République Algérienne Démocratique et Populaire

Ministère de l'Education Nationale

ECOLE NATIONALE POLYTECHNIQUE

Projet de fin d'Etudes

en vue d'obtention du

Diplôme d'Ingénieur d'Etat

en

Génie Electrique

Option : Electrotechnique

Sujet

Réalisation d'un Environnement Graphique avec Base de Données pour l'Analyse et la Simulation de Réseaux Electriques

Présenté par :

Lotfi BAGHLI

Sous la Direction de :

Monsieur A. Hellal

Promotion

Juin 1994

E.N.P. 10, Avenue Pasteur, HASSEN BADI, El-Harrach, 16200 Alger, ALGERIE

Table des matières

Résumé

Le travail que nous avons effectué consiste en le développement d'un environnement graphique avec base de données pour l'analyse et la simulation de réseaux électriques.

Le logiciel que nous avons élaboré s'appelle **POWER DESIGNER**. Il est entièrement orienté objets, ce qui lui confère une structure souple et évolutive. **POWER DESIGNER** est programmé pour Microsoft Windows et bénéficie d'une interface utilisateur puissante, agréable et interactive.

POWER DESIGNER permet de saisir graphiquement le schéma unifilaire d'un système de puissance et de ses composants, de l'éditer, de l'archiver et d'opérer des simulations d'écoulement de puissance et de stabilité transitoire avec plusieurs méthodes de calcul.

Les résultats de ces simulations sont rendus sous forme texte et graphique.

Abstract

The present work consists in the development of a data based graphical package for analysing and simulating electrical networks.

The software package we have elaborated is called **POWER DESIGNER**, and is completely object-oriented. This endows it with a flexible and versatile structure.

POWER DESIGNER is intended for Microsoft Windows and has a powerful user interface which is agreeable and interactive.

With **POWER DESIGNER**, one-line diagrams of power systems can be graphically constructed, edited and stored. Load flow and transient stability simulations can be carried with various methods.

The results of these simulations are displayed in text and graphical forms.

Remerciements

Je tiens à exprimer ma profonde gratitude envers mon directeur de mémoire, Monsieur A. Hellal, pour avoir dirigé ce travail et prodigué de nombreux et judicieux conseils.

Je remercie tout particulièrement Monsieur Djamel-Eddine Bourouba pour avoir testé les versions successives de **POWER DESIGNER** et pour ses précieux conseils.

Je voudrais aussi exprimer ma vive reconnaissance envers tous les enseignants de l'Ecole Nationale Polytechnique et ses travailleurs ainsi que tous ceux qui ont participés à ma formation.

Dédicaces

Je dédie ce mémoire

à toute ma famille

et à tous mes amis

Lotfi

Table des matières

Introduction

L'avènement de micro-ordinateurs individuels de plus en plus puissants (386, 486, Pentium) a poussé les développeurs à passer du simple système d'exploitation qui gère les tâches de bas niveau, aux systèmes beaucoup plus sophistiqués et proches de l'utilisateur. Des interfaces graphiques ont commencé à apparaître sur le Macintosh puis elles ont envahi les environnements de développement. Sur les IBM PC et compatibles tournant sous DOS, il y a Windows, sur ceux qui utilisent OS/2, il y a Presentation Manager, l'Atari ST utilise GEM, les machines sous UNIX utilisent un système XWindow, les stations de travail SUN utilisent NeWS et le cube NeXT utilise NextStep.

Mais Windows a fini par s'imposer sur les PC car il requiert une configuration relativement modeste. Avec sa version 3.0 et l'actuelle 3.1, Windows a acquis une grande popularité chez les utilisateurs. Ils bénéficient d'un environnement graphique multitâche où plusieurs programmes peuvent s'exécuter simultanément et échanger des informations, chose qui n'existait pas sous DOS.

De leur coté, les électrotechniciens et plus particulièrement ceux qui s'occupent du mouvement d'énergie dans les systèmes de puissance, ont développé beaucoup de programmes de calcul et d'analyse de systèmes électriques de puissance. Mais ces programmes, le plus souvent, ne permettaient à chaque fois que l'analyse d'un seul aspect des opérations sur le système de puissance (Ecoulement de puissance, défauts ...). De plus, les données qu'ils utilisent exigent souvent un format spécifique, ce qui limite l'interchangeabilité entre les différents programmes.

La gestion des programmes développés d'une manière non modulaire et non structurée n'est pas aisée. L'ingénieur trouve beaucoup de difficultés à les revoir et les faire évoluer afin de les maintenir à jour avec la technologie toujours en évolution du hardware et du software.

L'arrivée de mini et micro-ordinateurs puissants, dotés d'interfaces graphiques multitâche a permis de revoir les programmes déjà écrits, de les restructurer en modules pouvant communiquer entre eux et de leur ajouter des possibilités d'utilisation interactives plus agréables.

C'est dans ce cadre que s'inscrit le présent projet. A l'instar des systèmes de Conception Assistée par Ordinateur (CAO) qui servent à concevoir les composants électroniques, les circuits intégrés VLSI et les cartes de circuits imprimés, nous avons voulu développer quelque chose de similaire pour les systèmes de puissance.
Toutes les tâches de préparation des données, de simulation et de présentation des résultats sont effectuées par ordinateur. Elles peuvent donc êtres intégrées en un seul produit (package). On y gagne ainsi en temps et on évite les erreurs de manipulation.

Le logiciel que nous avons réalisé, **POWER DESIGNER**, intègre l'édition, la simulation et la présentation des résultats. Ce logiciel est programmé pour Windows. Il permet de partager les ressources qu'offre cet environnement avec les autres applications Windows dans un cadre multitâche et bénéficie des possibilités d'échange de données.

POWER DESIGNER permet de construire graphiquement les réseaux électriques de puissance, de les modifier, d'opérer des simulations et de sortir les résultats dans des formats texte et graphique.

POWER DESIGNER se veut aussi être un outil pédagogique, qui permet à l'étudiant d'agir de manière interactive sur la topologie du réseau et sur les simulations.

Il est difficile de trouver la juste mesure entre décrire le logiciel pour les programmeurs qui voudraient faire quelque chose de similaire et le décrire pour les utilisateurs de manière à ce qu'ils exploitent **POWER DESIGNER** au maximum de ses possibilités.

Dans ce mémoire, nous présentons dans un premier chapitre les éléments essentiels de la Programmation Orientée Objets, de l'environnement Windows et de la programmation sous cet environnement. Les notions introduites dans ce chapitre sont nécessaires pour une bonne compréhension du chapitre II. Ce dernier présente la structure et la hiérarchie des objets que nous avons mis au point pour le logiciel **POWER DESIGNER**. Nous montrons les développements successifs de cette hiérarchie afin de mettre en évidence l'extensibilité des logiciels basés sur les concepts de la Programmation Orientée Objets.

Nous présentons également quelques commandes et possibilités de **POWER DESIGNER**. Cependant, vu leur grand nombre, il n'est pas possible de les détailler sans alourdir le présent document. L'utilisateur devra activer l'Aide fournie avec **POWER DESIGNER** pour plus de détails sur l'utilisation du logiciel.

Nous évoquons aussi dans le chapitre II quelques aspects de la gestion de la base de données constituée des composants du réseau. Ce sont les liens électriques entre les composants, l'archivage de cette base de données à l'aide des flux et la mise en oeuvre des fonctions " Couper / Coller ". Ils ont nécessité un effort particulier et il nous a semblé nécessaire d'en présenter les principes. Ceux-ci sont introuvables dans la bibliographie.

Dans les chapitres III et IV, nous présentons les applications de notre logiciel dans l'étude d'écoulement de puissance et de stabilité transitoire des réseaux électriques.

Nous mettons en évidence les modèles utilisés et les hypothèses simplificatrices ainsi que les algorithmes de calcul que nous avons développés. Des exemples viennent ensuite valider ces études.

Enfin, une conclusion générale récapitule le travail que nous avons effectué et les principaux résultats que nous avons obtenus.

Chapitre I : Généralités

1. Outils de développement

Les programmes de cette envergure ne peuvent être faits en un seul bloc, ils sont généralement divisés en unités plus petites afin de faciliter le développement et surtout le débogage (mise au point). Chaque unité est en fait constituée d'un ensemble de routines analogues. Un concept encore plus révolutionnaire permet une meilleure structuration sans la figer; c'est la Programmation Orientée Objets (POO).

Nous avons voulu faire bénéficier **POWER DESIGNER** d'une interface graphique de haut niveau. Naturellement, cette interface doit supporter la gestion des applications, des fenêtres, de la souris, des périphériques...
Windows 3.1 de Microsoft s'est imposé dans le monde du PC comme étant l'environnement graphique de référence pour le système d'exploitation DOS et il requiert une configuration plus modeste que Presentation Manager pour OS/2 d'IBM.

Il faut cependant, pour générer des applications Windows, disposer du compilateur adéquat. Le seul outil disponible à notre niveau était le Borland Pascal Objects 7.0 de Borland.

POWER DESIGNER se veut être évolutif, extensible avec un minimum d'effort. On doit donc pouvoir lui ajouter d'autres modules de simulation, d'autres composants sans modifications majeures aux programmes existants.

La Programmation Orientée Objets (POO) offre une solution en or à ce problème, en ce sens qu'elle permet, grâce à des concepts d'héritage et d'encapsulation, une structuration des données et des méthodes et une souplesse d'extensibilité sans égal.

Des langages de POO pure tels que SmallTalk ou Modula 2 existent, mais ils ne sont pas d'un usage général. Le Turbo Pascal depuis sa version 5.5 intègre quelques concepts de POO et la version 7.0 s'est vue rajeunir avec de nouvelles instructions et a atteint un stade qui lui permet de s'affirmer comme un langage de Programmation Orientée Objets.

POWER DESIGNER se doit aussi d'être indépendant du réseau électrique étudié et de sa taille. Seule la gestion dynamique de la mémoire autorise la réservation de l'espace mémoire requis pour le stockage des variables au moment de l'exécution. Elle répond donc exactement à notre cahier de charge, à la différence d'une gestion statique où la réservation de la mémoire se fait à la compilation de manière fixe; ce qui limite le domaine d'application.

Le Borland C++ offre une excellente gestion de la mémoire, il est considéré comme un langage puissant et général.

POWER DESIGNER comprend :

Deux (2) exécutables Windows, programmés avec le Borland Pascal Objects 7.0 :

- PowerD qui gère l'environnement, l'édition et les appels de simulations.
- GraphD, un outil de récapitulation graphique pour PowerD.

- Sept (7) exécutables DOS, programmés avec le Borland C++ 3.1 afin de bénéficier de la gestion dynamique de la mémoire et des flux de données. Ces programmes sont des modules de calcul d'écoulement de puissance et de stabilité transitoire que PowerD appelle directement d'un simple clic sur le menu correspondant.

Avant d'aborder la structure des objets de **POWER DESIGNER**, un rappel sur les concepts de Programmation Orientée Objets s'impose.

2. La Programmation Orientée Objets

La POO vise à mettre au point des modules réutilisables pour aboutir à un ensemble de "composants logiciels", par analogie avec les composants électroniques. L'espoir ultime étant de parvenir à bâtir un nouveau produit logiciel à partir d'un "catalogue" de composants élémentaires [1].

La POO est basée sur le concept de **type objet** (appelé aussi classe dans d'autres langages orientés objets comme le C++). Un type objet est plus général qu'un type enregistrement. Si ce dernier ne peut contenir que des champs de données, le type objet peut, en outre, contenir des méthodes, c'est à dire des sous programmes.

Une fois un type objet défini, on pourra déclarer des variables de ce type qu'on qualifiera d'objets. Au même titre que :

```
Var I : Integer;
```

déclare une variable I du type entier.

En POO pure, seules les méthodes d'un objet ont accès aux données de cet objet, on parle alors d'**encapsulation**; mais certains langages (le Turbo Pascal et le C++) autorisent cet accès. Ces données seront considérées comme des variables locales indépendantes des variables globales même si elles portent le même identificateur.

Un des concepts les plus importants de la POO est celui d'**héritage**. Il permet de définir un nouveau type objet à partir d'un type objet déjà existant auquel on ajoute de nouveaux champs de données et de nouvelles méthodes. On bénéficie ainsi des "aptitudes" de l'ancien objet et on le "spécialise" pour une tâche ciblée.

Ces concepts étant communs à tous les langages orientés objets, nous ne parlerons par la suite que de la POO en Turbo / Borland Pascal.

Précisons que les Turbo Pascal 5.5 à 7.0 permettent de générer des applications qui tournent sous DOS, tandis que le Borland Pascal Objects 7.0 est un produit complet qui fournit en plus du Turbo Pascal 7.0, le BP 7.0 et le BP Windows 7.0. Le BP est un IDE (Environnement de Développement Intégré) sous DOS et le BPW est un IDE sous Windows. Ces deux produits permettent de compiler vers le DOS, le DOS en Mode Protégé (DPMI) et vers Windows.

2.1. Exemple d'un enregistrement en Turbo Pascal

```
type    Position = Record
        X : real;
        Y : real;
        End;
```

Position est un type enregistrement qui comporte 2 champs réels : X et Y.
Une fois des variables A et B déclarées du type Position par :

```
Var A, B : Position;
```

A.X donne accès au champ X de la variable A et A.Y donne accès au champ Y.
De même pour B.X et B.Y.

2.2. Exemple d'un type objet en Turbo Pascal

```
type    Emplacement = object
        X : real;
        Y : real;
        procedure Init( X0, Y0 : real);
        procedure Deplace_de( dX, dY : real);
        procedure Situation;
        End;
```

Emplacement est un type objet qui comporte 2 champs réels et 3 méthodes. Il faut de plus expliciter ces méthodes dans le corps du programme, en écrivant par exemple :

```
procedure Emplacement.Init( X0, Y0 : real);
begin
X:=X0;
Y:=Y0;
end;

procedure Emplacement.Deplace_de( dX, dY : real);
begin
X:=X+dX;
Y:=Y+dY;
end;

procedure Emplacement.Situation;
begin
WriteLn( 'Je suis en X=',X,' et Y=',Y);
end;
```

Une fois que la variable A a été déclarée comme objet de type Emplacement, on peut l'utiliser et elle représente une instance de type Emplacement.

```
Var A : Emplacement;
Begin
A.Init( 2.3, -5.12); { initialise les champs X et Y de A à 2.3 et -5.12 }
A.Situation;         { affiche la valeur des champs X et Y de A }
A.Deplace_de( 3, 3); { ajoute 3 aux champs X et Y de A }
A.Situation;
End.
```

En sortie, nous obtenons :

```
Je suis en X= 2.3000000000E+00 et Y=-5.1200000000E+00
Je suis en X= 5.3000000000E+00 et Y=-2.1200000000E+00
```

2.3. Héritage entre objets

A partir d'un type objet défini, on peut créer des héritiers :

```
type Point = object ( Emplacement)
    Couleur : byte;
    procedure Colore_toi( C : byte);
    procedure Situation;
    End;

procedure Point.Colore_toi( C : byte);
begin
Couleur:=C;
end;

procedure Point.Situation;
begin
WriteLn( 'Je suis un point en X=',X,' et Y=',Y);
WriteLn( ' et de couleur=', Couleur);
end;

Var B : Point;
Begin
B.Init( 2.3, -5.12);
B.Colore_toi( 0);
B.Situation;
B.Deplace_de( 3, 3);
B.Situation;
B.Colore_toi( 4);
B.Situation;
End.
```

En sortie, nous obtenons :

```
Je suis un point en X= 2.3000000000E+00 et Y=-5.1200000000E+00
 et de couleur=0
Je suis un point en X= 5.3000000000E+00 et Y=-2.1200000000E+00
 et de couleur=0
Je suis un point en X= 5.3000000000E+00 et Y=-2.1200000000E+00
 et de couleur=4
```

Nous dirons que le type objet Point **hérite** du type objet Emplacement ou encore que Point est un **descendant** de Emplacement. De même, nous dirons que Emplacement est un **ancêtre** ou un **ascendant** de Point [1].

Le type objet Point a 3 champs et 4 méthodes, dont une (Situation) a été redéfinie.

2.4. Pointeurs et objets

Il y a deux manières de créer une instance d'un type objet :

- Soit en déclarant une variable de ce type et qui devient un objet comme nous l'avions déjà vu avec :

  Var B : Point;

- Soit par le biais d'un **pointeur** sur un objet.

Un pointeur est une variable statique qui contient une adresse sur une donnée dynamique. A la différence des variables statiques et automatiques, les variables dynamiques (ou les objets dynamiques) sont gérés par le programmeur et il faut leur allouer de l'espace en mémoire avant d'être utilisés. Ceci se fait par l'instruction **New**. Après leur utilisation, il faut libérer (désallouer) cet espace par l'instruction **Dispose**.

Nous utilisons le pointeur **PB** vers un objet du type Point.
PB est le pointeur sur cet objet et **PB^** est l'objet lui-même.

```
Var PB : ^Point;
Begin
new( PB);                { alloue l'espace dynamique }
PB^.Init( 2.3, -5.12);
PB^.Colore_toi( 0);
PB^.Situation;
Dispose( PB); { libère l'espace dynamique }
End.
```

En sortie, nous obtenons :

```
Je suis un point en X= 2.3000000000E+00 et Y=-5.1200000000E+00
 et de couleur=0
```

2.5. Compatibilité entre types objet

Si A et B sont deux objets du type objet Point, l'instruction A:=B; provoque la recopie des champs de B dans ceux de A. Ceci équivaut à :

```
A.X:=B.X;
A.Y:=B.Y;
A.Couleur:=B.Couleur;
```

Si PA et PB sont des pointeurs sur des objets du type Point,
PA^:=PB^; a le même effet que ci-dessus.

Par contre, PA:=PB; ne provoque pas la recopie des champs de PB^ dans ceux de PA^, mais change l'adresse pointée par PA en celle pointée par PB.
Après une telle instruction, l'ancien objet PA^ est perdu et les PA et PB pointent vers le même objet.

Nous pouvons maintenant présenter une règle de POO extrêmement utilisée dans **POWER DESIGNER** et qui stipule qu'un type objet descendant peut être utilisé là où on avait besoin de son ancêtre.

Prenons un exemple simple. Soit PE un pointeur sur notre objet de type Emplacement, et PA un pointeur sur un objet de type Point, que nous redéfinissons pour l'occasion comme suit :

```
type Point = object ( Emplacement)
    Couleur : byte;
    procedure Init( X0, Y0 : real; C0 : byte);
```

```
    procedure Colore_toi( C : byte);
    procedure Situation;
    End;
```

C'est-à-dire que nous redéfinissons la méthode Init dans le descendant Point de manière à ce qu'elle initialise même la couleur. De plus, dans cette redéfinition, nous faisons appel à la méthode Init de l'ancêtre Emplacement.

```
procedure Point.Init( X0, Y0 : real; C0 : byte);
begin
Emplacement.Init( X0, Y0);
Couleur:=C0;
end;
...


Var PE : ^Emplacement;
   PA : ^Point;
...
PE^.Init( 1, 1);
PE^.Situation;
```

En sortie, nous obtenons :

```
Je suis en X= 1.0000000000E+00 et Y= 1.0000000000E+00
```

```
PA^.Init( 2.13, -5.12, 2);
PA^.Situation;
```

En sortie, nous obtenons :

```
Je suis un point en X= 2.1300000000E+00 et Y=-5.1200000000E+00
 et de couleur=2
```

Revenons aux histoires de recopie, avec :

```
PE:=PA;
```

Le pointeur PE pointe maintenant sur l'objet PA^. Bien que pour le compilateur, PE reste toujours un pointeur sur un objet du type Emplacement. De ce fait seul les champs définis dans l'ancêtre Emplacement sont accessibles par PE^. De même, la méthode Situation appelée par

```
PE^.Situation;
```

sera celle du type objet Emplacement bien qu'utilisant la valeur des champs X, Y de l'objet PA^.

En sortie, nous obtenons donc :

```
Je suis en X= 2.1300000000E+00 et Y=-5.1200000000E+00
```

C'est une restriction assez sévère. Elle est due à la manière dont le compilateur a généré l'appel de la méthode Situation. On est en face d'un cas de **ligature statique**.

Ceci est très gênant. En effet, supposons (et c'est le cas dans notre projet) que nous ayons besoin d'une liste chaînée dynamique[1] d'objets de types différents (Générateurs, noeuds, résistances, lignes...) mais qui ont tous un ancêtre commun (Composant).

Nous pouvons parcourir la liste avec un pointeur du type pointeur sur Composant. Mais il n'est pas possible d'appeler, par le biais de ce simple pointeur, une méthode si elle est redéfinie dans certains de ses descendants.

Or, l'un des avantages de la POO est de permettre le **polymorphisme**. Ce terme signifie que l'on attribue un même nom à une méthode qui s'adapte au type objet concerné.
Tant que les objets sont statiques (définis à la compilation), le problème ne se pose pas, mais si les objets sont dynamiques (alloués par **new**), ils peuvent s'avérer incompatibles avec le polymorphisme.

2.6. Polymorphisme et méthodes virtuelles

La solution apportée par Turbo Pascal à cette ambiguïté réside dans l'emploi de **méthodes virtuelles** là où l'on voudrait que le compilateur ne fige pas les appels des méthodes.

La méthode réellement appelée est déterminée alors en fonction du type de l'objet qui appelle. En revenant à notre exemple, si à un instant donné PE pointe sur un type Emplacement, PE^.Situation appellera Emplacement.Situation.
Si plus tard PE pointe sur un type Point, PE^.Situation appellera alors Point.Situation. On parle dans ce cas de **ligature dynamique**.

Cependant, pour que Turbo Pascal puisse gérer ces appels, il faut que :

- Chaque méthode, dont l'appel ne peut être défini au moment de la compilation, soit déclarée comme virtuelle (mot clé **virtual**) dans toute la hiérarchie, là où elle est redéfinie.
- Tout objet comportant au moins une méthode virtuelle se doit de comporter une méthode dite constructeur (mot clé **constructor**) qui permet d'initialiser l'objet et de créer une table des méthodes virtuelles (VMT : Virtual Memory Table).
  Cette méthode doit être appelée en premier avant de commencer à utiliser l'objet. On lui affecte la tâche d'initialiser certains champs de l'objet.

Notre exemple devient :

```
{ Polymorphisme et méthodes virtuelles }
Program Demo3;

uses crt;

type Emplacement = object
    X : real;
    Y : real;
    constructor Init( X0, Y0 : real);
    procedure Deplace_de( dX, dY : real);
    procedure Situation; virtual;
```

[1] Voir le chapitre II pour plus d'informations sur la liste chaînée NetWork utilisée dans POWER DESIGNER.

```
    End;

type Point = object ( Emplacement)
    Couleur : byte;
    constructor Init( X0, Y0 : real; C0 : byte);
    procedure Colore_toi( C : byte);
    procedure Situation; virtual;
    End;

{ Definition de Emplacement }
constructor Emplacement.Init( X0, Y0 : real);
begin
X:=X0;
Y:=Y0;
end;

procedure Emplacement.Deplace_de( dX, dY : real);
begin
X:=X+dX;
Y:=Y+dY;
end;

procedure Emplacement.Situation;
begin
WriteLn( 'Je suis en X=',X,' et Y=',Y);
end;

{ Definition de Point }
constructor Point.Init( X0, Y0 : real; C0 : byte);
begin
Emplacement.Init( X0, Y0);
Couleur:=C0;
end;

procedure Point.Colore_toi( C : byte);
begin
Couleur:=C;
end;

procedure Point.Situation;
begin
WriteLn( 'Je suis un point en X=',X,' et Y=',Y);
WriteLn( ' et de couleur=', Couleur);
end;

Var PE : ^Emplacement;
   PA : ^Point;

Begin
clrscr;
new( PE, Init( 1, 1));
PE^.Situation;

new( PA, Init( 2.13, -5.12, 2));
PA^.Situation;
```

```
{ Bon polymorphisme }
PE:=PA;
PE^.Situation;

Dispose( PE);
{ Dispose( PA); }
End.
```

En sortie, nous obtenons bien :

```
Je suis en X= 1.0000000000E+00 et Y= 1.0000000000E+00
Je suis un point en X= 2.1300000000E+00 et Y=-5.1200000000E+00
 et de couleur=2
Je suis un point en X= 2.1300000000E+00 et Y=-5.1200000000E+00
 et de couleur=2
```

Nous encourageons ceux qui veulent approfondir leur connaissances en POO à consulter les références [1] et [2].

3. Environnement Windows

Windows est un environnement graphique multitâche à base de fenêtres où peuvent s'exécuter des programmes spécialement écrits pour lui.

Les programmes Windows sont similaires en apparence et leurs commandes sont homogènes. Par exemple :

- On sort de n'importe quel programme Windows par la commande Alt+F4 ou son équivalent menu.
- Tous les programmes qui présentent une zone d'affichage plus grande que la dimension de leur fenêtre disposent d'ascenseurs (encore des objets) qui permettent de faire défiler la fenêtre et qui fonctionnent de la même manière d'un programme à l'autre.

Nous invitons l'utilisateur à se rapprocher d'un PC sur lequel est installé Windows et d'actionner l'Aide encore une fois disponible de la même manière sur les programmes Windows.

Windows peut également exécuter des programmes qui ne sont pas spécialement conçus pour lui. Ainsi, les applications MSDOS peuvent être lancées dans ce que l'on appelle une session DOS. Plusieurs sessions DOS peuvent être ouvertes simultanément en fonction de la mémoire disponible et du mode de fonctionnement de Windows : [5]

- Mode réel : PC à base d'un microprocesseur 8086 (ou 286/386 avec moins de 1 MO de RAM).
- Mode standard (protégé) : 286 et au moins 1 MO de RAM (ou 386 avec moins de 2 MO de RAM).
- Mode 386 étendu : 386 et plus avec au moins 2 MO de RAM.

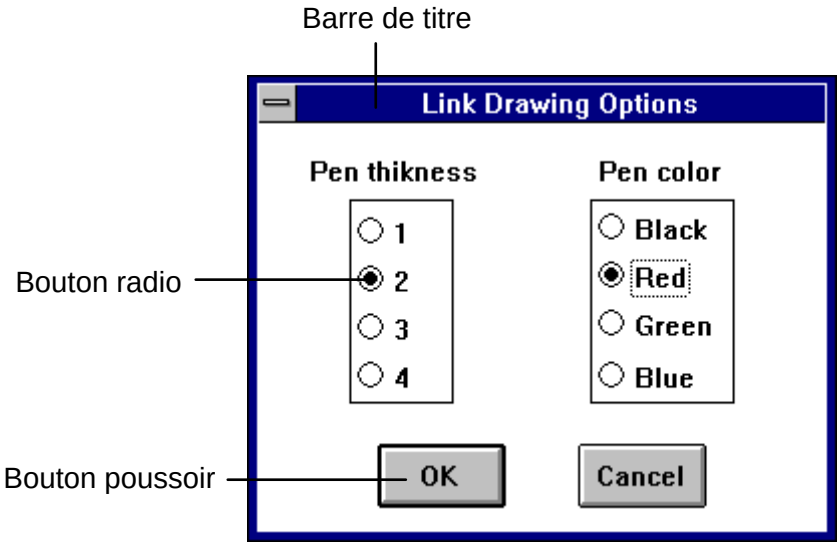
Windows procure aux développeurs de programmes une relative facilité quand à l'implantation de menus déroulants, boîtes de dialogue, barres de défilement. Un langage de programmation graphique est également disponible (GDI) et permet d'afficher des primitives graphiques (lignes, rectangles, ellipses, polygones, textes) avec des attributs de couleur, de trait et de remplissage très divers. La gestion des périphériques (clavier, souris,

affichage vidéo, imprimantes, modem...) est entièrement assurée par Windows de sorte que le programmeur n'a pas à se préoccuper de la plate-forme matérielle dont disposera l'utilisateur de son programme.

4. Programmer pour Windows

Ecrire un programme pour Windows est plus difficile que d'en écrire un pour le DOS. Ce qui se présente clairement et de manière intuitive à l'utilisateur est complexe à programmer. Windows est un tout, ses éléments sont très interconnectés; par exemple, pour dessiner un graphique sur l'écran, on a besoin d'un élément qui se nomme "Handle de Contexte de Dispositif (HDC)"- une sorte de pointeur -, pour lequel on a besoin d'un "Handle de fenêtre". Pour l'obtenir, il faut créer une fenêtre et la préparer à recevoir des "messages". Pour recevoir et prendre en charge ces messages dans une fenêtre, on doit créer une fonction de fenêtre...

Quand on programme sous Windows, on manipule des objets Windows même si le langage de programmation n'est pas orienté objets. Le plus important de ces objets est l'objet **fenêtre**. Il prend en compte les actions de l'utilisateur provenant du clavier et de la souris et affiche les informations à l'intérieur de sa zone client. L'objet fenêtre contient en outre, une barre de titre, des boutons  et des bordures pour modifier la taille de la fenêtre et un menu pour sélectionner les commandes. Les boîtes de dialogue sont des fenêtres spécifiques qui comportent de nombreux "enfants" tel que des boutons poussoir, boutons radio, cases de pointage, champs de saisie...



Exemple d'une des boîtes de dialogue de POWER DESIGNER

L'utilisateur agit sur ces éléments à l'aide de la souris et du clavier. Windows recueille ces informations et les transforme sous forme de **messages** qu'il envoie à la fenêtre active. La notion de message est fondamentale dans la programmation Windows. Une "procédure de fenêtre" se charge spécialement de traiter ces messages.

De nombreux outils existent pour programmer sous Windows. Microsoft propose SDK (Software Development Kit) qui contient les bibliothèques des routines et des fichiers en-têtes qu'il faut utiliser avec un compilateur C et plus récemment Visual Basic 3.0 et Visual C++. En fait, le choix nous a été imposé par la disponibilité du compilateur et il s'est porté sur

le Borland Pascal Object 7.0. C'est un outil très complet et comportant de nombreux exemples de programmes.

De plus, c'est un langage orienté objets et ses développeurs ont construit une bibliothèque d'objets appelée **Object Windows Library** (OWL) qui facilite grandement la gestion des messages et permet de les rediriger vers les méthodes des objets d'OWL ou de leurs descendants.

Naturellement, le programmeur est libre de ne pas utiliser l'OWL, mais il lui faut prévoir des procédures de branchement par lesquelles Windows envoie ses messages ainsi que les initialisations dont les objets OWL se chargeaient.

Pour que **POWER DESIGNER** puisse être évolutif et extensible, la Programmation Orientée Objets devait répondre présent à tous les niveaux. Nous avons donc choisi d'utiliser les objets de l'OWL comme ancêtres à nos objets plus spécifiques; ce qui est présenté dans le chapitre suivant.

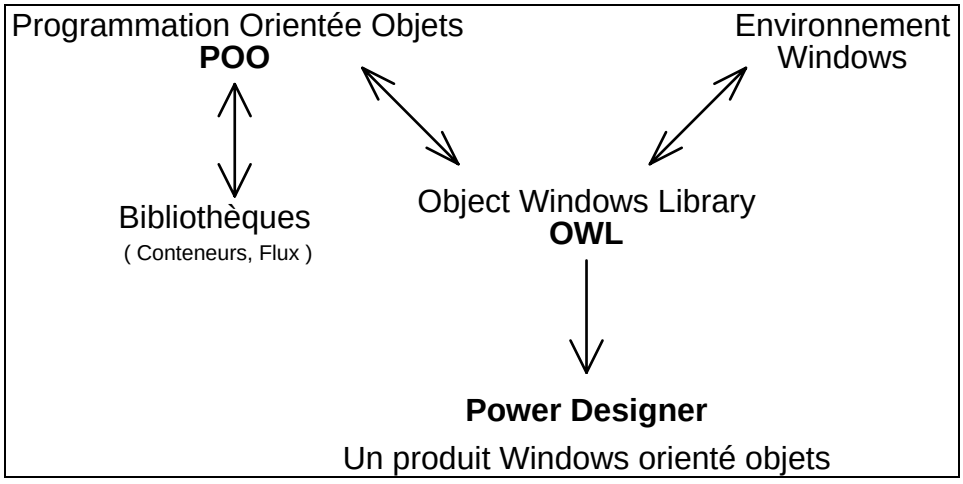


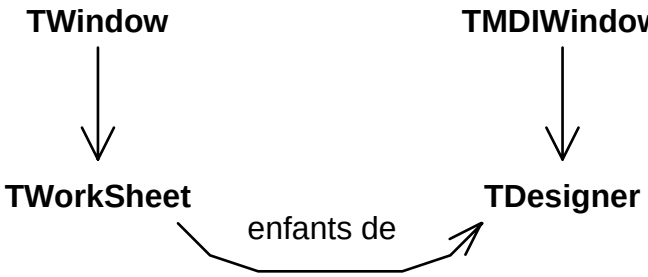
Schéma récapitulatif des outils utilisés pour l'élaboration de POWER DESIGNER

CHAPITRE II : POWER DESIGNER : structure et utilisation

1. Structure de POWER DESIGNER

POWER DESIGNER est une application pour Windows. Elle est constituée dans son interface extérieure d'une fenêtre **TDesigner**[2] descendant du type objet **TMDIWindow**.

TMDIWindow (pour Multiple Document Interface) est une fenêtre objet qui permet de gérer plusieurs documents à la fois appelés Enfants (Child). **TDesigner** est donc une fenêtre MDI puisqu'il hérite des propriétés de son ascendant.



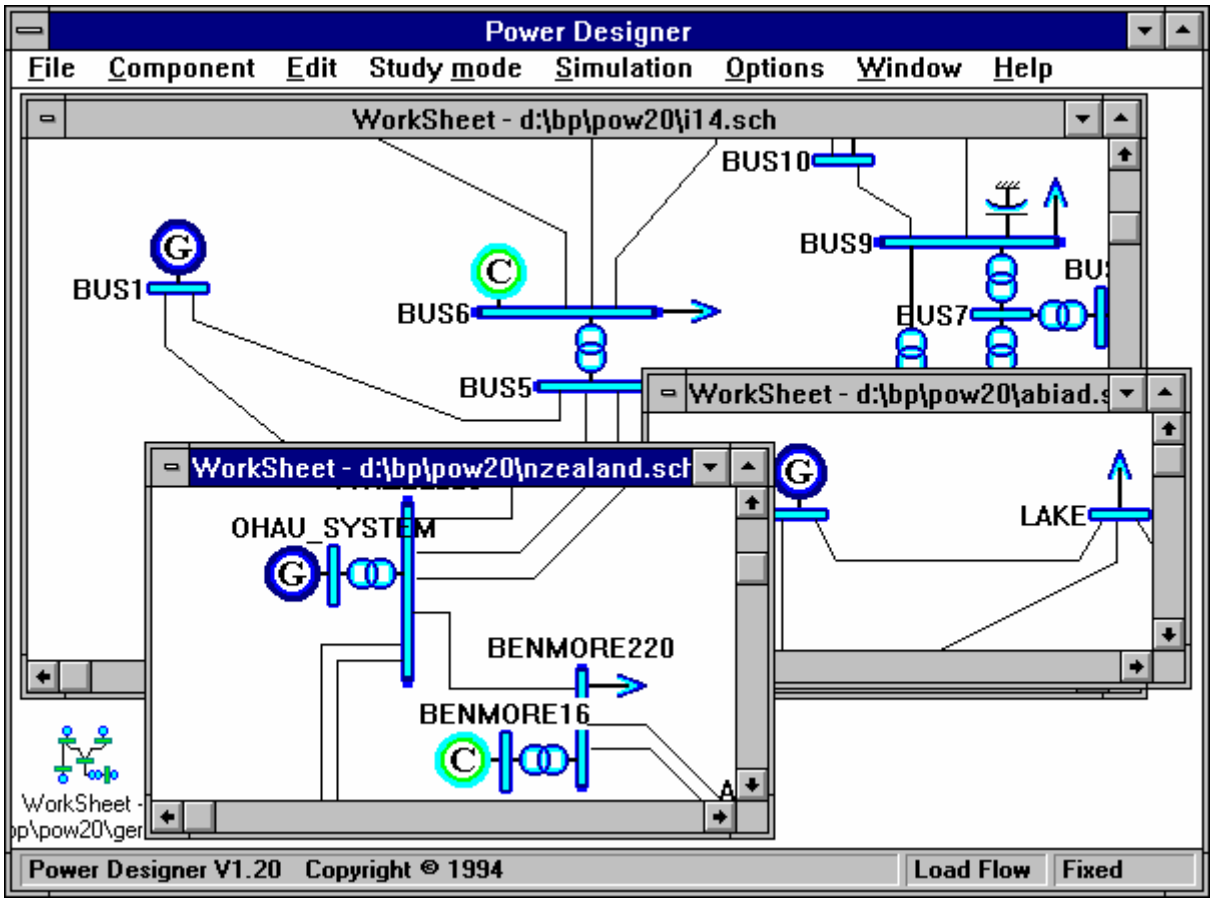
Les enfants d'une MDI sont des fenêtres **TWindow** ou (comme la règle de POO le stipule: là où on attend un objet, son descendant peut faire l'affaire) son descendant **TWorkSheet** (littéralement feuille de travail). C'est le plus grand des objets par le nombre de ses champs et de ses méthodes et la longueur du code associé (voir Annexes 1 et 3). **TWorkSheet** représente le document, c'est-à-dire le schéma unifilaire d'un réseau électrique de puissance avec ses composants.

TWorkSheet s'occupe entre autres de :

- La gestion des éléments du réseau.
- L'édition de ces éléments et de leur données.
- L'échange d'éléments avec un autre **TWorkSheet**.
- La gestion des options.
- L'archivage de la base de données ainsi constituée.
- La préparation des données.
- L'appel des programmes de calculs.
- La récupération des résultats.
- La mise à jour de ces résultats dans les composants.
- L'appel de la récapitulation graphique.
- L'appel de l'Aide...

Il faut rappeler que **TDesigner** et **TWorkSheet** sont des types objet et qu'un objet est une instance d'un type objet. Quand vous ouvrez dans **POWER DESIGNER** 3 documents, vous avez 3 objets (instances) du type objet **TWorkSheet** qui sont gérés par un objet du type objet **TDesigner**.

[2] Par convention, on choisit des identificateurs commançant par "**T**" pour les objets, par "**P"** pour les pointeurs et par "**R**" pour les objets enregistrés (Stream Registration).

