Solving Power Flow Problems with a Matlab Implementation of the Power System Applications Data Dictionary

Fernando L. Alvarado
alvarado@engr.wisc.edu
ECE Department, The University of Wisconsin, Madison, Wisconsin 53705

Abstract

This paper implements a power flow application and variations using the IEEE Power System Application Data Dictionary within a Matlab environment. It describes a number of useful data and implementation techniques for a variety of applications. The techniques include the use of very compact and efficient code for the computation of Power Transfer Distribution Factors. Power Transfer Distribution Factors are an important element of present and proposed congestion management strategies for power systems, particularly when these systems must operate in a deregulated environment. Keywords: databases, IEEE Common Format, data exchange.

1. Introduction

This paper provides a brief overview of the major elements of the IEEE Power System Application Data Dictionary (PSADD). It utilizes PSADD elements toward the creation of flexible general-purpose power flow environments that have great expandability and flexibility. The PSADD is intended primarily as a means for exchanging application data among power system analytical applications of various types. The paper then uses an environment closely based on PSADD concepts toward the creation of sample applications in Matlab. The two main applications considered are: a simple "vanilla" power flow and a rapid and efficient computation of Power Transfer Distribution Factors (PTDF). The paper is intended as a means of documenting efforts conducted by PSerc[1] toward the creation of a unified environment for power flow analysis, as a means for illustrating and popularizing the Data Dictionary itself, and as a means for illustrating Matlab vectorized computation techniques and methods. The paper contains much actual code because it is intended for use in its electronic form, where users are expected to be able to "cut and paste" examples to a Matlab environment directly from the text of the paper.

[1] The Power Systems Engineering Research Center, an NSF and industry sponsored university consortium.

2. The Data Dictionary

The PSADD is an attempt by IEEE to extend and renovate the old standby of the power industry, the IEEE Common Format [1], which served the industry well for many years. The Data Dictionary is meant primarily as a definition of terms used for analytical applications within power system models. It is organized around the notion of objects. Within this paper, we consider objects to be structures within the Matlab environment. Each object contains a number of attributes or fields. As such, each object constitutes a relational database. The following is a list of all data types defined in the dictionary. Boldface indicates data types that are used in the present paper. Some of the data types are defined but declared obsolescent

- **Analog Data**
- Area
- **Bus Data**
- Capacitor Data: obsolete, see shunt device data.
- Company Data
- Connectivity Node Data
- Data Exchange Data
- Equivalent AC Series Device Data
- Equivalent Injection/Shunt Device Data
- Generator Data: obsolete, see machine data.
- Interchange Control Data
- **Line Data**
- **Load Data**
- **Machine (Synchronous) Data**
- **Miscellaneous Application Data**
- Ownership Data
- Rating Data
- Reactor Data: obsolete, see shunt device data.
- Set Data
- **Shunt (Nonsynchronous) Device Data**
- Station Data
- Status Data
- Switching Device Data
- TieSwitch Data
- System
- Terminal
- Transaction Data
- Transformer Data

0-7695-0001-3/99 $10.00 (c) 1999 IEEE

Each field of each object type has a type: integer, text, real or date. We only consider text or real types. In some cases, two alternative types are permitted in the PSADD for certain fields. We do not allow this. Only a small subset of all available PSADD fields is used in this paper. Fields or data types mentioned but not used are indicated with an asterisk*. T denotes Text (string) data, R denotes real data, I denotes integer data. The Transformer and Load data types have subtypes not described here.

