach to the Digital Simulation of 2-Phase Induction Motors," *Proceedings of NAPS*, Purdue University, pp. 277-284, September 26-27, 1988.

[10] A. Keyhani, "Development of an Interactive Power System Research Simulator," *IEEE Transactions on Power Apparatus and Systems*, PAS Vol. 103, No.3, pp. 515-521, March, 1984.

[11] Cohen, S.J. Fink, F.J.D. Serduke and J.Z. Kriloff, *A Guide to TSO Speakeasy*, Argonne National Lab., ANL 75-44, 1974.

[12] H.M. Saadat, "Time Domain Simulation of Synchronous Machine Imbalanced Faults Using PC-Matlab," *Proceeding of NAPS*, Purdue University, pp. 285-291, September 26-27, 1988.

[13] The MathWorks Inc., *PC-Matlab Reference Manual*, Version 3.13, September, 1987.

[14] F. L. Alvarado, *SOLVER-Q Instruction Manual*, Software Development and Distribution Center, The University of Wisconsin, Madison, WI, 1987.

Fernando L. Alvarado (M'69 - SM'78) was born in Lima, Peru in 1945. He received the BEE and PE degrees from the National University of Engineering in Lima, Peru, the MS degree from Clarkson College (now Clarkson University) in Potsdam, New York, and the Ph.D. degree from the University of Michigan in 1972. Dr. Alvarado joined the University of Toledo-Ohio, as an assistant professor in 1972. Since 1975 he has been with the University of Wisconsin in Madison, where he is currently a Professor in the Department of Electrical and Computer Engineering. Dr. Alvarado has authored numerous papers in the area of computer applications to engineering in general and power systems in particular. He is the author of the SOLVER-Q computer program and manual. His main interests are in the area of large scale nonlinear networks, power system economics and control, symbolic computation and, more recently, computing paradigms and robotic control.

Claudio A. Cañizares (S'88) was born in Mexico D.F. in 1960. He received the BEE from the National Polytechnic School (EPN) of Quito-Ecuador in 1984, where he has been working since 1983, and the MS in Electrical Engineering from the University of Wisconsin-Madison in May of 1988. Currently, Mr. Cañizares holds the position of Assistant Professor at the EPN. Since August 1986 he has been on leave from the EPN and working toward his Ph.D. degree at the University of Wisconsin-Madison, under EPN and Fulbright Scholarships. He has been collaborating with Dr. Fernando L. Alvarado as a research assistant, and works as a teaching assistant for the University of Wisconsin at the Electrical and Computer Engineering Department.

Ali Keyhani (M'67) received the Ph.D. degree from Purdue University, West Lafayette, Indiana in 1975. From 1967 to 1973, he worked for Hewlett-Packard Co. on the computer-aided design of electronic transformers. From 1970 to 1973, he worked for Columbus and Southern Ohio Electric Co. on computer applications for power system engineering problems. In 1975, he joined TRW Controls and worked on the development of computer programs for energy control centers. In 1976 he joined the Tehran Polytechnic in Tehran, Iran. Since 1980 he has been an Associate Professor of Electrical Engineering at The Ohio State University. Dr. Keyhani's research interest is in control of power systems, environmental systems, motion control and modeling, parameter estimation, failure detection of electrical machines, transformers and drive systems.

Bryan Coates (S'89) was born in Oberlin, Ohio on June 5, 1965. He received the BS degree from Ohio State University, Columbus, Ohio, in 1988. Currently he is working towards his MS degree in Electrical Engineering at the Ohio State University.