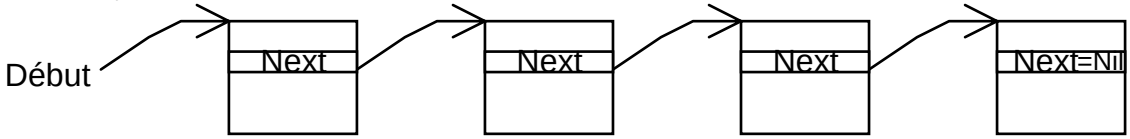


POWER DESIGNER gérant plusieurs documents

TWorkSheet contient un champ nommé **NetWork** qui est un pointeur sur un objet du type **TCollection**. TCollection est un type objet Borland de la classe des **conteneurs**, il gère une **liste chaînée** d'objets. Cette dernière est un concept plus général qu'un tableau. Alors qu'un tableau est de dimension fixe et ne contient qu'un seul type de données, une liste chaînée présente beaucoup plus d'avantages comme le décrit l'exemple suivant :

Soit l'objet TPoint du chapitre I avec un nouveau champ **Next** qui est un pointeur sur un autre objet TPoint :

```
type    PPoint = ^TPoint;
        TPoint = object ( Emplacement)
        Next : PPoint;
        ... { reste de la définition de l'objet de type TPoint }
        End;
```



Structure d'une liste chaînée

Nous pouvons maintenant construire une liste chaînée de ces objets. Il suffit de connaître le pointeur de type PPoint sur le premier objet de type TPoint de notre liste. Cet objet contient dans son champ Next le pointeur sur le deuxième objet de la liste et ainsi de

suite jusqu'au dernier objet de la liste. Ce dernier a son champ Next=**Nil** (Not in List); c'est un mot réservé qui veut dire que le pointeur ne pointe vers rien.

TObject
- **TApplication**
- TWindowsObject
  - TDialog
    - TDlgWindow
    - TPrinterAbortDlg
    - TInputDialog
    - TPrintDialog
    - TPrinterSetupDlg
    - TFileDialog
  - **TWindow**
    - **TMDIWindow**
    - TMDIClient
    - TControl
      - TButton
      - TScrollBar
      - TStatic
        - TEdit
      - TListBox
        - ComboBox
      - TGroupBox
      - TCheckBox
        - TRadioButton
    - TEditWindow
      - TFileWindow
- TScroller
- TStream
  - TEmsStream
  - **TDosStream**
    - TBufStream
  - TMemoryStream
- **TCollection**
  - TSortedCollection
    - TStringCollection
    - TStrCollection
- TPrinter
- TPrintout
  - TEditPrintout
  - TWindowPrintout
- TValidator
  - TPXPictureValidator
  - TFilterValidator
    - TRangeValidator
  - TLookupValidator
    - TStringLookupValidator

Hiérarchie des objets d'ObjectWindows V1.0
(Extrait de l'Aide de Borland Pascal Object 7.0)

L'objet de type TCollection gère une liste chaînée d'objets différents mais parents, c'est-à-dire qu'ils ont un ascendant commun, généralement l'objet fondamental de la

hiérarchie des objets Borland : le type objet **TObject**. D'ailleurs, même TCollection, TWindow, TMDIWindow héritent de TObject comme le montre la hiérarchie de OWL V1.0

TCollection contient, en plus de la liste chaînée, des méthodes permettant de gérer ses éléments, de les détacher, de parcourir la collection à la recherche d'un élément répondant à un certain critère etc...

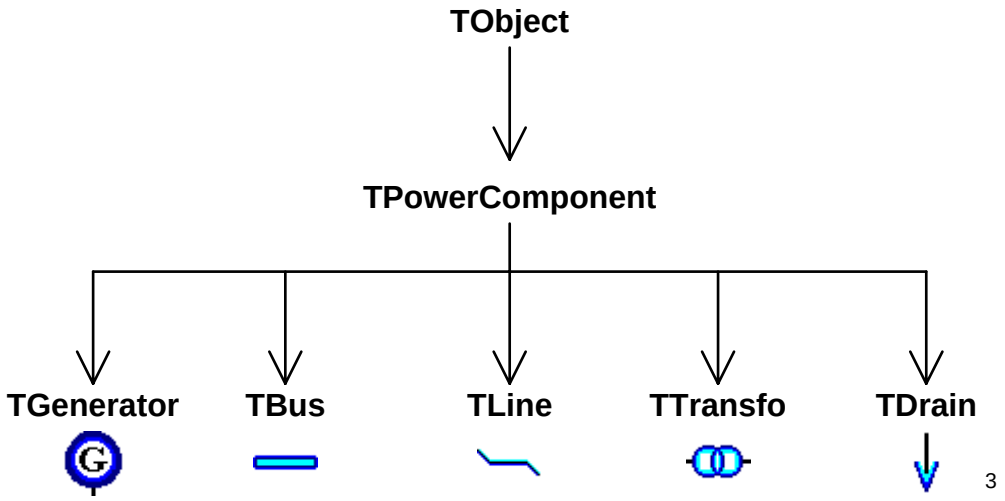
Ainsi, la collection des composants du réseau est pointée par NetWork qui est le champ principal de l'objet de type **TWorkSheet**.

Les études générales les plus importantes des systèmes de puissance concernent l'écoulement de puissance, la stabilité statique, dynamique et transitoire. Ces études nécessitent la modélisation des composants du réseau et définissent les paramètres des modèles.

C'est sur cette base que nous avons élaboré une bibliothèque d'objets (**Power components**) destinés à représenter les composants d'un réseau électrique de puissance.

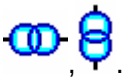
Au début de notre projet, nous avions comme premier objectif de faire des applications sur l'écoulement de puissance. Nous avons donc élaboré la hiérarchie suivante.

Partant du type objet TObject (OWL), nous avons défini un type objet de base le **TPowerComponent** ancêtre de tous les composants de puissance.



[3]

Tous les champs et méthodes communs aux différents composants sont regroupés dans **TPowerComponent** :

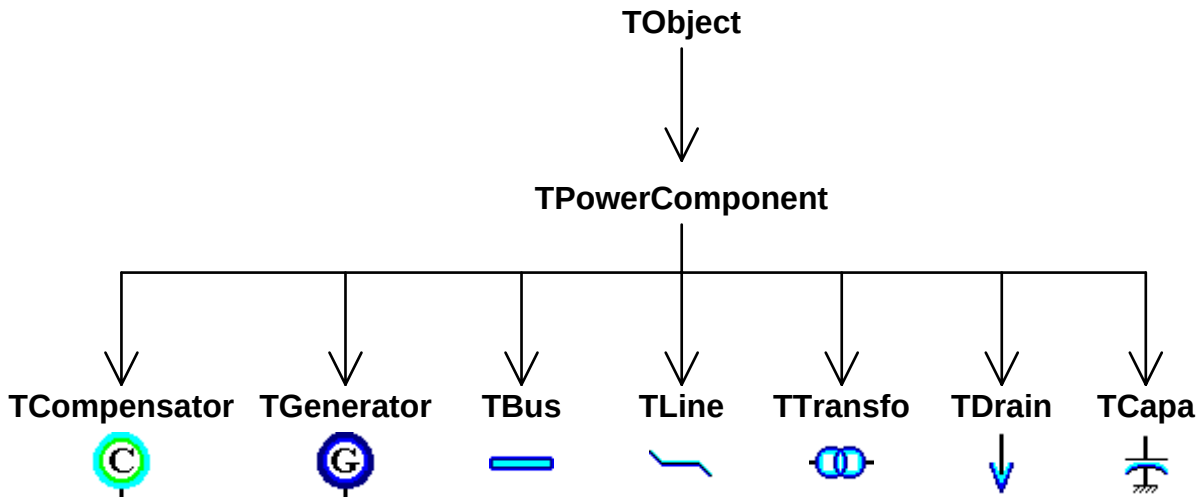
- Le nom du composant de puissance : ALGER220...
- Son type : Generator, Bus, Line...
- Sa position (X, Y : Integer) sur le document **TWorkSheet**.
- Sa Rotation :  .

Les objets descendants du type objet **TPowerComponent** contiennent des champs plus spécifiques. **TGenerator**, qui représente un générateur synchrone, a pour l'écoulement

[3] **TDrain** est utilisé à la place de **TLoad** pour représenter les charges, car Load est utilisé en Borland Pascal pour les accès aux flux (Streams) qui permettent l'enregistrement des champs des objets recensés. Cependant, l'interface utilisateur emploie le mot **Load** pour désigner la charge.

de puissance les champs Pg, Qgmin, Qgmax, Vsch qui sont du type Real et représentent respectivement la puissance active générée, les limites de production de la puissance réactive et la tension spécifiée, le tout en unités relatives (per unit).

Par la suite, nous avons voulu doter notre application d'autres composants : les compensateurs synchrones et les capacités pour la production d'énergie réactive. Il nous a suffi d'ajouter à notre hiérarchie déjà existante les deux type objets **TCompensator** et **TCapa**.



Les avantages de la POO commencent à apparaître avec la gestion de certaines tâches communes à tous les objets. Par exemple, l'objet du type **TLine** ne se dessine pas de la même manière que les autres objets.

Les objets qui ne sont pas des lignes sont représentés par une image (BitMap) peinte avec la fonction **BitBlt** du GDI Windows. Il est alors judicieux de ne placer la méthode **Paint** que dans **TPowerComponent** (déclarée virtuelle bien sûr) et de ne la redéfinir que dans les descendants qui se dessinent autrement, comme **TLine** qui est représentée sur le schéma unifilaire par une succession de traits.

Voici un exemple simple et concret de l'application de la POO et de ses concepts : L'objet du type **TWorkSheet** propriétaire de la collection **NetWork** ne dessine pas lui-même les éléments du réseau actuel. Il "décentralise" les opérations et délègue cette tâche à la collection. **NetWork** appelle alors chacun de ses éléments à travers un pointeur (**MyComponent**) sur l'objet ancêtre **TPowerComponent** et lui demande de se dessiner par :

```
MyComponent^.Paint(...);
```

Grâce au polymorphisme et à la table des méthodes virtuelles (VMT), chaque composant "sait" comment se dessiner; si c'est une ligne, c'est la méthode TLine.Paint qui est appelée, autrement la méthode de l'ancêtre TPowerComponent.Paint sera appelée.

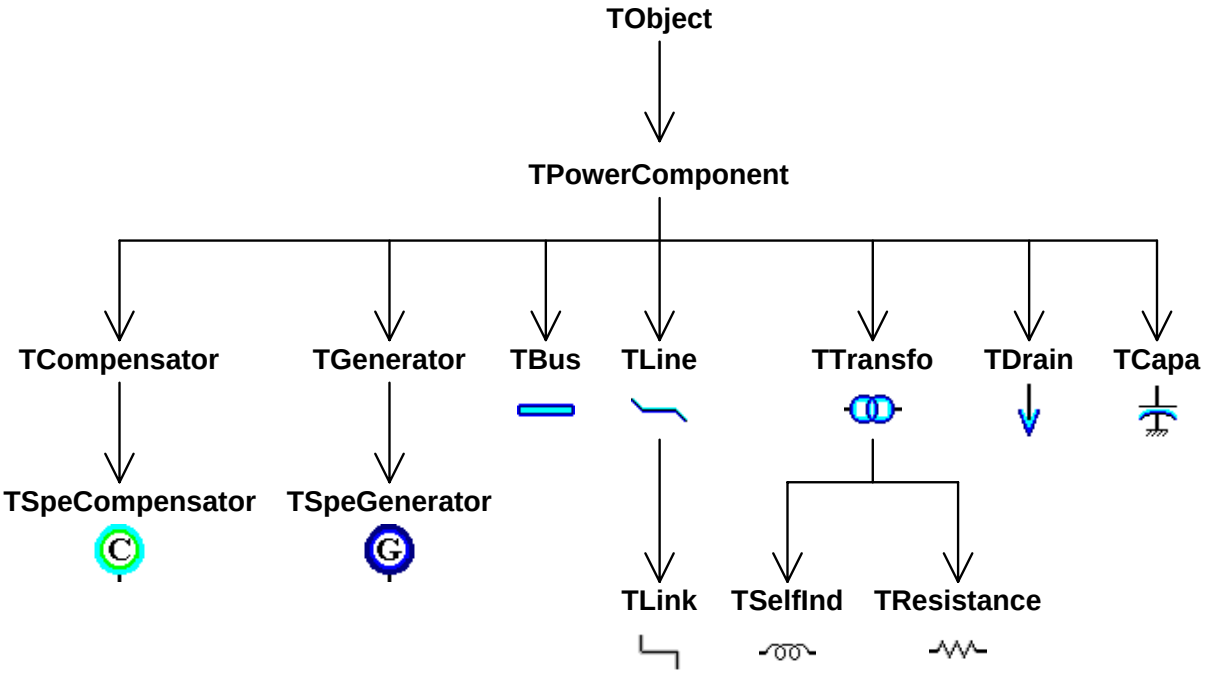
Cet aspect de la programmation permet de subdiviser le programme en unités contenant les déclarations et les définitions des types objet (Power Components) utilisés par d'autres objets (**TWorkSheet**). Une fois la structure établie, ajouter un nouveau composant ne demande pas beaucoup de changements.

Et c'est précisément ce qui s'est passé quand l'implantation des études d'écoulement de puissance a été achevée et que nous avons voulu ajouter des études de stabilité transitoire.

Ces études réclament d'autres paramètres (Ra, X'_d, H, D sur lesquels nous reviendrons dans le chapitre IV) que ceux déjà existant pour l'écoulement de puissance.

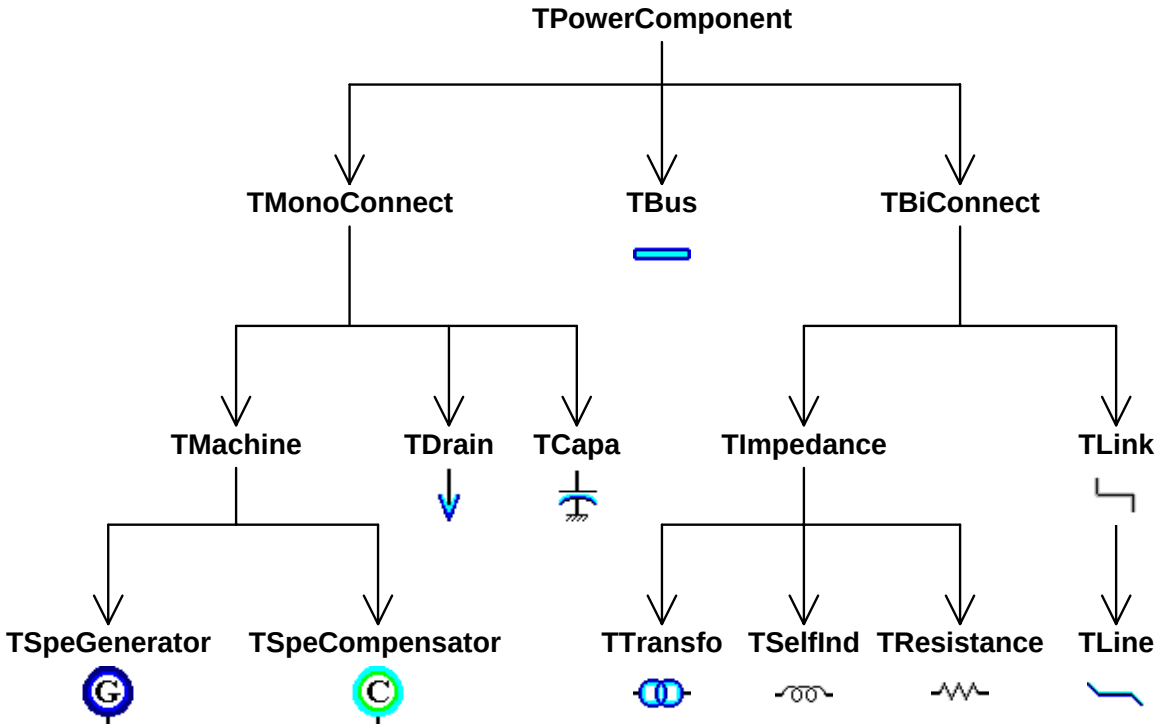
Au lieu de modifier les types objets **TGenerator** et **TCompensator**, il a été plus judicieux de dériver de nouveaux type objets **TSpeGenerator** et **TSpeCompensator** et de leur ajouter les champs de données et les méthodes nécessaires.

Nous avons aussi ajouté de nouveaux composants de puissance à notre hiérarchie; **TResistance**, **TSelfInd** dérivés de **TTransfo** et **TLink** de **TLine**. La hiérarchie finale se présente comme suit :



Remarquons que ce n'est pas la seule hiérarchie qui peut répondre à cette étude. Avec cette nouvelle hiérarchie, nous avons montré qu'il est possible et même aisé d'enrichir ce que nous avons déjà élaboré sans changer les méthodes de base. Cela laisse le champ libre à d'éventuelles extensions au logiciel **POWER DESIGNER**.

Nous proposons même une hiérarchie qui simplifie davantage la gestion des connexions (entre les différents composants).



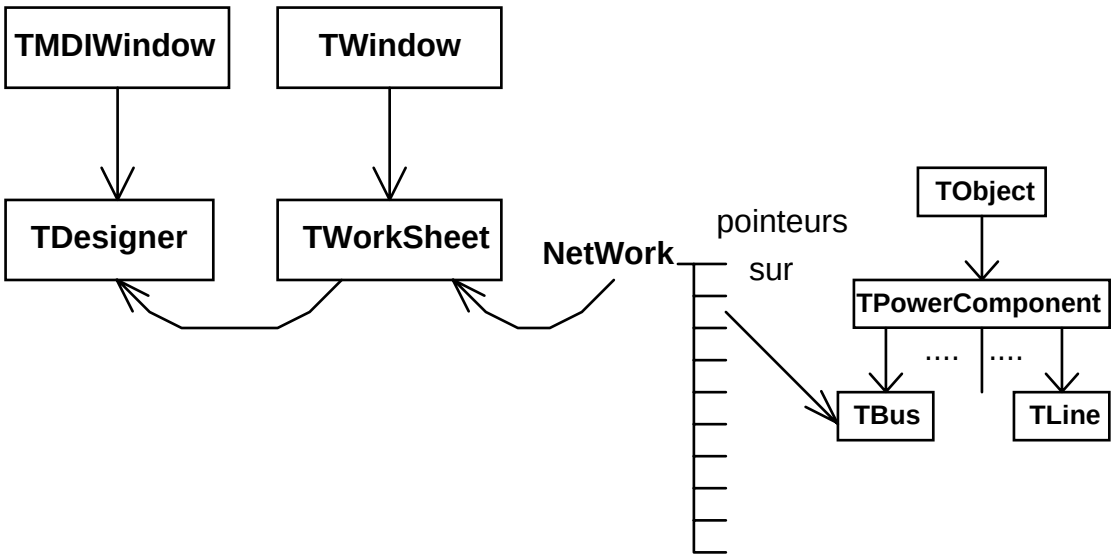
Il ne faut toutefois pas trop abuser de l'héritage car bien que l'on gagne en clarté de structure et de cohérence, la déclaration des méthodes virtuelles (obligatoire pour éviter les effets de bord évoqués au chapitre I) entraîne une augmentation du temps de réponse de l'application due à la recherche dans la VMT de la bonne méthode à appeler.

Le développeur voulant continuer ce travail et exploiter nos .TPW (Unités au format Borland Pascal compilées pour Windows) trouvera en Annexe 1 les différents champs des types objet que nous avons mis au point, les en-têtes des méthodes et leur paramètres telles qu'elles existent dans la version 1.20 de **POWER DESIGNER**.

A titre d'exemple, les deux méthodes du type objet **TWorkSheet** suivantes sont listées en Annexe 3 :

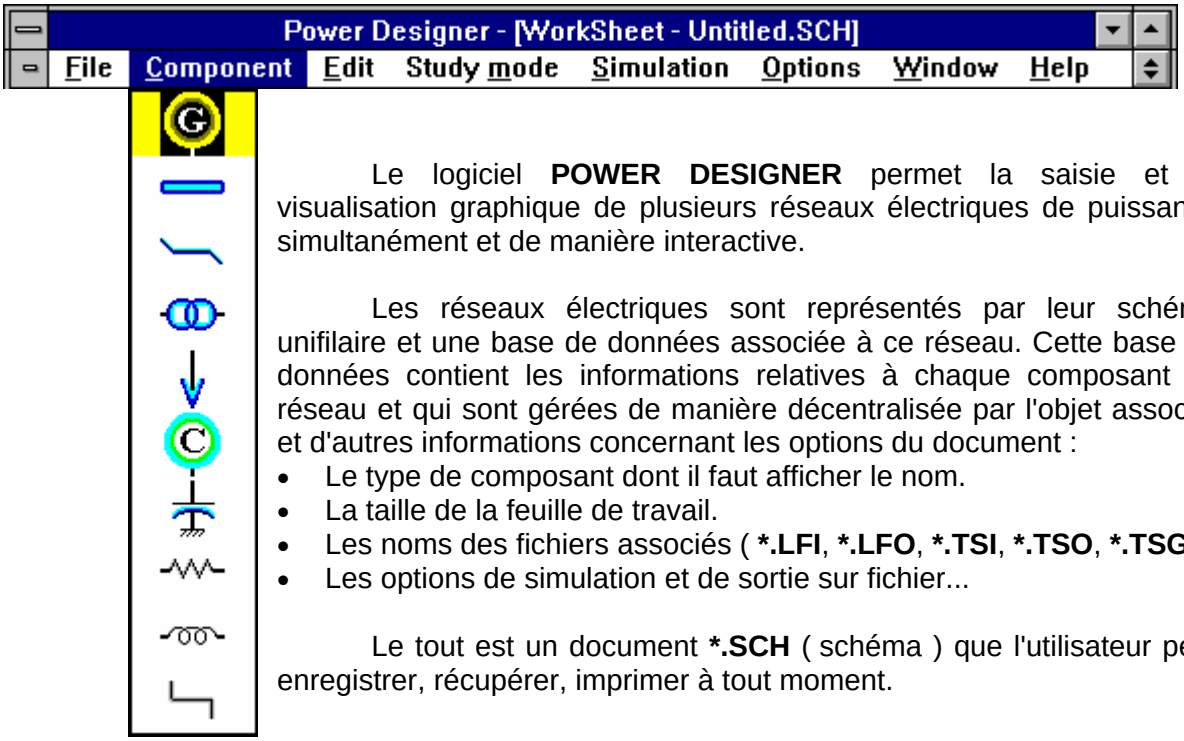
- **LButtonDownAct** : montre ce que fait le programme quand il reçoit le message wm_LButtonDown. Ce message correspond à une pression sur le bouton gauche de la souris. Il est intercepté par la méthode WMLButtonDown qui appelle LButtonDownAct.
  La méthode LButtonDownAct est aussi appelée par la méthode WMKeyDown lorsque les touches Space ou Enter sont sollicitées.
- **MakeData** : correspond à la commande du menu **Simulation | Make Data**. Son action est décrite dans le paragraphe 2.

La figure ci-dessous récapitule la structure générale de **POWER DESIGNER**. Elle n'inclut pas l'objet **TRibbonWindow** qui est une ligne d'état (Status Line) servant à afficher diverses informations quand l'utilisateur accède aux éléments du menu et le mode de simulation actuel (voir l'Aide fournie avec **POWER DESIGNER**).



Récapitulation de la structure générale de POWER DESIGNER

2. Utilisation et possibilités du logiciel POWER DESIGNER

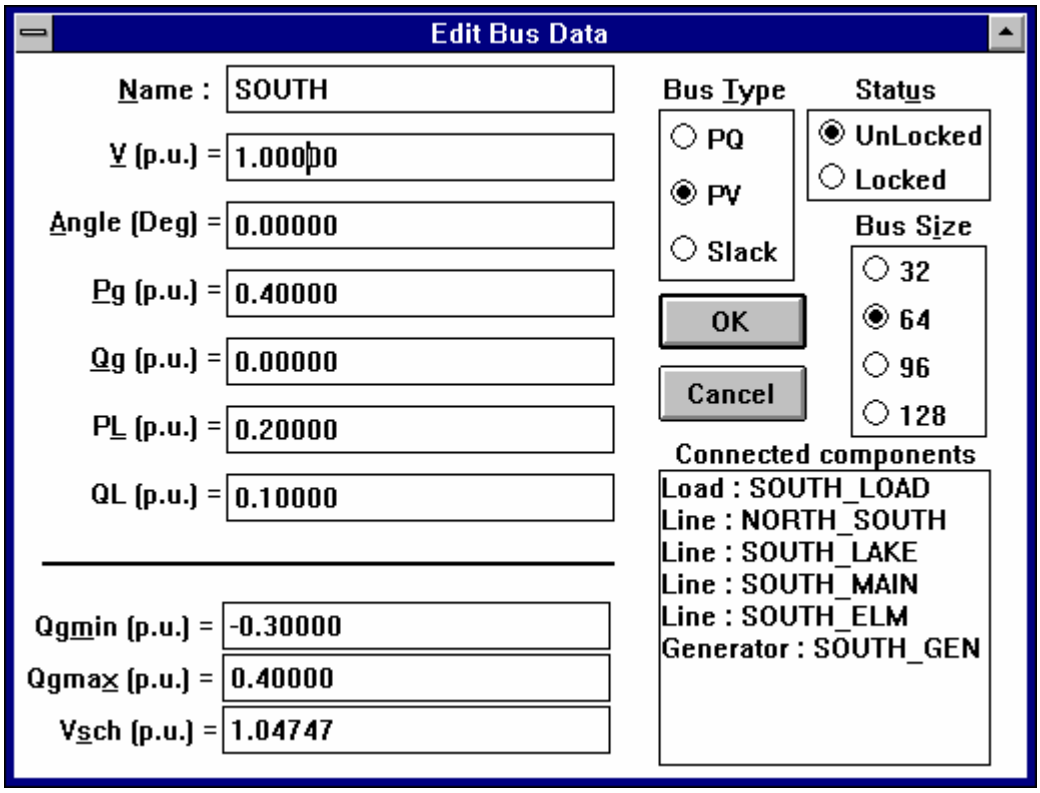


Le logiciel **POWER DESIGNER** permet la saisie et la visualisation graphique de plusieurs réseaux électriques de puissance simultanément et de manière interactive.

Les réseaux électriques sont représentés par leur schéma unifilaire et une base de données associée à ce réseau. Cette base de données contient les informations relatives à chaque composant du réseau et qui sont gérées de manière décentralisée par l'objet associé, et d'autres informations concernant les options du document :

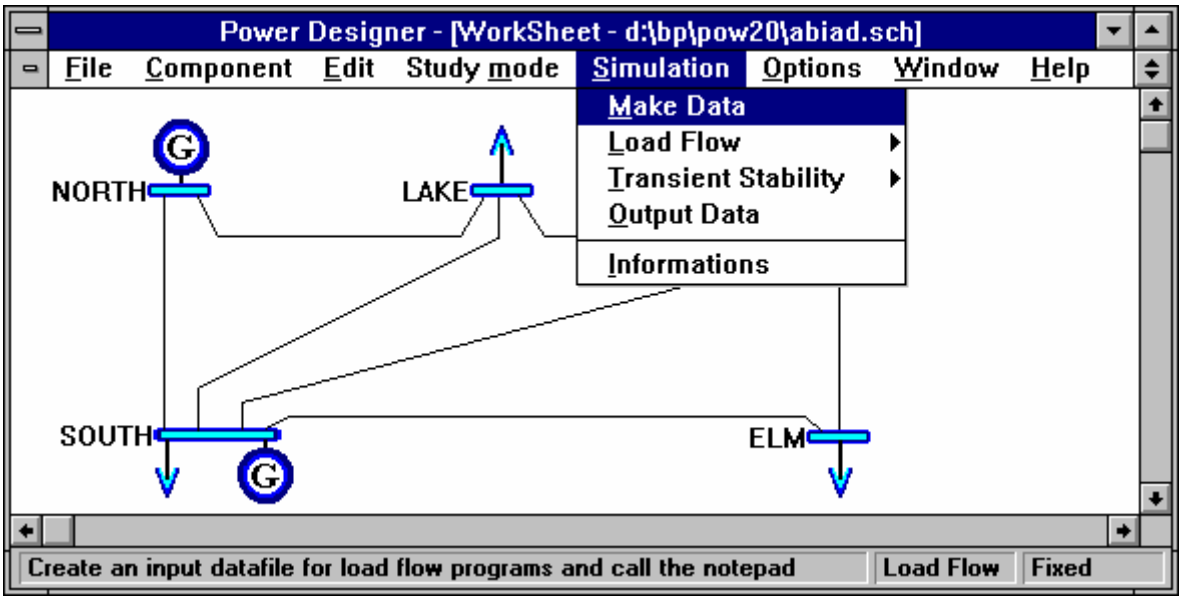
- Le type de composant dont il faut afficher le nom.
- La taille de la feuille de travail.
- Les noms des fichiers associés (***.LFI**, ***.LFO**, ***.TSI**, ***.TSO**, ***.TSG**)
- Les options de simulation et de sortie sur fichier...

Le tout est un document ***.SCH** (schéma) que l'utilisateur peut enregistrer, récupérer, imprimer à tout moment.



Boîte de dialogue associée au composant noeud (Bus)

Une fois le réseau et ses composants saisis, l'utilisateur appelle la commande **Make Data** du menu **Simulation**.



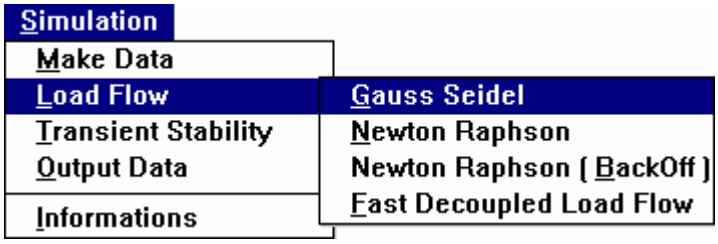
Son action est double. Elle vérifie l'intégrité des données du réseau (si tous les éléments sont nommés, connectés...), puis dresse une liste (la collection BusColl) des noeuds du réseau en commençant par le noeud de référence (Slack Bus) puis les noeuds contrôlés (PV buses) suivis des noeuds de consommation (PQ buses).

Make Data crée ensuite un fichier d'entrée aux programmes de calcul en fonction du mode d'étude (Study Mode) actuel et l'affiche sur l'écran grâce au Bloc Note (Note Pad) de Windows.

POWER DESIGNER dispose actuellement de deux modes d'étude :

2.1. Mode d'étude : écoulement de puissance

Make Data produit un fichier Load Flow Input (**.LFI**).
L'utilisateur a le choix entre 4 méthodes de calcul de l'écoulement de puissance : Gauss-Seidel, Newton-Raphson, Newton-Raphson avec une variante "Back Off" et Fast Decoupled Load Flow (voir le chapitre III).



Le lancement d'une session DOS avec le programme de calcul de l'écoulement de puissance se fait automatiquement à partir du menu. Le programme **PowerD** transmet au programme de calcul sollicité le nom des fichiers d'entrée **.LFI** et de sortie **.LFO**.

Signalons que pendant qu'une session DOS s'exécute, l'utilisateur peut récupérer la main et faire un autre travail en multitâche.

Une fois le calcul fini, l'utilisateur est invité à appeler **Output Data**. Cette commande sort le fichier **.LFO** sur le Bloc Note de Windows et met à jour les grandeurs calculées dans les noeuds du réseau, c'est-à-dire Vmag, delta, Pg (du noeud de référence) et Qg (des noeuds PV). Ceci est primordial pour une éventuelle étude de stabilité transitoire.

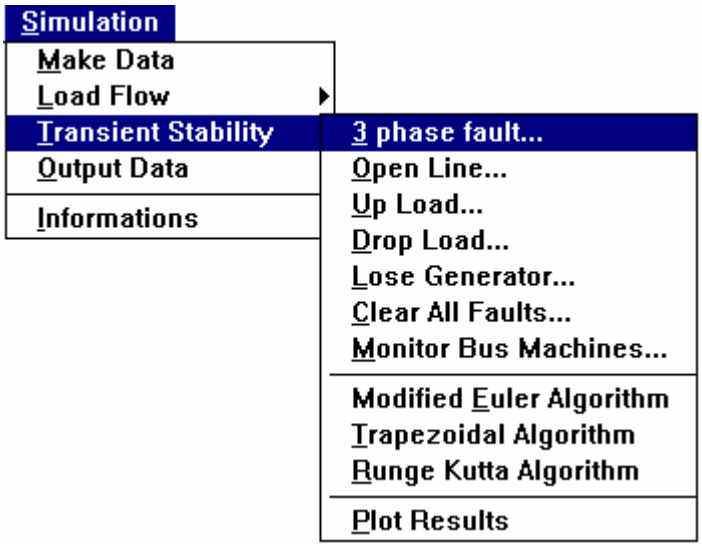
2.2. Mode d'étude : stabilité transitoire

L'étude de la stabilité transitoire vis-à-vis d'une perturbation peut être entreprise après que le point de fonctionnement dans les conditions normales ait été déterminé par une simulation d'écoulement de puissance.

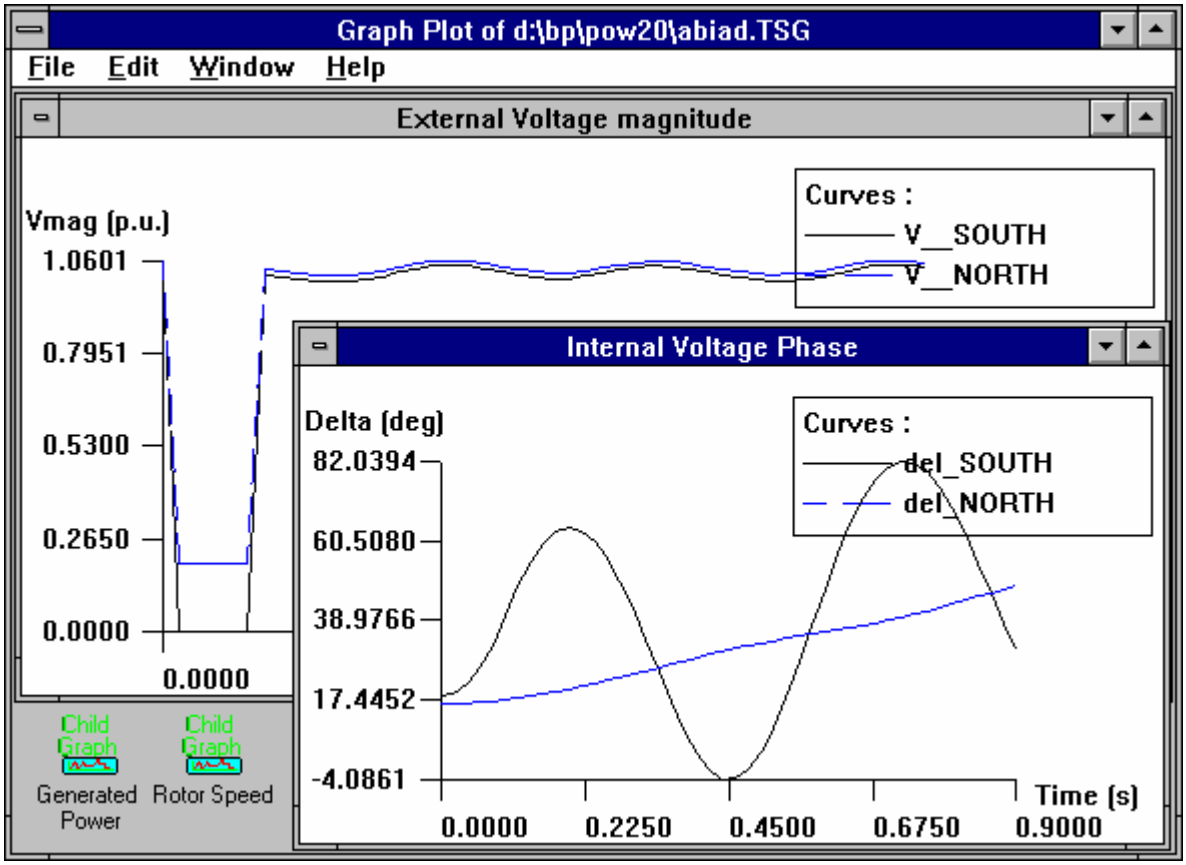
Make Data produit alors un fichier Transient Stability Input (**.TSI**) comportant : les données du réseau, les tensions aux noeuds avant défaut, les paramètres du défaut et les machines à surveiller.

Il faut ensuite appeler l'une des 3 méthodes de calcul proposées (voir le chapitre III).

Après le calcul, l'utilisateur peut sortir le fichier **.TSO** sur le Bloc Note et appeler, par **Plot Results**, le programme de récapitulation graphique **GraphD**.



Ce programme est une application Windows que nous avons développé pour que les utilisateurs n'ayant pas des logiciels comme **Grapher**, puissent au moins visualiser et imprimer les courbes (phase de Eg, Wr, Pg et Vmag en fonction du temps) des machines surveillées lors de la simulation.



GraphD : Programme de récapitulation graphique

Il est à noter que tous les programmes du logiciel **POWER DESIGNER** (PowerD, GraphD, GSP, NRP, NRPB, FDL, TSP, TSPT et RKP) sont indépendants du réseau étudié et s'adaptent de manière dynamique à la dimension du réseau, sans qu'il ne faille changer ou recompiler les programmes comme c'est souvent le cas avec les programmes courants.

POWER DESIGNER est un outil très adaptable, agréable d'emploi et doté de nombreuses options (voir l'Aide fournie avec le logiciel).

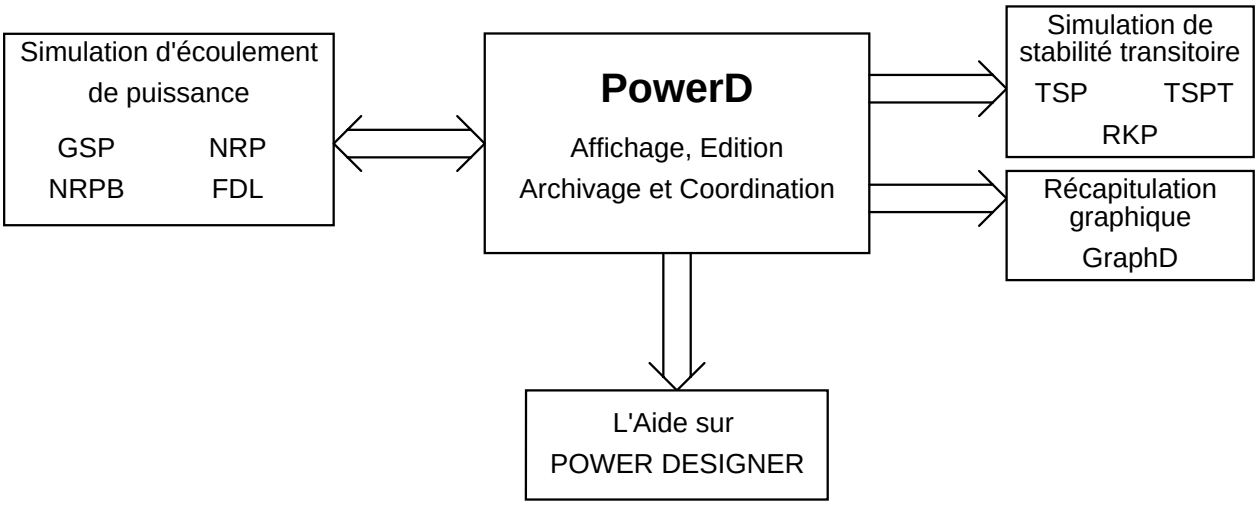


Schéma des échanges d'information entre
les différents programmes de POWER DESIGNER

3. Organisation des liens électriques entre les éléments de la Base de données

La base de données est constituée principalement de la collection (NetWork) des composants de puissance, de la collection (MonitoredBuses) des noeuds à surveiller lors d'une simulation de stabilité transitoire, des options de simulation (Load Flow & Transient Stability) et des options de sortie sur fichier (Output Options).