The PSADD integrates two viewpoints: a bus-oriented viewpoint common to analytical studies, and an equipment-oriented viewpoint common to measured values and system operation. The Miscellaneous Application Data describes this information. Here is a *subset* of the Miscellaneous Application Data table:

Name	Description	
Misc.Title	User defined case title	T
Misc.BaseMVA	MVA base	R
Misc.ModelConnectivityType	BM or Bus.Model BS or BusSection* BO or Both*	T
Misc.ContingencyGenMWResponse	RF = Use response factors PC = Proportional to capacity PO = Proportional to output	T
Misc.DeviceRefMethod	Method (i.e., Number or Name) used as for referencing a device.	T

Four other tables of generic interest are those for Analog Data, Status Data, Ratings Data and Set Data. The Set Data is useful for grouping information into Zones, Areas, Companies, Locations and almost any other user-desired grouping. It is not used here. The Ratings Data specifies rating types. This paper assumes all flow ratings are in MVA, thus no such data is required. Status Data is used to denote the status of a device or component. We use 1 to denote in service, active or closed, 0 otherwise. Thus, we do not need Status Data as a separate data type. Analog Data is used to specify "metered" information (as opposed to "parameters" of components). Rather than entering this information as part of each device type, it is entered as "analog data" with a pointer to the device type where it is used. Here is a subset of the definition of Analog Data:

Name	Description	
Analog.Number	Unique number	I
Analog.Type	VOLT=voltage magnitude FLOW=Power or current MW = Active Power MVAR = Reactive Power TAP = Transformer Tap ANGL = Voltage Angle	T
Analog.AssignDevice	Device type associated with value (complete spelling or abbreviation of one of the root entities). Some possibilities are: BS=Bus LN=Line TR=Xformer SW=Switch MA=Machine LD=Load SH=Shunt	T
Analog.DeviceRef	Device assignment (Bus.Number, Line.Number) of value on device for device type.	I
Analog.Value	Numeric value	R

The next important category is the Bus type:

Name	Definition	
Bus.Number	Bus number	I
Bus.Name	Bus name	T
Bus.BasekV	Base voltage RMS	R
Bus.Status		I
Bus.Vmag	Magnitude of bus voltage. Solved value.	R
Bus.Vangle	Phase angle of bus voltage. Solved value	R
Bus.MWMismatch	Net Mw not assigned to generator, load or shunt	R
Bus.MvarMismatch	Net Mvar not assigned to generator, load or shunt	R
Bus.TypeCode	*L or Load (no control) *CB or ControlToBand *CT or ControlToTarget *S = Swing (or reference) *I = Isolated	T
Bus.VoltTarget	Voltage magnitude setpoint	R
Bus.LowVoltLim	Voltage control band lower limit	R
Bus.UpVoltLim	Voltage control band upper limit	R
Bus.ControlPriorityCode	T= transformer S= Synchronous machine N =nonsynchronous shunt	T

The nodal injections types used in this paper include Machine, Load and Shunt Element, defined next:

Name	Description	
Machine.Number	Unique Number	I
Machine.BusRef	Bus.Number to which machine is connected	I
Machine.Id	Identifier	T
Machine.Status		I
Machine.MW	MW output, solved value	R
Machine.Mvar	MVAR output, solved value	R
Machine.ControlMode	PQ = hold P and Q constant PV = adjust Q to hold V NL = adjust Q, no limits SW = case swing Machine	I
Machine.ControlledBusRef	Bus (Bus.Number) whose voltage is controlled	I
Machine.RatingId	Identification of ratings	T
Machine.RatingValue	A vector of possible generator maximum MW values	R
Machine.RatingMinimum(+)	A vector of possible generator minimum MW values	R
Machine.MinQOutput	Lower limit for reactive	R

	power	
Machine.MaxQOutput	Upper limit for reactive power	R
Machine.MinOperatingVolt	Machine minimum operating voltage	R
Machine.MaxOperatingVolt	Machine maximum operating voltage	R
Machine.PostContRespFact	Percent of Machine MW response to system MW imbalance. Values are normalized for each island	R

(+) Not in the current version of the Data Dictionary.

