Journal of ELECTRICAL ENGINEERING, VOL. 57, NO. 3, 2006, 161–166

A TEMPLATE SPARSE LIBRARY FOR POWER SYSTEMS

Amrane Tolba — Louiza Benfarhi — Mohamed Belkacemi *

This paper present a Template Sparse Library for Power Systems (TSLPS) developed in C++. Its aim is to provide a coherent platform for many power system applications such as Load Flow(LF), Short Circuit(SC), Optimal Power Flow(OPF) and Transient Stability (TS). The TSLPS is inspired by the recent template numerical library but considers only power system characteristics, *eg* for a class of large sparse symmetric matrices over real and complex numbers. Template facility of C++ is used to write a generic program on float, double and complex data types. The paper describes the generic code developed and its impact on the power system applications. LF and SC analysis have been successfully applied to the solution of several power systems and presented good computational performance.

K e y w o r d s: power systems, sparse matrix, template, LU decomposition, object oriented programming

1 INTRODUCTION

Modern power systems have grown both in size and complexity. The planning design and operation of electrical power systems require simulation analyses to evaluate the current and future system performance, reliability, safety and ability to grow with production or operating requirements. These have led to search of better and good integrated analysis tools and new equipment to respond to those requirements.

Power system analysis vary in complexity from one type of analysis to another. SC and LF analysis are the most frequently performed in the industry, as they answer fundamental questions relative to the system's state of operation. Software used presently in power system analysis is inherently complex because of large scale systems to be modelled, volume of data to be handled, and visualization requirements. Object Oriented Programming (OOP) has gained widespread acceptance to implement complex problems as its advantages of flexibility and ease of maintenance have been recognized [1, 2, 3]. In this area of research, several papers have evaluated the OOP for power system software analysis into:

Data Objects : Data objects model transmission lines, transformers, generators *etc.*

Computational Objects : Examples of computational objects are matrix, sparse matrix and graph objects. They are far more complex than the data objects. Their complexity can be classified into algorithmic complexity and overall abstraction complexity.

Application Objects : Application objects refer to power system analysis applications like LF and SC analysis. They use aggregation or composition of existing data objects and computational objects to create bigger objects like Network container.

Power system computing requires many abstractions, large sparse linear system solvers, and sparse matrix optimization. In recent years there has been a proliferation of libraries which provide abstractions. However, the code generated by such library sacrifices too much performance when applied to a special case of interest such as power systems. On the other side every application such as LF, SC, OPF, and TS has its own requirements on data structures. So the TSLPS developed and described here is based on template facilities and considers only power system characteristics (matrices).

The aim of the TSLPS is to provide a coherent platform for power system computing. The kernel of this platform is based on template facilities, and provides data structures and classes for the manipulation of basis objects, such as vectors and matrices needed for power system computing. We chose C++ as a programming language since it has full support for object oriented design, yet it does not enforce it. The flexibility of C++ allows a software designer to choose the appropriate tools for each particular software component [3].

In this paper we apply the fundamental generic programming approaches the domain of power system analysis. The resulting library TSLPS provides comprehensive functionality with a small number of fundamental algorithms, while at the same time achieving high performance. High performance generic algorithms are a new development made possible by the powerful template and object oriented features of the C++ language and compiler [3].

2 POWER SYSTEM MATRICES

Transmission and distribution networks have a very low degree of connectivity. As such, power system ma-

* Dept. of Electrical Engineering, Faculty of Engineering, Batna University Batna, Algeria, E-mail: l_benfarhi@yahoo.com

ISSN 1335-3632 © 2006 FEI STU

trices tend to be very large and very sparse like the admittance matrix Y_{bus} in SC analysis and the Jacobian matrix J in LF analysis. The Y_{bus} is complex symmetric and positive definite. Each diagonal element of Y_{bus} is nonzero, and each branch connected between busses occurs twice in Y_{bus}, so that a branch between busses j and k appears in y_{jk}, and y_{kj}. Additional parallel branches and shunt elements do not create elements additionally. The Jacobian matrix is real, symmetric (per block), and usually positive definite. It represents a linear approximation to the relationship between small changes in the network potentials and the network out-flows for each node. The pattern of entries in the Jacobian matrix is the same as the matrix representation of the graph of the network.

In most cases, the coding of large computer programs involves tradeoffs between execution time and memory requirements. Sparsity storage techniques, however, reduce both execution time and memory requirements [4]. The only drawback of sparsity programming is that it is more difficult to code than full matrix programming. An efficient implementation implies a numerically stable implementation with minimum computation complexity and a memory requirement proportional to the number of non-zeros in the original matrix.