NetWork est une collection à base de liste chaînée. A chaque élément de cette collection est attribué un numéro : son ordre dans la collection. Cet ordre est unique et c'est celui des éléments entrés successivement par l'utilisateur.

Un composant de type **TBus,** par exemple, doit à tout moment connaître quels sont les composants qui lui sont connectés (générateur, ligne...). Cette information doit aussi lui donner accès aux champs de ces objets. Cela ne servirait à rien de connaître juste le nom du composant connecté. En effet, au moment de générer le fichier de données, il y aura une grande perte de temps à comparer les noms de tous les composants afin de trouver le composant connecté, si toutefois le nom est saisi...

Le choix d'un pointeur comme information de connexion parait idéal. En effet, un pointeur sur l'objet connecté donne accès aux différents champs (P^.Nom, P^.Pg, P^.Qgmin...) et on peut ainsi faire, par exemple, le bilan de toutes les puissances générés au niveau d'un noeud par ses générateurs.

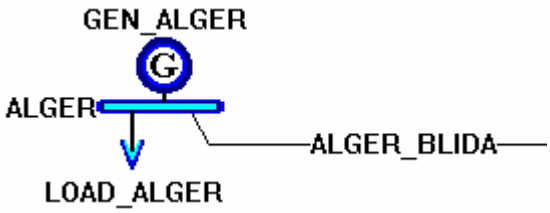
Seulement le pointeur pose des problèmes à l'enregistrement et à la lecture de la collection NetWork à partir du disque comme nous l'exposerons par la suite.

Nous avons eu l'idée d'utiliser l'ordre des composants dans la collection comme information de connexion.

Supposons que l'objet **TBus** de nom Alger soit un noeud auquel sont connectés :

- Un générateur : Gen_Alger.
- Une charge : Load_Alger.
- Une ligne : Alger_Blida.

Ordre des composants	**Type objet**	**Nom**
0	TSpeGenerator	Gen_Alger
1	TBus	Alger
2	TLoad	Load_Alger
3	TLine	ALger_Blida



Nous avons attribué à chaque composant des champs dits de connexion.

- **TGenerator**, **TCompensator** (et donc **TSpeGenerator** et **TSpeCompensator**), **TDrain**, et **TCapa** ont un seul champ : **Con** de type Integer.
- **TLine**, **TTransfo**, **TResistance** et **TSelfInd** ont deux champs : **Con1** et **Con2** de type Integer.

Ces différents champs sont mis à **-1** si la connexion n'est pas établie et à l'ordre du Bus auquel ils sont connectés dans le cas contraire.

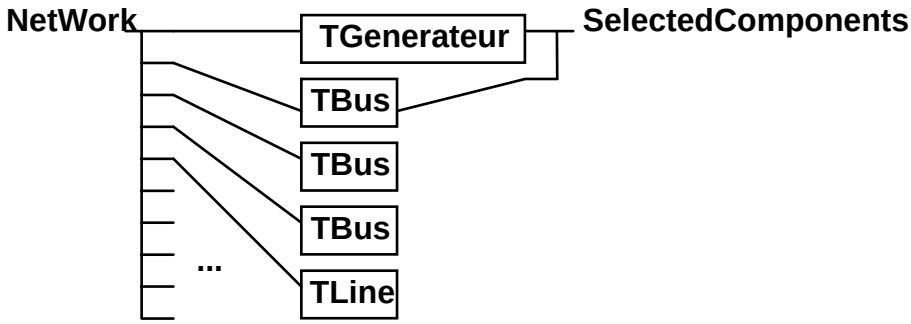
Pour l'objet de type **TBus,** c'est plus compliqué vu que ce composant peut avoir plusieurs composants connectés.
TBus a une collection de pointeurs **ConColl** sur des objets de type **TBusCon** (voir Annexe 1 : Déclaration des types objet).

Quand l'utilisateur sélectionne deux composants et les connecte, la procédure est la suivante :

- L'objet de type **TWorkSheet** propriétaire du réseau NetWork appelle la méthode **Connect** du premier objet de la sélection.
- Si ce premier objet est de type **TBus** :

  Il inverse l'ordre des deux composants dans la sélection (non pas dans NetWork mais dans SelectedComponents: la collection des composants sélectionnés).

  Ensuite il appelle la méthode **Connect** du premier objet de la sélection.



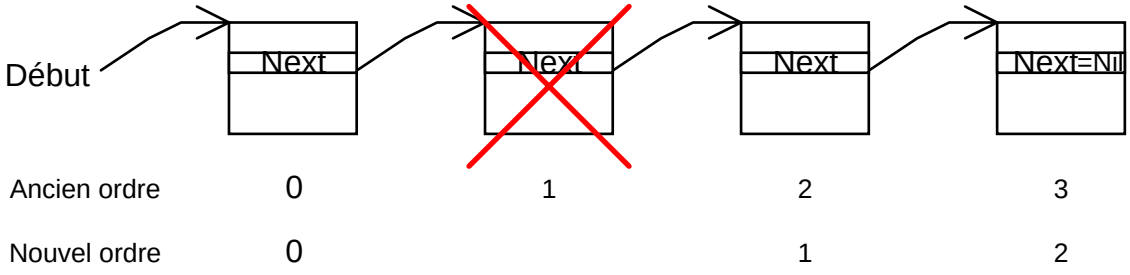
- Le premier objet de la sélection (appelons-le **A**) vérifie si la connexion est physiquement possible. En cas d'erreur, l'utilisateur est averti que la connexion n'est pas possible (Générateur avec Ligne par exemple).
- Si **A** est déjà connecté avec un autre Bus, alors **A** va chercher dans la collection **ConColl** de ce Bus son numéro d'ordre et le supprime de cette collection.
- **A** va alors affecter à son champ **Con** l'ordre de l'objet **B** (le Bus de la sélection).
- **A** crée aussi un nouvel objet de type **TBusCon**, portant l'ordre de **A** et va ajouter le pointeur sur ce **TBusCon** à la collection **ConColl** du Bus **B**.

```
...
P:=SelectedComponents^.At(1);
Num:=NetWork^.IndexOf( @Self);
If Con<>-1 Then NetWork^.FirstThat( @HasEqualPcbNum);
...
Con:=NetWork^.IndexOf( P);
PBC:=New( PBusCon, Init( Num ));
PBus(P)^.ConColl^.Insert(PBC);
```

Cela se complique pour les composants à deux connexions. Le principe reste le même mais il faut prendre en considération l'ordre des connexions.

Cette manière de représenter les liens entre les éléments de la base de données par des champs de connexions nous oblige à effectuer une réindexation de ces champs lors d'une opération "**Delete Selected Components**" que l'utilisateur demande par la commande **Delete** du menu **Edit**.

En effet, en effaçant un élément da la collection NetWork, tous les éléments dont l'ordre est supérieur à l'élément en cours de suppression vont voir leur ordre dans NetWork décrémenté de un.



Il faut donc, avant d'opérer une suppression, mettre à jour les champs connexion des objets de NetWork en conséquence.

De même, les champs spéciaux de **TWorkSheet** doivent aussi être mis à jour. Ces champs correspondent à la collection **MonitoredBuses** et aux composants où un défaut va

être appliqué lors d'une étude de stabilité transitoire (voir Annexe 1 : Déclaration des types objet).

4. Enregistrement et lecture du réseau sur le flux

4.1. Introduction

Le flux est une abstraction logicielle représentant un flot de données entre une source (le producteur) et une cible (le consommateur). Il est utilisé comme canal pour :

- Recevoir de l'information dans le cas d'un flux de sortie.
- Fournir de l'information dans le cas d'un flux d'entrée.

4.2. Utilisation des flux dans POWER DESIGNER

Dans **POWER DESIGNER**, nous utilisons les flux de deux manières :

- Dans **PowerD**, nous employons les flux de l'OWL (Object Windows Library) du Borland Pascal Object 7.0 à travers leur représentant de type objet **TDosStream** (voir la hiérarchie des objets d'ObjectWindows au paragraphe 1. Structure de POWER DESIGNER) afin d'enregistrer et de lire la base de données.
- Dans les modules de calcul sous DOS (GSP, NRP, NRPB, FDL, TSP, TSPT et RKP), nous utilisons les flux du Borland C++ 3.1 par le biais de leurs objets **cout** pour la sortie sur la console par défaut, **ifstream** pour la lecture sur fichier et **ofstream** pour l'écriture sur fichier.

Nous ne décrirons que l'utilisation des flux pour stocker la base de données de **POWER DESIGNER**, le lecteur voulant approfondir l'aspect des flux vu du coté C++ devra se reporter à la référence [2].

L'objet de type **TStream** (et donc son descendant **TDosStream**) est doté de deux méthodes agissant sur des objets descendants du type objet **TObject**. Ce sont les méthodes **Put** et **Get**. Elles permettent l'enregistrement et la lecture des objets sur le flux.

Pour que cette action puisse avoir lieu, il faut :

- Que chaque objet soit d'un type descendant de **TObject** de l'OWL. C'est pour cela que nous étions obligés de commencer notre hiérarchie de Power Components à partir de l'objet ancêtre **TObject**.
- Que le constructeur :

  constructor Load(var S: TStream);

  et la méthode :

  procedure Store(var S: TStream);

  soient définis ou redéfinis dans les descendants de **TObject**, de manière à lire et à enregistrer sur le flux le contenu des champs du type objet concerné.
- Que chaque type objet fasse l'objet d'une déclaration (d'un enregistrement) dont le format est le suivant :

TStreamRec (record) (Objects unit)

Declaration

```
TStreamRec = record
  ObjType: Word;
  VmtLink: Word;
  Load: Pointer;
  Store: Pointer;
  Next: Word;
end;
```

The fields in the stream registration record are defined as follows:

Field	Contents
ObjType	A unique numerical id for the object type
VmtLink	A link to the object type's virtual method table entry
Load	A pointer to the object type's Load constructor
Store	A pointer to the object type's Store method
Next	A pointer to the next TStreamRec

Enregistrement TStreamRec
(Extrait de l'Aide de Borland Pascal Object 7.0)

Exemple de déclaration de **RLine** pour l'objet de type TLine :

```
RLine : TStreamRec = (
   ObjType : 156;
   VmtLink : Ofs( TypeOf(TLine)^);
   Load : @TLine.Load;
   Store : @TLine.Store );
```

Voir aussi l'annexe 2 : Déclaration des enregistrements des flux (Stream Registration).

Afin d'enregistrer la base de données, il convient d'ouvrir un fichier par:

```
TheFile.Init(FileName, stCreate);
```

où TheFile est un objet de type **TDosStream**, puis d'y inscrire les champs de **TWorkSheet** :

```
...
TheFile.Write(WSInfo.WSOpt, SizeOf(WSInfo.WSOpt));
TheFile.Write(LFOpt, SizeOf(LFOpt));
TheFile.Write(TSOpt, SizeOf(TSOpt));
TheFile.Write( WSInfo.TSInfo, SizeOf( WSInfo.TSInfo));
...
```

et d'y enregistrer la collection NetWork :

```
TheFile.Put(NetWork);
```

Cette instruction provoque la recherche du type d'enregistrement RType (type registration record) en comparant l'offset de la VMT avec le champ VmtLink de tous les objets déclarés à être enregistrés.

Ensuite elle inscrit le code identificateur ObjType et appelle la méthode **Store** de l'objet NetWork.

NetWork^.Store(TheFile) appelle alors pour chaque élément de la collection NetWork, la méthode TheFile.Put. Cette dernière exécute la séquence précédente mais cette fois sur l'élément de la collection : le **TPowerComponent**. Lui-même, s'il dispose d'autres objets (comme le **ConColl** de **TBus**) va leur demander de s'enregistrer sur le flux et ainsi de suite.

La lecture se passe de la même manière, sauf que dans ce cas ce seront les méthodes **Get** et **Load** qui seront appelées.

Mentionnons ici un point important. La méthode de lecture (**Load**) de l'objet doit être obligatoirement un **constructeur**. En effet, seul un constructeur permet d'initialiser un objet en lui créant la table des méthodes virtuelles (VMT) par laquelle la ligature dynamique est possible.

Nous avons évoqué dans le paragraphe 3 (Organisation des liens électriques entre les éléments de la Base de données) le problème que pouvait poser l'utilisation des pointeurs en guise d'information de connexion.

En effet, au moment d'enregistrer le champ de l'objet, le programme fournit une adresse mémoire comme donnée à enregistrer. Or, la carte des allocations mémoire à cet instant sera différente de celle d'une lecture ultérieure du NetWork à partir du disque.

A la lecture, une adresse non valide sera lue et le noyau Windows commandera l'arrêt pur et simple de l'application car elle risque de perturber les autres applications par ses pointeurs qui pointent vers autre chose que des objets (des segments de code par exemple).

4.3. Avantages et inconvénients de l'utilisation des flux comme canal de stockage

Le principal avantage est le même que l'on retrouve partout en Programmation Orientée Objets : "la décentralisation des actions". La collection NetWork demande à ses éléments de s'enregistrer sur le flux qu'elle leur passe comme paramètre. Chacun s'enregistre alors d'une manière qui lui est propre.

De plus, les données sont enregistrées sous forme binaire. Le fichier est par conséquent plus compact que si l'enregistrement était basé sur une forme texte.

Ceci pose néanmoins un problème de compatibilité entre les versions. Si une version future du logiciel ajoute de nouveaux champs, il faudrait qu'elle prévoie un filtre de conversion pour pouvoir accéder aux fichiers actuels.

Nous avons placé en en-tête des fichiers **.SCH** (schéma unifilaire et base de données) un identificateur de numéro de version afin que le filtrage puisse se faire aisément.

En conclusion, nous remarquons que les flux offrent des avantages immenses en matière d'échange d'information. Ils s'occupent du formatage et des tâches de bas niveau d'archivage de l'information.

5. Fonctions "couper / coller" entres les documents

5.1. Introduction

L'environnement Windows offre un outil très puissant en matière d'échange de données.

Cet outil se nomme le presse-papiers (ClipBoard). Il permet de couper un morceau de texte (dans les documents qui gèrent du texte), de le dupliquer et de le coller ailleurs, dans une autre application.

Le presse-papiers peut gérer plusieurs formats de données : textes, textes OEM, images BitMap, images au format TIFF, Metafile, sons au format WAVE et même les formats de données privés.

5.2. Elaboration de "couper / coller" dans POWER DESIGNER

Nous avons voulu doter **POWER DESIGNER** des possibilités de "couper / coller" (copy / paste) des parties du réseau (ensemble de composants) dans le même document **TWorkSheet** et avec d'autres documents de la fenêtre MDI **TDesigner**.

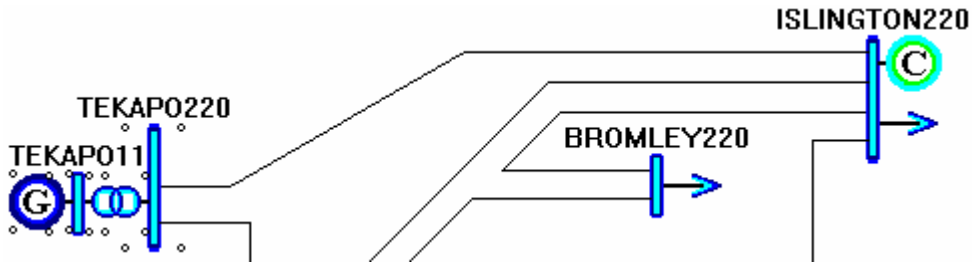
Le presse-papiers de Windows réclame des données allouées et bloquées dans le "Global Heap" à l'aide de la fonction Windows **GlobalAlloc**. Cette fonction demande des données contiguës de la forme **A**, sizeof(**A**).

Or, si nous voulons copier la collection SelectedComponents et ses éléments, il faudra la dupliquer et donner au presse-papiers le Handle (pointeur Windows) sur l'allocation du Global Heap. Malheureusement, la collection de part sa structure, ne tient pas en un seul bloc. Ses éléments sont éparpillés dans la mémoire et ne sont pas contigus. Nous n'avons que les pointeurs sur ces éléments, nous ne pouvons donc pas utiliser le presse-papiers de Windows avec cette méthode.

Nous avons alors choisi d'utiliser un fichier d'échange sur disque pour le stockage temporaire des données.

Une fois ce problème résolu, survient celui de l'indexation et des champs de connexion.

En effet, pour que les fonctions "couper / coller" de la sélection soient intéressantes, il faut que le programme détache électriquement (déconnecte) les éléments qui ne sont pas dans cette sélection et préserve les connexions entre les éléments de la sélection.



Les connections en amont du noeud TEKAPO220 doivent être préservées

Seulement, lors du collage, les champs (Con, Con1...) des connexions vont porter sur les indexes (ordres) de la collection NetWork du document source. Il faut donc les réajuster au niveau de NetWork du document destination.

Ce réajustement se fait de la manière suivante (sur pas moins de 243 lignes de programme) :

1. A l'aide du **TDosStream**, nous créons une copie conforme à la collection SelectedComponents et tous ses composants afin de ne pas perturber les connexions d'origine. Nous chargeons la nouvelle collection dans **TempColl**.

2. Nous mettons à jour tous les éléments de TempColl :
   - Si l'élément est connecté avec un élément faisant partie de la sélection, il faudra alors réindexer son champ selon l'ordre des composants dans TempColl.
   - Si l'élément est connecté avec un élément ne faisant pas partie de la sélection, il faudra alors déconnecter l'élément dans TempColl.

3. Nous enregistrons ensuite la collection TempColl et ses éléments dans un fichier (PowerD.CBD pour ClipBoard Data).

4. Si l'opération est "**copier**" (Copy : Ctrl+Insert) alors sortir.
   Si l'opération est "**couper**" (Cut : Shift+Delete) alors supprimer les composants faisant partie de la sélection de NetWork.

5. Nous libérons TempColl.

A la restitution, "**collage**" (Paste : Shift+Insert) :

1. Nous lisons la collection TempColl du fichier PowerD.CBD.

2. Nous ajoutons les éléments de TempColl à la collection NetWork destination, composant par composant, en réindexant les champs connexion de manière à avoir un ordre correspondant à la nouvelle collection NetWork.

3. Nous libérons TempColl.

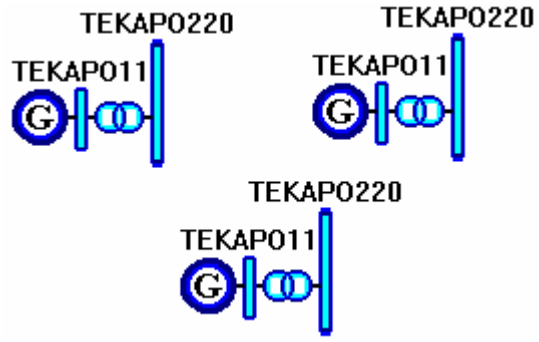
Plusieurs collages successifs sont alors possibles et on retrouve en transparence totale pour l'utilisateur, un presse-papiers interne pour les documents **TWorkSheet**.

5.3. Avantages et utilisations

- Supposons que nous ayons plusieurs stations de génération comprenant chacune un générateur, un noeud basse tension, un transformateur élévateur de tension et un noeud haute tension connecté au réseau (voir la figure ci-dessous).

  Il suffit de saisir cette portion de réseau une fois, de connecter électriquement ses composants, de les sélectionner puis de les copier dans notre presse-papiers.

  Pour les dupliquer, il suffit d'exécuter autant de fois que nécessaire la commande coller.



Duplications successives d'un même groupe de composants

- La deuxième application est encore plus intéressante lors de l'étude de l'interconnexion des réseaux :

  Soit un réseau **A**, que nous avons saisi et sur lequel nous effectuons un calcul d'écoulement de puissance avec des paramètres spécifiés (Vsch, Pg...).

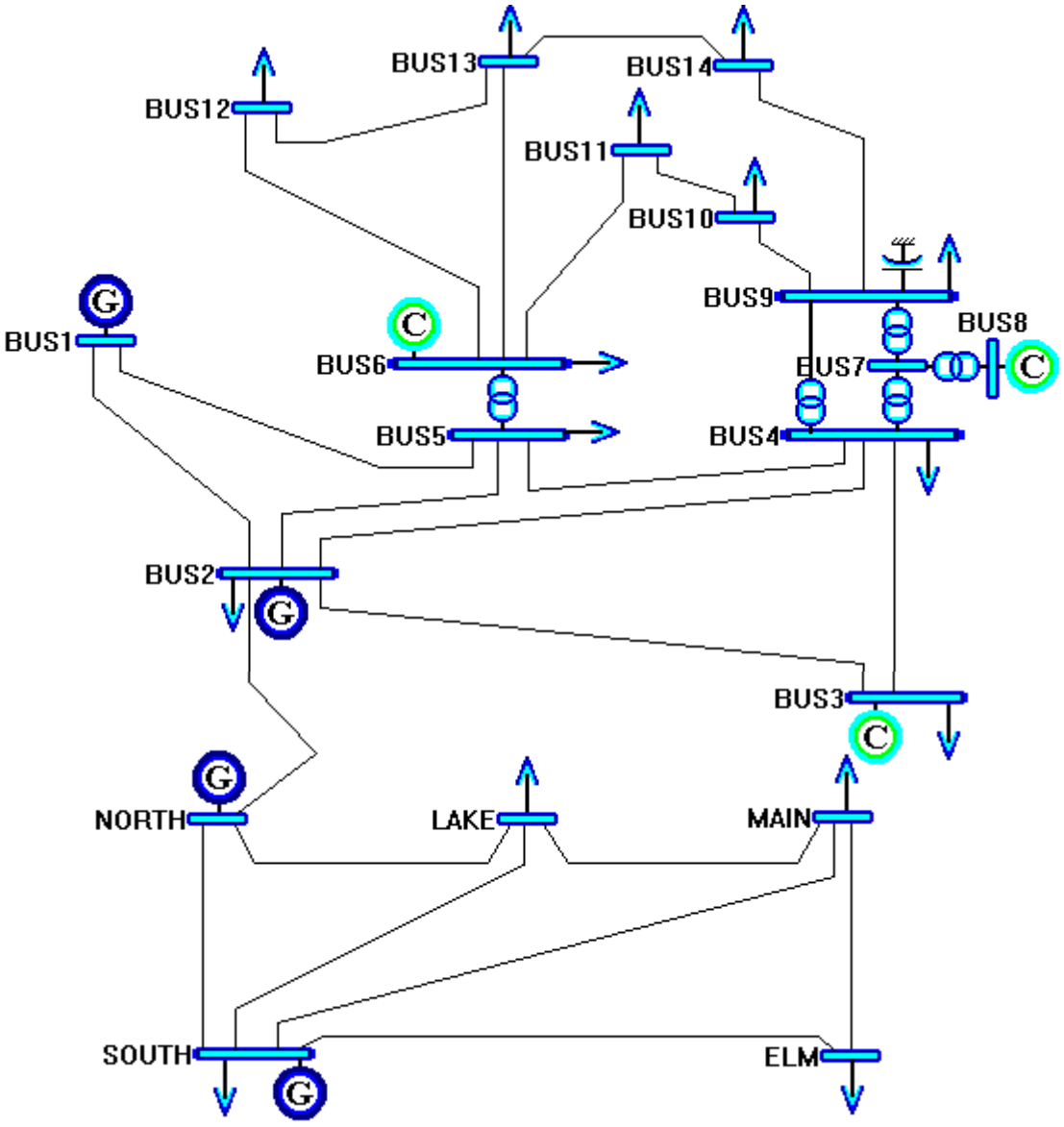
  Soit un autre réseau **B**, sur lequel nous avons fait des simulations similaires et nous avons obtenus pour chacun de ces réseaux un point de fonctionnement satisfaisant.

  Nous voulons maintenant étudier le réseau **C** formé par les deux réseaux **A** et **B** interconnectés par certaines lignes.

  Il suffit alors de copier les réseaux **A** et **B** dans un nouveau document **TWorkSheet** puis d'ajouter les composants d'interconnexion.

- L'étude de parties d'un réseau est aussi possible par cette méthode.

Les commandes "couper / coller" ajoutent un caractère professionnel à l'environnement d'édition de **POWER DESIGNER** afin de mieux répondre aux normes Windows en matière d'interface utilisateur.



Exemple d'interconnexion entre le réseau IEEE 14 noeuds et le réseau à 5 noeuds de la référence [7]

Chapitre III : Ecoulement de Puissance

1. Introduction

L'étude de l'écoulement de puissance (load flow) permet d'avoir la solution des grandeurs d'un réseau électrique en fonctionnement normal équilibré en régime permanent. Ces grandeurs sont les tensions aux noeuds, les puissances injectées aux noeuds et celles qui transitent dans les lignes. Les pertes et les courants s'en déduisent.

Les études de l'écoulement de puissance permettent de planifier la construction et l'extension des réseaux électriques ainsi que la conduite et le contrôle de ces réseaux.

Un grand nombre de mathématiciens, d'informaticiens et d'ingénieurs ont consacré des années de leur carrière à étudier les méthodes de calcul de l'écoulement de puissance. Il n'y a qu'à voir le nombre de publications dans ce domaine pour apprécier l'effort qu'ils ont développé.

Avant 1929 [10], les calculs de load flow (appelé power flow) se faisaient à la main. En 1929, des calculateurs de réseaux (de Westing House) ou des analyseurs de réseaux (de General Electric) furent employés pour les calculs d'écoulement de puissance. Ce n'est qu'en 1956 que fut développée la première méthode adéquate par Ward et Hale [11].

Les premières méthodes étaient basées sur la méthode itérative de Gauss-Seidel relative à la matrice admittance Y. Elle ne nécessite pas beaucoup d'espace mémoire et sa programmation est relativement simple. Mais, si les petits réseaux ne nécessitent que peu d'itérations pour converger, les grands réseaux, par contre, demandent un grand nombre d'itérations si toutefois ils convergent. Ce qui poussa les chercheurs à développer les méthodes basées sur la matrice impédance Z. Mais même si ces dernières avaient de meilleures caractéristiques de convergence, elles ont comme désavantage de nécessiter beaucoup d'espace mémoire dû au fait que la matrice Z n'est pas éparpillée ("sparse"), contrairement à son inverse Y.

Ce qui amena les chercheurs à développer la méthode de Newton-Raphson. Cette méthode nécessite plus de temps par itération que celle de Gauss-Seidel, alors qu'elle ne demande que quelques itérations même pour les grands réseaux. Cependant, elle requiert des capacités de stockage ainsi que des puissances de calcul importantes.

D'autres techniques sont apparues: celles qui exploitent l'éparpillement de la matrice Y et du Jacobien afin d'optimiser le stockage et celles qui tiennent compte du fort couplage entre P et Θ et entre Q et V (Fast Decoupled Load Flow) afin d'accélérer le temps de calcul.

2. POWER DESIGNER

Dans le cadre de notre travail, nous avons développé un ensemble de programmes que **POWER DESIGNER** appelle pour faire le calcul de l'écoulement de puissance.

Ces programmes sont externes au programme **POWER DESIGNER** pour plusieurs raisons :

- Le programme **POWER DESIGNER** est déjà assez important et il est difficile de gérer les programmes d'un seul bloc. La tendance actuelle est au modulaire, elle permet un entretien et une mise à jour plus facile.

- Les programmes de calcul (ceux de l'écoulement de puissance n'y échappent pas) ont besoin d'une bibliothèque mathématique importante et fournie. Ni le Pascal, ni même le Fortran ne peuvent rivaliser avec le C++ et ses bibliothèques multi-usages.

- Le C++ permet la création et la manipulation des tableaux dynamiques (dont la dimension est fixée non pas à la compilation mais à l'exécution). Cela rend le programme indépendant du réseau étudié. L'allocation dynamique permet la gestion optimale de la mémoire disponible.
  En effet, avec le modèle de compilation **Huge** du C++, l'espace dynamique maximum allouable est de 1 MO, alors qu'il n'est que de 64 KO pour les données statiques. Ce qui permet de faire fonctionner des réseaux allant jusqu'à 63 noeuds sur un PC et ceci sans même utiliser les techniques de gestion d'éparpillement (sparsity).

- Nous aurions aimé fournir des routines de calcul à lien dynamique pour Windows (les **DLL** : Dynamic Link Library) mais le BC++ 3.1 nous a généré des applications instables sous Windows.
  Nous avons alors compilé tous les modules de calcul sous DOS et **POWER DESIGNER** les appelle de Windows dans une session DOS d'une manière complètement transparente à l'utilisateur.

  Cela offre néanmoins l'avantage de pouvoir utiliser ces programmes de manière double. Ainsi, en TP par exemple, un utilisateur peut disposer d'une machine puissante sur laquelle est installé Windows afin de gérer **POWER DESIGNER** et ses modules. D'autres utilisateurs sur des PC ne disposant pas de Windows peuvent bénéficier (une fois que **POWER DESIGNER** a généré les fichiers .LFI) des programmes DOS et ils obtiennent des fichiers de sortie en format texte comme les programmes classiques d'écoulement de puissance.

3. Modèles et hypothèses simplificatrices

3. 1. Résistances

Une résistance est un composant de puissance reliant deux noeuds avec une résistance série de valeur **R**.

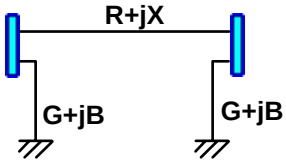
3. 2. Inductances

Une inductance est un composant de puissance reliant deux noeuds avec une inductance série de valeur **X**.

3. 3. Lignes

Les lignes sont représentées par leur schéma équivalent en Π.
Les grandeurs associées sont :
- L'impédance série R+jX.

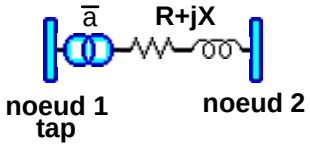
- L'admittance shunt à chaque bras G+jB.
- La puissance maximale de transit. Si la puissance de transit est supérieure à cette limite, un avertissement (Warning) est généré.



3. 4. Transformateurs

Les transformateurs sont représentés par leur matrice admittance.
Les grandeurs associées sont :

- Le rapport de transformation **a=Amag∠Adelta** qui peut être complexe ou non.
- L'impédance de fuite Z=R+jX.
- La puissance maximale de transit.



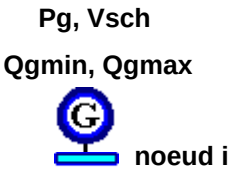
Dans le cas où l'argument de **a** n'est pas nul, un schéma équivalent en Π n'existe pas. Nous représentons le transformateur par sa matrice d'admittance qui est dans ce cas non symétrique ($Y_{21} \neq Y_{12}$). De ce fait, il nous faut stocker toute la matrice Y et non pas juste sa partie triangulaire supérieure.

$$\begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{Z \cdot |a^2|} & -\frac{1}{Z \cdot a^*} \\ -\frac{1}{Z \cdot a} & \frac{1}{Z} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix}$$

Voici comment est implémenté le transformateur dans la matrice Y :

```
/* Soit Rs0, Xs0, Amag0, Adelta0 les paramètres
du transformateur reliant le bus i1 ( Tap ) au bus i2 */

a0=polar( Amag0, Adelta0*M_PI/180);
if ( Rs0 !=0 || Xs0 !=0 )  y0=1/complex( Rs0, Xs0);
                           else y0=complex(0,0);

Y[i1*N1+i1]+=y0/a0/conj(a0);
Y[i2*N1+i2]+=y0;
Y[i1*N1+i2]-=y0/conj(a0);
Y[i2*N1+i1]-=y0/a0;
```

3. 5. Générateurs



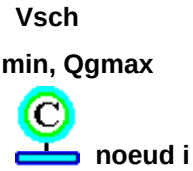
Un Générateur est représenté par une source de tension constante **Vmag=Vsch** qui injecte au niveau du noeud auquel il est connecté une puissance active **Pg** et réactive **Qg**.
Pg est constante durant tout le calcul, **Qg** par contre varie afin de maintenir **Vmag=Vsch**. Si **Qg** atteint une des deux limites (**Qgmin** ou **Qgmax**), elle se fixe à cette limite et **Vmag** se libère.

3. 6. Charges

La charge est modélisée par une impédance qui consomme une puissance active **PL** et réactive **QL** constante.

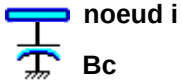
3. 7. Compensateurs synchrones

Un compensateur synchrone est un générateur de puissance réactive. Il est représenté par un générateur avec **Pg=0**.



3. 8. Compensateurs statiques

Le compensateur statique simple est modélisé par une capacité shunt dont l'admittance est **jBc=jCW** .



4. Les programmes

Nous avons développé 4 programmes en C++ pour le calcul de l'écoulement de puissance :

- **GSP** : Méthode de Gauss-Seidel en coordonnées rectangulaires.
- **NRP** : Méthode de Newton-Raphson en coordonnées polaires.
- **NRPB** : NRP avec une variante permettant aux noeuds PV devenus PQ lors d'un dépassement des limites de Qg, de redevenir PV si Qg revient à l'intérieur des limites spécifiées.
- **FDL** : Méthode du Fast Decoupled Load Flow.

Ces programmes sont appelés par **POWER DESIGNER** qui leur passe, comme paramètres de ligne de commande, le nom des fichiers d'entrée .LFI et de sortie .LFO.

La structure du fichier d'entrée .LFI est basée sur le modèle de la référence [15]. Notre modèle est à format libre, les champs de données doivent être séparés par des espaces ou des tabulations et chaque carte a un numéro d'identification spécifique (voir Annexe 3 : méthode MakeData du type objet TWorkSheet).

L'utilisateur de **POWER DESIGNER** à partir de Windows n'a pas à se préoccuper de ces cartes puisque la commande **Make Data** les génère elle même dans le format adéquat.

L'en-tête du fichier .LFI comporte les paramètres suivants :

- Le nombre de noeuds du réseau.
- Le nombre de noeuds PV (sans le noeud de référence) du réseau.
- Le nombre maximal d'itérations autorisées ainsi que la tolérance à assurer :
  - Pour la méthode de Gauss-Seidel, la tolérance est basée sur l'écart des tensions entre 2 itérations successives.

♦ Pour les autres méthodes, la tolérance est basée sur le mismatch de puissance (bilan des puissances injectées à partir d'un noeud dans le réseau) à chaque itération.

4. 1. Méthode de Gauss-Seidel (GSP)

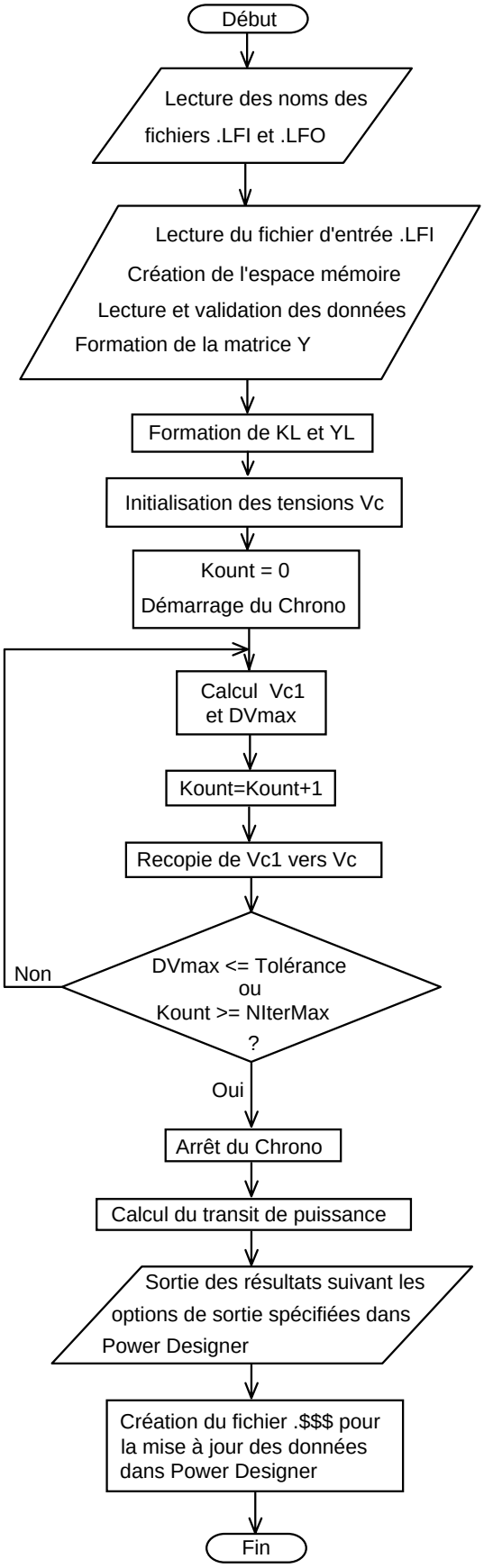
Le programme suit l'algorithme classique de la méthode de Gauss-Seidel avec une variante traitant les dépassements de puissance réactive Qg générée. De plus, la gestion dynamique de l'espace mémoire a requis certaines modifications.

Après avoir lu l'en-tête du fichier, le programme **GSP** demande l'allocation des tableaux[4]. Si l'allocation échoue par manque de mémoire, le programme s'arrête et affiche le message d'erreur correspondant.

GSP utilise le facteur d'accélération saisi dans la boîte de dialogue **Load Flow Options** de **POWER DESIGNER**. Chaque réseau a "son" facteur d'accélération optimal qui minimise le nombre d'itérations nécessaires à la convergence et donc minimise le temps de calcul. Ce facteur se situe généralement entre 1.4 et 1.8 [7].