Name	Description	
Load.Number	Unique Number	I
Load.BusRef	Bus.Number of bus to which load is connected	I
Load.Id	Identifier	T
Load.Status		I
Load.ModelType*	CONST = constant MW and MVAR VFUNC = voltage depend INDMOT=induction motor	T
Load.ModelMinVoltage*	Minimum model voltage for this models	R
Load.ModelMaxVoltage*	Maximum voltage for this load model	R
Load.MinMW*	Minimum real power	R
Load.MaxMW*	Maximum real power	R
Load.ReferenceMW*	Reference real power at one per unit (for scaling)	R
Load.MinMvar*	Minimum reactive power consumed	R
Load.MaxMvar*	Maximum reactive power	R
Load.ReferenceMvar*	Reference reactive power at one per unit (for scaling)	R
Load.Mvar	Magnitude of Load Mvar. Solved value	R
Load.Mw	Magnitude of Load MW. Solved value	R

Name	Description	
Shunt.Number	Unique Number	I
Shunt.BusRef	Bus.Number to which shunt is connected	I
Shunt.Id	Identifier	I
Shunt.Status		I
Shunt.BaseMVA*	MVA of this device	R
Shunt.BasekV*	Base voltage of this device	R
Shunt.Type	Identifier of this device (Capacitor, Reactor)	T
Shunt.ControlMode	NONE = fixed shunt value VOLT = hold voltage between limits MVAR=hold MVAR flow*	I
Shunt.ControlBusRef	Bus.Number at which voltage is to be controlled	I
Shunt.MaxRegVoltage	Value at which shunt will change out to lower voltage	R
Shunt.MinRegVoltage	Value at which shunt will change out to raise voltage	R
Shunt.BlockSusceptValueNo	Number of discrete values available. Zero for continuous adjustment	I
Shunt.BlockSusceptance	Susceptance of each block A vector.	R
Shunt.MW	Magnitude of Shunt output MW. A solved value	R
Shunt.Mvar	Magnitude of Shunt output Mvar. A solved value	R

Two series PSADD elements are implemented here:

Name	Definitions	
Line.Number	Unique number	I
Line.FromBusRef	Bus.Number of "from" bus	I
Line.ToBusRef	Bus.Number of "to" bus	I
Line.Circuit	Alphanumeric circuit ID	T
Line.Status		I
Line.SectionR	Series resistance	R
Line.SectionX	Series reactance	R
Line.ShuntConductance	Total uniformly shunt conductance of this section	R
Line.ShuntSusceptance	Total shunt susceptance of this section	R
Line.FromMW	Line MW flow at From end. A solved value	R
Line.FromMvar	Line Mvar flow at From end. A solved value.	R
Line.ToMW	Line MW flow at To end. A solved value	R
Line.ToMvar	Line Mvar flow at To end. A solved value	R
Line.TypeId	Type of equipment type (LINE, CAPACITOR*)	T
Line.Length*	Length	R
Line.LossAssignBusRef	Bus.Number to which losses should be assigned	I
Line.FromLumpedCond	Conductance of lumped shunt element at "from" bus	R
Line.FromLumpedSuscept	Susceptance of lumped shunt element at "from" bus	R
Line.ToLumpedCond	Conductance of lumped shunt element at "to" bus	R
Line.ToLumpedSucept	Susceptance of lumped shunt element at "to" bus	R
Line.RateId	ID of line rating value. A vector of text strings	T
Line.RateValue	Rating value (a vector)	R

Name	Definition	
Xformer.Number	Unique Number	I
Xformer.PrimaryBusRef	Bus.Number to which winding awinding A is connected	I
Xformer.SecondaryBusRef	Bus.Number to which winding B is connected	I
Xformer.Circuit	Identifier	T
Xformer.Type	FIXED Fix ratio and angle	T