3 TEMPLATES

Templates are type manipulations used to guide the compiler to produce optimized code. The original intent of templates was to support generic programming, which can be summarized as reuse though parameterization. Generic functions and objects have parameters which customize their behavior. These parameters must be known at compile time. This technique is particularly interesting for some linear algebra operations, where intensive but independent instructions are performed [5]. Templates allow programmers to develop classes and functions which are general purpose, yet retain the efficiency of statically configured code. Templates can be exploited to perform computations at compile time [5, 6]. The compile time programs can perform code generation by selectively inlining code as they are interpreted by the compiler.

In other words, a unique code which caters to many data types can be written by using template facility. For example, a class matrix may be required to support data types such as float, double and complex. This means that all methods like arithmetic operations defined for this class should work equally well on all these data types, without duplication of code. The algorithm written using template definition for class matrix is therefore written with data type. User program then specifies required data type. The use of templates adds flexibility at the cost of the compile time. The run time efficiency, however, is not sacrificed.

4 SPARSE LINEAR SYSTEM SOLVER

At the center of power network solution is the solution of the following sparse matrix equation:

$$AX = B \tag{1}$$

where, A is a non singular sparse matrix, B is a given vector and X is an unknown vector to be found. A direct method for solving the linear equation 1 is LU factorization [6]. Given the LU, we can perform forward course and backward course on B to solve for X. This is needed in LF calculation. With LU we can also get the inverse of A which is used in SC calculation. Because the A matrix (Y_{bus}, J) is symmetric, the upper triangular matrix is simply the transpose of the lower triangular one $(U = L^{\top})$, so that :

$$LL^{\top}X = B\,. \tag{2}$$

There are several techniques for sparsity programming. The method presented here is Compressed Sparse Row storage (CSR). Three vectors are used for storing and accessing the sparse matrix. When Y_{bus} or J are first built, the elements for each row are contiguous. However advantage of values that are zero comes at a cost. Factorization usually adds nonzero elements (called fills) in addition to those in the original matrix. Keeping track of only the original nonzero values and those created during factorization requires additional computations to identify the fill locations (symbolic factorization). Further, the number of fills depends on the order in which the algorithm processes the columns. For example, once a column becomes filled with non-zeros, the algorithm will replace any zeros originally present in subsequent columns with non-zeros. Therefore, additional computations to find a reasonable ordering that preserves sparsity are needed. Eventually, one comes to the conclusion that a matrix is sparse when it has enough zeros to make the additional overhead worthwhile.

The computation of the power system sparse equations (admittance or Jacobian equations) involves three main steps: *ordering*, to preserve sparsity and thus to reduce work and storage requirements, *factorization*, to decompose the reordered coefficient matrix into a product of factors from which the solution can be computed easily, and *solve*, to compute the solution from the factors.

Ordering: Optimal ordering is used to minimize the number of fills occurring during the factorization of the linearized power system equations. The commonly used methods were proposed by Tinney [7]. These methods specify a procedure relation that gives the conditions for a node to be factorized.

Factorization: The factorization step is split in two phases: symbolic and numerical [8]. The first computes the nonzero structure of the factors and the second computes the numerical values. Symbolic factorization builds the data structures that store references to the nonzero values in the factored matrix. The organization of the data structures significantly affects the performance of numerical factorization.

only occurs when a connector changes state, a relatively infrequent occurrence (on the order of minutes), allowing the extra cost of sorting the off-diagonal elements to be amortized over many solutions.

In the numerical factorization, the factor matrices are computed using the Cholesky method.

Solve: Once the elements in L are solved, unknown vector X can be determined by forward and backward course. The same data structures used during factorization are used in this step. In this paper, we consider a unique $LL^\top$ factorization for real and complex matrices. $LL^\top$ factorization involves following computations.

$$L_{i,j} = A_{i,j} - \sum_{k=1}^{j-1} L_{i,k}^* L_{j,k}\,, \tag{3}$$

$$L_{i,i} = A_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^* L_{i,k}\,. \tag{4}$$

5 IMPLEMENTATION

At the higher layers of our software, the goal was to make the code easy to understand, use, modify and extend. The TSLPS defines a set of data structures and other objects for representing linear algebra objects. A TSLPS matrix is constructed with layers of objects. Each layer is a collection of classes that are templated on the lower layer. At the bottom most layers are the basic numeric types (float, double, etc). The next layers consist of the one dimensional arrays, followed by two dimensional arrays.