Pour le calcul de **Vc1** (voir organigramme), le programme vérifie pour les noeuds PV dont les limites de production de réactif **Qgmin** et **Qgmax** ont été spécifiées, si **Qg** dépasse ces limites. Si c'est le cas, le noeud est basculé PQ et le module de la tension n'est plus fixé à **Vsch**.

[4] les matrices ont été gérées comme des tableaux puisque le Borland C++ ne peut allouer dynamiquement que des tableaux.



Organigramme de GSP

4. 2. Méthode de Newton-Raphson (NRP)

L'approche de lecture et de réservation d'espace mémoire est la même pour tous les programmes de calcul développés dans ce projet.

Notre principal apport dans la méthode de Newton-Raphson réside dans la manière dont nous formons le Jacobien.

Etant donné que nous traitons le cas de dépassement des limites de la puissance réactive générée, l'algorithme que nous utilisons est plus compliqué que l'algorithme standard [8].

Le système d'équations à résoudre est :

$$J \cdot \begin{bmatrix} \Delta\delta \\ \Delta V \end{bmatrix} = -\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} \quad \text{avec} \quad J = \begin{bmatrix} \frac{\partial \Delta P}{\partial \delta} & \frac{\partial \Delta P}{\partial V} \\ \frac{\partial \Delta Q}{\partial \delta} & \frac{\partial \Delta Q}{\partial V} \end{bmatrix}$$

Comme **Vmag** est fixé à **Vsch** pour les noeuds PV, il y a une partie du Jacobien (colonnes de N+1 à $N+N_{PV}$) qui ne doit pas exister puisqu'il n'y a pas de variations suivant **Vmag**. De même, les lignes du Jacobien de N+1 à $N+N_{PV}$ ne doivent pas exister puisque pour un noeud PV, **Qg** n'est pas fixée.

Au lieu de comprimer le Jacobien[5] de Npv (lignes et colonnes), il suffit d'affecter à cette zone des valeurs nulles sauf à la diagonale principale du Jacobien que nous mettons à l'unité. Nous annulons également les valeurs du vecteur mismatch pour i=N+1 à $N+N_{PV}$.

De manière générale, nous pouvons écrire le vecteur mismatch sous la forme :

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \begin{bmatrix} 0 \\ \Delta P_1 \\ . \\ \Delta P_N \\ \hline 0 \\ 0 \\ . \\ \Delta Q_{Npv+1} \\ . \\ \Delta Q_N \end{bmatrix}$$

et la matrice jacobienne sous la forme suivante :

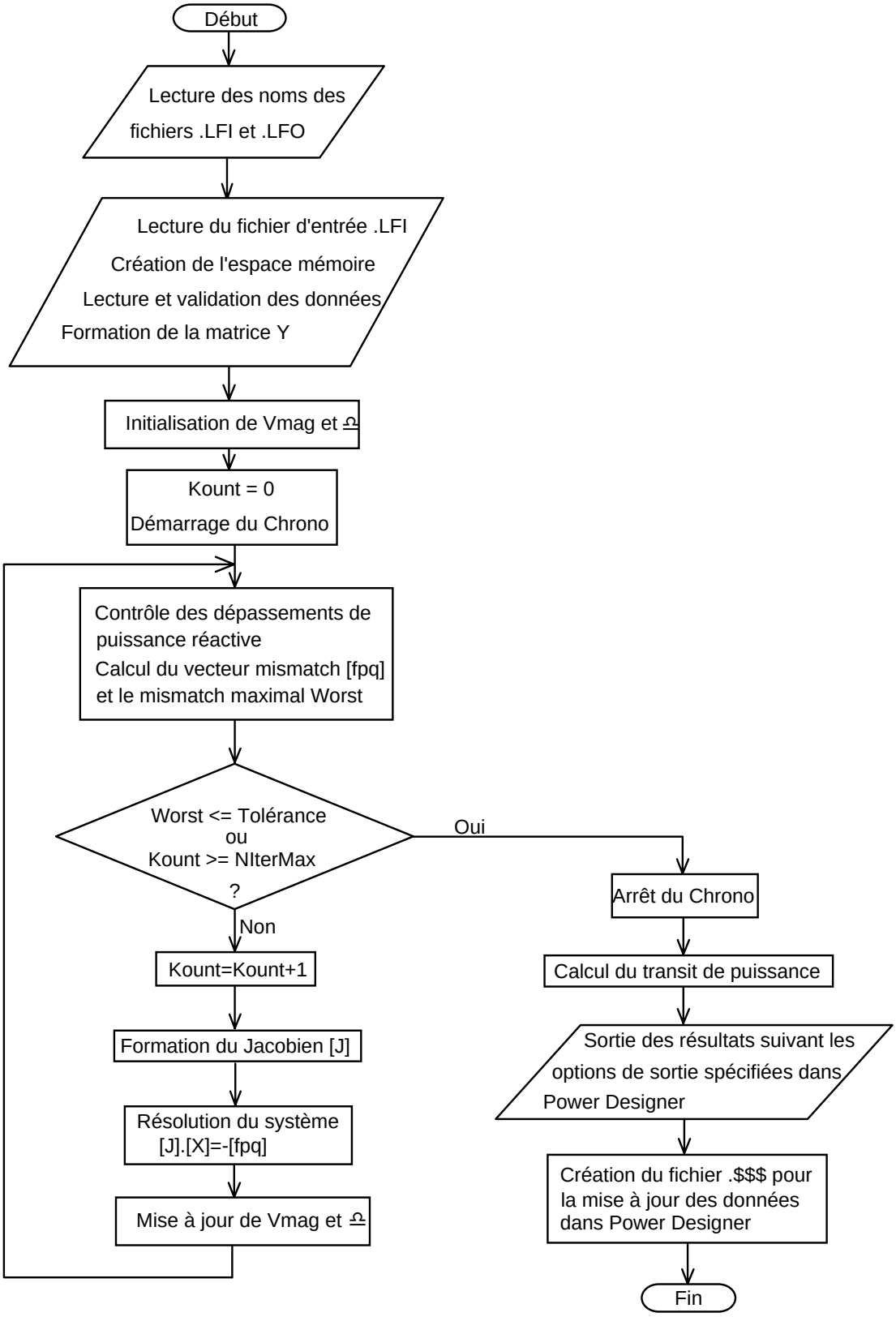
[5] Le premier axe correspond au bus de référence (slack bus). Il n'intervient pas dans les calculs, nous ne l'avons pas éliminé car l'indiçage actuel permet d'accélérer les accés.

indices de noeuds	0	1	·	·	·	·	N	$N+1$	·	$N+Npv$	$N+Npv+1$	·	$2N$
0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	$\frac{\partial \Delta P_1}{\partial \delta_1}$	·	·	·	·	$\frac{\partial \Delta P_1}{\partial \delta_N}$	0	0	·	$\frac{\partial \Delta P_1}{\partial V_{Npv+1}}$	·	$\frac{\partial \Delta P_1}{\partial V_N}$
·	0	·	·	·	·	·	·	0	0	·	·	·	·
·	0	·	·	·	·	·	·	0	0	·	·	·	·
·	0	·	·	·	·	·	·	0	0	·	·	·	·
·	0	·	·	·	·	·	·	0	0	·	·	·	·
N	0	$\frac{\partial \Delta P_N}{\partial \delta_1}$	·	·	·	·	$\frac{\partial \Delta P_N}{\partial \delta_N}$	0	0	·	$\frac{\partial \Delta P_N}{\partial V_{Npv+1}}$	·	$\frac{\partial \Delta P_N}{\partial V_N}$
$N+1$	0	0	0	0	0	0	0	1	0	0	0	0	0
·	0	0	0	0	0	0	0	0	1	0	0	0	0
$N+Npv$	0	·	·	·	·	·	·	0	0	·	·	·	·
$N+Npv+1$	0	$\frac{\partial \Delta Q_{Npv+1}}{\partial \delta_1}$	·	·	·	·	$\frac{\partial \Delta Q_{Npv+1}}{\partial \delta_N}$	0	0	·	$\frac{\partial \Delta Q_{Npv+1}}{\partial V_{Npv+1}}$	·	$\frac{\partial \Delta Q_{Npv+1}}{\partial V_N}$
·	0	·	·	·	·	·	·	0	0	·	·	·	·
$2N$	0	$\frac{\partial \Delta Q_N}{\partial \delta_1}$	·	·	·	·	$\frac{\partial \Delta Q_N}{\partial \delta_N}$	0	0	·	$\frac{\partial \Delta Q_N}{\partial V_{Npv+1}}$	·	$\frac{\partial \Delta Q_N}{\partial V_N}$

Jacobien de NRP

Cette manière de procéder demande nettement moins de temps de calcul que pour recopier les éléments du Jacobien lors d'une compression. De plus, il n'y a pas création d'une matrice jacobienne auxiliaire, ce qui permet d'économiser de l'espace mémoire.

En outre, lors du test de dépassement des limites en puissance réactive générée, si le noeud PV doit devenir PQ, c'est-à-dire que l'on va fixer sa puissance générée à **Qglim** (min ou max) et libérer **Vmag**, il suffit alors de ne pas annuler l'axe correspondant à ce noeud (i) dans le Jacobien et dans le vecteur mismatch (N+i).



Organigramme de NRP

4. 3. Méthode de Newton-Raphson modifiée (NRPB)

La principale différence avec le programme NRP réside dans un test que nous effectuons pour les noeuds PV devenus PQ lors d'un dépassement des limites de production de puissance réactive.

Si ce noeud PQ peut redevenir PV (avoir **Vmag=Vsch**) sans qu'il n'y ait à nouveau dépassement des limites de **Qg**, alors le noeud est remis PV.

Ce cas est toutefois rare et dépend de la topologie du réseau car nous ne commençons à tester les dépassements des limites de production de **Qg** qu'après une certaine convergence. Nous avons choisi de les tester après une itération, lorsque le mismatch de puissance est inférieur à 20 fois la tolérance spécifiée ou à 0.2 en absolu :

```
if ( Kount>=1 && ( Worst< Tol*20 || Worst< 0.2 ) )
        //... alors teste les Dépassements
```

4. 4. Méthode découplée rapide (FDL)

Le Fast Decoupled Load Flow que nous avons utilisé est celui décrit dans la référence [13].

soit :

$$Y_{ij} = G_{ij} + jB_{ij}$$

$$P_i = V_i \sum_{k=0}^{N} V_k (G_{ik} \cos(\delta_i - \delta_k) + B_{ik} \sin(\delta_i - \delta_k))$$

$$Q_i = V_i \sum_{k=0}^{N} V_k (G_{ik} \sin(\delta_i - \delta_k) - B_{ik} \cos(\delta_i - \delta_k))$$

$$\begin{cases} [\Delta P] = [P^{sch} - P] = -\left[\frac{\partial \Delta P}{\partial \delta}\right][\Delta \delta] \\ [\Delta Q] = [Q^{sch} - Q] = -\left[\frac{\partial \Delta Q}{\partial V}\right][\Delta V] \end{cases} \Rightarrow \begin{cases} [\Delta P] = \left[\frac{\partial P}{\partial \delta}\right][\Delta \delta] \\ [\Delta Q] = \left[\frac{\partial Q}{\partial V}\right][\Delta V] \end{cases}$$

avec les approximations :

$$G_{ij} \sin(\delta_i - \delta_j) << B_{ij} \ , \ \cos(\delta_i - \delta_j) \approx 1 \text{ et } Q_i << B_{ii} V_i^2$$

et

- en négligeant les éléments affectant l'écoulement de puissance réactif dans la formation de [B'], c'est-à-dire les réactances shunt et en considérant que les transformateurs fonctionnent à leur régime nominal.
- en négligeant les éléments affectant l'écoulement de puissance active dans la formation de [B''] et en omettant l'effet des transformateurs déphaseurs.
- en négligeant les résistances série lors de la formation de [B'].

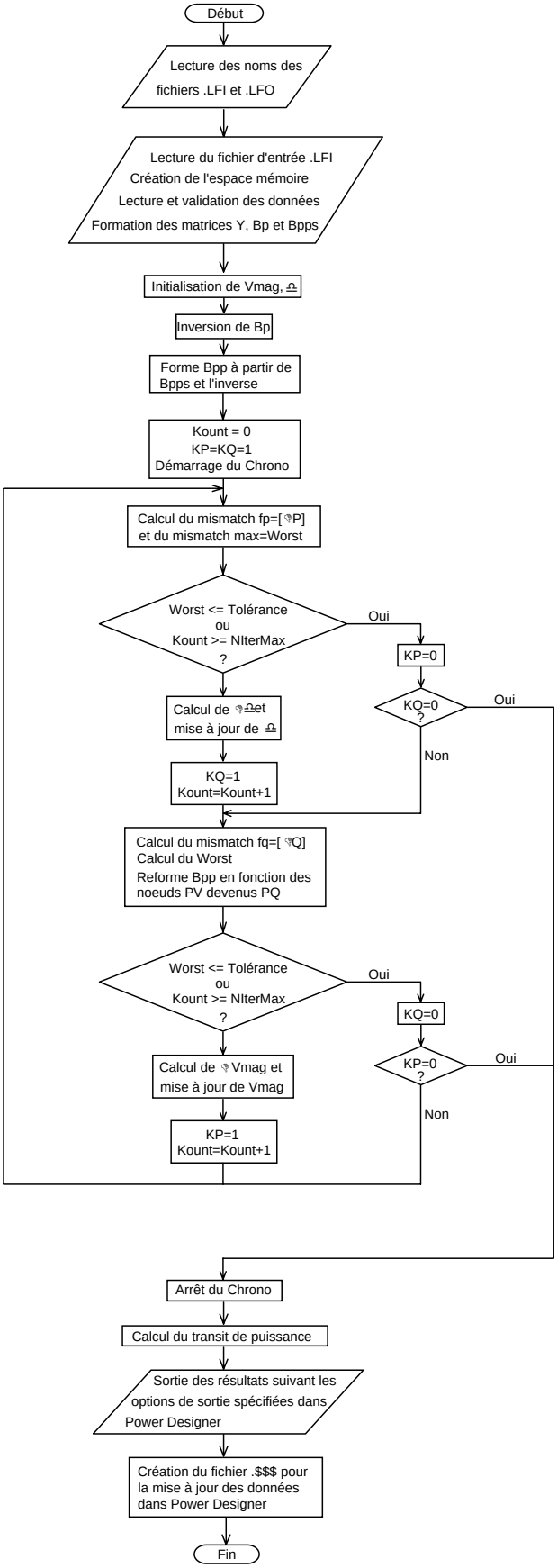
alors le système d'équations se ramène à :

$$\begin{cases} \left[\frac{\Delta P}{V}\right] = \left[B^{'}\right]\left[\Delta\delta\right] \\ \left[\frac{\Delta Q}{V}\right] = \left[B^{''}\right]\left[\Delta V\right] \end{cases}$$

avec $B_{ij}^{'} = -\frac{1}{X_{ij}}$ et $B_{ii}^{'} = \sum_{\substack{j=0 \\ j\neq i}}^{N} \frac{1}{X_{ij}}$

$B_{ij}^{''} = -imag(Y_{ij}) = -B_{ij}$

Remarque : Le test des noeuds PV devenant PQ modifie la matrice [B''] (les axes éliminés doivent êtres remis), il faut donc reformer et réinverser [B'']. Cette opération est coûteuse en temps de calcul et altère une des qualités essentielles de cette méthode; la rapidité.



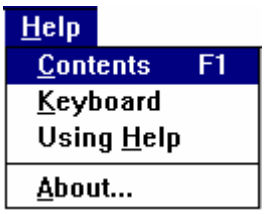
Organigramme de FDL

5. Applications

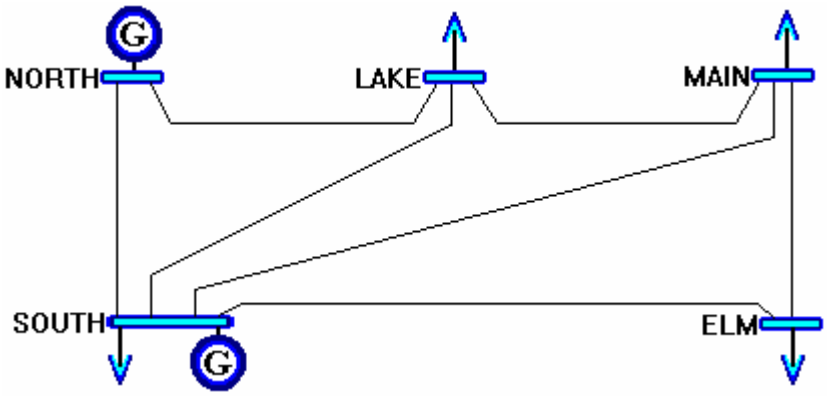
Afin d'illustrer les possibilités de notre logiciel, nous présentons deux exemples de réseaux :

- Un réseau à 5 noeuds de la référence [7]
- Le réseau à 14 noeuds IEEE (IEEE 14 Bus System) [12].

Pour la saisie du réseau et l'édition de ses données, l'utilisateur devra consulter l'Aide sous Windows de **POWER DESIGNER**, accessible à partir du menu **Help** ou simplement en appuyant sur la touche de fonction **F1**.



5. 1. Réseau à 5 noeuds



ABIAD.SCH

Schéma du réseau à 5 noeuds tel qu'il apparaît dans POWER DESIGNER

La commande **Simulation | Make Data** permet de générer le fichier **ABIAD.LFI** et l'affiche sur le Bloc Note :

```
5 1
25  5.0000000000E-05
10  1.0000000000E-03
8 ---------------  Input file for the load flow program  ---------------
8 --- Number of total Buses = 5  PV Buses = 1
8
8 --- for GS : Nb IterationsMax = 25  Tolerance =  5.0000000000E-05
8 --- for NR : Nb IterationsMax = 10  Tolerance =  1.0000000000E-03
8
8 ----------------------------  Bus Data  -------------------------------
8  BUS          Pg       Qg       Pl       Ql       Vb
1         NORTH  0.00000  0.00000  0.00000  0.00000  1.06000
1         SOUTH  0.40000  0.00000  0.20000  0.10000  1.04747
1          LAKE  0.00000  0.00000  0.45000  0.15000  1.00000
1          MAIN  0.00000  0.00000  0.40000  0.05000  1.00000
```

```
1          ELM  0.00000  0.00000  0.60000  0.10000  1.00000
8 ------------------------ Active Bus Limits ----------------------------
8  BUS          Qgmin    Qgmax
2         SOUTH  -0.30000  0.40000
8 ----------------------------  Line Data  -----------------------------
8  BUS          BUS          Rs       Xs       Gs       Bs       Smax
3         NORTH         LAKE  0.08000  0.24000  0.00000  0.02500  0.00000
3          LAKE         MAIN  0.01000  0.03000  0.00000  0.01000  0.00000
3          MAIN          ELM  0.08000  0.24000  0.00000  0.02500  0.00000
3         NORTH        SOUTH  0.02000  0.06000  0.00000  0.03000  0.00000
3         SOUTH         LAKE  0.06000  0.18000  0.00000  0.02000  0.00000
3         SOUTH         MAIN  0.06000  0.18000  0.00000  0.02000  0.00000
3         SOUTH          ELM  0.04000  0.12000  0.00000  0.01500  0.00000
8 ------------------------  Transformer Data  --------------------------
8  BUS          BUS          Rs       Xs       Amag     A(deg)  Smax
8 ------------------------  Capacitor Data  ---------------------------
8  BUS          Bc
8 ----------------------  Acceleration Factor GS  ----------------------
9   1.6000000000E+00
8 ----------------------  Load Flow Output Options  --------------------
10 369
8 ------------------------  END OF INPUT FILE  -------------------------
```

La commande **Simulation | Load Flow | Fast Decoupled Load Flow** appelle le programme FDL.EXE avec comme paramètres : ABIAD.LFI ABIAD.LFO.

Une fois la simulation finie, la commande **Simulation | Output Data** met à jour **ABIAD.SCH** et affiche le fichier **ABIAD.LFO** dans le Bloc Note.

```
8 --------------  Load flow studies  -------------
8 ------------  Fast Decoupled method  -----------
8 ---------------  Input file for the load flow program  ---------------
8 --- Number of total Buses = 5  PV Buses = 1
8
8 --- for GS : Nb IterationsMax = 25  Tolerance =  5.0000000000E-05
8 --- for NR : Nb IterationsMax = 10  Tolerance =  1.0000000000E-03
8
8 ----------------------------  Bus Data  -------------------------------
8  BUS          Pg       Qg       Pl       Ql       Vb
1  NORTH         0        0        0        0        1.06
1  SOUTH         0.4      0        0.2      0.1      1.04747
1  LAKE          0        0        0.45     0.15     1
1  MAIN          0        0        0.4      0.05     1
1  ELM           0        0        0.6      0.1      1
8 ------------------------ Active Bus Limits ----------------------------
8  BUS          Qgmin    Qgmax
2  SOUTH         -0.3     0.4
8 ----------------------------  Line Data  -----------------------------
8  BUS          BUS          Rs       Xs       Gs       Bs       Smax
3  NORTH         LAKE         0.08     0.24     0        0.025    0
3  LAKE          MAIN         0.01     0.03     0        0.01     0
3  MAIN          ELM          0.08     0.24     0        0.025    0
3  NORTH         SOUTH        0.02     0.06     0        0.03     0
3  SOUTH         LAKE         0.06     0.18     0        0.02     0
3  SOUTH         MAIN         0.06     0.18     0        0.02     0
3  SOUTH         ELM          0.04     0.12     0        0.015    0
```

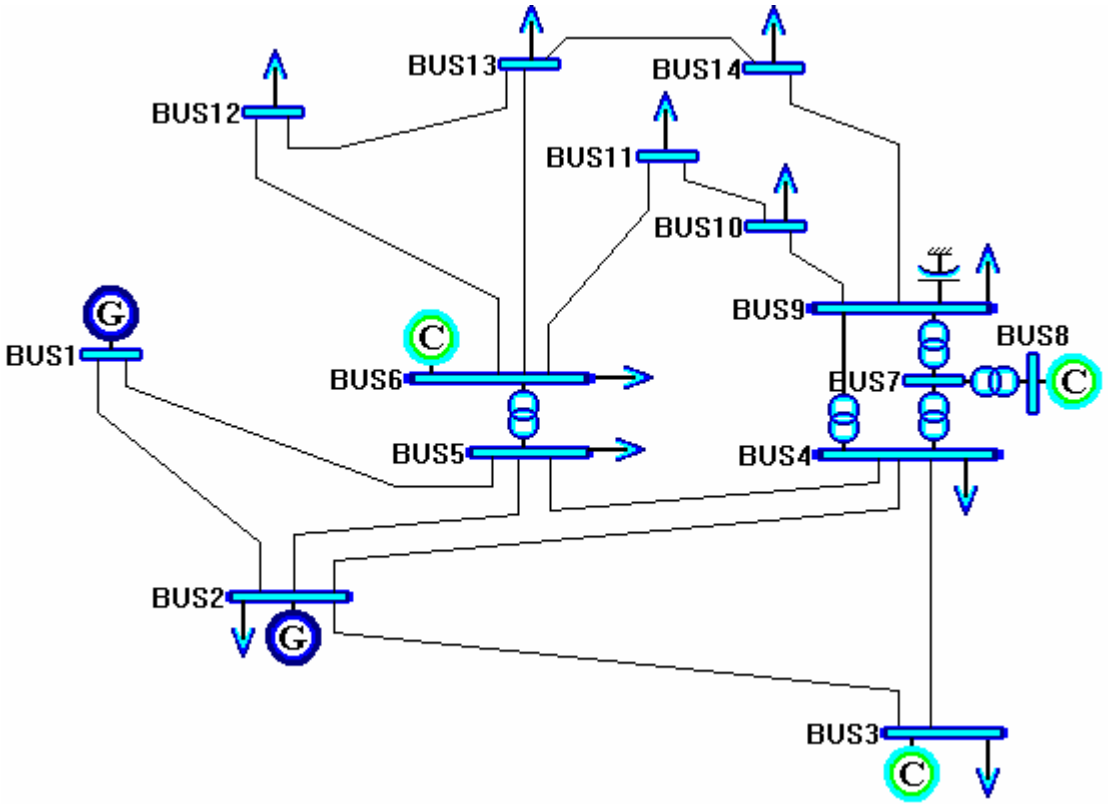
```
8 ------------------------  Transformer Data  --------------------------
8  BUS          BUS          Rs       Xs       Amag     A(deg)  Smax
8 -------------------------  Capacitor Data  ---------------------------
8  BUS          Bc
8 ----------------------  Acceleration Factor GS  ---------------------
9  1.6
8 ----------------------  Load Flow Output Options  --------------------
10 369
8 ------------------------  END OF INPUT FILE  ------------------------

---- YBUS matrix  ( no null elements )------
Busi        Busj              real        imag        magnitude   arg(deg)
NORTH       NORTH       Y[0 ,0 ]=6.25       -18.695     19.71207    -71.5145
NORTH       SOUTH       Y[0 ,1 ]=-5         15          15.81139    108.43495
NORTH       LAKE        Y[0 ,2 ]=-1.25      3.75        3.95285     108.43495
SOUTH       SOUTH       Y[1 ,1 ]=10.83333   -32.415     34.17738    -71.51999
SOUTH       LAKE        Y[1 ,2 ]=-1.66667   5           5.27046     108.43495
SOUTH       MAIN        Y[1 ,3 ]=-1.66667   5           5.27046     108.43495
SOUTH       ELM         Y[1 ,4 ]=-2.5       7.5         7.90569     108.43495
LAKE        LAKE        Y[2 ,2 ]=12.91667   -38.695     40.79391    -71.54062
LAKE        MAIN        Y[2 ,3 ]=-10        30          31.62278    108.43495
MAIN        MAIN        Y[3 ,3 ]=12.91667   -38.695     40.79391    -71.54062
MAIN        ELM         Y[3 ,4 ]=-1.25      3.75        3.95285     108.43495
ELM         ELM         Y[4 ,4 ]=3.75       -11.21      11.8206     -71.50374

Iteration Kount=0
Max_Mismatch=0.6332 at Bus n' 4 : ELM

Iteration Kount=1
Max_Mismatch=0.37154 at Bus n' 4 : ELM

Iteration Kount=2
Max_Mismatch=0.20515 at Bus n' 1 : SOUTH

Iteration Kount=3
Max_Mismatch=0.01597 at Bus n' 4 : ELM

Iteration Kount=4
Max_Mismatch=0.00403 at Bus n' 1 : SOUTH

Iteration Kount=5
Max_Mismatch=0.00034 at Bus n' 2 : LAKE

Iteration Kount=5
Max_Mismatch=0.00025 at Bus n' 1 : SOUTH

-----------------------  Output Bus Data  -------------------------
--------------------------------------------------------------------
Bus name : NORTH              Bus number : 0
                real       imag       magnitude   arg(deg)
  Voltage       : 1.06       0          1.06        0
  Generated power : 1.29559    -0.07472    1.29774     -3.30079
  Power demand    : 0          0          0           0
  Power transfer  : Bus name     Bus n'     real        imag
       to       : SOUTH        1          0.88841     -0.05259
       to       : LAKE         2          0.40718     0.03966
            Total Mismatch              real        imag
                                  0          0
--------------------------------------------------------------------
Bus name : SOUTH              Bus number : 1
```

```
                real        imag        magnitude   arg(deg)
  Voltage         : 1.04621     -0.05128    1.04747     -2.80608
  Generated power : 0.4         0.30091     0.50055     36.95338
  Power demand    : -0.2        -0.1        0.22361     -153.43495
  Power transfer  : Bus name     Bus n'     real        imag
       to         : NORTH        0          -0.87431    0.09488
       to         : LAKE         2          0.24696     0.05756
       to         : MAIN         3          0.27937     0.05168
       to         : ELM          4          0.54824     0.09005
            Total Mismatch                   real        imag
                                    -0.0003     2.7756e-17
---------------------------------------------------------------------
Bus name : LAKE                Bus number : 2
                real        imag        magnitude   arg(deg)
  Voltage         : 1.02029     -0.0892     1.02418     -4.9963
  Generated power : 0           0           0           0
  Power demand    : -0.45       -0.15       0.47434     -161.56505
  Power transfer  : Bus name     Bus n'     real        imag
       to         : NORTH        0          -0.39526    -0.00391
       to         : SOUTH        1          -0.24344    -0.04701
       to         : MAIN         3          0.18869     -0.04173
            Total Mismatch                   real        imag
                                    1.0376e-05  0.0003
---------------------------------------------------------------------
Bus name : MAIN                Bus number : 3
                real        imag        magnitude   arg(deg)
  Voltage         : 1.01916     -0.09506    1.02358     -5.3285
  Generated power : 0           0           0           0
  Power demand    : -0.4        -0.05       0.40311     -172.87498
  Power transfer  : Bus name     Bus n'     real        imag
       to         : SOUTH        1          -0.27496    -0.03844
       to         : LAKE         2          -0.18834    0.0428
       to         : ELM          4          0.06332     0.00334
            Total Mismatch                   real        imag
                                    -2.7047e-05  -7.7923e-05
---------------------------------------------------------------------
Bus name : ELM                 Bus number : 4
                real        imag        magnitude   arg(deg)
  Voltage         : 1.01209     -0.10905    1.01795     -6.14953
  Generated power : 0           0           0           0
  Power demand    : -0.6        -0.1        0.60828     -170.53768
  Power transfer  : Bus name     Bus n'     real        imag
       to         : SOUTH        1          -0.53698    -0.0563
       to         : MAIN         3          -0.06302    -0.00242
            Total Mismatch                   real        imag
                                    -1.0023e-06  0.0002
---------------------------------------------------------------------

----------  Bus voltage  ----------
Bus               real        imag        magnitude   arg(deg)
NORTH          V[0 ]=1.06        0           1.06        0
SOUTH          V[1 ]=1.04621     -0.05128    1.04747     -2.80608
LAKE           V[2 ]=1.02029     -0.0892     1.02418     -4.9963
MAIN           V[3 ]=1.01916     -0.09506    1.02358     -5.3285
ELM            V[4 ]=1.01209     -0.10905    1.01795     -6.14953

-------  Injected power at all buses  ----------
  Generated power :
Bus               real        imag        magnitude   arg(deg)
NORTH          Sg[0 ]=1.29559     -0.07472    1.29774     -3.30079
```

```
SOUTH        Sg[1 ]=0.4        0.30091     0.50055     36.95338
LAKE         Sg[2 ]=0          0           0           0
MAIN         Sg[3 ]=0          0           0           0
ELM          Sg[4 ]=0          0           0           0

  Power demand :
Bus              real       imag       magnitude   arg(deg)
NORTH        Sd[0 ]=0          0           0           0
SOUTH        Sd[1 ]=-0.2       -0.1        0.22361     -153.43495
LAKE         Sd[2 ]=-0.45      -0.15       0.47434     -161.56505
MAIN         Sd[3 ]=-0.4       -0.05       0.40311     -172.87498
ELM          Sd[4 ]=-0.6       -0.1        0.60828     -170.53768
---------------------------------------------------------------------
                 real       imag       magnitude   arg(deg)
Total generation     1.69559     0.22619     1.71061     7.59836
Total demand         -1.65       -0.4        1.69779     -166.37301
AC losses            0.04559     -0.17381    0.17969     -75.30274

-----  Line power transfer ( bus i --> j at i  -----
Busi         Busj              real       imag       mag (pu)    arg(deg)
NORTH        SOUTH        S[0 ,1 ]=0.88841     -0.05259    0.88997     -3.38741
NORTH        LAKE         S[0 ,2 ]=0.40718     0.03966     0.4091      5.56355
SOUTH        NORTH        S[1 ,0 ]=-0.87431    0.09488     0.87945     173.80651
SOUTH        LAKE         S[1 ,2 ]=0.24696     0.05756     0.25358     13.11954
SOUTH        MAIN         S[1 ,3 ]=0.27937     0.05168     0.28411     10.48034
SOUTH        ELM          S[1 ,4 ]=0.54824     0.09005     0.55558     9.32825
LAKE         NORTH        S[2 ,0 ]=-0.39526    -0.00391    0.39528     -179.43275
LAKE         SOUTH        S[2 ,1 ]=-0.24344    -0.04701    0.24794     -169.07068
LAKE         MAIN         S[2 ,3 ]=0.18869     -0.04173    0.19325     -12.47038
MAIN         SOUTH        S[3 ,1 ]=-0.27496    -0.03844    0.27763     -172.04207
MAIN         LAKE         S[3 ,2 ]=-0.18834    0.0428      0.19314     167.19742
MAIN         ELM          S[3 ,4 ]=0.06332     0.00334     0.06341     3.02021
ELM          SOUTH        S[4 ,1 ]=-0.53698    -0.0563     0.53993     -174.0152
ELM          MAIN         S[4 ,3 ]=-0.06302    -0.00242    0.06306     -177.80082
  No excess power transfer

Number of iterations : 5
Total time : 0.16484 s
Mean time per iteration : 0.03297 s
```

Remarque : Le temps d'exécution est différent de celui du tableau comparatif des temps d'exécution (voir paragraphe 6.2) à cause des options spécifiées d'affichage et de sortie sur fichier et de l'action du cache-disque (en écriture et en lecture) **Smart Drive** de Microsoft dont Windows a besoin pour un fonctionnement optimum. De plus, si d'autres applications Windows sont en cours d'exécution, la gestion multitâche de Windows affecte aussi ce temps d'exécution.

5. 2. Réseau IEEE 14 noeuds



I14.SCH

Schéma du réseau IEEE 14 noeuds tel qu'il apparaît dans POWER DESIGNER

Le fichier de sortie **I14.LFO** :

```
8 --------------  Load flow studies  -------------
8 ------------  Newton Raphson method  ------------
8 ----------------  Input file for the load flow program  ----------------
8 --- Number of total Buses = 14  PV Buses = 4
8
8 --- for GS : Nb IterationsMax = 50  Tolerance =  1.0000000000E-05
8 --- for NR : Nb IterationsMax = 10  Tolerance =  1.0000000000E-04
8
8 ----------------------------  Bus Data  -------------------------------
8  BUS          Pg       Qg       Pl       Ql       Vb
1  BUS1         0        0        0        0        1.06
1  BUS2         0.4      0        0.217    0.127    1.045
1  BUS3         0        0        0.942    0.19     1.01
1  BUS6         0        0        0.112    0.075    1.07
1  BUS8         0        0        0        0        1.09
1  BUS4         0        0        0.478    -0.039   1
1  BUS5         0        0        0.076    0.016    1
1  BUS7         0        0        0        0        1
1  BUS9         0        0        0.295    0.166    1
1  BUS10        0        0        0.09     0.058    1
1  BUS11        0        0        0.035    0.018    1
1  BUS12        0        0        0.061    0.016    1
1  BUS13        0        0        0.135    0.058    1
1  BUS14        0        0        0.149    0.05     1
```

```
8 ------------------------ Active Bus Limits ---------------------------
8  BUS          Qgmin    Qgmax
2  BUS2         -0.4     0.5
2  BUS3         0        0.4
2  BUS6         -0.06    0.24
2  BUS8         -0.06    0.24
8 ----------------------------- Line Data  -----------------------------
8  BUS          BUS          Rs       Xs       Gs       Bs       Smax
3  BUS1         BUS2         0.01938  0.05917  0        0.0264   0
3  BUS2         BUS3         0.04699  0.19797  0        0.0219   0
3  BUS3         BUS4         0.06701  0.17103  0        0.0173   0
3  BUS2         BUS4         0.05811  0.17632  0        0.0187   0
3  BUS2         BUS5         0.05695  0.17388  0        0.017    0
3  BUS4         BUS5         0.01335  0.04211  0        0.0064   0
3  BUS1         BUS5         0.05403  0.22304  0        0.0246   0
3  BUS9         BUS10        0.03181  0.0845   0        0        0
3  BUS10        BUS11        0.08205  0.19207  0        0        0
3  BUS6         BUS11        0.09498  0.1989   0        0        0
3  BUS9         BUS14        0.12711  0.27038  0        0        0
3  BUS13        BUS14        0.17093  0.34802  0        0        0
3  BUS12        BUS13        0.22092  0.19988  0        0        0
3  BUS6         BUS12        0.12291  0.25581  0        0        0
3  BUS6         BUS13        0.06615  0.13027  0        0        0
8 ------------------------ Transformer Data ---------------------------
8  BUS          BUS          Rs       Xs       Amag     A(deg)   Smax
4  BUS5         BUS6         0        0.25202  0.932    0        0
4  BUS4         BUS7         0        0.20912  0.978    0        0
4  BUS7         BUS9         0        0.11001  1        0        0
4  BUS7         BUS8         0        0.17615  1        0        0
4  BUS4         BUS9         0        0.55618  0.969    0        0
8 ------------------------ Capacitor Data ---------------------------
8  BUS          Bc
5  BUS9         0.19
8 ----------------------- Acceleration Factor GS ----------------------
9  1.8
8 ---------------------- Load Flow Output Options ---------------------
10 336
8 ------------------------ END OF INPUT FILE ------------------------

Iteration Kount=0
Max_Mismatch=0.80159 at Bus n' 2 : BUS3

Iteration Kount=1
Max_Mismatch=0.10271 at Bus n' 6 : BUS5

Iteration Kount=2
Max_Mismatch=0.00098 at Bus n' 6 : BUS5

Iteration Kount=3
Max_Mismatch=8.35493e-08 at Bus n' 6 : BUS5

----------  Bus voltage  ----------
Bus              real        imag        magnitude   arg(deg)
BUS1        V[0 ]=1.06        0          1.06        0
BUS2        V[1 ]=1.04105     -0.09073    1.045       -4.98095
BUS3        V[2 ]=0.98522     -0.22235    1.01        -12.71797
BUS6        V[3 ]=1.0372      -0.26289    1.07        -14.22265
BUS8        V[4 ]=1.06047     -0.25202    1.09        -13.36825
BUS4        V[5 ]=1.00213     -0.18256    1.01862     -10.32422
BUS5        V[6 ]=1.0083      -0.15578    1.02026     -8.78258
```

```
BUS7        V[7 ]=1.03318    -0.24553    1.06195    -13.36825
BUS9        V[8 ]=1.02061    -0.27245    1.05635    -14.9466
BUS10       V[9 ]=1.01501    -0.27395    1.05133    -15.10432
BUS11       V[10]=1.02203    -0.26994    1.05708    -14.79526
BUS12       V[11]=1.01889    -0.27449    1.05522    -15.07742
BUS13       V[12]=1.01389    -0.27469    1.05044    -15.15894
BUS14       V[13]=0.99548    -0.28618    1.03579    -16.03893

-------  Injected power at all buses  ----------
  Generated power :
Bus              real       imag       magnitude   arg(deg)
BUS1        Sg[0 ]=2.32386    -0.16889    2.32999    -4.15672
BUS2        Sg[1 ]=0.4        0.42396     0.58288    46.66594
BUS3        Sg[2 ]=0          0.23394     0.23394    90
BUS6        Sg[3 ]=0          0.1224      0.1224     90
BUS8        Sg[4 ]=0          0.17357     0.17357    90
BUS4        Sg[5 ]=0          0           0          0
BUS5        Sg[6 ]=0          0           0          0
BUS7        Sg[7 ]=0          0           0          0
BUS9        Sg[8 ]=0          0           0          0
BUS10       Sg[9 ]=0          0           0          0
BUS11       Sg[10]=0          0           0          0
BUS12       Sg[11]=0          0           0          0
BUS13       Sg[12]=0          0           0          0
BUS14       Sg[13]=0          0           0          0

  Power demand :
Bus              real       imag       magnitude   arg(deg)
BUS1        Sd[0 ]=0          0           0          0
BUS2        Sd[1 ]=-0.217     -0.127      0.25143    -149.66155
BUS3        Sd[2 ]=-0.942     -0.19       0.96097    -168.59652
BUS6        Sd[3 ]=-0.112     -0.075      0.13479    -146.19204
BUS8        Sd[4 ]=0          0           0          0
BUS4        Sd[5 ]=-0.478     0.039       0.47959    175.33557
BUS5        Sd[6 ]=-0.076     -0.016      0.07767    -168.11134
BUS7        Sd[7 ]=0          0           0          0
BUS9        Sd[8 ]=-0.295     -0.166      0.3385     -150.63303
BUS10       Sd[9 ]=-0.09      -0.058      0.10707    -147.20047
BUS11       Sd[10]=-0.035     -0.018      0.03936    -152.78389
BUS12       Sd[11]=-0.061     -0.016      0.06306    -165.30268
BUS13       Sd[12]=-0.135     -0.058      0.14693    -156.75021
BUS14       Sd[13]=-0.149     -0.05       0.15717    -161.44977
---------------------------------------------------------------------
                 real       imag       magnitude   arg(deg)
Total generation    2.72386    0.78498    2.83471    16.07629
Total demand        -2.59      -0.735     2.69227    -164.15692
AC losses           0.13386    0.04998    0.14289    20.47483

Number of iterations : 3
Total time : 0.32967 s
Mean time per iteration : 0.10989 s
```

6. Temps d'exécution

Nous avons établi un tableau comparatif des temps d'exécution du calcul d'écoulement de puissance par les différents programmes développés.