	VOLT Fix angle var ratio MVAR Fix angle var ratio MW – Fix ratio var angle	
Xformer.TapRatio	Tap ratio value	R
Xformer.TapAngle	Tap angle value	R
Xformer.TapRatioMin	Minimum Tap ratio value of the transformer	R
Xformer.TapRatioMax	Maximum Tap ratio value of the transformer	R
Xformer.TapRatioStep	Tap ratio step size value of the transformer	R
Xformer.TapAngleMin	Minimum Tap angle value of the transformer	R
Xformer.TapAngleMax	Maximum Tap angle value of the transformer	R
Xformer.TapAngleStep	Tap angle step size value of the transformer	R
Xformer.Status		I
Xformer.BaseMVA	*	R
Xformer.FromMw	Mw flow From bus. Solved value	R
Xformer.FromMvar	Mvar flow From bus. Solved value	R
Xformer.ToMw	Mw flow To bus. Solved value	R
Xformer.ToMvar	Mvar flow To bus. Solved value	R
Xformer.LossAssignBusRef	Bus.Number to which losses should be assigned	I
Xformer.Pri-SecR	s.c. resistance winding A to B	R
Xformer.Pri-SecX	s.c. reactance winding A to B	R
Xformer.Pri-SecZeroR*	s.c. zero sequence resistance winding A to B	R
Xformer.Pri-SecZeroX*	s.c. zero sequence reactance winding A to B	R

3. Rendition in Matlab

The implementation of the Data Dictionary in Matlab is done by means of sparse arrays, structures and cells of types real and character. For the sake of clarity, all concepts will be illustrated by means of an example. The example in question corresponds to the 6-bus system from Wood and Wollenberg. The Common Format File (CFF) describing this data is illustrated in the appendix. The CFF is not sufficient to describe all the necessary information. Thus, additional assumptions will be made as needed.

To implement the data dictionary, one needs to first establish the necessary structures for all the base data types and the corresponding fields (or attributes) of each data type. Some data types (such as **Misc**) contain only scalar or simple text attributes. The complete definition for the **Misc** data type subset is:

```
Misc.Title='This is a sample case';
Misc.BaseMVA=100; Misc.ConnectivityType='BM';
Misc.ContingencyGenMWResponse='PC';
Misc.DeviceRefMethod='Number';
```

Most other data types include vector or even sets of vectors as their data elements. The question arises immediately in Matlab whether it is better to represent these types as a single structure with vector structure entries, or to represent each data type entry as a structure and to represent the entire data type as a vector of structures. Either answer is correct and will work, but because of the vectorized nature of Matlab computations, our experiments indicate that it is quite efficient to represent data types as structures of vectors. Furthermore, for the sake all consistence almost all vectors will be column vectors. When it is necessary to represent sets (actually vectors) of variable-length vectors, this is done by using the notion of cells, which are vectors of arbitrary entry types. This is the convention adopted in the data dictionary.

Start with a definition of all buses along with, loads, machines and shunts connected to them.

```
% Initialization of Bus Data Type
Bus.Number=[];       Bus.Name='';
Bus.BasekV=[];       Bus.Status=[];
Bus.Vmag=[];         Bus.Vangle=[];
Bus.MwMismatch=[];   Bus.MvarMismatch=[];
Bus.TypeCode='';     Bus.VoltTarget=[];
Bus.LowVoltLim=[];   Bus.UpVoltLim=[];

% Initialization of Machine Data Type
Machine.Number=[]; Machine.BusRef=[];
Machine.Id='';     Machine.Status=[];
Machine.Mw=[];     Machine.Mvar=[];
Machine.ControlMode='';
Machine.ControlledBusRef=[];
Machine.MinQOutput=[]; Machine.MaxQOutput=[];
Machine.MinOperatingVolt=[];
Machine.MaxOperatingVolt=[];
Machine.PostContRespFact=[];

% Initialization of Load Data Type
Load.Number=[];   Load.Id='';
Load.BusRef=[];   Load.Status=[];
Load.Mvar=[];     Load.Mw=[];

% Initialization of Shunt Data Type
Shunt.Number=[]; Shunt.BusRef=[];
Shunt.Id='';     Shunt.Status=[];
Shunt.Type='';   Shunt.ControlMode=[];
Shunt.ControlBusRef='';
Shunt.MaxRegVoltage=[];
Shunt.MinRegVoltage=[];
Shunt.BlockSusceptValueNo=[];
Shunt.BlockSusceptance=[];
Shunt.Mw=[];     Shunt.Mvar=[];
```