One source of performance loss are complex numbers since they are not a built-in C++ data type as in Fortran. There is a template complex class in the Standard C++ library. However we defined our own class for complex numbers but we make minimal use of complex arithmetic operators, which are overloaded.

5.1 Array1D class

The one dimensional arrays Array1D in the TSLPS have two purposes. They are used as vector objects as well as the building blocks for two dimensional arrays. The following data type is defined as a base array class.

```
template <class T>
class Array1D
{
private :
int n_; T* data_;
 ..............................
void set_(T* begin, T* end, const T& val);
public:
Array1D(); // Null constructor
Array1D(int n, const T &a); //create a new n array
inline Array1D(const Array1D &A);
//Copy constructor
inline Array1D & operator=(const T &a);
inline Array1D & operator=(const Array1D &A);
inline T& operator[](int i);
inline const T& operator[](int i) const;
Array1D & copy(const Array1D & A);
inline int dim() const; //Dimension of the array
 ..............................
~Array1D(); //Destructor
}
```

5.2 Array2D class

We construct a two dimensional arrays Array2D from the basic TSLPS arrays like Array1D as follows.

```
template <class T>
class Array2D {
private:
Array1D<T> data_;
int m_, n_;
 ..............................
public:
Array2D(); //Null constructor
Array2D(int m, int n, const T &a);
//Create a new array
inline Array2D(const Array2D &A);
// Copy constructor
inline Array2D & operator=(const T &a);
inline Array2D & operator=(const Array2D &A);
inline T* operator[](int i);
inline const T* operator[](int i) const;
Array2D &copy(const Array2D & A);
inline int dim() const;
 ..............................
 Array2D(); //Destructor
}
```

5.3 SparseSymMatrix class

The sparse symmetric matrix SparseSymMatrix can be constructed from Array1D and Array2D. Array Val is used to store the non-zeros of the matrix row by row, while another array Colind of the same size is used to

store the column positions of these entries. A third array Rowptr has one entry for each row of the matrix, and it stores the position in values of the first nonzero element of each row of the matrix. The following data type is a generic class for sparse symmetric matrix.

```
template <class T>
class SparseSymMatrix
{
private:
Array1D<T> val_; // data values (nz_ elements)
Array1D<int> rowptr_; // row_ptr
Array1D<int> colind_; // col_ind (nz_ elements)
int dim_; // number of rows
.....
public:
SparseSymMatrix (const SparseSymMatrix &Sm);
SparseSymMatrix (int N, int nz, const T *val,
const int *r, const int *c);
//Constructs a symmetric sparse matrix
inline const T& val(int i) const ;
.....
inline const int& row_ptr(int i) const ;
inline const int& col_ind(int i) const ;
inline int dim() const ; //Dimension of the matrix
int NumbNonzeros() const ;
.....
 SparseSymMatrix() ; //Destructor
}
```

5.4 LL$^\top$ Solve class

The Cholesky ($LL^\top$) solver LL$^\top$Solve operates on Array1D and Array2D. The generic class for solving sparse equations is defined as follows.

```
template <class T> class LL^T Solve
{
private
Array2D<T> L_; // lower triangular factor
.....
public:
Array2D<T> OrderTinney( ... );
//Ordering function
LL^T Solve (); //Constructor
LL^T Solve (const Array2D¡T¿ &A);
//Constructs a lower triangular matrix L
Array2D<T> getL() const; //Returns L
Array1D<T> solve(const Array1D<T> &B);
// Solves a linear system A*X=B and returns X
.....
~ LL^T Solve (); //Derstructor
}
```

The bus number is rearranged by using the OrderTinney( ) function. This ordering function is implemented in the LL$^\top$ Solve in order to be available to all power system applications. The feature of the TSLPS is that, for the most part, each of the algorithms is implemented with just one template function. A single algorithm is used whether the array is single, double, or complex. From a software maintenance standpoint, this reuse of code gives TSLPS a fewer lines of code and a huge advantage.

6 POWER SYSTEM APPLICATIONS

The Network container is the basic class in the object oriented power system model [9, 10]. This class, not described here, aggregates all objects that correspond to the physical elements of the power system such as buses, lines, transformers, generators and loads. The class Network is the top class from which the abstract classes LoadFlow and ShortCircuit are derived. Since the LoadFlow and ShortCircuit classes are subclasses of the Network class, they have direct access to the physical objects. Each class has been implemented as a generic as possible. This means that the classes and their methods can be reused over and over again. Additional methods defined in the subclasses implement the higher level functionality.