6. 1. Configuration machine

Ces temps sont ceux relatifs à une machine compatible PC 486 DX 33 avec coprocesseur mathématique intégré.

Les programmes ont été exécutés sous DOS et le cache disque **Smart Drive** de Microsoft a été inhibé afin que les essais soient reproductibles.
En effet, **Smart Drive** utilise une partie de la mémoire pour y stocker temporairement ce qu'il a déjà lu et/ou ce qu'il va écrire sur le disque dur.

Le temps d'exécution est compté entre le moment où les itérations commencent et le moment où elles finissent. Les options d'affichage et de sortie sur fichier **.LFO** ont été réduites au minimum.

6. 2. Tableau comparatif

Temps moyen par itération (s)	**Ti**
Temps d'exécution total (s)	**Tt**
Nombre d'itérations	**K**

pour une tolérance :

- GSP $\Delta V_i <= 5.E-05$ p.u.
- NRP, FDL $\max(\Delta P_i, \Delta Q_i) <= 1.E-03$ p.u.

Réseau	GSP	NRP	FDL
5 noeuds [7]	**Ti**=0 **Tt**=0 **K**=13 **accel**=1.6	**Ti**=0 **Tt**=0 **K**=2	**Ti**=0 **Tt**=0 **K**=5
IEEE 14 noeuds	**Ti**=0.00597 **Tt**=0.27473 **K**=46 **accel**=1.8	**Ti**=0.10989 **Tt**=0.21978 **K**=2	**Ti**=0.03663 **Tt**=0.21978 **K**=6
IEEE 30 noeuds	**Ti**=0.02437 **Tt**=3.24176 **K**=133 **accel**=1.8	**Ti**=0.65934 **Tt**=1.31868 **K**=2	**Ti**=0.21062 **Tt**=1.26374 **K**=6
IEEE 57 noeuds	**Ti**=0.08254 **Tt**=17.91209 **K**=217 **accel**=1.8	Mémoire insuffisante	**Ti**=1.05769 **Tt**=12.69231 **K**=12

Remarques :

- Le programme NRPB donne les mêmes temps de calcul que NRP pour ces réseaux.
- Le nombre d'itérations de FDL correspond au nombre des demi-itérations de l'algorithme décrit dans la référence [13].

7. Discussion des résultats

Les résultats obtenus montrent clairement la supériorité des algorithmes de Newton-Raphson et particulièrement du Fast Decoupled Load Flow sur celui de Gauss-Seidel. En effet, bien que ce dernier affiche le temps le plus court par itération, le nombre d'itérations nécessaires à la convergence est nettement plus important. De ce fait, il se retrouve en dernier.

De plus, le nombre d'itérations croit pour la méthode de Gauss-Seidel avec le nombre de noeuds du réseau alors que ce nombre est pratiquement indépendant du réseau pour NRP. Ceci est principalement dû à la manière d'évaluation de la méthode de Newton-Raphson qui prend en compte toutes les interactions en même temps, la convergence est alors quadratique.

La méthode FDL affiche les mêmes temps d'exécution que celle de N-R pour les très petits réseaux. Cependant, elle devient plus rapide pour les réseaux plus importants et pour les tolérances habituelles (0.01 à 0.001).

Ces caractéristiques sont dues au découplage qui permet de ne traiter que la moitié du Jacobien. De plus, les approximations faites permettent d'avoir [B'] constante. Elle n'est alors inversée qu'une seule fois.

Le temps de calcul pour une itération Q lors du calcul du vecteur mismatch fq (voir l'organigramme de FDL) peut être réduit si la matrice [B''] n'est pas inversée à chaque fois. Son inverse doit être alors corrigé lors des conversions noeud PV vers noeud PQ de manière à en tenir compte [14].

8. Conclusion

A cause de la convergence quadratique de la méthode de Newton-Raphson, une solution de haute précision peut être obtenue en quelques itérations seulement. Le découplage P de V et Q de δ permet d'accroître ces performances au détriment quelques fois de la stabilité de la méthode par rapport à celle de N-R, mais des approximations (qui ont donné lieu au FDL) viennent consolider cette stabilité.

De plus, ces méthodes peuvent facilement être adaptées afin d'inclure les transformateurs de réglage et la production optimale.
Ajoutons à cela que le test de convergence basé sur le mismatch (N-R) est plus robuste que celui basé sur les écarts de tension (G-S).

Ces caractéristiques font le succès du Fast Decoupled Load Flow et de la méthode de Newton-Raphson.

En outre, les programmes de calcul d'écoulement de puissance que nous avons élaboré pour **POWER DESIGNER** tiennent compte des limites de production en puissance réactive, ce qui est plus proche de la réalité physique. Nous avons même prévu dans le programme NRPB, pour les réseaux mal conditionnés, la possibilité pour les noeuds PV devenus PQ de revenir PV par la suite s'il n'y a pas de dépassement des limites de production de puissance réactive.

L'environnement **POWER DESIGNER** associé à Windows apporte une grande souplesse d'utilisation dans la simulation de l'écoulement de puissance et permet de changer rapidement les paramètres et la topologie du réseau.
De plus, la gestion multitâche permet de revenir à **POWER DESIGNER** ou à une autre application Windows pendant que le calcul d'écoulement de puissance continue à se faire.

Notons que dans sa version actuelle, bien que **POWER DESIGNER** puisse gérer plus de 16000 composants de réseau (noeuds, lignes, transformateurs...), les programmes de calcul d'écoulement de puissance ne peuvent assimiler que des réseaux de moins de 64 noeuds à cause des limitations de la capacité mémoire du PC. La voie est donc ouverte à d'autres élèves ingénieurs pour le développement de programmes de calcul d'écoulement de puissance basés sur l'éparpillement du Jacobien (sparsity methods) afin d'optimiser la gestion de l'espace mémoire et de pouvoir faire marcher des réseaux plus grands sur un PC. Ces programmes s'intégreront très facilement à **POWER DESIGNER**.

Chapitre IV : Stabilité Transitoire

1. Introduction

Les systèmes de puissance deviennent de plus en plus complexes, très interconnectés et comportent un grand nombre de machines qui peuvent interagir énergiquement par le biais de leur réseau d'Extra et d'Ultra haute tension.

L'étude de la stabilité fait partie des études qui permettent de planifier la construction et l'extension des réseaux électriques ainsi que la conduite et le contrôle de ces réseaux.

Quand une perturbation qui a causé momentanément un changement de vitesse des machines a été éliminée, les machines du système de puissance doivent à nouveau fonctionner à la vitesse de synchronisme. Si une machine ne reste pas au synchronisme avec les autres machines du système, de forts courants de circulation vont apparaître et dans un système bien dimensionné, des relais et des disjoncteurs doivent découpler la machine du système.

Les études de stabilité concernent surtout la stabilité statique, dynamique et transitoire du réseau.

Les études de stabilité statique et dynamique concernent [8] un petit groupe de machines sur lesquelles on effectue de lents ou légers changements des conditions d'exploitation. Ces études se préoccupent de la réponse du réseau suite à des variations incrémentales autour d'un point d'équilibre.

Les études de stabilité transitoire s'occupent des grandes perturbations. Ces perturbations peuvent être d'origines diverses : pertes de groupes, de lignes, branchement, débranchement de charges, défaut shunt (court-circuit), franc ou via une impédance de défaut ...

Ces grandes perturbations ne permettent pas la linéarisation des équations autour d'un point de fonctionnement comme c'est le cas pour les études de stabilité statique et dynamique.
Il faut donc recourir à des méthodes directes ou itératives pour la résolution de ces équations différentielles.

Les études de stabilité transitoire permettent de connaître l'évolution des grandeurs critiques d'un réseau électrique pendant et après la perturbation. Les programmes de stabilité transitoire doivent donc fournir les tensions, puissances, courants et plus spécialement vitesses, angles internes et couples des machines du réseau.

Les problèmes de stabilité transitoire peuvent être subdivisés en deux catégories :

- Stabilité après la première oscillation (first-swing stability).
  Dans ce cas, l'étude est basée sur un modèle simple de générateur sans les systèmes de régulation.
  On ne considère alors que la première seconde suivant la perturbation. Si les machines restent stables après cette seconde, le système de puissance est considéré comme stable pour cette perturbation.

- Stabilité après plusieurs oscillations (multiswing stability)
  Cette fois, l'étude se prolonge sur plusieurs secondes après la perturbation. Elle exige des modèles plus rigoureux et la prise en compte des systèmes de régulation afin de reproduire le plus fidèlement possible le comportement des machines du système.

Dans **POWER DESIGNER**, nous avons voulu intégrer un module de simulation de stabilité transitoire des réseaux de puissance. Nous avons opté pour le premier mode. C'est-à-dire, la stabilité après la première oscillation.

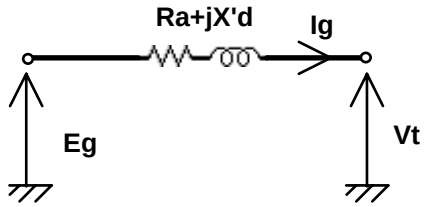
Dans notre étude, nous ne considérerons que les défauts momentanés symétriques suivants : court-circuit triphasé, ouverture de ligne, augmentation brutale de la charge, diminution brutale de la charge et perte de générateur. Ces défauts peuvent apparaître simultanément et se chevaucher dans le temps.

Après simulation, une comparaison pourra être faite, facilement grâce aux graphes, pour juger de la sévérité du défaut suivant les cas.

2. Modélisation et hypothèses simplificatrices

Nous avons adopté le modèle de base [7],[8] du générateur (et du compensateur synchrone).

- La puissance mécanique d'entraînement est considérée comme constante durant toute l'étude (absence de régulation de vitesse).
- La machine est représenté par une f.e.m. Eg derrière l'impédance transitoire de la machine Ra+jXd'.



Circuit équivalent de la représentation simplifiée de la machine synchrone

- Eg est d'amplitude constante (absence de régulation de tension).

Le choix d'un système de référence est très important. Pour une machine synchrone, on considère un système d'axes lié au rotor, l'angle interne étant représenté par l'angle entre la f.e.m. Eg et la tension de sortie Vt.

Mais dans un système multi-machines en régime transitoire, chaque machine tourne indépendamment l'une de l'autre. Il faut donc choisir un système de référence indépendant pour tout le réseau.

On choisit généralement un système d'axes tournant à la vitesse de synchronisme et qui coïncide avec celui de la machine du noeud de référence avant la perturbation [6].

Les angles qu'on affiche sont ceux des f.e.m. Eg par rapport à ce système d'axes. Ils sont nommés "Internal Voltage phase" dans le module de récapitulation graphique **GraphD** de **POWER DESIGNER**.

2.1. Equations des machines

$$\theta_m = \omega_{sm} t + \delta_m$$
$$\theta = \omega_s t + \delta \quad \text{avec} \quad \theta = p\theta_m \text{ et } \delta = p\delta_m$$

$$J\frac{d\omega_m}{dt} = J\frac{d^2\theta_m}{dt^2} = J\frac{d^2\delta_m}{dt^2} = C_m - C_e$$
$$J\omega_{sm}\frac{d^2\delta_m}{dt^2} = P_m - P_e \text{ car } \omega_m \approx \omega_{sm}$$
$$\frac{1}{S_b}\left(\frac{1}{2}J\omega_{sm}^2\right)\frac{2}{\omega_{sm}}\frac{d^2\delta_m}{dt^2} = \frac{P_m - P_e}{S_b} \text{ (W)}$$
$$\text{soit } H = \frac{1}{S_b}\left(\frac{1}{2}J\omega_{sm}^2\right)$$

H définit le rapport de l'énergie cinétique stockée à la vitesse de synchronisme sur la puissance de base en MVA.

$$\frac{2H}{\omega_{sm}}\frac{d^2\delta_m}{dt^2} = P_m - P_e \text{ (p.u.)}$$

comme ω_{sm} et δ_m sont tous deux mécaniques, il en résulte l'équation :

$$\frac{2H}{\omega_s}\frac{d^2\delta}{dt^2} = P_m - P_e \text{ (p.u.)}$$

avec $\omega_s = 2\pi f$

De plus, en y incluant un facteur d'amortissement (damping factor), l'équation devient :

$$\frac{2H}{\omega_s}\frac{d^2\delta}{dt^2} = P_m - P_e - D\frac{d\delta}{dt} \text{ (p.u.)}$$

Les équations à résoudre deviennent :

$$\frac{d\delta}{dt} = \omega - \omega_s$$
$$\frac{d^2\delta}{dt^2} = \frac{\omega_s}{2H}\left(P_m - P_e - D\frac{d\delta}{dt}\right)$$

Les grandeurs au niveau du générateur sont :

$$E_g = V_t + \frac{I_g}{y_g} \quad \text{avec} \quad y_g = \frac{1}{R_a + jX'_d} \quad \text{et} \quad I_g = \left(\frac{S_g}{V_t}\right)^*$$
$$P_e = \mathrm{Re}(E_g * I_g^*)$$

Le vecteur d'état du système multi-machines [Xv] est égal à :

$$X_V = \begin{bmatrix} \delta_0 \\ \delta_1 \\ . \\ \delta_{Npv} \\ \hline \omega_0 \\ \omega_1 \\ . \\ \omega_{Npv} \end{bmatrix}$$

Il est initialisé à t=0 comme suit :

$$Xv[i] = \arg\left(E_g[i]\right)$$
$$Xv[N_{PV} + i] = \omega_s$$

i=0 à Npv

2.2. Modélisation des charges

Les charges sont représentées par des admittances calculées à la tension du point de fonctionnement avant défaut. Ce point de fonctionnement est obtenu par un calcul d'écoulement de puissance.

$$Y_L = \frac{P_L - jQ_L}{V^2}$$

2.3. Modélisation des défauts

2.3.1. Court-circuit triphasé

Le court-circuit triphasé est représenté par une admittance shunt très importante ($10^{20}\ \Omega^{-1}$) que nous ajoutons au niveau du noeud de défaut pendant la durée spécifiée.
Cette représentation offre comme avantage de pouvoir simuler un court-circuit (non franc) via une admittance de défaut (Gf+jBf) dont la valeur est fixée dans la boîte de dialogue du menu **Options | Transient Stability**.

```
if (FBFlag==1) Y[If*N1+If]+=complex( Gf, Bf);
```

2.3.2. Ouverture de ligne

L'ouverture de ligne est simulée en supprimant l'effet de la ligne dans la matrice Y. Nous ajoutons, pendant la durée de ce défaut, une ligne fictive de caractéristiques négatives mais égales en module à la ligne d'origine :

```
if (LOFlag==1) {      Y[Ip*N1+Ip]-=ys_OpenIpIq+ysh_OpenIpIq;
                      Y[Iq*N1+Iq]-=ys_OpenIqIp+ysh_OpenIqIp;
                      Y[Ip*N1+Iq]+=ys_OpenIpIq;
                      Y[Iq*N1+Ip]+=ys_OpenIqIp;  }
```

2.3.3. Augmentation brutale de la charge

L'augmentation brutale de la charge est simulée en ajoutant une admittance shunt au niveau du noeud de défaut pendant la durée spécifiée.
Cette admittance est égale à son équivalent charge rapportée à une tension nominale.

```
if (SLFlag==1) Y[ISL*N1+ISL]+=complex( SLPL, -SLQL);
```

2.3.4. Diminution brutale de la charge

Cette fois, l'admittance est de signe contraire.

```
if (DLFlag==1) Y[IDL*N1+IDL]-=complex( DLPL, -DLQL);
```

2.3.5. Perte de générateur

La perte du générateur à un noeud spécifié se simule :

- En enlevant de la matrice Y, l'admittance transitoire du générateur yg.

```
if (LGFlag==1) Y[ILG*N1+ILG]-=yg[ILG];
```

- En supprimant l'injection du courant du générateur (yg[i]*Eg[i]) au niveau du noeud **i** dans le calcul de l'écoulement de puissance.

```
sum=complex(0,0);
for ( j=0; j<=N; j++)   if ( i!=j) sum+=Y[i*N1+j]*Vc[j];

if ( i<=Npv && !( i==ILG && LGFlag==1) )
                Vc[i]=(-sum+yg[i]*Eg[i])/Y[i*N1+i];
  else          Vc[i]= -sum/Y[i*N1+i];
```

- Et en forçant la puissance du générateur à zéro dans la résolution des équations différentielles du système.

```
Ig=( Eg[i]-polar( Vmag[i], delta[i]) )*yg[i];
if ( LGFlag==1 && i==ILG ) Ig=0;  // Gen débranché
Pg[i]=real( Eg[i]*conj( Ig) );
```

Tous ces défauts ont leur propre moment d'apparition et d'élimination. Les défauts peuvent donc apparaître simultanément ou se chevaucher dans le temps.

3. Possibilités de POWER DESIGNER

POWER DESIGNER comporte un ensemble d'options et de caractéristiques visant à apporter une souplesse dans la mise en oeuvre et l'exécution de simulations dans l'étude de la stabilité transitoire des réseaux.

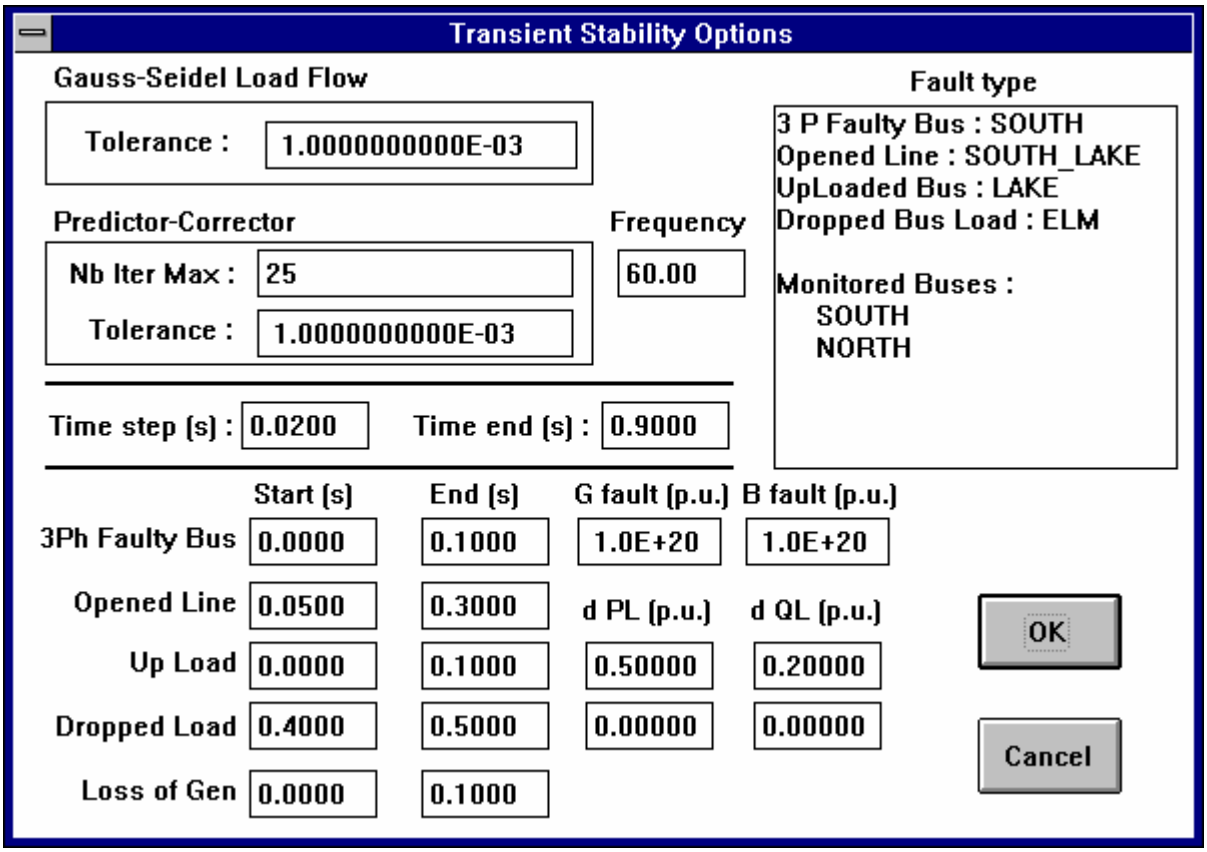
Avant qu'une étude de stabilité transitoire ne puisse être entreprise, il faut d'abord faire un calcul d'écoulement de puissance et appeler **Output Data** pour la mise à jour de la base de données en vue d'obtenir le point de fonctionnement du réseau.

Remarque : Il est à noter qu'il faut spécifier une tolérance adéquate pour le calcul de l'écoulement de puissance de manière à avoir des données aussi précises que possible. A partir de ce point de fonctionnement, une simulation de stabilité transitoire sans aucun défaut doit donner des courbes constantes.

Il faut spécifier un défaut et sa localisation :

- Court-circuit triphasé au niveau d'un noeud du réseau.
- Ouverture d'une ligne.
- Augmentation brutale de la charge en un noeud du réseau.
- Diminution brutale de la charge en un noeud du réseau.
- Perte d'un groupe de génération.

Il faut spécifier les moments d'apparition et d'élimination de ces défauts ainsi que d'autres paramètres dans la boîte de dialogue du menu **Options | Transient Stability**.



De plus, lorsque le mode d'étude (**Study Mode**) est sur **Transient Stability**, les défauts apparaissent sur le schéma unifilaire comme suit :

Court-circuit triphasé au niveau d'un noeud du réseau.

Ouverture d'une ligne.

Augmentation brutale de la charge en un noeud du réseau.

Diminution brutale de la charge en un noeud du réseau.

Perte d'un groupe de génération.

Il faut aussi spécifier les noeuds dont les machines seront surveillées (**Monitored Buses**). Il suffit de sélectionner les noeuds à surveiller et d'activer la commande **Simulation | Transient Stability | Monitor Bus machines** (voir l'Aide de **POWER DESIGNER**).

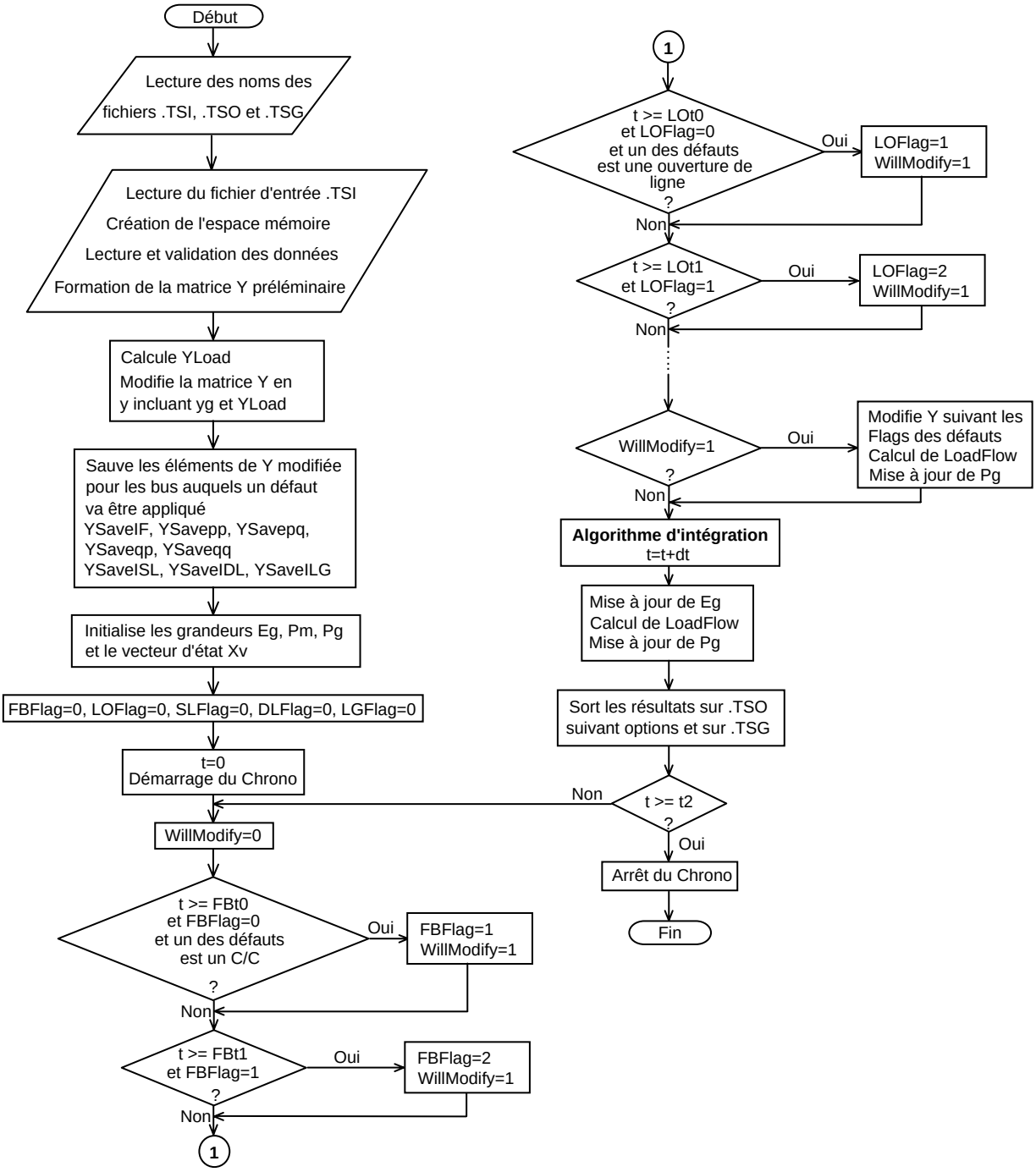
4. Les programmes

Nous avons développé trois programmes de calcul de stabilité transitoire. Comme leurs équivalents de l'écoulement de puissance, ils sont appelés par **POWER DESIGNER** à partir de son menu et s'exécutent dans une session DOS.

Ces programmes se différencient par l'algorithme de la méthode itérative utilisée dans la résolution des équations différentielles.

- **TSP** : Méthode d'Euler modifiée [7].
- **TSPT** : Méthode du prédicteur-correcteur (trapézoïdale).
- **RKP** : Méthode de Runge-Kutta du 4ème ordre.

Dans ces trois programmes, le calcul de l'écoulement de puissance pour la mise à jour des tensions lors des itérations se fait par la méthode de Gauss-Seidel.
Il est facile d'ajouter le terme yg[i]*Eg[i] dans le calcul de la tension au noeud **i**, car c'est juste une sommation qu'on introduit pour les noeuds PV (qui ont une machine) et qu'on élimine pour les noeuds PQ et le noeud PV qui a perdu momentanément son générateur.



Organigramme des programmes de stabilité transitoire

4.1. Méthode d'Euler modifiée (TSP)

La lecture du fichier d'entrée se fait de la même manière que dans les modules d'écoulement de puissance.
Cependant, certaines données ne sont pas utilisées dans la simulation de stabilité transitoire (Qgmin, Qgmax, Vsch, Smax).
D'autres données, par contre, sont utilisées pour obtenir le point initial (Sg=Pg+jQg et V=Vmag$\angle\delta$, voir le fichier d'entrée **ABIAD.TSI** de la simulation 5.1). Ces données proviennent de la simulation d'écoulement de puissance.

L'algorithme d'intégration utilisé dans TSP est celui d'Euler modifié [7]. Il se présente en deux parties :

- La partie prédicteur :

```
for ( int i=0; i<=Npv; i++)
  {
    dXvp[i]=Xv[Npv1+i]-Ws;
    dXvp[Npv1+i]=Ws/2/H[i]*( Pm[i]-Pg[i]-D[i]*dXvp[i]);
    Xvp[i]=Xv[i]+dXvp[i]*dt;
    Xvp[Npv1+i]=Xv[Npv1+i]+dXvp[Npv1+i]*dt;
  }
```

- La partie correcteur :

Après avoir effectué une mise à jour de Eg, un calcul d'écoulement de puissance, puis une mise à jour de Pg, nous calculons le vecteur Xv :

```
for ( i=0; i<=Npv; i++)
  {
    dXvc[i]=Xvp[Npv1+i]-Ws;
    dXvc[Npv1+i]=Ws/2/H[i]*( Pm[i]-Pg[i]-D[i]*dXvc[i]);
    Xv[i]+=( dXvp[i]+dXvc[i] )*dt/2;
    Xv[Npv1+i]+=( dXvp[Npv1+i]+dXvc[Npv1+i] )*dt/2;
    Eg[i]=polar( abs( Eg[i]), Xv[i] );
  }
```

4.2. Méthode du prédicteur-correcteur (TSPT)

La principale différence avec le programme TSP réside dans l'algorithme d'intégration. Nous utilisons dans TSPT l'algorithme prédicteur-correcteur (trapézoïdal).

Au lieu de ne faire qu'un seul passage dans la correction (méthode d'Euler modifiée), nous effectuons autant d'itérations que nécessaire de manière à rendre l'erreur inférieure à la tolérance spécifiée.

La partie correcteur devient :

```
for ( i=0; i<=Npv; i++)
 {
 do   {
      a=0;  // Flag de sortie
      dXvc[i]=Xvp[Npv1+i]-Ws;
      dXvc[Npv1+i]=Ws/2/H[i]*( Pm[i]-Pg[i]-D[i]*dXvc[i]);
      Xvc[i]=Xv[i]+( dXvp[i]+dXvc[i] )*dt/2;
      Xvc[Npv1+i]=Xv[Npv1+i]+( dXvp[Npv1+i]+dXvc[Npv1+i] )*dt/2;

      if ( fabs( Xvp[i]-Xvc[i])<=Tol )  a++;
      if ( fabs( Xvp[Npv1+i]-Xvc[Npv1+i])<=Tol && a)  break;

      Xvp[i]=Xvc[i];
      Xvp[Npv1+i]=Xvc[Npv1+i];
      Eg[i]=polar( abs( Eg[i]), Xvp[i] );
      LFGauss();
      Ig=( Eg[i]-polar( Vmag[i], delta[i]) )*yg[i];
      if ( LGFlag==1 && i==ILG ) Ig=0;  // Gen debranche
      Pg[i]=real( Eg[i]*conj( Ig) );
      } while(1);
 // apres convergence de l'arg et de la vitesse
 Xv[i]=Xvc[i];
 Xv[Npv1+i]=Xvc[Npv1+i];
 }
```

4.3. Méthode de Runge-Kutta (RKP)

Dans ce programme, l'algorithme d'intégration utilisé est celui de Runge-Kutta du 4ème ordre. C'est l'algorithme le plus précis des trois, mais il est aussi le plus gourmand en temps de calcul.

```
// Boucle K1
   for ( i=0; i<=Npv; i++)
      {
      dXvp[i]=Xv[Npv1+i]-Ws;
      dXvp[Npv1+i]=Ws/2/H[i]*( Pm[i]-Pg[i]-D[i]*dXvp[i]);
      K1[i]=dXvp[i]*dt;
      K1[Npv1+i]=dXvp[Npv1+i]*dt;
      Xvp[i]=Xv[i]+K1[i]/2;
      Xvp[Npv1+i]=Xv[Npv1+i]+K1[Npv1+i]/2;
      Eg[i]=polar( abs( Eg[i]), Xvp[i] );
      }
   LFGauss();
   CalcPg();

// Boucle K2
   for ( i=0; i<=Npv; i++)
      {
```

```
      dXvp[i]=Xvp[Npv1+i]-Ws;
      dXvp[Npv1+i]=Ws/2/H[i]*( Pm[i]-Pg[i]-D[i]*dXvp[i]);
      K2[i]=dXvp[i]*dt;
      K2[Npv1+i]=dXvp[Npv1+i]*dt;
      Xvp[i]=Xv[i]+K2[i]/2;
      Xvp[Npv1+i]=Xv[Npv1+i]+K2[Npv1+i]/2;
      Eg[i]=polar( abs( Eg[i]), Xvp[i] );
      }
   LFGauss();
   CalcPg();

// Boucle K3
   for ( i=0; i<=Npv; i++)
      {
      dXvp[i]=Xvp[Npv1+i]-Ws;
      dXvp[Npv1+i]=Ws/2/H[i]*( Pm[i]-Pg[i]-D[i]*dXvp[i]);
      K3[i]=dXvp[i]*dt;
      K3[Npv1+i]=dXvp[Npv1+i]*dt;
      Xvp[i]=Xv[i]+K3[i];
      Xvp[Npv1+i]=Xv[Npv1+i]+K3[Npv1+i];
      Eg[i]=polar( abs( Eg[i]), Xvp[i] );
      }
   LFGauss();
   CalcPg();

// Boucle K4 et fin
   for ( i=0; i<=Npv; i++)
      {
      dXvp[i]=Xvp[Npv1+i]-Ws;
      dXvp[Npv1+i]=Ws/2/H[i]*( Pm[i]-Pg[i]-D[i]*dXvp[i]);
      K4[i]=dXvp[i]*dt;
      K4[Npv1+i]=dXvp[Npv1+i]*dt;
      //UpDate Xv
      Xv[i]+=( K1[i]+2*( K2[i]+K3[i] )+K4[i] )/6;
      Xv[Npv1+i]+=(K1[Npv1+i]+2*(K2[Npv1+i]+K3[Npv1+i])+K4[Npv1+i])/6;
      Eg[i]=polar( abs( Eg[i]), Xv[i] );
      }
   LFGauss();
   CalcPg();
t+=dt; // On incremente une fois pour toute le t externe
```

5. Applications

Afin d'illustrer les possibilités de notre logiciel dans ce domaine, nous proposons d'effectuer des simulations sur le réseau à 5 noeuds de la référence [7] et le réseau à 9 noeuds de la référence [17].

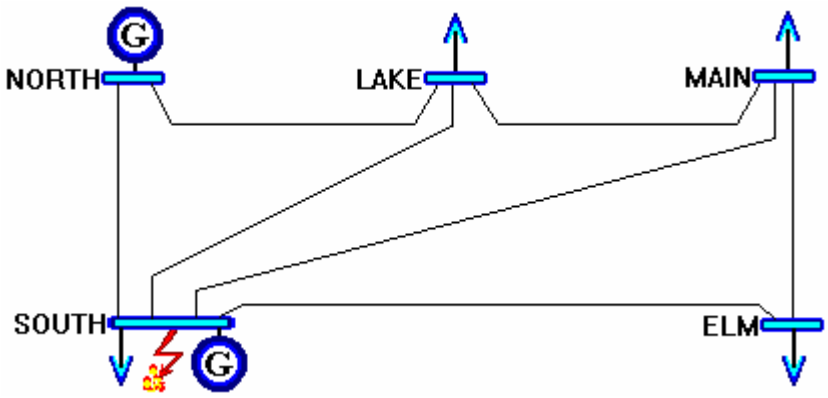
Réseau à 5 noeuds [7]

- **Simulation 1** : Défaut shunt triphasé (court-circuit) d'une durée de 0.1 sec au noeud South.
- **Simulation 2** : Défaut shunt triphasé d'une durée de 0.2 sec au noeud South pour comparaison avec la Simulation 1.
- **Simulation 3** : Défauts simultanés :
  Augmentation brutale de la charge de ΔP=0.2 p.u. entre t=0 s et t=0.1 s au noeud Lake.
  Ouverture de ligne entre les noeuds South et Lake à t=0.05 s, refermeture à t=0.2 s.
  Simulation sur une durée de 3 s.
- **Simulation 4** : Simulation 3 mais avec les trois programmes de calcul pour une comparaison des méthodes d'intégration.

Réseau à 9 noeuds [17]

- **Simulation 5** : Défaut shunt triphasé (court-circuit) d'une durée de 0.1 sec au noeud Ouest.
  Simulation sur une durée de 3 s.