Likewise, all structures for lines can be initialized:

```
Line.Number=[];     Line.FromBusRef=[];
Line.ToBusRef=[];   Line.Circuit='';
Line.Status=[];
Line.SectionR=[];   Line.SectionX=[];
Line.ShuntConductance=[];
Line.ShuntSusceptance=[];
Line.FromMw=[];     Line.FromMvar=[];
Line.ToMw=[];       Line.ToMvar=[];
Line.TypeId='';     Line.LossAssignBusRef=[];
```

```
Line.FromLumpedCond=[];
Line.FromLumpedSuscept=[];
Line.ToLumpedCond=[];Line.ToLumpedSuscept=[];
Line.RateId='';     Line.RateValue=[];
```

Since no transformers are used in the example, details of transformers are not illustrated.

At this point, the table values can be loaded for the example at hand more or less directly from the Data Dictionary. Some assumptions are made:

- All machine minimum powers are 10% of the machine maximum powers.
- All machine maximum powers are 200 MW (or 2pu).
- We prefer to work in per-unit everywhere, even when the Data Dictionary calls for Mw or Mvar.
- Minimum voltages anywhere are specified at 0.85 pu, maximum voltages at 1.2 pu.
- All shunt susceptances are assumed to be adjustable in exactly two equal-sized steps. Since there are no susceptances in this example, a capacitive susceptance with a value of 0.02 pu is assumed at bus 5.
- Normally, data types would be constructed using a function that reads raw data from a database, from a common format data file, or some other available source. Here, data is constructed directly for the example at hand, without resorting to a reading and translation function.

```
% Bus data specification
n=6;   Bus.Number=(1:n)';
Bus.Name=strvcat('BUS1','BUS2',...
  'BUS3','BUS4','BUS5','BUS6');
Bus.BasekV=138*ones(n,1);
Bus.Status=ones(n,1);
Bus.Vmag=[1.05;1.05;1.07;1;1;1];
Bus.Vangle=zeros(n,1);
Bus.TypeCode=...
  strvcat('S','CT','CT','L','L','L');
Bus.VoltTarget=Bus.Vmag;
Bus.LowVoltLim=0.85*ones(n,1);
Bus.UpVoltLim=1.2*ones(n,1);

% Machine data specification
ng=3;   Machine.Number=(1:ng)';
Machine.Id=strvcat(Machine.Id,'1','1','1');
Machine.Status=ones(ng,1);
Machine.ControlMode=strvcat('SW','PV','PV');
Machine.BusRef=[1;2;3];
Machine.ControlledBusRef=[1;2;3];
Machine.RatingId=...
  strvcat('NORMAL','NORMAL','NORMAL');
Machine.RatingValue=[1.5;1.5;1.5];
Machine.RatingMinimum=[0.15;0.15;0.15];
Machine.MinQOutput=[-0.25;-0.25;-0.25];
Machine.MaxQOutput=[0.75;0.75;0.75];
Machine.MinOperatingVolt=0.85*ones(ng,1);
Machine.MaxOperatingVolt=1.2*ones(ng,1);
Machine.PostContRespFact=[1;1;1];

% Load data specification
nl=3;   Load.Number=(1:3)';
Load.BusRef=[4;5;6];
Load.Id=strvcat('1','1','1');
Load.Status=[1;1;1];

% Shunt data specification
nsh=1;   Shunt.Number=[1];   Shunt.BusRef=5;
Shunt.Id='1';   Shunt.Status=1;
Shunt.Type='Capacitor';
Shunt.ControlMode='NONE';
Shunt.ControlBusRef=5;
Shunt.MaxRegVoltage=1.25*ones(nsh,1);
Shunt.MinRegVoltage=0.85*ones(nsh,1);
Shunt.BlockSusceptValueNo={2}; % CELL ARRAY
Shunt.BlockSusceptValue={[0.1,0.1]}; % "

% Line data specification
nL=11;  Line.Number=(1:nL)';
Line.FromBusRef=[1;1;1;2;2;2;3;3;4;5];
Line.ToBusRef=[2;4;5;3;4;5;6;5;6;5;2];
Line.Circuit=strvcat('1','1','1','1',...
  '1','1','1','1','1','1','1');
Line.SectionR=[.1;.05;.008;.05;.05; ...
  .1;.07;.12;.02;.02;.1];
Line.SectionX=[.2;.2;.3;.25;.1;.3; ...
  .2;.26;.1;.4;.3];
Line.ShuntConductance=zeros(nL,1);
Line.ShuntSusceptance=[.02;.02;.03; ...
  .02;.01;.02;.025;.025;.01;.04;.03];
Line.TypeId=char(ones(nL,1)*double('Line'));
Line.LossAssignBusRef=Line.FromBusRef;
Line.FromLumpedCond=zeros(nL,1);
Line.FromLumpedSuscept=zeros(nL,1);
Line.ToLumpedCond=zeros(nL,1);
Line.ToLumpedSuscept=zeros(nL,1);
ratid=strvcat('NORMAL','EMERG');
ratval=[1.5;2];
for k=1:nL,
  Line.RateId{k,1}=ratid;
  Line.RateValue{k,1}=ratval;
end
```