6.1 Load flow application

LF analysis provide answers to fundamental questions on voltage drop, loading of feeders and power apparatus, and power loss in the network. Voltage, current, and power relationships are governed by power balance nonlinear algebraic equations. Newton-Raphson (NR), and Fast Decoupled (FD) iterative numerical methods are the most commonly used to solve the LF equations simultaneously, by forcing bus voltage and MVA to the solution. The NR method is gradient based, where voltage gradient terms (Jacobian) are used to determine the sensitivity of change in power to the change in voltage. FDLF is an extension of NRLF method, as it relies on the weak coupling between real and reactive power to form decoupled gradient terms, so a fewer equations have to be solved simultaneously.

The Jacobian equations are based on the type of buses. For specified voltage (V, θ) and/or generation (P_g, Q_g) at generator buses, and specified load (P_l, Q_l) at load buses, find a set of bus voltage by iteration such that the mismatch between specified quantities and calculated quantities is within a required tolerance. The mismatch is computed based on Y_{bus} and bus voltage. The following code is defined for a LF analysis.

```
template <class T>
class LoadFlow: public Network
{
private:
double Tol, MaxIter;
//tolerance and max iteration
Array1D<T> Vsol_;//Solution array
SparseSymMatrix<T> JacMat;
//Jacobian matrix
.....
public:
LoadFlow() ; //Constructor
Array1D<T> PowerEq( ... );
//power equations
Array1D<T> MaxMismatch( ... );
//mismatch array
SparseSymMatrix FormJacMat( ... );
//returns Jacobian matrix
Array1D<T> VolCorrection( ... );
.....
LLTSolve<T> SolveLF( ... );
//solves the LF equations
~ LoadFlow();//Destructor
}
```

The MaxMismath( ) is for finding the maximum bus mismatch. The FormJacMat( ) is for forming Jacobian matrix which is stored into the SparseSymMatrix. The SolveLF( ) function is a generic and is implemented in terms of bus, mismatch and the Jacobian element templates. The load flow is calculated by using the sparse linear solver $LL^{\top}$Solve.

6.2 Short circuit analysis

SC for balanced faults, 3 phase, are calculated simply by first reducing the network to the point of the fault into a single impedance, then simulating a SC condition assuming a rated voltage before the fault occurred. Unbalanced faults, however, require applying symmetrical components techniques to permit single phase analysis. Positive, negative, and zero sequence equivalent impedances are calculated and combined in different ways to satisfy faults conditions. The grounding configuration of the network is of paramount significance in unbalanced fault calculations, as it determines the effective zero sequence impedance and the flow of ground currents.

SC analysis require more data than LF analysis. However, the equations solved are linear and their solution doesn't require any iterations. Forming the sequence matrices is the most computationally demanding step of the fault calculations algorithm. The following code is defined for SC analysis where the ShortCicuit is a subclass of Network class.

```
template <class T>
class ShortCircuit : public Network
{
private:
int BusFault; //Bus fault
SparseSymMatrix<T> Ybus, Zbus;
//admittance matrix
Array1D<T> Ibusf ; // Solution array
.....
public:
ShortCircuit(); //Constructor
SparseSymMatrix <T> InvAdm();
//Inverses Ybus LL^T
Solve<T> SolveSymSC ( ... );
//solves the symmetrical SC
LL^T Solve<T> SolveUnsymSC ();
//solves unsymmetrical SC
.....
ShortCircuit(); //Destructor
}
```

In all the cases, the main computation burden appears to be in the storage and inversion of the admittance matrix. Inversion of the admittance matrix is required for the determination of the impedance matrix elements. The InvAdm() is for forming the impedance matrix which is stored into the SparseSymMatrix. The SolveSymSC () and SolveUnsymSC () functions calculate the short circuit by using the sparse linear solver $LL^{\top}$Solve.

7 PERFORMANCE TEST RESULTS

Our implementation of the TSLPS is ongoing. We compared code produced by the TSLPS using templates with our implementations of matrices using simple classes and general matrices, for the row compressed storage applied to solving LF and SC problems. The results presented are obtained on Intel Pentium Processor at 200 MHz using a C++ Builder 5. Four power systems (6bus, 30bus, 59bus and 118bus) are employed in the tests presented in this section. The 59bus is the Algerian power system. Tables 1 and 2 show the CPU time for these power systems and performance of the TSLPS is compared with our programs, developed to analyze LF and SC analysis, written in C++ but without templates.