5.1. Simulation 1 : C/C 3ϕ au noeud South, durée 0.1s



ABIAD.SCH

Schéma du réseau à 5 noeuds tel qu'il apparaît dans POWER DESIGNER dans le mode Transient Stability

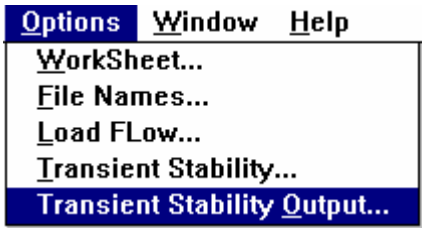
La commande **Simulation | Make Data** permet de générer le fichier **ABIAD.TSI** et l'affiche sur le Bloc Note :

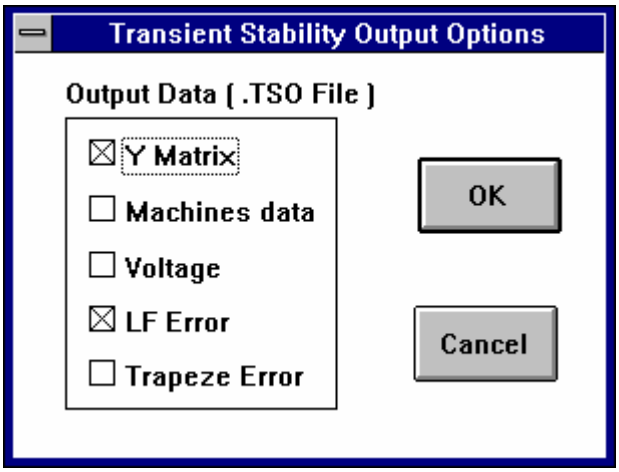
```
5 1
8 ----------  Input file for the Transient Stability program  ----------
8 --- Number of total Buses = 5  PV Buses = 1
8
8
8 ---------------------------  Bus Data  ------------------------------
8  BUS          Pg       Qg       Pl       Ql       Vb
1         NORTH  1.29570  -0.07489  0.00000  0.00000  1.06000
1         SOUTH  0.40000  0.30040  0.20000  0.10000  1.04747
1          LAKE  0.00000  0.00000  0.45000  0.15000  1.00000
1          MAIN  0.00000  0.00000  0.40000  0.05000  1.00000
1           ELM  0.00000  0.00000  0.60000  0.10000  1.00000
8 ----------------------------  Line Data  ----------------------------
8  BUS          BUS          Rs       Xs       Gs       Bs       Smax
3         NORTH          LAKE  0.08000  0.24000  0.00000  0.02500  0.00000
3          LAKE          MAIN  0.01000  0.03000  0.00000  0.01000  0.00000
3          MAIN           ELM  0.08000  0.24000  0.00000  0.02500  0.00000
3         NORTH         SOUTH  0.02000  0.06000  0.00000  0.03000  0.00000
3         SOUTH          LAKE  0.06000  0.18000  0.00000  0.02000  0.00000
3         SOUTH          MAIN  0.06000  0.18000  0.00000  0.02000  0.00000
3         SOUTH           ELM  0.04000  0.12000  0.00000  0.01500  0.00000
8 ------------------------  Transformer Data  -------------------------
8  BUS          BUS          Rs       Xs       Amag     A(deg)  Smax
8 -------------------------  Capacitor Data  --------------------------
8  BUS          Bc
8 ----------------------  Acceleration Factor GS  ---------------------
9   1.6000000000E+00
8 ----------------  Bus voltage from Load Flow study  ------------------
8  Bus          mag      arg(deg)
11        NORTH  1.06000  0.00000
11        SOUTH  1.04747  -2.80649
11         LAKE  1.02421  -4.99667
11         MAIN  1.02361  -5.32873
11          ELM  1.01798  -6.14962
8 ----------------------  Machine Characteristics  --------------------
8  Bus          H         D         R         Xdp
16        NORTH  50.00000   0.00000  0.00000  0.25000
16        SOUTH   1.00000   0.00000  0.00000  1.50000
8 -------------------------  Monitored Buses  -------------------------
8  Bus
17        SOUTH
17        NORTH
8 --------------------------  3 Phase Faulty Bus  ----------------------
8  Bus          Start   End     Gf        Bf
18        SOUTH  0.0000  0.1000   1.0E+20   1.0E+20
8 ------------------------  Simulation Parameters  ---------------------
8  dt      End     Nimax Tol       eps       Fault freq
22 0.0200  0.9000    25   1.0E-03   1.0E-03     1   60.00
8 ----------------  Transient Stability Output Options  ----------------
23 0
8 ------------------------  END OF INPUT FILE  -------------------------
```

La commande **Simulation | Transient Stability | Modified Euler Algorithm** appelle le programme TSP.EXE avec comme paramètres : ABIAD.TSI ABIAD.TSO ABIAD.TSG.

Une fois la simulation finie, la commande **Simulation | Output Data** affiche le fichier **ABIAD.TSO** dans le Bloc Note.

Ce fichier reporte selon les options définies dans la boîte de dialogue **Transient Stability Output Options.** L'accès à cette boîte de dialogue se fait par la commande **Options | Transient Stability Output** :





Boîte de dialogue des options de sortie sur fichier de la simulation de stabilité transitoire

La commande **Simulation | Transient Stability | Plot Results** appelle le module de récapitulation graphique **GraphD** et lui transmet comme paramètre le nom de fichier **ABIAD.TSG** qui contient l'évolution des grandeurs sous forme de tableau.

On peut aussi utiliser le logiciel **Grapher** pour afficher ces courbes et les imprimer, car le fichier **.TSG** que notre logiciel **POWER DESIGNER** génère est standard et peut être lu par tous les tableurs.

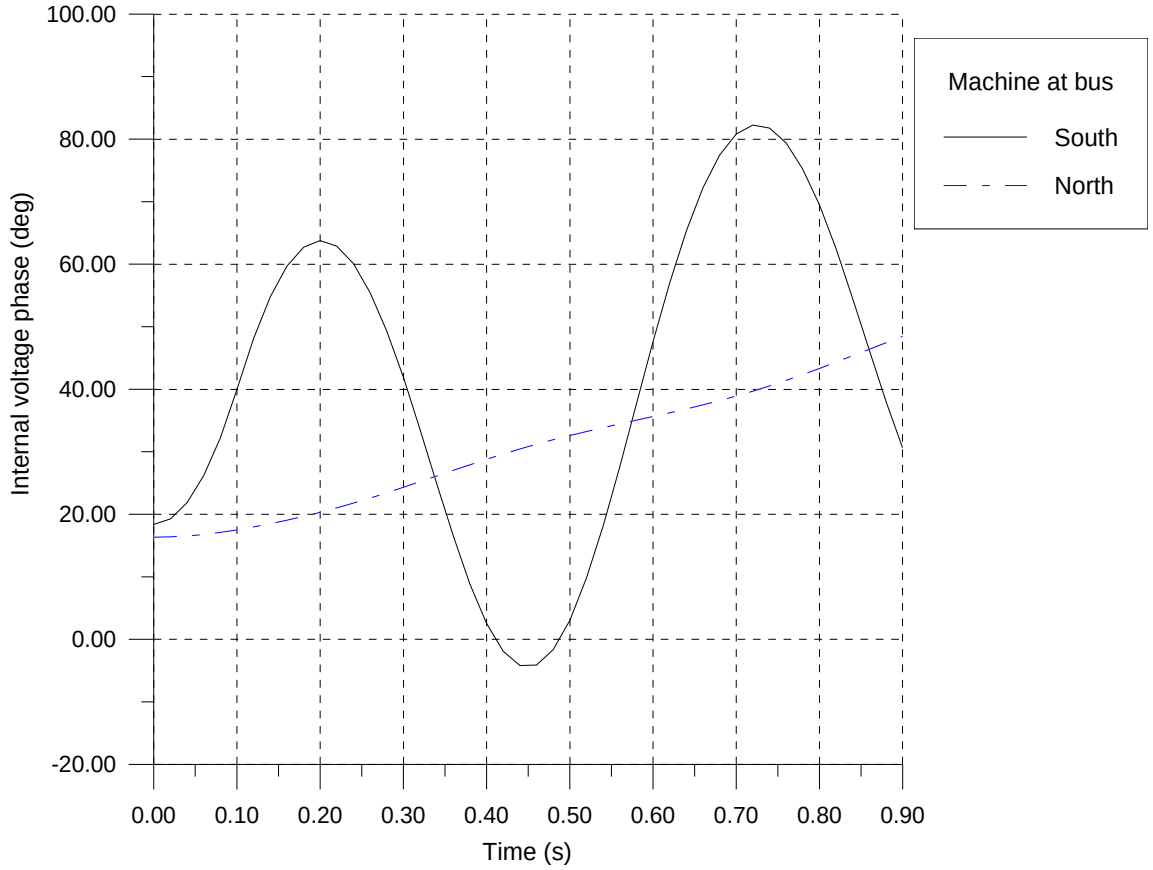


Figure 4.1 Phases de la f.e.m. des machines durant la simulation 1

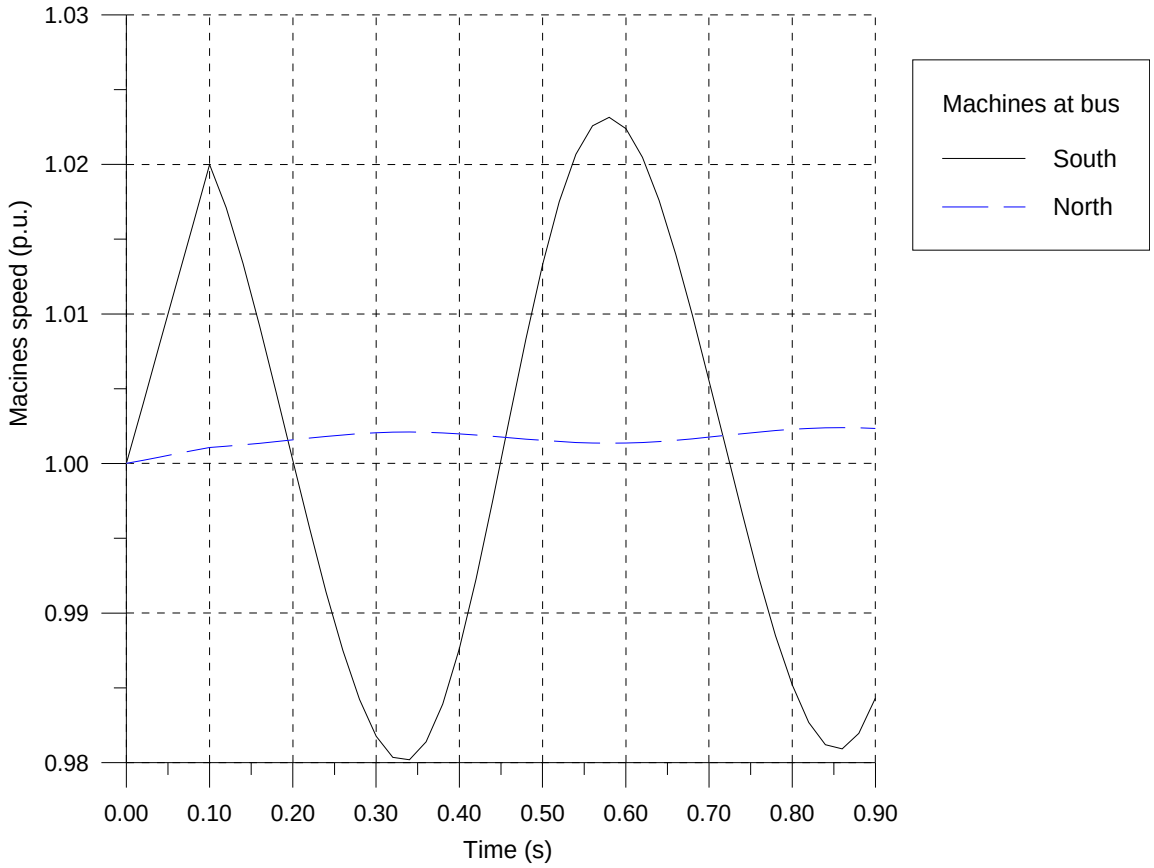


Figure 4.2 Vitesses des machines durant la simulation 1

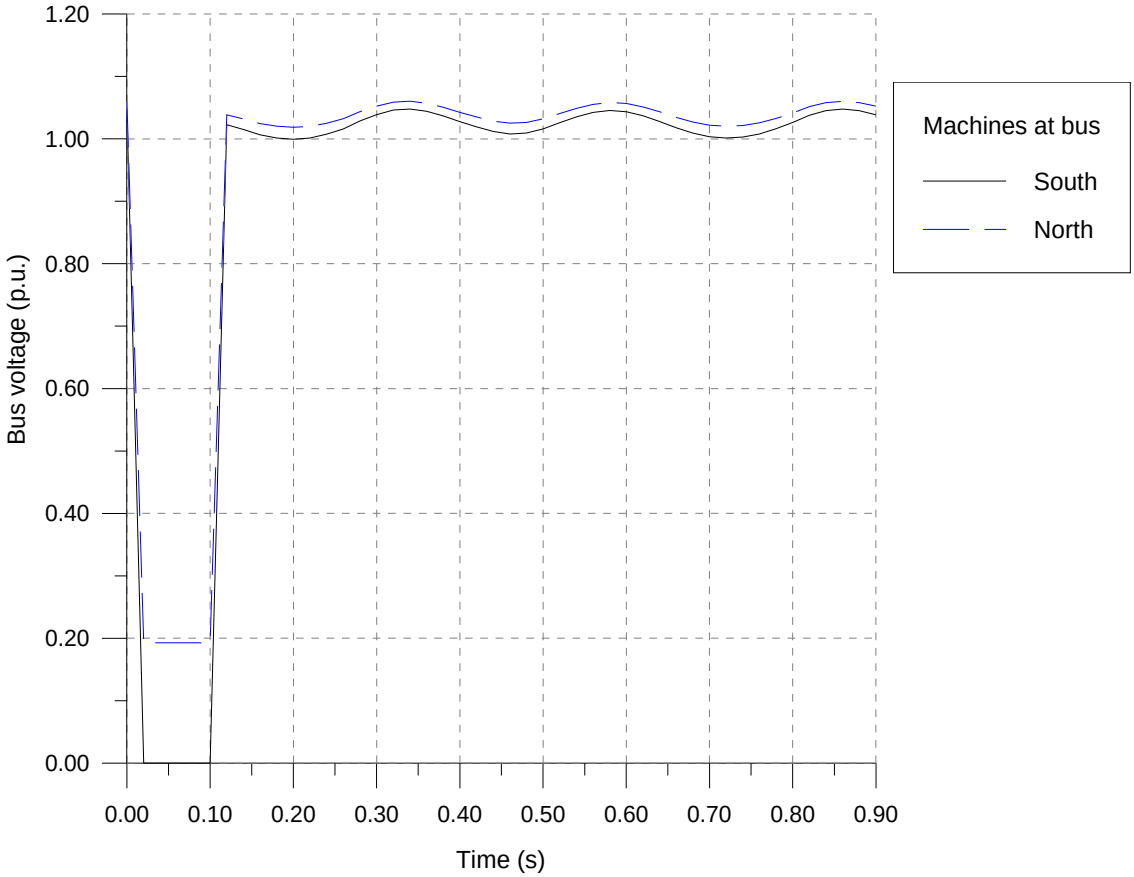


Figure 4.3 Tensions aux noeuds des machines durant la simulation 1

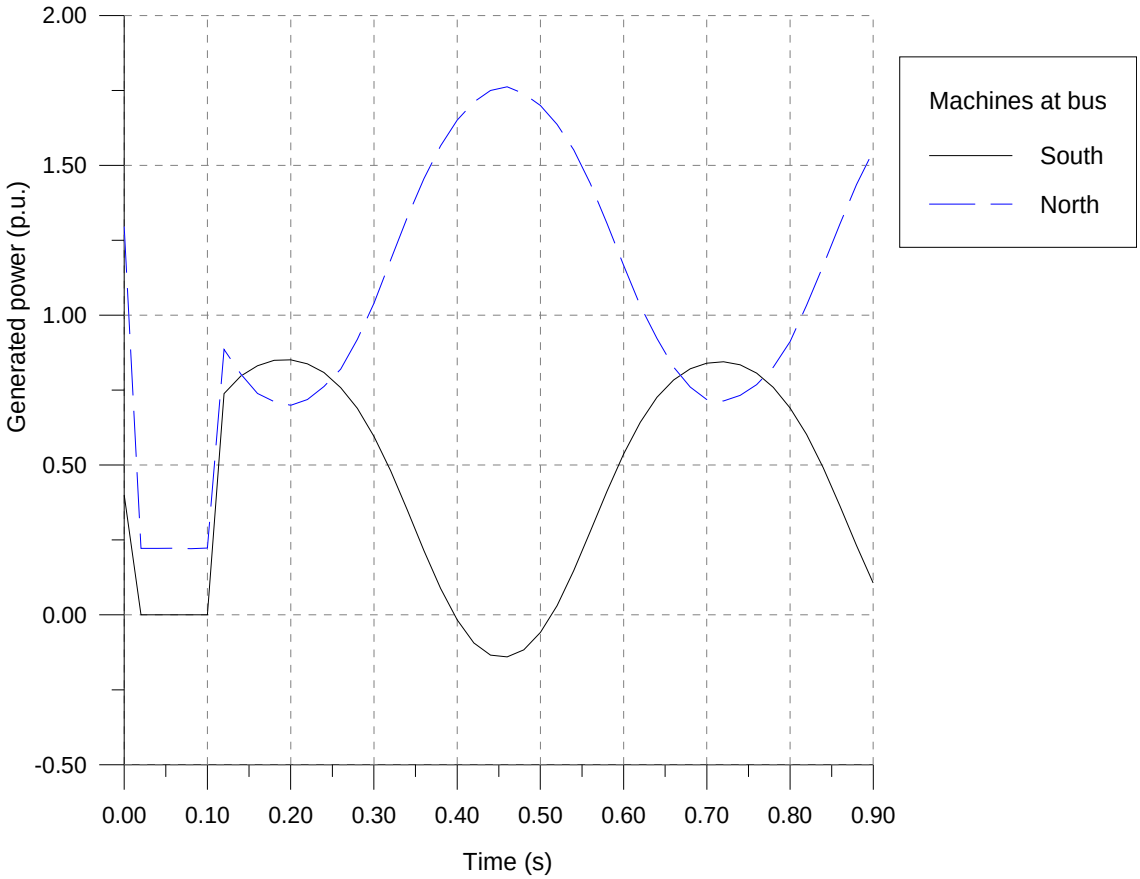


Figure 4.4 Puissances générées durant la simulation 1

5.2. Simulation 2 : C/C 3ϕ au noeud South, durée 0.2s

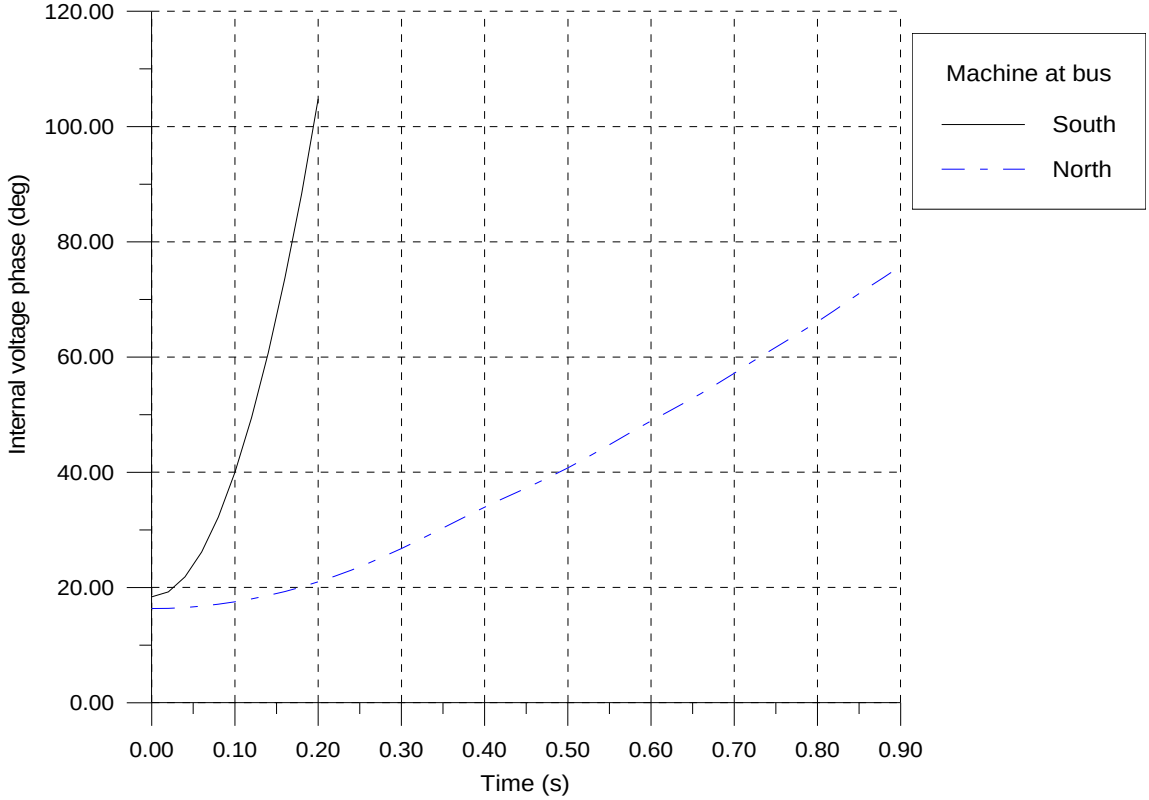


Figure 4.5 Phases de la f.e.m. des machines durant la simulation 2

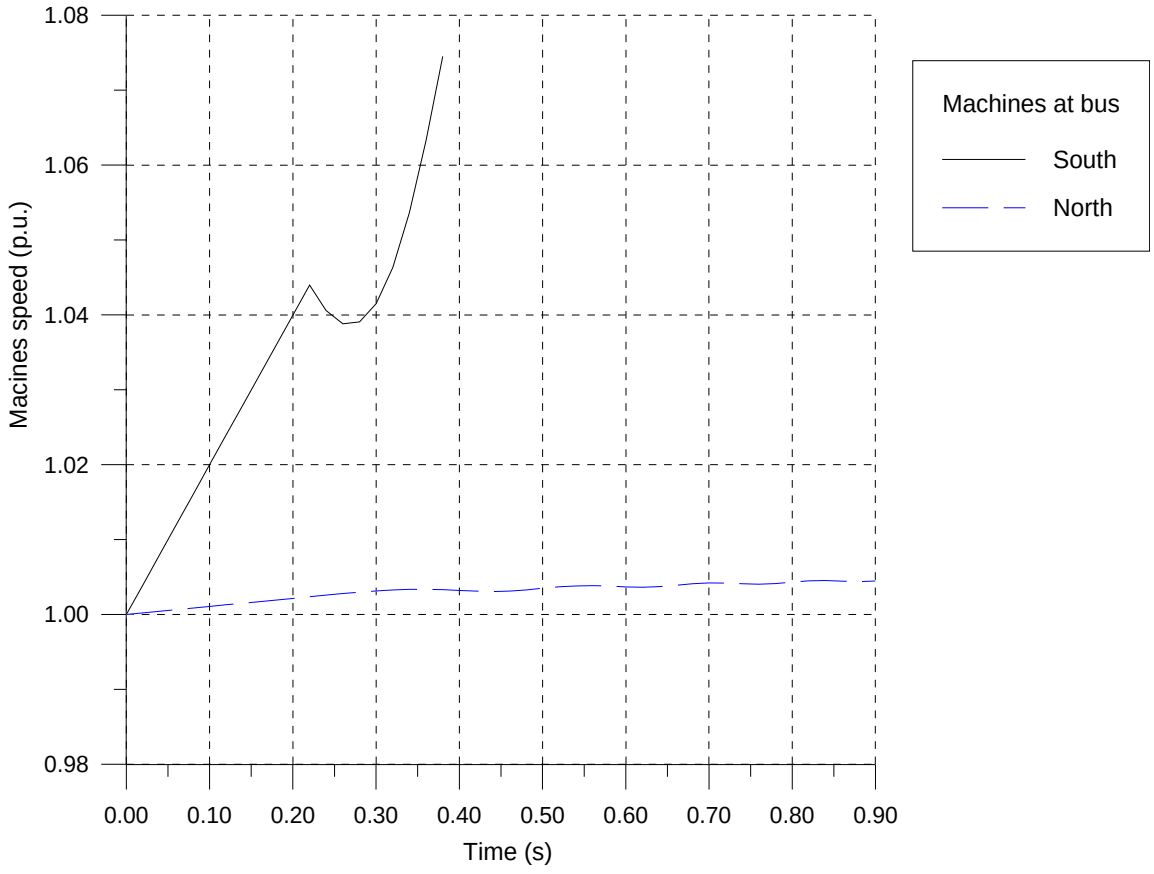


Figure 4.6 Vitesses des machines durant la simulation 2

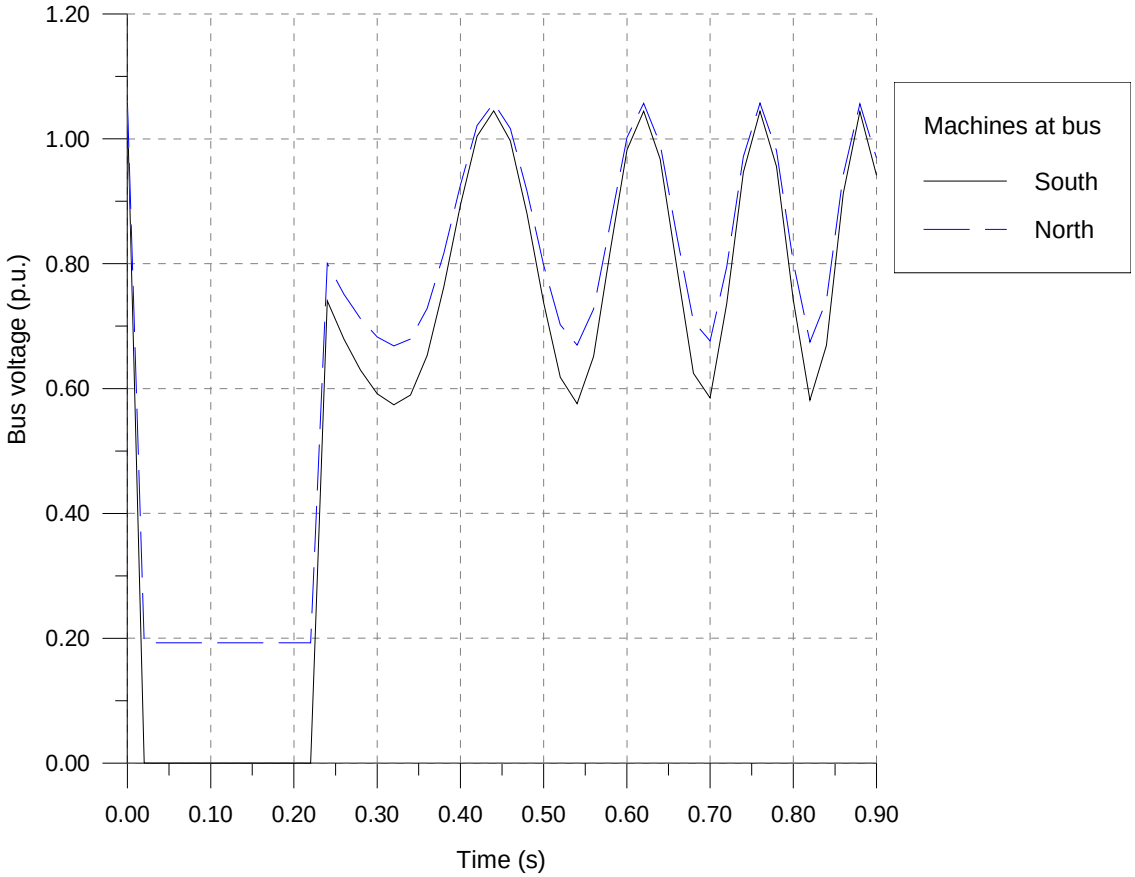


Figure 4.7 Tensions aux noeuds des machines durant la simulation 2

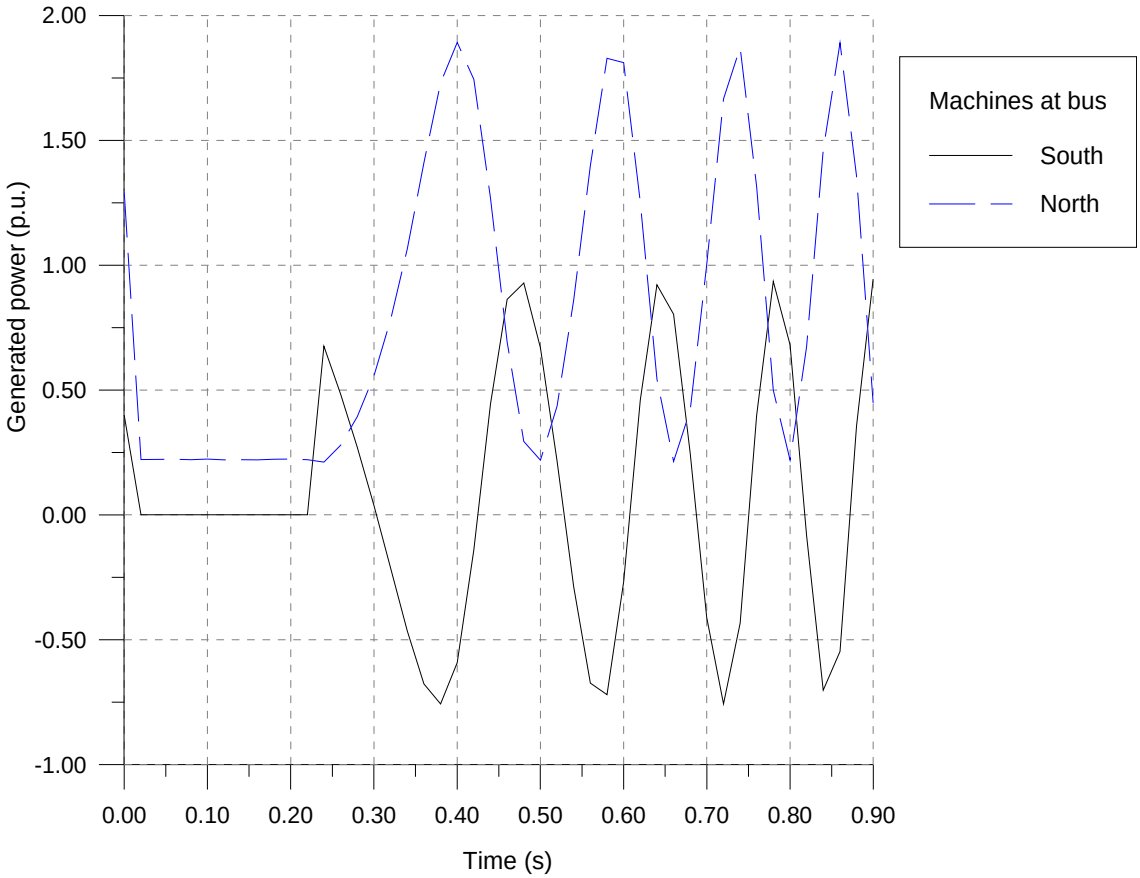
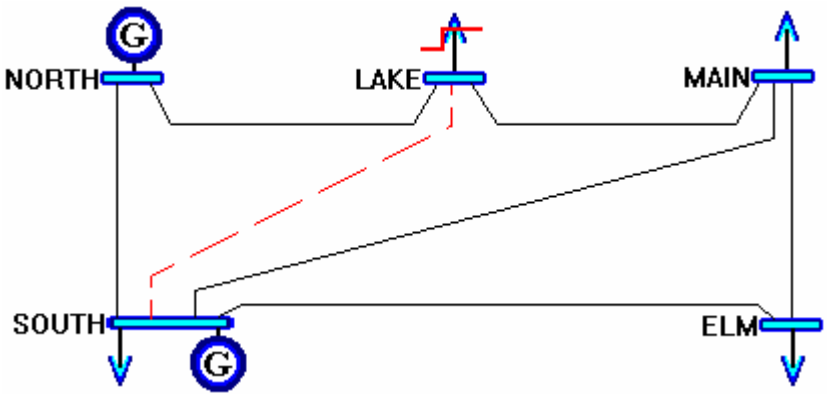


Figure 4.8 Puissances générées durant la simulation 2

5.3. Simulation 3 : défauts simultanés



ABIAD.SCH

Schéma du réseau à 5 noeuds tel qu'il apparaît dans POWER DESIGNER dans le mode Transient Stability pour ces défauts simultanés

```
8 ----------------------------  Opened Line  --------------------------
8  BUS          BUS          Start   End
19        SOUTH          LAKE  0.0500  0.2000
8  BUS          BUS          Rs       Xs       Gs       Bs
25        SOUTH          LAKE  0.06000  0.18000  0.00000  0.02000
8 -------------------------  Increased Bus Load  ----------------------
8  Bus          Start   End     Pl       Ql
20         LAKE  0.0000  0.1000  0.20000  0.00000
8 ------------------------  Simulation Parameters  ---------------------
8  dt       End      Nimax Tol       eps       Fault freq
22 0.0200  3.0000    25   1.0E-03   1.0E-03     6   60.00
```

Partie du fichier ABIAD.TSI correspondant à la simulation 3

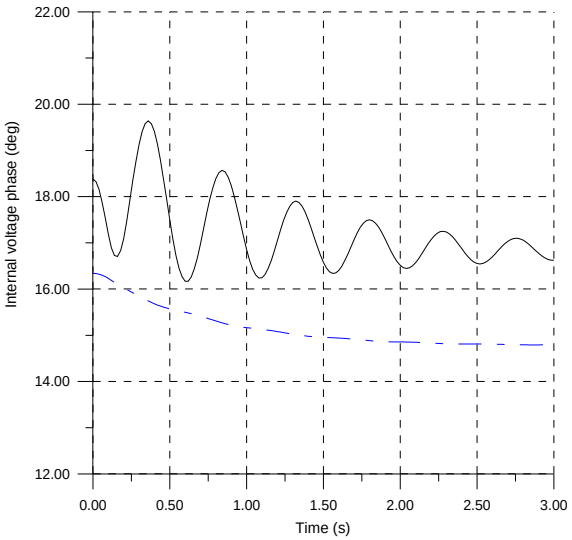


Figure 4.9 Phases de la f.e.m. des machines durant la simulation 3

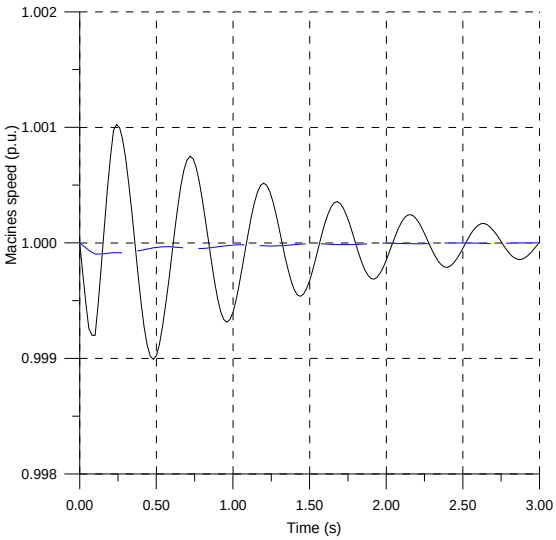


Figure 4.10 Vitesses des machines durant la simulation 3

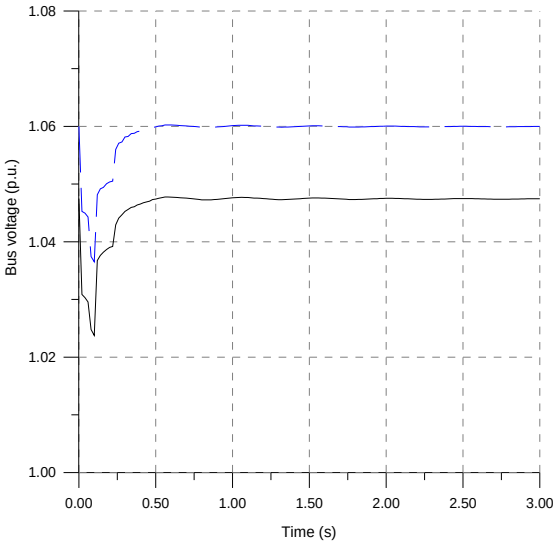


Figure 4.11 Tensions aux noeuds des machines durant la simulation 3

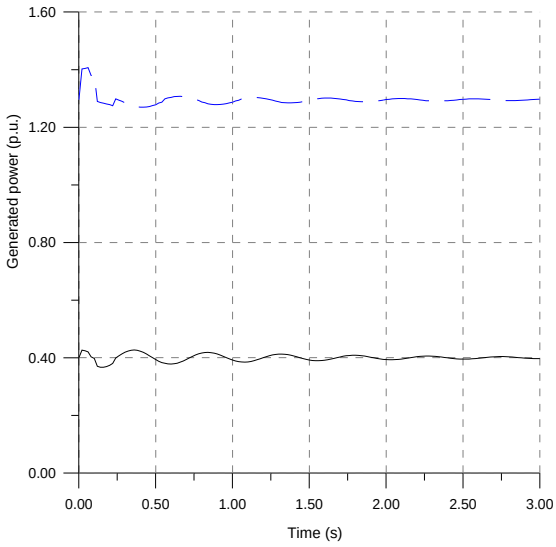
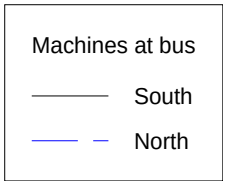


Figure 4.12 Puissances générées durant la simulation 3



5.4. Simulation 4 : Comparaison des méthodes d'intégration

Dans cette simulation nous avons essayé les trois programmes TSP, TSPT et RKP sur le même exemple (Simulation 3) afin de comparer les algorithmes d'intégration utilisés dans ces programmes.