The specification of metered parameters for all data types is done by means of the **Analog** data type. We consider first the specification of data values for all loads, followed by the specification of all machines, all shunts, all lines and all transformers (this example has no transformers). Ordinarily this would also be accomplished by an input routine that would translate appropriate database entries, common format file information and/or other such available data into the Data Dictionary structures. Here, the data is specified directly for the example of interest, starting with an initialization of all Analog data types:

```
%Initialization of all Analog data
Analog.Number=[];      Analog.Type='';
Analog.AssignDevice='';
Analog.DeviceRef=[];   Analog.Value=[];

% Specification of all Load operating Analog
data
for k=1:nl,
 na=length(Analog.Number);
 Analog.Number=[Analog.Number;na+1;na+2];
Analog.Type=strvcat(Analog.Type,'MW','MVAR');
 Analog.AssignDevice=...
 strvcat(Analog.AssignDevice,'Load','Load');
```

```
 Analog.DeviceRef=[Analog.DeviceRef;k;k];
end
Analog.Value=[Analog.Value;0.7;0.7;...
  0.7;0.7;0.7;0.7];

% Specification of all Machine operating
Analog data
for k=1:ng,
  na=length(Analog.Number);
  Analog.Number=[Analog.Number;na+1];
  Analog.Type=strvcat(Analog.Type,'MW');
  Analog.AssignDevice = ...
    strvcat(Analog.AssignDevice,'Machine');
end
Analog.DeviceRef=[Analog.DeviceRef;1;2;3];
Analog.Value=[Analog.Value;0;0.5;0.6];
```

4. A "Vanilla" Newton Power Flow

The above structures can be used directly in the implementation of a basic Newton power flow. Convenient computation in Matlab requires a few additional ideas:

- Most data types used for computation should be complex. Thus, a few data types are added to the definition of the Bus data type (and other data types as needed).
- An Internal data type is defined and used to contain all important intermediate matrix variables.
- An overall encapsulation of all data types into a single master data types is done to be able to refer to all the variables and data for a single study by means of a single variable. This is a structure of structures.