Table 1. Computation time for FDLF analysis

Test System	CPU time in seconds: Non generic Sparse Matrix +	CPU time in seconds: Template Sparse Matrix
6	0.201	0.200
30	0.227	0.219
59	0.596	0.516
118	1.412	1.209

Table 2. Computation time for SC analysis

Test System	CPU time in seconds: Non generic Sparse Matrix	CPU time in seconds: Template Sparse Matrix
6	0.100	0.100
30	0.103	0.109
59	0.176	0.146
118	0.298	0.203

The CPU time test results indicate that the template implementation is faster than the non generic one especially for the 59 bus and 118 bus power systems. The results show that template is an alternative for the development of large software for power systems applications even when high computational performance is a critical requirement.

8 CONCLUSION

A number of projects in the numerical analysis community have exploited template to support sparse matrix computation. This paper contributes with the development of a sparse template library for power system TSLPS. The ability to construct and manipulate types has proven extraordinarily useful in writing TSLPS. OOP reduce the efforts for development and maintenance of large software systems. Often objects oriented programs require more time to execute. The use of template has shown that this time can be effectively reduced. This implies that template is attracting attention of power system analyzers and developers using OOP as it reduces time computations. The most difficult part of power system programming is to deal with the network relationship. Load flow and short circuit analysis have been successfully applied to the solution of several power systems and presented good computational performance.

References

[1] ZHU, J.—LUBKEMAN, D. L.: Object Oriented Development of Software System for Power System Simulations, IEEE Trans. on Power Systems **12** No. 2 (May 1997), 1002–1007.

[2] GAING, Z. L.—LU, C. N.—CHANG, B. S.—CHENG, C. L.: An Object Oriented Approach For Implementing Power System Restoration Package, IEEE Trans. on Power Systems **11** No. 1 (February 1996), 483–489.

[3] ZHOU, E. Z.: Object Oriented Programming, C++ and Power System Simulation, IEEE Trans. on Power Systems **11** No. 1 (February 1996), 206–215.

[4] HAKAVIK, B.—HOLEN, A. T.: Power System Modelling and Sparse Matrix Operations Using Object Oriented Programming, IEEE Trans. On Power Systems **9** No. 2 (1994), 1045–1052.

[5] ZHOU, E. Z.: Design of Generic Direct Sparse Linear System Solver in C++ for Power System Analysis, IEEE Trans. on Power Systems **16** No. 4 (November 2001), 206–215.

[6] PANDIT, S.—SOMAN, S. A.—KHAPARDE, S. A.: Object Oriented Design for Power System Applications, IEEE Computer Applications in Power (October 2000), .43–47.

[7] TINNEY, W. F.—WALKER, I. W.: Direct Solution of Sparse Network Equations by Optimally Ordered Triangular Factorization, Proc. of the IEEE **55** (November 1967), 1801–1809.

[8] ESQUIVEL, CR F.—ACHA, E.—TAN, SG—RICO, J. J.: Efficient Object Oriented Power Systems Software for the Analysis of Large Scale Networks Containing FACTS-Controlled Branches, IEEE Trans. on Power Systems, **13** No. 2 (May 1998), 464–472.

[9] NEYER, A. F.—WU, F. F.—IMHOF, K.: Object Oriented Programming For Flexible Software: Example of a Load Flow, IEEE Trans. on Power Systems **5** No. 3 (August 1990), 689–696.

[10] RUMBAUGH, J.*et al*: Object Modelling and Design, Prentice Hall, Englewood Cliffs, New Jersy, 1991.

Received 13 December 2004

Amrane Tolba received the Master Degrees (2000) from Batna University, and now is a teaching assistant. He is working for PhD at the same University. His research interests include power system automation and power system software development.

Louiza Benfarhi received the Master Degrees (1997) from Batna University, and now is a teaching assistant. She is working for PhD Her research interests include OOP and power system software development.

Mohamed Belkacemi is a Professor at Batna University. He received the Master Degrees (1987) and PhD from UMIST, UK (1991). His research interests include sparse matrix computation and power system software development.

SLOVART G.T.G. s.r.o. GmbH
EXPORT - IMPORT

EXPORT - IMPORT
of ***periodicals*** and of non-periodically
printed matters, ***books*** and ***CD - ROMs***

Krupinská 4 PO BOX 152, 852 99 Bratislava 5,Slovakia
tel.: ++421 2 638 39 472-3, *fax.:* ++421 2 63 839 485
e-mail: gtg@internet.sk, http://www.slovart-gtg.sk

SLOVART G.T.G. s.r.o. GmbH
EXPORT - IMPORT