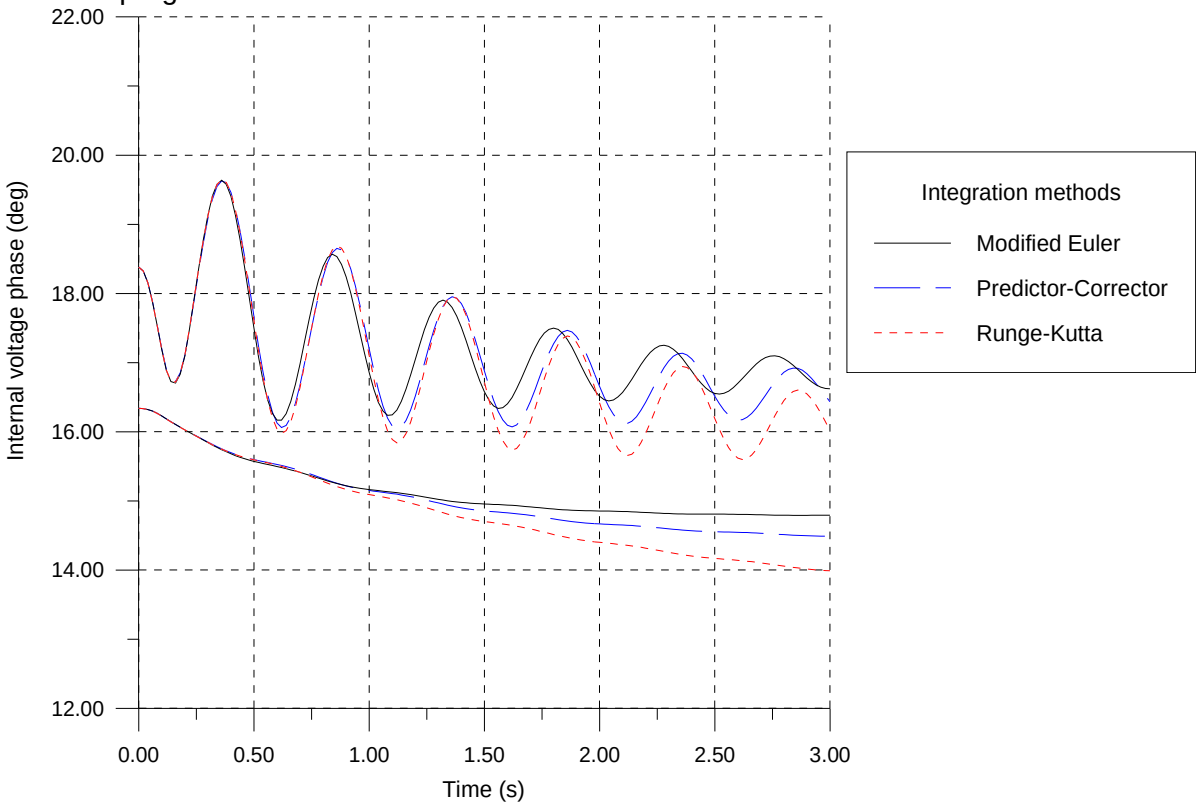
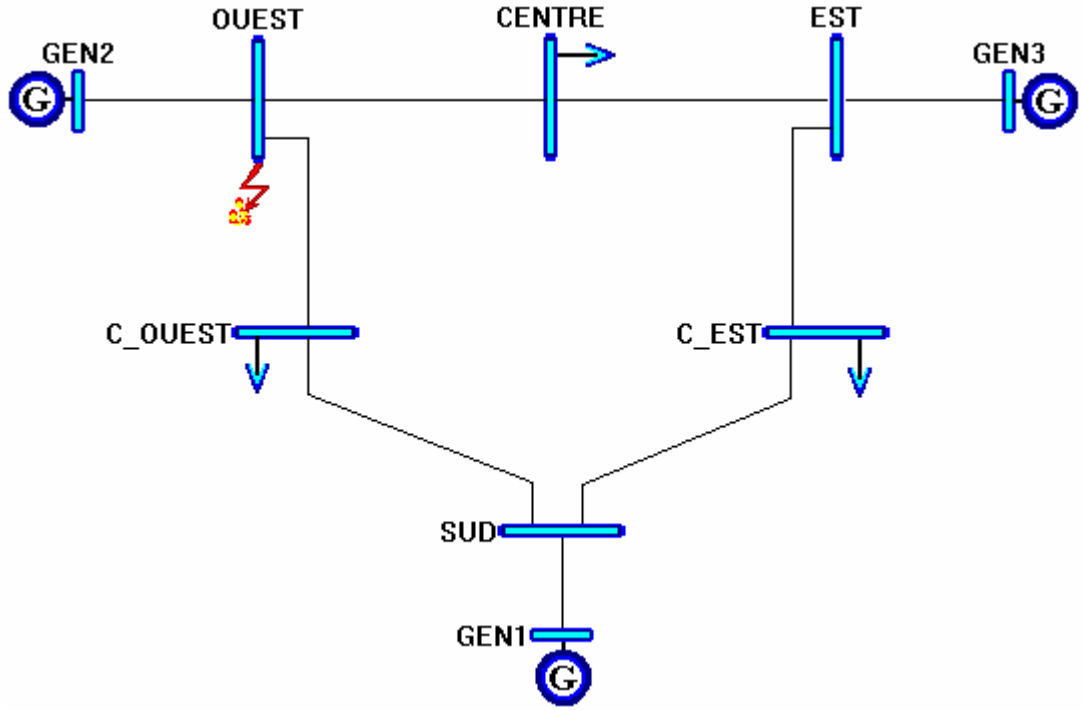


Figure 4.13 Phases de la f.e.m. des machines durant la simulation 4 Comparaison des méthodes d'intégration

5.5. Simulation 5 : C/C 3ϕ au noeud Ouest du réseau à 9 noeuds



TRANS.SCH

Schéma du réseau à 9 noeuds tel qu'il apparaît dans POWER DESIGNER dans le mode Transient Stability

Le fichier **TRANS.TSI** est illustré par l'affichage suivant :

```
9 2
8 ----------  Input file for the Transient Stability program  ----------
8 --- Number of total Buses = 9  PV Buses = 2
8
8 ----------------------------  Bus Data  -------------------------------
8  BUS          Pg       Qg       Pl       Ql       Vb
1         GEN1  0.56453  0.72072  0.00000  0.00000  1.00000
1         GEN2  1.65000  0.55809  0.00000  0.00000  1.00000
1         GEN3  1.00000  0.40000  0.00000  0.00000  1.00000
1        OUEST  0.00000  0.00000  0.00000  0.00000  1.00000
1       CENTRE  0.00000  0.00000  1.00000  0.35000  1.00000
1      C_OUEST  0.00000  0.00000  1.25000  0.50000  1.00000
1          EST  0.00000  0.00000  0.00000  0.00000  1.00000
1        C_EST  0.00000  0.00000  0.90000  0.30000  1.00000
1          SUD  0.00000  0.00000  0.00000  0.00000  1.00000
8 ----------------------------  Line Data  -----------------------------
8  BUS          BUS          Rs       Xs       Gs       Bs       Smax
3         GEN2         OUEST  0.00000  0.06250  0.00000  0.00000  0.00000
3        OUEST        CENTRE  0.00850  0.07200  0.00000  0.00745  0.00000
3       CENTRE           EST  0.01190  0.10080  0.00000  0.01045  0.00000
3         GEN3           EST  0.00000  0.05860  0.00000  0.00000  0.00000
3         GEN1           SUD  0.00000  0.05760  0.00000  0.00000  0.00000
3      C_OUEST         OUEST  0.03200  0.16100  0.00000  0.01530  0.00000
```

```
3        C_EST          EST  0.03900  0.17000  0.00000  0.01790  0.00000
3         SUD      C_OUEST  0.01000  0.08500  0.00000  0.00880  0.00000
3         SUD        C_EST  0.01700  0.09200  0.00000  0.00790  0.00000
8 ------------------------  Transformer Data  -------------------------
8  BUS          BUS          Rs       Xs       Amag     A(deg)  Smax
8 -------------------------  Capacitor Data  --------------------------
8  BUS          Bc
8 ----------------------  Acceleration Factor GS  ---------------------
9   1.0000000000E+00
8 ----------------  Bus voltage from Load Flow study  ------------------
8  Bus          mag      arg(deg)
11        GEN1  1.00000  0.00000
11        GEN2  1.00000  11.81563
11        GEN3  1.00000  7.70589
11       OUEST  0.97061  5.71660
11      CENTRE  0.95291  2.61985
11     C_OUEST  0.91584  -3.58590
11         EST  0.97832  4.27189
11       C_EST  0.93261  -2.95344
11         SUD  0.95904  -1.94303
8 ----------------------  Machine Characteristics  ---------------------
8  Bus          H        D        R        Xdp
16        GEN2   3.33330   0.07000  0.00000  0.11980
16        GEN1   9.55150   0.00000  0.00000  0.06080
16        GEN3   2.35160   0.05000  0.00000  0.18130
8 --------------------------  Monitored Buses  -------------------------
8  Bus
17        GEN1
17        GEN2
17        GEN3
8 --------------------------  3 Phase Faulty Bus  ----------------------
8  Bus          Start   End     Gf        Bf
18       OUEST  0.0000  0.1000   1.0E+20   1.0E+20
8 ------------------------  Simulation Parameters  ----------------------
8  dt      End     Nimax Tol       eps       Fault freq
22 0.0200  0.9000    25   1.0E-03   1.0E-03     1   50.00
8 ----------------  Transient Stability Output Options  ----------------
23 0
8 ------------------------  END OF INPUT FILE  -------------------------
```

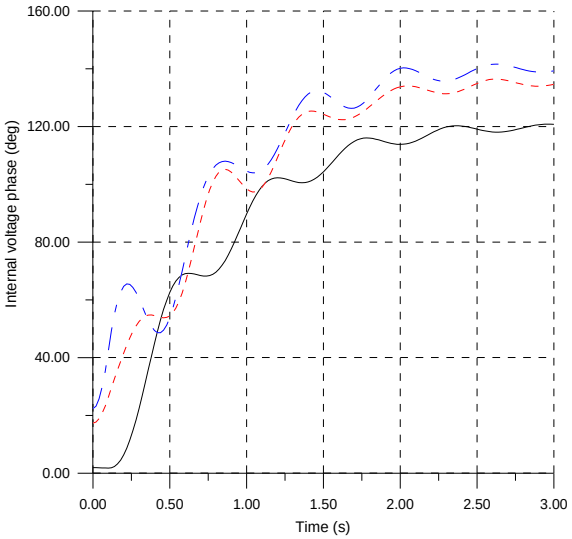


Figure 4.14 Phases de la f.e.m. des machines durant la simulation 5

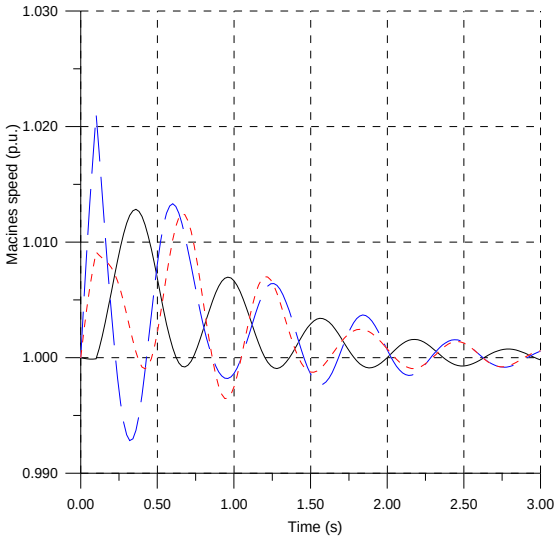


Figure 4.15 Vitesses des machines durant la simulation 5

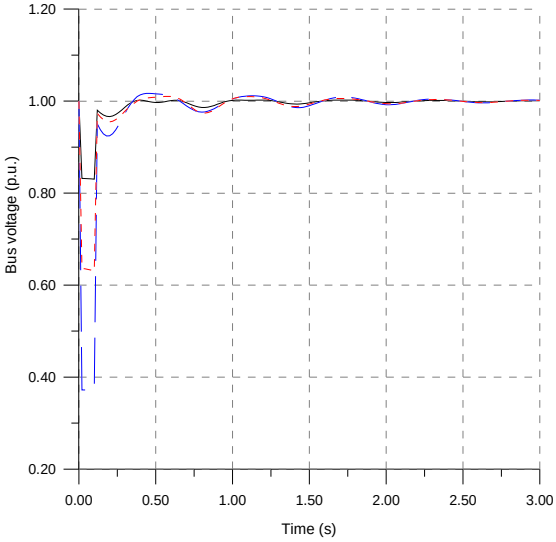


Figure 4.16 Tensions aux noeuds des machines durant la simulation 5

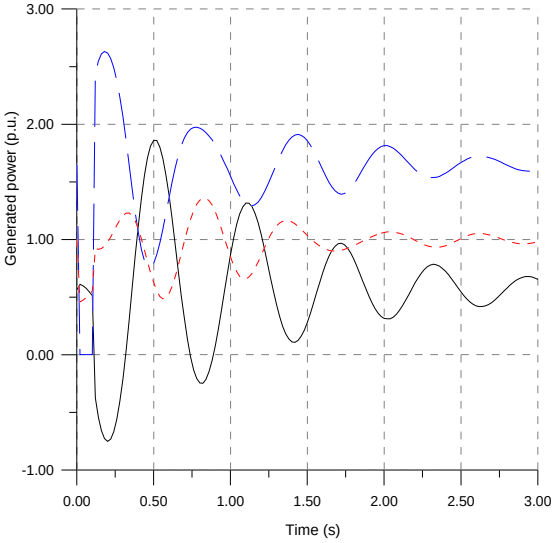
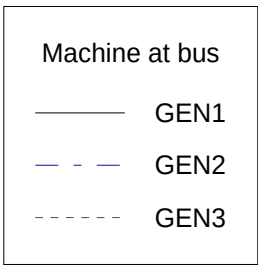


Figure 4.17 Puissances générées durant la simulation 5



6. Discussions

Nous remarquons, tout d'abord, que pour l'ensemble des simulations sur le réseau à 5 noeuds, la machine du noeud North est nettement moins affectée que celle du noeud South.
Ceci est dû principalement à deux raisons :

- La machine du noeud North a une constante **H** (représentant l'inertie de la machine) beaucoup plus grande que celle de la machine du noeud South.
- Pour les simulations 1 et 2 (C/C 3ϕ), il est clair qu'étant donné que le défaut est appliqué au noeud South, c'est sa machine qui est la plus perturbée. Si le noeud North était relié par plusieurs lignes (à travers plusieurs noeuds) au noeud South , il serait encore moins affecté.

En fait, les noeuds du réseau se soutiennent, chacun avec une pondération qui dépend étroitement des paramètres de sa machine et de la "puissance" de la liaison avec le noeud de défaut (impédance des lignes, nombre de noeuds intermédiaires...).

Entre les défauts shunt et les défauts série, nous remarquons que le réseau est très peu affecté par une ouverture de ligne par rapport à un défaut shunt (surtout s'il s'agit d'impédance très basse, voire un court-circuit franc).

En effet, le défaut triphasé engendre une plus grande perturbation sur les machines. Celles-ci accélèrent plus durant le défaut et l'amplitude des oscillations est plus importante.

Physiquement, lors d'un court-circuit, la puissance électrique aux bornes de la machine chute et le couple électromagnétique aussi; il ne s'oppose plus en totalité au couple mécanique de la turbine et la machine accélère pondérée par son moment d'inertie.
L'énergie cinétique acquise pendant toute la durée du défaut va devoir être dissipée à travers le réseau pour revenir au régime permanent stable.

C'est donc la durée du défaut qui joue le rôle le plus important quand à la réaction des machines (et donc du réseau) vis-à-vis d'une perturbation [7].

La comparaison des simulations 1 et 2 met en évidence l'importance de la durée du défaut sur la perte du synchronisme.

Les courbes sont identiques pour (0.<= t <= 0.1 sec). Passé ce point, le défaut est éliminé dans la simulation 1 et le système entreprend des oscillations; la *puissance synchronisante* empêche la perte du synchronisme. Le réseau est stable pour cette perturbation et *à fortiori* pour les défauts moins sévères (bi et monophasés).

Par contre, lors de la simulation 2, le défaut n'est éliminé qu'à (t=0.2 sec) et les machines ont accumulé une telle énergie cinétique, qu'il leur est impossible de revenir à la stabilité après la disparition du défaut et c'est le décrochage. Les protections doivent donc être dimensionnées de manière à intervenir avant d'en arriver là.

Nous remarquons aussi qu'après un régime transitoire pour lequel le réseau est resté stable, les grandeurs Vmag (External Voltage magnitude), Wr (Rotor Speed) et Pg (Generated Power) retournent vers leurs valeurs d'avant défaut. Les phases de Eg (Internal Voltage Phase) de toutes les machines se retrouvent alors toutes décalées d'un certain angle par rapport à la référence. Mais la différence entre les phases de Eg des machines prises deux à deux est la même que celle avant le défaut. Ce qui témoigne d'un retour vers le même état stable. Ceci est surtout visible à travers les simulations 3 et 5 où le temps de simulation a été étendu à 3 secondes afin de mettre en évidence ces caractéristiques.

A travers la simulation 4, nous constatons que durant la première seconde, les courbes des trois algorithmes restent confondues, puis l'écart se creuse entre elles. Ceci est dû aux algorithmes d'intégration numériques. Les erreurs dues aux approximations faites sur les intervalles [t, t+dt] s'accumulent tout au long des itérations et font la différence.

Nous remarquons que la méthode du prédicteur-correcteur est plus précise que celle d'Euler modifiée; sa courbe se situe entre celles d'Euler modifiée et de Runge-Kutta.

La méthode de Runge-Kutta employée dans le programme RKP est celle du 4ème ordre. Elle est la plus précise des trois méthodes du logiciel mais requiert plus de calcul et donc plus de temps.

Il est à noter aussi l'importance du pas de simulation sur la fidélité des courbes calculées. La méthode de Runge-Kutta tolère un pas plus grand que celles d'Euler modifiée et du prédicteur-correcteur pour une même précision.
Ce pas joue aussi le rôle d'une période d'échantillonnage. Les moments de début (t0) et de fin (t1) des perturbations sont pris en compte par les programmes de calcul à un pas de calcul (dt) près. Il convient donc de vérifier les instants de modification de la matrice Y dans le fichier **.TSO**.

Pour les études envisagées avec ce logiciel et qui concernent seulement la stabilité après la première oscillation, nous ne recommandons pas d'aller au delà de la première seconde. En effet, les systèmes de régulation et de contrôle interviennent et les courbes de réponse réelles seraient différentes de celles calculées par ce logiciel.

7. Conclusion

Nous pouvons dire que la méthode du prédicteur-correcteur est recommandable pour sa facilité de programmation et ses relatives précision et rapidité dans l'étude de la stabilité transitoire [6].

En effet, elle se comporte comme la méthode d'Euler modifiée, mais quand le besoin se fait sentir, lors de brusques variations, elle augmente sa précision en augmentant le nombre d'itérations.

Les défauts shunt sont beaucoup plus sévères que les défauts série et les défauts triphasés le sont plus par rapport aux défauts biphasés et ainsi par rapport aux défauts monophasés. Un réseau qui est resté stable après un défaut triphasé le sera aussi pour des défauts mono et biphasés.

La durée du défaut est d'une importance capitale quand au pouvoir du réseau à surmonter la perturbation.

Si le modèle de la machine synchrone utilisé suffit pour une telle étude, il n'en est pas de même s'il faut considérer l'étude de la stabilité après plusieurs oscillations. D'autres paramètres tels que la saturation ou l'introduction de régulateurs (A.V.R. pour Automatic Voltage Regulator, S.G. pour Speed Governor ...) doivent intervenir. Dans ce cas, il faut recourir à d'autres modèles. Des modules de calcul pourraient être développés en ce sens et viendraient s'intégrer à l'environnement de travail **POWER DESIGNER**.

Nous dirons qu'un système de puissance est stable pour une perturbation donnée si l'angle interne (représenté ici par la phase de Eg) revient avant la première seconde.
Si c'est le cas, la stabilité est assurée à plus forte raison quand des systèmes de régulation sont présents.

Conclusion générale

Le travail que nous avons effectué consiste en le développement d'un environnement graphique avec base de données pour l'analyse et la simulation de réseaux électriques.

Le logiciel **POWER DESIGNER** ainsi réalisé permet de saisir un réseau électrique par son schéma unifilaire et d'y réaliser toutes sortes d'opérations d'édition et d'archivage. Il permet aussi d'effectuer les simulations les plus demandées dans l'étude des systèmes de puissance, à savoir, l'écoulement de puissance et la stabilité transitoire.

Par le choix des concepts de développement les plus récents en matière de programmation et d'interface utilisateur (Programmation Orientée Objets et sous Windows), nous avons élaboré un produit agréable, puissant et très extensible.

Dans ce travail, nous avons abordé plusieurs problèmes tels que l'archivage de la base de données des réseaux électriques de puissance, la représentation graphique du schéma unifilaire et de ses composants, l'édition de ces composants et de leur paramètres, la gestion des commandes de l'utilisateur, l'exploitation de la librairie d'objets "Object Windows" et le développement de nouveaux objets plus spécifiques.

Les programmes d'écoulement de puissance déjà existants à notre niveau étaient écrits en Fortran et dépendaient de la dimension du réseau étudié. Ils ne pouvaient pas être exploités par **POWER DESIGNER**. Nous avons alors écrit des programmes en C++ gérant des allocations dynamiques de la mémoire qui leur permettent d'être indépendants du réseau étudié. De plus, ces programmes couvrent quatre méthodes de calcul (Gauss-Seidel, Newton-Raphson, une variante Back Off et le Fast Decoupled Load FLow) et disposent de raffinements en programmation et en sortie qui rendent aisées les opérations de simulation et de changement de paramètres.

Les programmes de simulation de stabilité transitoire sont eux aussi indépendants du réseau et du nombre de ses machines. Ils sont bâtis autour de trois algorithmes d'intégration numérique et permettent de simuler des défauts simultanés dont les moments d'apparition et d'élimination peuvent se chevaucher dans le temps. Un module de récapitulation graphique a aussi été développé afin de visualiser graphiquement et d'imprimer les résultats de ces simulations sous Windows.

Pour des raisons de clarté, nous n'avons pas pu exposer dans ce document toutes les possibilités de **POWER DESIGNER**. Les utilisateurs trouverons plus d'explications sur la manière de l'utiliser ainsi que sur les commandes et les options dans l'Aide sous Windows de **POWER DESIGNER**.

Le projet **POWER DESIGNER** est très extensible, la structure orientée objets des données permet d'ajouter de nouveaux composants de puissance avec leurs représentations graphiques et d'autre modules de simulation. Ainsi, des modèles de machines plus rigoureux tenant compte de la régulation et de la saturation peuvent être introduits. Il est également possible de développer des modules de calcul d'écoulement de puissance optimal, triphasé, d'analyse des défauts, de propagation d'harmoniques... Ces modules s'intégreront comme des sous-menus supplémentaires au menu **Simulation** de **POWER DESIGNER**.

L'avantage d'un tel logiciel est aussi d'être un produit "fait maison" et non une boîte noire fournie par un organisme étranger.

POWER DESIGNER a été développé pour fonctionner sur des PC dotés de Windows, mais il peut également être adapté, à l'aide des compilateurs adéquats, à des stations de travail comme SUN SparcStation ou Ardent Titan. Ces dernières ont une puissance de calcul et une capacité mémoire très importantes et répondent aux besoins des exploitants des systèmes de puissance.

Ainsi, même des modules d'analyse en temps réel peuvent être ajoutés au logiciel. Les systèmes de mesure et de contrôle du système de puissance peuvent lui envoyer des "messages" signalant des changements de consommation ou des dépassements des limites d'exploitation. Le logiciel entreprendrait alors des calculs débouchant sur les nouveaux paramètres d'exploitation.

POWER DESIGNER peut aussi être utilisé à des fins pédagogiques et il est même prévu de l'utiliser pour des Travaux Pratiques en graduation dans les années à venir.

Bibliographie

[1] Delannoy, C., Programmer en Turbo Pascal 7.0. Berti Editions, Eyrolles, 1993.

[2] Delannoy, C., Programmer en Turbo C++. Eyrolles, 1991.

[3] Fournioux, L., MS DOS Approfondi. Berti Editions, Edition Ellipse, 1992.

[4] Microsoft, Microsoft Windows Software Development Kit : Guide to Programming Version 3.0. Microsoft Corp., 1990.

[5] Petzold, C., Programmer Sous Windows 3. P.S.I., Dunod, Microsoft Press, 1991.

[6] Arrillaga, J. and Arnold, C.P., Computer Analysis of Power Systems. John Wiley & Sons, 1990.

[7] Stagg, G.W. and El-Abiad, A.h., Computer Methods in Power System Analysis. Mc Graw Hill, 1968.

[8] Stevenson, W.D.Jr., Elements of Power System Analysis. Mc Graw Hill, 1985.

[9] Elgerd, O.I., Electric Energy Systems Theory : An Introduction. Mc Graw Hill, 1971.

[10] Turan, G., Modern Power System Analysis. John Wiley & sons, 1988.

[11] Ward, J.B. and Hale, H.W., "Digital Computer Solution of Power Flow Problems". Trans.Am.Inst.Elect.Eng., Part 3 75 398-404, 1956.

[12] Freris, L.L. and Sasson, A.M., "Investigation of the Load Flow Problem". Proc IEE, Vol. 115, No10, October 1968, pp. 1459-1470.

[13] Stott, B. and Alsac, O., "Fast Decoupled Load Flow". IEEE Trans.Power Appar. Sys. PAS-87, 1974, pp. 859-867.

[14] Chang, S.K. and Brandwajn, V., "Adjusted Solutions in Fast Decoupled Load Flow", IEEE Transactions on Power Systems. Vol. 3, No 2, May 1988, pp.726-733.

[15] IEEE Committee Report, "Common Format for Exchange of Solved Load Flow Data", IEEE Trans.Power Appar. Sys. PAS-92, No 6, Nov-Dec 1988, pp.726-733.

[16] Korichi, D., "Analyse et Techniques des Plans de Délestage sur les Systèmes de Puissance", Projet de Fin d'Etudes, E.N.P., juin 1993.

[17] Mekhnache, K., et Khati, K., "Etude de la Stabilité Transitoire d'un Réseau Electrique H.T.", Projet de Fin d'Etudes, E.N.P., juin 1991.

[18] Laib, M., et Ouabri, H., "Etude Comparative de Différentes Méthodes de Calcul d'Ecoulement de Puissance et Application", Projet de Fin d'Etudes, E.N.P., juillet 1993.

ANNEXES

Annexe 1 : Déclaration des types objet

Cette annexe présente les déclarations des types objet (champs et méthodes) que nous avons développés pour **POWER DESIGNER**.

TLPoint

```
PLPoint = ^TLPoint;
TLPoint = object(TObject)
X, Y: Integer;
constructor Init(AX, AY: Integer);
constructor Load(var S: TStream);
procedure Store(var S: TStream);
end;
```

TBusCon

```
PBusCon = ^TBusCon;
TBusCon = object(TObject)
Con : Integer;
constructor Init( ACon : Integer);
constructor Load(var S: TStream);
procedure Store(var S: TStream);
end;
```

TPowerComponent

```
PPowerComponent = ^TPowerComponent;
TPowerComponent = Object( TObject)
HisType       : PowSymbType;
X,Y, Rotation : Integer;
Name  : Array [0..12] of Char;
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Destructor Done; Virtual;
Procedure Paint( PaintDC : HDC; WSInfo : WorkSheetInfoRec;
              Var PaintInfo : TPaintStruct); Virtual;
Procedure PaintSelected( PaintDC : HDC; Var PaintInfo : TPaintStruct);
         Virtual;
Function IsInIt( T : TPoint): Boolean; Virtual;
Function AreYouIn( R : TRect): Boolean; Virtual;
Procedure UpDate( Decal : TPoint; UnDC : HDC;
               WSInfo : WorkSheetInfoRec; Var R : TRect); Virtual;
Procedure Rotate( UnDC : HDC; WSInfo : WorkSheetInfoRec; Var R : TRect); Virtual;
Procedure Connect( SelectedList, NetWorkList : PCollection);
        Virtual;
Procedure UpDateCon( Num : Integer); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
             ; Study : Word); Virtual;
```

```
Procedure UnConnect( NetWorkList : PCollection); Virtual;
End;
```

TGenerator

```
PGenerator = ^TGenerator;
TGenerator = Object( TPowerComponent)
PG, Qgmin, Qgmax, Vsch  : Real;
Con              : Integer;
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Procedure Paint( PaintDC : HDC; WSInfo : WorkSheetInfoRec;
            Var PaintInfo : TPaintStruct); Virtual;
Procedure UpDate( Decal : TPoint; UnDC : HDC;
             WSInfo : WorkSheetInfoRec; Var R : TRect); Virtual;
Procedure Rotate( UnDC : HDC; WSInfo : WorkSheetInfoRec;Var R : TRect); Virtual;
Procedure Connect( SelectedList, NetWorkList : PCollection);
      Virtual;
Procedure UpDateCon( Num : Integer); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
          ; Study : Word); Virtual;
Procedure UnConnect( NetWorkList : PCollection); Virtual;
End;
```

TCompensator

```
PCompensator = ^TCompensator;
TCompensator = Object( TPowerComponent)
Qgmin, Qgmax, Vsch  : Real;
Con              : Integer;
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Procedure Paint( PaintDC : HDC; WSInfo : WorkSheetInfoRec;
            Var PaintInfo : TPaintStruct); Virtual;
Procedure UpDate( Decal : TPoint; UnDC : HDC;
             WSInfo : WorkSheetInfoRec;Var R : TRect); Virtual;
Procedure Rotate( UnDC : HDC; WSInfo : WorkSheetInfoRec;Var R : TRect); Virtual;
Procedure Connect( SelectedList, NetWorkList : PCollection);
      Virtual;
Procedure UpDateCon( Num : Integer); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
          ; Study : Word); Virtual;
Procedure UnConnect( NetWorkList : PCollection); Virtual;
End;
```

TBus

```
PBus = ^TBus;
TBus = Object( TPowerComponent)
V, Angle, Pg, Qg, PL, QL, Qgmin, Qgmax, Vsch : Real;
BusType, Status, Size       : Integer;
```

```
ConColl       : PCollection;
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Destructor Done; Virtual;
Procedure Paint( PaintDC : HDC; WSInfo : WorkSheetInfoRec;
            Var PaintInfo : TPaintStruct); Virtual;
Procedure PaintSelected( PaintDC : HDC; Var PaintInfo : TPaintStruct);
        Virtual;
Function IsInIt( T : TPoint): Boolean; Virtual;
Function AreYouIn( R : TRect): Boolean; Virtual;
Procedure UpDate( Decal : TPoint; UnDC : HDC;
             WSInfo : WorkSheetInfoRec;Var R : TRect); Virtual;
Procedure Rotate( UnDC : HDC; WSInfo : WorkSheetInfoRec;Var R : TRect); Virtual;
Procedure Connect( SelectedList, NetWorkList : PCollection);
      Virtual;
Procedure UpDateCon( Num : Integer); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
            ; Study : Word); Virtual;
Procedure UnConnect( NetWorkList : PCollection); Virtual;
End;
```

TTransfo

```
PTransfo = ^TTransfo;
TTransfo = Object( TPowerComponent)
Amag, Adelta, ZR, ZX, Smax : Real;
Con1, Con2  : Integer;
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Procedure Paint( PaintDC : HDC; WSInfo : WorkSheetInfoRec;
            Var PaintInfo : TPaintStruct); Virtual;
Procedure UpDate( Decal : TPoint; UnDC : HDC;
             WSInfo : WorkSheetInfoRec;Var R : TRect); Virtual;
Procedure Rotate( UnDC : HDC; WSInfo : WorkSheetInfoRec;Var R : TRect); Virtual;
Procedure Connect( SelectedList, NetWorkList : PCollection);
      Virtual;
Procedure UpDateCon( Num : Integer); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
            ; Study : Word); Virtual;
Procedure UnConnect( NetWorkList : PCollection); Virtual;
End;
```

TDrain

```
PDrain = ^TDrain;
TDrain = Object( TPowerComponent)
PD, QD  : Real;
Con     : Integer;
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Procedure Paint( PaintDC : HDC; WSInfo : WorkSheetInfoRec;
```

```
            Var PaintInfo : TPaintStruct); Virtual;
Procedure UpDate( Decal : TPoint; UnDC : HDC;
            WSInfo : WorkSheetInfoRec;Var R : TRect); Virtual;
Procedure Rotate( UnDC : HDC; WSInfo : WorkSheetInfoRec;Var R : TRect); Virtual;
Procedure Connect( SelectedList, NetWorkList : PCollection);
      Virtual;
Procedure UpDateCon( Num : Integer); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
            ; Study : Word); Virtual;
Procedure UnConnect( NetWorkList : PCollection); Virtual;
End;
```

TLine

```
PLine = ^TLine;
TLine = Object( TPowerComponent)
LinePoints    : PCollection;
ZR, ZX, B, G, Smax    : Real;
Con1, Con2    : Integer;
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Destructor Done; Virtual;
Procedure Paint( PaintDC : HDC; WSInfo : WorkSheetInfoRec;
            Var PaintInfo : TPaintStruct); Virtual;
Procedure PaintSelected( PaintDC : HDC; Var PaintInfo : TPaintStruct);
      Virtual;
Function IsInIt( T : TPoint): Boolean; Virtual;
Function AreYouIn( R : TRect): Boolean; Virtual;
Procedure UpDate( Decal : TPoint; UnDC : HDC;
            WSInfo : WorkSheetInfoRec; Var R : TRect); Virtual;
Procedure InvalidateLineRect( UnDC : HDC; WSInfo : WorkSheetInfoRec;
                  Var R : TRect); Virtual;
Procedure Rotate( UnDC : HDC; WSInfo : WorkSheetInfoRec; Var R : TRect); Virtual;
Procedure Connect( SelectedList, NetWorkList : PCollection);
      Virtual;
Procedure UpDateCon( Num : Integer); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
            ; Study : Word); Virtual;
Procedure UnConnect( NetWorkList : PCollection); Virtual;
End;
```

TCapa

```
PCapa = ^TCapa;
TCapa = Object( TPowerComponent)
Bc  : Real;
Con : Integer;
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Procedure Paint( PaintDC : HDC; WSInfo : WorkSheetInfoRec;
            Var PaintInfo : TPaintStruct); Virtual;
Procedure UpDate( Decal : TPoint; UnDC : HDC;
```

```
                WSInfo : WorkSheetInfoRec; Var R : TRect); Virtual;
Procedure Rotate( UnDC : HDC; WSInfo : WorkSheetInfoRec; Var R : TRect); Virtual;
Procedure Connect( SelectedList, NetWorkList : PCollection);
        Virtual;
Procedure UpDateCon( Num : Integer); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
              ; Study : Word); Virtual;
Procedure UnConnect( NetWorkList : PCollection); Virtual;
End;
```

TSpeGenerator

```
PSpeGenerator = ^TSpeGenerator;
TSpeGenerator = Object( TGenerator)
H, D, R, Xdp  : Real;
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
              ; Study : Word); Virtual;
End;
```

TSpeCompensator

```
PSpeCompensator = ^TSpeCompensator;
TSpeCompensator = Object( TCompensator)
H, D, R, Xdp  : Real;
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
              ; Study : Word); Virtual;
End;
```

TResistance

```
PResistance = ^TResistance;
TResistance = Object( TTransfo)
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
              ; Study : Word); Virtual;
End;
```

TSelfInd

```
PSelfInd = ^TSelfInd;
TSelfInd = Object( TTransfo)
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
```

```
            ; Study : Word); Virtual;
End;
```

TLink

```
PLink = ^TLink;
TLink = Object( TLine)
Epaisseur : Word;
Couleur : TColorRef;
Constructor Init( AType : PowSymbType);
Constructor Load ( Var S : TStream);
Procedure Store ( Var S : TStream); Virtual;
Procedure Paint( PaintDC : HDC; WSInfo : WorkSheetInfoRec;
            Var PaintInfo : TPaintStruct); Virtual;
Procedure Edit( PWork : PWindowsObject; NetWorkList : PCollection
            ; Study : Word); Virtual;
Procedure Connect( SelectedList, NetWorkList : PCollection);
       Virtual;
Procedure UpDateCon( Num : Integer); Virtual;
Procedure UnConnect( NetWorkList : PCollection); Virtual;
End;
```