The first step is to convert pertinent Data Dictionary structures into Matlab structures suitable for computation. Also, it is extremely useful to associate any analog measurement data from the Analog data types and associate it with the pertinent data type in a manner suitable for computation. The following code performs these tasks and further encapsulates all information:

```
j=sqrt(-1);
kPL=intersect(strmatch('Load',Analog.AssignDe
vice),strmatch('MW',Analog.Type));
kQL=intersect(strmatch('Load',Analog.AssignDe
vice),strmatch('MVAR',Analog.Type));
kPM=intersect(strmatch('Machine',Analog.Assig
nDevice),strmatch('MW',Analog.Type));
Bus.SL=zeros(n,1);   Bus.PM=zeros(n,1);
iPL=Load.BusRef(Analog.DeviceRef(kPL));
iQL=Load.BusRef(Analog.DeviceRef(kQL));
iPM=Machine.BusRef(Analog.DeviceRef(kPM));
Bus.SL(iPL)=Bus.SL(iPL)+Analog.Value(kPL);
Bus.SL(iQL)=Bus.SL(iQL)+j*Analog.Value(kQL);
Bus.PM(iPM)=Bus.PM(iPM)+Analog.Value(kPM);
Line.Z=Line.SectionR+j*Line.SectionX;
Line.ShuntY2=(Line.ShuntConductance+j*Line.Sh
untSusceptance)/2;
Line.YI=Line.FromLumpedCond+j*Line.FromLumped
Suscept;
Line.YJ=Line.ToLumpedCond+j*Line.ToLumpedSusc
ept;
% Shunt devices at bus not considered here
PQlist=strmatch('L',Bus.TypeCode);
PVlist=strmatch('CT',Bus.TypeCode);
SlackList=strmatch('S',Bus.TypeCode);
GenList=union(SlackList,PVlist);
NonSlack=union(PQlist,PVlist);
```

Next we construct the Y-bus matrix:

```
n=length(Bus.Number);
Ybus=sparse([],[],[],n,n);
Ybus=Ybus+sparse(Line.FromBusRef,Line.FromBus
Ref,1./Line.Z+Line.ShuntY2,n,n);
Ybus=Ybus+sparse(Line.FromBusRef,Line.ToBusRe
f,-1./Line.Z,n,n);
Ybus=Ybus+sparse(Line.ToBusRef,Line.FromBusRe
f,-1./Line.Z,n,n);
Ybus=Ybus+sparse(Line.ToBusRef,Line.ToBusRef,
1./Line.Z+Line.ShuntY2,n,n);
Ybus=Ybus+sparse(Line.FromBusRef,Line.FromBus
Ref,Line.YI,n,n);
Ybus=Ybus+sparse(Line.ToBusRef,Line.ToBusRef,
Line.YJ,n,n);
```

We then illustrate the code to construct the complete power flow Jacobian for any system (Note: this highly vectorized version of the code is due to Christopher DeMarco of the University of Wisconsin):

```
function [dSdd,dSdv]=pflowjac(Yb,Vb)
Ib=Yb*Vb;
dSdd=j*diag(conj(Ib).*Vb)-
j*diag(Vb)*conj(Yb)*diag(conj(Vb));
dSdv=diag(conj(Ib).*(Vb./abs(Vb)))+diag(Vb)*c
onj(Yb)*diag(conj(Vb)./abs(Vb));
```

Finally, the complete master vanilla Newton solver is:

```
Vmag=Bus.Vmag;   Vang=Bus.Vangle*pi/180;
for iter=1:10,
  V=Vmag.*exp(j*Vang);
  [dSdd,dSdv]=pflowjac(Ybus,V);
  misvect=V.*conj(Ybus*V)+Bus.SL-Bus.PM;
  rmiss=[real(misvect(NonSlack)); ...
    imag(misvect(PQlist))];
  mismatch=max(abs(rmiss));
  if mismatch<0.0001, break; end
  rjac=[real(dSdd(NonSlack,NonSlack)) ...
    real(dSdv(NonSlack,PQlist)); ...
    imag(dSdd(PQlist,NonSlack)) ...
    imag(dSdv(PQlist,PQlist))];
  dx=rjac\rmiss;
  dxAng=dx(1:length(NonSlack));
  dxMag=dx(length(NonSlack)+1:length(dx));
  Vang(NonSlack)=Vang(NonSlack)-dxAng;
  Vmag(PQlist)=Vmag(PQlist)-dxMag;
end
Bus.Vmag=Vmag;   Bus.Vangle=Vang*180/pi;
```

To run the example in this paper, "cut and paste" every Matlab code segment to a Matlab environment. Store function code segments in m-files.

Extrapolation of these techniques to considerably more complex cases and problems is immediate. We illustrate only one such extension.

5. Power Transfer Distribution Factors

As an example of the usefulness of this approach to practical power system computations, we illustrate the code to compute Power Transfer Distribution Factors for an increase in active power demand at any location in the network, relative to the power being supplied at the slack generator. By having the PTDFs for any two (or more) locations in the grid, the PTDF for any bilateral (or multilateral) transaction not involving the slack generator can also be readily found, although this is not done here. For the example above, the PTDF code requires the determination of a Jacobian matrix that relates the flows at either end of a line to changes in voltage magnitudes and angles. The vectorized code to construct such matrix for the flow at the "from" end of every line in the system is given next. Here **Vb** is the vector of complex nodal voltages and **Vs** refers to the sending end voltages of the lines of interest. **Yf** is either **YflowI** or **YflowJ,** the from or to end "flow" admittance matrices.

```
function [YflowI,YflowJ]=bldyflow(Bus,Line)
n=length(Bus.Number);
nb=length(Line.FromBusRef); nbe=(1:nb)';
YflowI=sparse(nbe,Line.FromBusRef,1./Line.Z+i
*Line.ShuntY2+Line.YI,nb,n);
YflowI=YflowI+sparse(nbe,Line.ToBusRef,-
1./Line.Z,nb,n);
YflowJ=sparse(nbe,Line.FromBusRef,-
1./Line.Z,nb,n);
YflowJ=YflowJ+sparse(nbe,Line.ToBusRef,1./Lin
e.Z+i*Line.ShuntY2+Line.YJ,nb,n);
```

```
function [dFdd,dFdv]=flowjac(Yf,Line,Vb,Vs)
nb=length(Line.FromBusRef);  n=length(Vb);
If=Yflow*Vb;
dFdd=j*sparse((1:nb)',Line.FromBusRef,(conj(I
f).*Vs),nb,n)-
j*diag(Vs)*conj(Yf)*diag(conj(Vb));
dFdv=sparse((1:nb)',Line.FromBusRef,(conj(If)
.*(Vs./abs(Vs))),nb,n)+diag(Vs)*conj(Yf)*diag
(conj(Vb)./abs(Vb));
```

This code can be used to determine the sensitivity of flows on any line to any injections at a given operating point. Sample code to do this calculation for our example for an injection at bus 5 would be:

```
[YflowI,YflowJ]=bldyflow(Bus,Line);
[dFdd,dFdv]=flowjac(YflowI,Line,V,V(Line.From
BusRef));
[dSdd,dSdv]=pflowjac(Ybus,V);
rJ=[real(dSdd(NonSlack,NonSlack)),real(dSdv(N
onSlack,PQlist));imag(dSdd(PQlist,NonSlack)),
imag(dSdv(PQlist,PQlist))];
rJf=[real(dFdd(:,NonSlack)),real(dFdv(:,PQlis
t));imag(dFdd(:,NonSlack)),imag(dFdv(:,PQlist
))];
e=zeros(n,1);   e(5)=1;
er=[real(e(NonSlack));imag(e(PQlist))];
dFP5=rJf*(rJ\er)
```

At this point, the Matlab environment contains already a wealth of useful information for answering a large number of questions about the system.

6. Conclusions

Power System Application Data Dictionary data types have been used as the basis for a direct implementation of a simple power flow in Matlab. The implementation is remarkably efficient and easy to extend to any desired additional purpose: new methods can be tested with ease, new functions and objectives implemented, all while maintaining the ability to communicate between and among diverse applications in a standardized manner.

Acknowledgements

Professor Christopher DeMarco made available the key Jacobian routine to this author. Joann Staron has been for years instrumental in developing the PSADD. Thanks are also due to Scott Greene and Taiyou Yong.

References

[1] IEEE Committee Report, "Common format for exchange of solved load flow data," IEEE Transactions on Power Apparatus and Systems, Volume PAS-92, Number 6, pages 1916-1925, November/December, 1973.