TWorkSheet

```
PWorkSheet = ^TWorkSheet;
TWorkSheet = object(TWindow)
DragDC: HDC;
FileName, InName, OutName,
TSInName, TSOutName,TSResultName   : Array [0..fsPathName] of char;
GS_IterMax, NR_IterMax : Integer;
GS_Tol, GS_AccelFact, NR_Tol : Real;
LFOpt, TSOpt   : Word;
WSInfo  :  WorkSheetInfoRec; { Voir sa def dans WSHEET19.PAS }

Hcroix, HSaveCursor : HCursor;
LButtonDown, HasChanged, IsNewFile,
FirstPlace, LineCreating, ModifyLine,
ControlOn, SelectArea, Recup, VoltageKnown    : Boolean;
CapturedPoint          : PLPoint;
Prec, Actuel, Old, OldM : TPoint;
Order, Rep             : Integer;
Printer: PPrinter;
NetWork, SelectedComponents,
BusColl, MonitoredBuses       : PCollection;

constructor Init(AParent: PWindowsObject; ATitle: PChar);
destructor Done; virtual;
function CanClose: Boolean; virtual;
procedure GetWindowClass(var AWndClass: TWndClass); virtual;
procedure WMLButtonDown(var Msg: TMessage);
  virtual wm_First + wm_LButtonDown;
procedure LButtonDownAct;
procedure WMLButtonUp(var Msg: TMessage);
  virtual wm_First + wm_LButtonUp;
```

```
procedure LButtonUpAct;
procedure WMRButtonDown(var Msg: TMessage);
  virtual wm_First + wm_RButtonDown;
procedure RButtonDownAct;
procedure WMMouseMove(var Msg: TMessage);
  virtual wm_First + wm_MouseMove;
Procedure DrawCapture;
Procedure MoveComponent ( Decal : TPoint);
Procedure ConvertRect( Var R : TRect);

procedure WMActivate( var Msg: TMessage);
  virtual wm_First +wm_Activate;
procedure SetNames(NewName: PChar);

procedure FileSave(var Msg: TMessage);
  virtual cm_First + cm_Save;
procedure FileSaveAs(var Msg: TMessage);
  virtual cm_First + cm_SaveAs;
function LoadFile : Integer; { V1.19 only }
procedure SaveFile;          { V1.19 only }
procedure CMPrint(var Msg: TMessage);
  virtual cm_First + cm_Print;
procedure CMSetup(var Msg: TMessage);
  virtual cm_First + cm_Setup;
Procedure Paint( PaintDC : HDC; Var PaintInfo : TPaintStruct); virtual;

Procedure NoMoreMemory;
function IsItFixed : Boolean;
Procedure CMCreGenerator(var Msg: TMessage);
  virtual cm_First + cm_Generator;
Procedure CMCreCompensator(var Msg: TMessage);
  virtual cm_First + cm_Compensator;
Procedure CMCreBus(var Msg: TMessage);
  virtual cm_First + cm_Bus;
Procedure CMCreLine(var Msg: TMessage);
  virtual cm_First + cm_Line;
Procedure CMCreTransfo(var Msg: TMessage);
  virtual cm_First + cm_Transfo;
Procedure CMCreDrain(var Msg: TMessage);
  virtual cm_First + cm_Drain;
Procedure CMCreCapa(var Msg: TMessage);
  virtual cm_First + cm_Capa;
Procedure CMCreResistance(var Msg: TMessage);
  virtual cm_First + cm_Resistance;
Procedure CMCreSelfInd(var Msg: TMessage);
  virtual cm_First + cm_SelfInd;
Procedure CMCreLink(var Msg: TMessage);
  virtual cm_First + cm_Link;

Procedure CreateComponent ( MyComponent : PPowerComponent);
Procedure Rotate(var Msg: TMessage);
  virtual cm_First + cm_Rotate;

Procedure CMModifyLine(var Msg: TMessage);
  virtual cm_First + cm_ModifyLine;
```

```
Procedure WMKeyDown(var Msg: TMessage);
  virtual wm_First + WM_KEYDOWN;
Procedure WMKeyUp(var Msg: TMessage);
  virtual wm_First + WM_KEYUp;
Procedure InsertPoint;
Procedure DeletePoint;

Procedure Performdelete;
Procedure Delete(var Msg: TMessage);
  virtual cm_First + cm_Delete;
Procedure Connect(var Msg: TMessage);
  virtual cm_First + cm_Connect;
Procedure Edit(var Msg: TMessage);
  virtual cm_First + cm_Edit;
Procedure CMSelectArea(var Msg: TMessage);
  virtual cm_First + cm_SelectArea;
Procedure UnConnect(var Msg: TMessage);
  virtual cm_First + cm_UnConnect;
Procedure SelectAll(var Msg: TMessage);
  virtual cm_First + cm_SelectAll;

procedure CMEditCut(var Msg: TMessage);
  virtual cm_First + cm_EditCut;
procedure CMEditCopy(var Msg: TMessage);
  virtual cm_First + cm_EditCopy;
procedure PerformEditCopy;
procedure CMEditPaste(var Msg: TMessage);
  virtual cm_First + cm_EditPaste;

Procedure MakeData(var Msg: TMessage);
  virtual cm_First + cm_MakeData;

Function IsStudyLoadFlow : Boolean;
Procedure GaussSeidelCall(var Msg: TMessage);
  virtual cm_First + cm_GaussSeidel;
Procedure NewtonRaphsonCall(var Msg: TMessage);
  virtual cm_First + cm_NewtonRaphson;
Procedure NRBackOffCall(var Msg: TMessage);
  virtual cm_First + cm_NewtonRaphsonBackOff;
Procedure FDLCall(var Msg: TMessage);
  virtual cm_First + cm_FDL;
Procedure LFOutPutData;
Procedure TSOutPutData;
Procedure OutPutDataCall(var Msg: TMessage);
  virtual cm_First + cm_OutputData;
Procedure CMInfos(var Msg: TMessage);
  virtual cm_First + cm_Informations;

Procedure CMLoadFlowStudy(var Msg: TMessage);
  virtual cm_First +  cm_LoadFlowStudy;
Procedure CMTransStabStudy(var Msg: TMessage);
  virtual cm_First +  cm_TransStabStudy;

Procedure WorkSheetOpt(var Msg: TMessage);
  virtual cm_First +  cm_OptWorkSheet;
```

```
Procedure LFFileNames;
Procedure STFileNames;
Procedure OptFileNames(var Msg: TMessage);
  virtual cm_First +  cm_OptFileNames;

Procedure LoadFlowOptions(var Msg: TMessage);
  virtual cm_First +  cm_OptLoadFlow;
Procedure TransStabOptions(var Msg: TMessage);
  virtual cm_First + cm_OptTransStab;
Procedure TransStabOutputOptions(var Msg: TMessage);
  virtual cm_First + cm_OptTransStabOut;

Function IsStudyTransStab : Boolean;
Procedure CMFaultBus(var Msg: TMessage);
  virtual cm_First + cm_FaultBus;
Procedure CMOpenLine(var Msg: TMessage);
  virtual cm_First + cm_OpenLine;
Procedure CMSwitchLoad(var Msg: TMessage);
  virtual cm_First + cm_SwitchLoad;
Procedure CMDropLoad(var Msg: TMessage);
  virtual cm_First + cm_DropLoad;
Procedure CMLossGen(var Msg: TMessage);
  virtual cm_First + cm_LossGen;
Procedure CMClearAll(var Msg: TMessage);
  virtual cm_First + cm_ClearAll;
Procedure CMMonitorBus(var Msg: TMessage);
  virtual cm_First + cm_MonitorBus;
Procedure CMTSP(var Msg: TMessage);
  virtual cm_First + cm_TSP;
Procedure CMTSPT(var Msg: TMessage);
  virtual cm_First + cm_TSPT;
Procedure CMRK(var Msg: TMessage);
  virtual cm_First + cm_RK;
Procedure CMPlotResult(var Msg: TMessage);
  virtual cm_First + cm_PlotResult;
End;
```

TDesigner

```
PDesigner = ^TDesigner;
TDesigner = object(TMDIWindow)
 StatusLine : PRibbonWindow;
 constructor Init(ATitle: PChar; AMenu: HMenu);
 procedure WMDestroy(var Msg: TMessage);
   virtual wm_First + wm_Destroy;
 destructor Done; virtual;
 procedure SetInfoFlag( PWS : PWorkSheet );
 procedure SetStudyFlag( PWS : PWorkSheet );
 procedure AdjustMenu( PWS : PWorkSheet );
 procedure ModifMenuToLoadFlow;
 procedure ModifMenuToTransStab;
 procedure SetUpWindow; virtual;
 procedure FileOpen(var Msg: TMessage);
```

```
    virtual cm_First + cm_Open;
  procedure CMHelpContent(var Msg: TMessage);
    virtual cm_First + cm_Content;
  procedure CMHelpKeyBoard(var Message: TMessage);
    virtual cm_First + cm_HelpKeyBoard;
  procedure CMHelpHelp(var Message: TMessage);
    virtual cm_First + cm_HelpHelp;
  procedure CMAbout(var Msg: TMessage);
    virtual cm_First + cm_About;
  function InitChild: PWindowsObject; virtual;
  procedure WMMenuSelect(var Msg: TMessage);
    virtual wm_First + wm_MenuSelect;
  procedure WMSize(var Msg: TMessage);
    virtual wm_First + wm_Size;
  procedure GetWindowClass(var AWndClass: TWndClass); virtual;
 end;
```

Annexe 2 : Déclaration des enregistrements des flux (Stream Registration)

Cette annexe présente les déclarations des enregistrements des flux pour les différents types objet développés pour **POWER DESIGNER**.

```
RLPoint: TStreamRec = (
   ObjType: 150;
   VmtLink: Ofs(TypeOf(TLPoint)^);
   Load: @TLPoint.Load;
   Store: @TLPoint.Store);

RPowerComponent : TStreamRec = (
   ObjType : 151;
   VmtLink : Ofs( TypeOf(TPowerComponent)^);
   Load : @TPowerComponent.Load;
   Store : @TPowerComponent.Store );

RGenerator : TStreamRec = (
   ObjType : 152;
   VmtLink : Ofs( TypeOf(TGenerator)^);
   Load : @TGenerator.Load;
   Store : @TGenerator.Store );

RBus : TStreamRec = (
   ObjType : 153;
   VmtLink : Ofs( TypeOf(TBus)^);
   Load : @TBus.Load;
   Store : @TBus.Store );

RTransfo : TStreamRec = (
   ObjType : 154;
   VmtLink : Ofs( TypeOf(TTransfo)^);
   Load : @TTransfo.Load;
   Store : @TTransfo.Store );

RDrain : TStreamRec = (
   ObjType : 155;
   VmtLink : Ofs( TypeOf(TDrain)^);
   Load : @TDrain.Load;
   Store : @TDrain.Store );

RLine : TStreamRec = (
   ObjType : 156;
   VmtLink : Ofs( TypeOf(TLine)^);
   Load : @TLine.Load;
   Store : @TLine.Store );
RBusCon : TStreamRec = (
   ObjType : 157;
   VmtLink : Ofs( TypeOf(TBusCon)^);
   Load : @TBusCon.Load;
```

```
    Store : @TBusCon.Store );

RCompensator : TStreamRec = (
    ObjType : 158;
    VmtLink : Ofs( TypeOf(TCompensator)^);
    Load : @TCompensator.Load;
    Store : @TCompensator.Store );

RCapa : TStreamRec = (
    ObjType : 159;
    VmtLink : Ofs( TypeOf(TCapa)^);
    Load : @TCapa.Load;
    Store : @TCapa.Store );

RSpeGenerator : TStreamRec = (
    ObjType : 160;
    VmtLink : Ofs( TypeOf(TSpeGenerator)^);
    Load : @TSpeGenerator.Load;
    Store : @TSpeGenerator.Store );

RSpeCompensator : TStreamRec = (
    ObjType : 161;
    VmtLink : Ofs( TypeOf(TSpeCompensator)^);
    Load : @TSpeCompensator.Load;
    Store : @TSpeCompensator.Store );

RResistance : TStreamRec = (
    ObjType : 162;
    VmtLink : Ofs( TypeOf(TResistance)^);
    Load : @TResistance.Load;
    Store : @TResistance.Store );

RSelfInd : TStreamRec = (
    ObjType : 163;
    VmtLink : Ofs( TypeOf(TSelfInd)^);
    Load : @TSelfInd.Load;
    Store : @TSelfInd.Store );

RLink : TStreamRec = (
    ObjType : 164;
    VmtLink : Ofs( TypeOf(TLink)^);
    Load : @TLink.Load;
    Store : @TLink.Store );
```

Annexe 3 : Deux méthodes du type objet TWorkSheet

- **LButtonDownAct** : traite les actions sur le bouton gauche de la souris et sur les touches Space ou Enter.
- **MakeData** : Génère les fichiers **.LFI** et **.TSI** selon le mode d'étude.

TWorkSheet.LButtonDownAct

```
{ performs actions for LButton Down, SpaceBar and ReturnKey }
Procedure TWorkSheet.LButtonDownAct;
Var  T : TPoint;
    P : PLPoint;
    DragComponent : PPowerComponent;

 Function Exist( MyC : PPowerComponent) : Boolean; Far;
 begin
 IF MyC=DragComponent Then Exist:=True
                 Else Exist:=False;
 end;

 { Recherche du component pointe par le curseur}
 Function IsPointed( MyC : PPowerComponent) : Boolean; Far;
 begin
 IF MyC^.IsInIt( T) And (NetWork^.IndexOf(MyC)>Order) Then
                                  Begin
                                  IsPointed:=True;
                                  Order:=NetWork^.IndexOf(MyC)
                                  end
                                  Else IsPointed:=False;
 end;

{ Procedure principale }
Begin
GetCursorPos( T);
ScreenToClient( HWindow, T);
T.X:=T.X+Scroller^.XPos*Scroller^.XUnit;
T.Y:=T.Y+Scroller^.YPos*Scroller^.YUnit;

IF FirstPlace and LineCreating Then
                        begin
                        FirstPlace:=False;
                        Prec.X:=PLine(SelectedComponents^.At(0))^.X;
                        Prec.Y:=PLine(SelectedComponents^.At(0))^.Y;
                        Exit;
                        End;
IF LineCreating And Not(LButtonDown) Then
            Begin
            LButtonDown:=True;
            P:=New( PLPoint, Init( Actuel.X-PLine(SelectedComponents^.At(0))^.X,
                              Actuel.Y-PLine(SelectedComponents^.At(0))^.Y) );
            if P=nil then NoMoreMemory
              else
              begin
```

```
                PLine(SelectedComponents^.At(0))^.LinePoints^.Insert(P);
                Prec:=Actuel;
                end;
              Exit;
              End;
If FirstPlace Then begin
                FirstPlace:=False;
                ReleaseDC ( HWindow, DragDC);
                ReleaseCapture;
                { Des-Affiche Captured }
                PDesigner( Parent)^.SetInfoFlag( @Self);
                InvalidateRect ( HWindow, Nil, True);
{                UpDateWindow( HWindow);}
                Exit;
                end;

IF ModifyLine And Not(LButtonDown) Then
              Begin
              LButtonDown:=True;
              if CapturedPoint=nil then
                  begin
              T.X:=PLine(SelectedComponents^.At(0))^.X;
              T.Y:=PLine(SelectedComponents^.At(0))^.Y;
                  end
              else begin
              T.X:=CapturedPoint^.X+PLine(SelectedComponents^.At(0))^.X;
              T.Y:=CapturedPoint^.Y+PLine(SelectedComponents^.At(0))^.Y;
                   end;
              T.X:=T.X-Scroller^.XPos*Scroller^.XUnit;
              T.Y:=T.Y-Scroller^.YPos*Scroller^.YUnit;
              ClientToScreen( HWindow, T);
              SetCursorPos( T.X, T.Y);
              HSaveCursor:=SetCursor( HCroix);
              Exit;
              End;
{IF ModifyLine And LButtonDown Then Exit;  evite Pbs }
IF SelectArea And Not(LButtonDown) Then
              Begin
              LButtonDown:=True;
              Prec:=T;
              Actuel:=T;
              Exit;
              End;
{ last action }
If Not ControlOn Then SelectedComponents^.DeleteAll;
  If Not(LButtonDown) Then
            begin
            LButtonDown:=True;
            If ( Old.X<>T.X) or ( OLd.Y<>T.Y) Then
                        begin
                        Old:=T;
                        Order:=-1;
                        end;
            DragComponent:=NetWork^.FirstThat( @IsPointed);
            IF DragComponent=Nil Then Begin
```

```
                        If SelectedComponents^.Count=0 Then
                                LButtonDown:=False;
                        Order:=-1;

                        InvalidateRect ( HWindow, Nil, True);
                    {    UpDateWindow( HWindow);}
                        Exit;
                        End;
          If SelectedComponents^.FirstThat( @Exist)<>Nil
            Then SelectedComponents^.Delete( DragComponent)
            Else SelectedComponents^.Insert( DragComponent);

          InvalidateRect ( HWindow, Nil, True);
{          UpDateWindow( HWindow);}
          SetCapture( HWindow);
          DragDC:=GetDC( HWindow);
          HasChanged:=True;
          PDesigner( Parent)^.SetInfoFlag( @Self); {Affiche Captured}
          End;
End;
```

TWorkSheet.MakeData

```
Procedure TWorksheet.MakeData(var Msg: TMessage);
Var  SlackExist     : Boolean;
    I, N, Npv, Npq,
    Error, K       : Integer;
    InFile         : Text;
    Name           : Array [0..fsPathName] of char;
    SS             : Array [0..300] of char;
    PB1, PB2       : PBus;
    PL             : PLine;


  Procedure UpDateBus ( MyC : PPowerComponent); Far;
  Var GenExist, CompensatorExist : Boolean;
     PB : PBus;

    Procedure UpDatePower( PBC : PBusCon); Far;
    Var  P : PPowerComponent;
    begin
    P:=NetWork^.At( PBC^.Con);
    If P^.HisType=Generator Then Begin
                          PB^.Pg:=PB^.Pg+PGenerator(P)^.Pg;
                          PB^.Qgmin:=PB^.Qgmin+PGenerator(P)^.Qgmin;
                          PB^.Qgmax:=PB^.Qgmax+PGenerator(P)^.Qgmax;
                { verifie si tous les Generators sont au meme Vsch }
                if (GenExist or CompensatorExist)
                   and (PB^.Vsch<>PGenerator(P)^.Vsch) then
                                begin
                                MessageBox( HWindow,
                        'There are different Vsch at the same bus',
                        'Error in data', MB_Ok or MB_IconExclamation);
                                Error:=3;
```

```
                                end;
                            { continue }
                            PB^.Vsch:=PGenerator(P)^.Vsch;
                            GenExist:=True;
                            End;
   If P^.HisType=Compensator Then
                    Begin
                    PB^.Qgmin:=PB^.Qgmin+PCompensator(P)^.Qgmin;
                    PB^.Qgmax:=PB^.Qgmax+PCompensator(P)^.Qgmax;
                { verifie si tous les Generators sont au meme Vsch }
                if (GenExist or CompensatorExist)
                  and (PB^.Vsch<>PCompensator(P)^.Vsch) then
                                MessageBox( HWindow,
                        'There are different Vsch at the same bus',
                        'Error in data', MB_Ok or MB_IconExclamation);
                    { continue }
                    PB^.Vsch:=PCompensator(P)^.Vsch;
                    CompensatorExist:=True;
                    End;
   If P^.HisType=Drain Then Begin
                        PB^.PL:=PB^.PL+PDrain(P)^.PD;
                        PB^.QL:=PB^.QL+PDrain(P)^.QD;
                        End;

   end;

begin
Error:=0;
if MyC^.HisType<>Bus Then Exit;
PB:=PBus(MyC);
if StrEnd(PB^.Name)-PB^.Name=0Then
                        Begin MessageBox( HWindow,
                        'Buses are not all named',
                        'Error in data', MB_Ok or MB_IconExclamation);
                        Error:=1;
                        Exit;
                        end;
if PB^.BusType=2 then begin
                    If Not(SlackExist) Then begin
                                SlackExist:=True;
                                BusColl^.Insert( PB);
                                end
                            Else
                            begin
                            MessageBox( HWindow,
                        'There is more than one slack bus specified',
                        'Error in data', MB_Ok or MB_IconExclamation);
                                Error:=2;
                                end;
                    Exit;
                    end;
{ si PB^.Status=1  ==> ne touche pas aux donnees du Bus }
if PB^.Status<>0 then exit;
PB^.Pg:=0;
if WSInfo.Study=id_LoadFlow then PB^.Qg:=0;
```

```
PB^.PL:=0;
PB^.QL:=0;
PB^.Qgmin:=0;
PB^.Qgmax:=0;
GenExist:=False;
CompensatorExist:=False;
PB^.ConColl^.ForEach( @UpDatePower);
If GenExist or CompensatorExist Then PB^.BusType:=1  { PV }
          Else PB^.BusType:=0; { PQ }
end;

Procedure LookForPVBuses ( MyC : PPowerComponent); Far;
Var PB : PBus;
begin
if MyC^.HisType<>Bus Then Exit;
PB:=PBus(MyC);
if PB^.BusType<>1 then Exit;  { ie PV only }
BusColl^.Insert( PB);
Inc (Npv);
end;

Procedure LookForPQBuses ( MyC : PPowerComponent); Far;
Var PB : PBus;
begin
if MyC^.HisType<>Bus Then Exit;
PB:=PBus(MyC);
if PB^.BusType<>0 then Exit;  { ie PQ only }
BusColl^.Insert( PB);
Inc (Npq);
end;

Procedure PutLineDatas ( MyC : PPowerComponent); Far;
Var PB : PLine;
   PR : PResistance;
   PS : PSelfInd;
   S : Array[0..80] of char;
begin
Case MyC^.HisType of
{ for the Line }
     Line : Begin
PB:=PLine(MyC);
if StrEnd(PB^.Name)-PB^.Name=0 Then
             begin
             Error:=5;
             MessageBox( HWindow, 'Lines are not all named',
                      'Warning in data', MB_Ok or MB_IconExclamation);
             end;
if (PB^.Con1=-1) or (PB^.Con2=-1) then
             begin
             Error:=4;
             StrCopy( S,'Line : ');
             StrCat( S, PB^.Name);
             StrCat( S, #13#10'is not connected ');
             MessageBox( HWindow, S,
                      'Error in data', MB_Ok or MB_IconExclamation);
```

```
            Exit;
            end;
PB1:=NetWork^.At(PB^.Con1);
PB2:=NetWork^.At(PB^.Con2);
WriteLn( InFile, '3  ',PB1^.Name:12,'  ',PB2^.Name:12,'  ',
        PB^.ZR:7:5,'  ', PB^.ZX:7:5,'  ',
        PB^.G:7:5,'  ',PB^.B:7:5,'  ', PB^.Smax:7:5);
             End;
{ for the Resistance }
    Resistance : Begin
PR:=PResistance(MyC);
if StrEnd(PR^.Name)-PR^.Name=0 Then
            begin
            Error:=5;
            MessageBox( HWindow, 'Resistances are not all named',
                    'Warning in data', MB_Ok or MB_IconExclamation);
            end;
if (PR^.Con1=-1) or (PR^.Con2=-1) then
            begin
            Error:=4;
            StrCopy( S,'Resistance : ');
            StrCat( S, PR^.Name);
            StrCat( S, #13#10'is not connected ');
            MessageBox( HWindow, S,
                    'Error in data', MB_Ok or MB_IconExclamation);
            Exit;
            end;
PB1:=NetWork^.At(PR^.Con1);
PB2:=NetWork^.At(PR^.Con2);
WriteLn( InFile, '3  ',PB1^.Name:12,'  ',PB2^.Name:12,'  ',
        PR^.ZR:7:5,'  ', PR^.ZX:7:5,'  ',
        PR^.ZX:7:5,'  ',PR^.ZX:7:5,'  ', PR^.Smax:7:5);
        { i.e. zero }
             End;
{ for the Self }
    SelfInd : Begin
PS:=PSelfInd(MyC);
if StrEnd(PS^.Name)-PS^.Name=0 Then
            begin
            Error:=5;
            MessageBox( HWindow, 'Self inductances are not all named',
                    'Warning in data', MB_Ok or MB_IconExclamation);
            end;
if (PS^.Con1=-1) or (PS^.Con2=-1) then
            begin
            Error:=4;
            StrCopy( S,'Self inductance : ');
            StrCat( S, PS^.Name);
            StrCat( S, #13#10'is not connected ');
            MessageBox( HWindow, S,
                    'Error in data', MB_Ok or MB_IconExclamation);
            Exit;
            end;
PB1:=NetWork^.At(PS^.Con1);
PB2:=NetWork^.At(PS^.Con2);
```

```
WriteLn( InFile, '3  ',PB1^.Name:12,'  ',PB2^.Name:12,'  ',
        PS^.ZR:7:5,'  ', PS^.ZX:7:5,'  ',
        PS^.ZR:7:5,'  ',PS^.ZR:7:5,'  ', PS^.Smax:7:5);
        { i.e. zero }
             End;
      else    Exit
      end; { du case }
end;

Procedure PutTransfoDatas ( MyC : PPowerComponent); Far;
Var PB : PTransfo;
   S : Array[0..80] of char;
begin
if MyC^.HisType<>Transfo Then Exit;
PB:=PTransfo(MyC);
if StrEnd(PB^.Name)-PB^.Name=0 Then
            begin
            Error:=5;
            MessageBox( HWindow, 'Transfos are not all named',
                    'Warning in data', MB_Ok or MB_IconExclamation);
            end;
if (PB^.Con1=-1) or (PB^.Con2=-1) then
            begin
            Error:=4;
            StrCopy( S,'Transfo : ');
            StrCat( S, PB^.Name);
            StrCat( S, #13#10'is not connected ');
            MessageBox( HWindow, S,
                    'Error in data', MB_Ok or MB_IconExclamation);
            Exit;
            end;
PB1:=NetWork^.At(PB^.Con1);
PB2:=NetWork^.At(PB^.Con2);
WriteLn( InFile, '4  ',PB1^.Name:12,'  ',PB2^.Name:12,'  ',
        PB^.ZR:7:5,'  ',PB^.ZX:7:5,'  ',
        PB^.Amag:7:5,'  ', PB^.Adelta:6:2,'  ', PB^.Smax:7:5);
end;

Procedure PutCapaDatas ( MyC : PPowerComponent); Far;
Var PB : PCapa;
   S : Array[0..80] of char;
begin
if MyC^.HisType<>Capa Then Exit;
PB:=PCapa(MyC);
if StrEnd(PB^.Name)-PB^.Name=0 Then
            begin
            Error:=5;
            MessageBox( HWindow, 'Capacitors are not all named',
                    'Warning in data', MB_Ok or MB_IconExclamation);
            end;
if PB^.Con=-1  then
            begin
            Error:=4;
            StrCopy( S,'Capacitor : ');
            StrCat( S, PB^.Name);
```

```
            StrCat( S, #13#10'is not connected ');
            MessageBox( HWindow, S,
                    'Error in data', MB_Ok or MB_IconExclamation);
            Exit;
            end;
PB1:=NetWork^.At(PB^.Con);
WriteLn( InFile, '5  ',PB1^.Name:12,'  ',
       PB^.Bc:7:5);
end;

Procedure PutMachinesDatas( MyC : PPowerComponent); Far;
Var Con : Integer;
   PB  : PBus;
   PG  : PSpeGenerator;
   PC  : PSpeCompensator;
   S : Array[0..80] of char;
begin
if ( MyC^.HisType<>Generator) And ( MyC^.HisType<>Compensator) Then Exit;
Case MyC^.HisType of
 Generator   : begin
            PG:=PSpeGenerator( MyC);
            Con:=PG^.Con;
            PB:=PBus( NetWork^.At( Con));
            WriteLn( InFile, '16 ',PB^.Name:12,'  ',
                    PG^.H:8:5,'  ',PG^.D:8:5,'  ',
                    PG^.R:7:5,'  ',PG^.Xdp:7:5 );
            if PG^.H=0 then
                    begin
                    Error:=6;
                    StrCopy( S,'Generator : ');
                    StrCat( S, PG^.Name);
                    StrCat( S, #13#10'has H=0 ');
                    MessageBox( HWindow, S,
                     'Error in data', MB_Ok or MB_IconExclamation);
                    Exit;
                    end;
            end;
 Compensator : begin
            PC:=PSpeCompensator( MyC);
            Con:=PC^.Con;
            PB:=PBus( NetWork^.At( Con));
            WriteLn( InFile, '16  ',PB^.Name:12,'  ',
                    PC^.H:8:5,'  ',PC^.D:8:5,'  ',
                    PC^.R:7:5,'  ',PC^.Xdp:7:5 );
            if PC^.H=0 then
                    begin
                    Error:=6;
                    StrCopy( S,'Compensator ');
                    StrCat( S, PC^.Name);
                    StrCat( S, #13#10'has H=0 ');
                    MessageBox( HWindow, S,
                     'Error in data', MB_Ok or MB_IconExclamation);
                    Exit;
                    end;
            end;
```

```
    end;
  end;

{ Corps de MakeData }
Begin
BusColl^.DeleteAll;
SlackExist:=False;
Npv:=0;
Npq:=0;
NetWork^.ForEach( @UpDateBus);
if Error<>0 Then Exit;
If Not(SlackExist) Then begin MessageBox( HWindow,
                    'There is no slack bus specified', 'Error in data',
                    MB_Ok or MB_IconExclamation);
                    Exit;
                    end;
NetWork^.ForEach( @LookForPVBuses);
NetWork^.ForEach( @LookForPQBuses);
N:=Npv+Npq;
case WSInfo.Study of
 id_LoadFlow :  begin
              Assign( InFile, InName);
              {$I-}
              ReWrite(InFile);
              {$I+}
              If IOResult<>0 Then begin
                 MessageBox (HWindow, InName
                 ,'Openning File Error',mb_Ok+mb_ICONExclamation);
                 Exit;
                 end;
              end;
 id_TransStab : begin       { le nom du fichier differe }
              Assign( InFile, TSInName);
              {$I-}
              ReWrite(InFile);
              {$I+}
              If IOResult<>0 Then begin
                 MessageBox (HWindow, TSInName
                 ,'Openning File Error',mb_Ok+mb_ICONExclamation);
                 Exit;
                 end;
              end;
 end;

WriteLn( InFile, N+1,' ',Npv);
if WSInfo.Study=id_LoadFlow then
                  begin
                  WriteLn( InFile, GS_IterMax,' ', GS_Tol);
                  WriteLn( InFile, NR_IterMax,' ', NR_Tol);
                  end;
{
for Newton-Raphson and Gauss-Seidel specefic NIterMax & Tol
}
case WSInfo.Study of
 id_LoadFlow :  WriteLn( InFile,
```

```
'8 ---------------  Input file for the load flow program  ---------------');
 id_TransStab : WriteLn( InFile,
'8 ----------  Input file for the Transient Stability program  ----------');

 end;
WriteLn( InFile, '8 --- Number of total Buses = ',N+1,'  PV Buses = ',Npv);
WriteLn( InFile, '8');
if WSInfo.Study=id_LoadFlow then
                  begin
                  WriteLn( InFile,
                  '8 --- for GS : Nb IterationsMax = ',GS_IterMax,
                  '  Tolerance = ',GS_Tol);
                  WriteLn( InFile,
                  '8 --- for NR : Nb IterationsMax = ',NR_IterMax,
                  '  Tolerance = ',NR_Tol);
                  end;
WriteLn( InFile, '8 ');
{ Card 1 : Buses }
WriteLn( InFile,
'8 ---------------------------  Bus Data  ------------------------------');
WriteLn( InFile,
       '8  BUS          Pg       Qg       Pl       Ql       Vb');
for I:=0 to N do
  begin
  PB1:=BusColl^.At(I);
  WriteLn( InFile, '1  ', PB1^.Name:12,'  ',PB1^.Pg:7:5,'  ',
        PB1^.Qg:7:5,'  ', PB1^.PL:7:5,'  ',PB1^.QL:7:5,'  ',PB1^.Vsch:7:5);
  end;
{ Card 2 : Active Buses }
if WSInfo.Study=id_LoadFlow then
                  begin
                  WriteLn( InFile,
'8 ------------------------ Active Bus Limits ----------------------------');
                  WriteLn( InFile, '8  BUS          Qgmin    Qgmax');
                  for I:=1 to Npv do
                     begin
                     PB1:=BusColl^.At(I);
                     WriteLn( InFile, '2  ',PB1^.Name:12,'  ',
                           PB1^.Qgmin:7:5,'  ',PB1^.Qgmax:7:5);
                     end;
                  end;

{ Card 3 : Lines }
WriteLn( InFile,
'8 ---------------------------  Line Data  -----------------------------');
WriteLn( InFile,
  '8  BUS          BUS          Rs       Xs       Gs       Bs       Smax');
NetWork^.ForEach( @PutLineDatas);

{ Card 4 : Transfos }
WriteLn( InFile,
'8 ------------------------  Transformer Data  --------------------------');
WriteLn( InFile,
'8  BUS          BUS          Rs       Xs       Amag     A(deg)  Smax');
NetWork^.ForEach( @PutTransfoDatas);
```

```
{ Card 5 : Capas }
WriteLn( InFile,
'8 -------------------------  Capacitor Data  ---------------------------');
WriteLn( InFile,
        '8  BUS          Bc');
NetWork^.ForEach( @PutCapaDatas);

{ Card 9 : Acceleration Factor GS }
WriteLn( InFile,
'8 ----------------------  Acceleration Factor GS  ---------------------');
WriteLn( InFile, '9  ', GS_AccelFact);

{ Card 10 : LoadFlow Output Options }
if WSInfo.Study=id_LoadFlow then
               begin
               WriteLn( InFile,
'8 ----------------------  Load Flow Output Options  ---------------------');
               WriteLn( InFile, '10 ', LFOpt);
               end;

{ Now Only TransStab }
if WSInfo.Study=id_TransStab then
   begin
   if VoltageKnown=False then
        begin
        StrCopy( SS,'Load Flow simulation hasen''t been done yet.');
        StrCat ( SS,#13#10'You should call Load Flow and OutputData');
        StrCat ( SS,#13#10'to update initial conditions ( Vmag, delta, Qg... )');
        StrCat ( SS,#13#10'for Transient Stability simulation otherwise');
        StrCat( SS,
        #13#10'Transient Stability simulation wouldn''t be done correctly');
        MessageBox( HWindow, SS, 'Warning', MB_OK+MB_ICONEXCLAMATION );
        end;
   { remet a zero le code de l'etude }
   WSInfo.TSInfo.Code:=0;

{ Card 11 : Bus voltage from Load Flow study }
   WriteLn( InFile,
'8 ----------------  Bus voltage from Load Flow study  -------------------');
   WriteLn( InFile, '8  Bus          mag      arg(deg)');
   for I:=0 to BusColl^.Count-1 do
      begin
      PB1:=BusColl^.At(I);
      WriteLn( InFile, '11 ', PB1^.Name:12,'  ',PB1^.V:7:5,'  ',
                                    PB1^.Angle:7:5);
      end;

{ Card 16 : Machines Characteristics H, R, Xdp }
   WriteLn( InFile,
'8 ----------------------  Machine Characteristics  ---------------------');
   WriteLn( InFile, '8  Bus          H        D        R        Xdp');
   NetWork^.ForEach( @PutMachinesDatas);

{ Card 17 : Monitored Buses }
```

```
  WriteLn( InFile,
'8 -------------------------  Monitored Buses  -------------------------');
  WriteLn( InFile, '8  Bus');
  if MonitoredBuses^.Count<>0 then
    for I:=0 to MonitoredBuses^.Count-1 do
          begin
          K:=PBusCon(MonitoredBuses^.At(I))^.Con;
          WriteLn( InFile, '17 ', PBus( NetWork^.At( K ) )^.Name:12);
          end
    else            begin
                    MessageBox( HWindow,
                    'There are no monitored buses specified',
                    'Error in data',
                    MB_Ok or MB_IconExclamation);
                    end;

{ Card 18 : Faulty Bus }
  if WSInfo.TSInfo.FaultedBus<>-1 then
                    begin
                    WSInfo.TSInfo.Code:=WSInfo.TSInfo.Code or id_FaultedBus;
  WriteLn( InFile,
'8 -------------------------  3 Phase Faulty Bus  ----------------------');
  WriteLn( InFile,'8  Bus          Start   End     Gf        Bf');
  WriteLn( InFile, '18 ',
          PBus( NetWork^.At( WSInfo.TSInfo.FaultedBus))^.Name:12,'  ',
          WSInfo.TSInfo.FBt0:6:4,'  ', WSInfo.TSInfo.FBt1:6:4,'  ',
          WSInfo.TSInfo.Gf:8,'  ', WSInfo.TSInfo.Bf:8 );
                    end;

{ Card 19 : Fault Parameters }
  if WSInfo.TSInfo.OpenedLine<>-1 then
    begin
        WSInfo.TSInfo.Code:=WSInfo.TSInfo.Code or id_OpenedLine;
        PL:=PLine( NetWork^.At(WSInfo.TSInfo.OpenedLine));
  WriteLn( InFile,
'8 ---------------------------  Opened Line  -------------------------');
  WriteLn( InFile,'8  BUS          BUS          Start   End');
  WriteLn( InFile, '19 ', PBus( NetWork^.At(PL^.Con1))^.Name:12,'  ',
          PBus( NetWork^.At(PL^.Con2))^.Name:12,'  ',
          WSInfo.TSInfo.OLt0:6:4,'  ', WSInfo.TSInfo.OLt1:6:4 );
  WriteLn( InFile,
  '8  BUS          BUS          Rs       Xs       Gs       Bs');
  WriteLn( InFile, '25 ', PBus( NetWork^.At(PL^.Con1))^.Name:12,'  ',
          PBus( NetWork^.At(PL^.Con2))^.Name:12,'  ',
          PL^.ZR:7:5,'  ', PL^.ZX:7:5,'  ',
          PL^.G:7:5,'  ', PL^.B:7:5);
        end;

{ Card 20 : SwitchedLoad }
  if WSInfo.TSInfo.SwitchedLoad<>-1 then
          begin
          WSInfo.TSInfo.Code:=WSInfo.TSInfo.Code or id_SwitchedLoad;
  WriteLn( InFile,
'8 ------------------------  Increased Bus Load  ----------------------');
  WriteLn( InFile,'8  Bus          Start   End     Pl       Ql       ');
```

```
   WriteLn( InFile, '20 ',
            PBus( NetWork^.At( WSInfo.TSInfo.SwitchedLoad))^.Name:12,'  ',
            WSInfo.TSInfo.SLt0:6:4,'  ', WSInfo.TSInfo.SLt1:6:4,'  ',
            WSInfo.TSInfo.SLPL:7:5,'  ',WSInfo.TSInfo.SLQL:7:5 );
           end;

{ Card 21 : DroppedLoad }
   if WSInfo.TSInfo.DroppedLoad<>-1 then
           begin
           WSInfo.TSInfo.Code:=WSInfo.TSInfo.Code or id_DroppedLoad;
   WriteLn( InFile,
'8 -------------------------  Dropped Bus Load  ------------------------');
   WriteLn( InFile,'8  Bus          Start   End     Pl       Ql       ');
   WriteLn( InFile, '21 ',
            PBus( NetWork^.At( WSInfo.TSInfo.DroppedLoad))^.Name:12,'  ',
            WSInfo.TSInfo.DLt0:6:4,'  ', WSInfo.TSInfo.DLt1:6:4,'  ',
            WSInfo.TSInfo.DLPL:7:5,'  ',WSInfo.TSInfo.DLQL:7:5 );
           end;

{ Card 24 : LossGen }
   if WSInfo.TSInfo.LossGen<>-1 then
           begin
           if BusColl^.IndexOf(
                       PBus( NetWork^.At( WSInfo.TSInfo.LossGen)))>Npv
                           then
                           begin
                           MessageBox( HWindow,
                   'The Bus which will lose his Generator is not a PV Bus',
                   'Error in data',
                           MB_Ok or MB_IconExclamation);
                           end;
           WSInfo.TSInfo.Code:=WSInfo.TSInfo.Code or id_LossGen;
   WriteLn( InFile,
'8 -------------------------  Lost Bus Generator  ------------------------');
   WriteLn( InFile,'8  Bus          Start   End');
   WriteLn( InFile, '24 ',
            PBus( NetWork^.At( WSInfo.TSInfo.LossGen))^.Name:12,'  ',
            WSInfo.TSInfo.LGt0:6:4,'  ', WSInfo.TSInfo.LGt1:6:4);
           end;

{ Card 22 : Simulation Parameters }
   WriteLn( InFile,
'8 ------------------------  Simulation Parameters  ----------------------');
   WriteLn( InFile,
'8  dt      End     Nimax Tol       eps       Fault freq');
   WriteLn( InFile, '22 ',
    WSInfo.TSInfo.dt:6:4,'  ', WSInfo.TSInfo.t2:6:4,'  ',
    WSInfo.TSInfo.NIterMax:4,'  ',WSInfo.TSInfo.Tol:8,'  ',
    WSInfo.TSInfo.eps:8,'  ',
    WSInfo.TSInfo.Code:4,'  ',WSInfo.TSInfo.freq:6:2 );
    if WSInfo.TSInfo.dt< 0.0001 then
                    begin
                    MessageBox( HWindow,
                      'The time step ( dt ) is too small',
                      'Error in data',
```

```
                    MB_Ok or MB_IconExclamation);
                    end;

{ Card 23 : TransStab Output Options }
   WriteLn( InFile,
'8 ----------------  Transient Stability Output Options  ----------------');
   WriteLn( InFile, '23 ', TSOpt);

  end;{end for Trans Stab study }
{end for every study }
WriteLn( InFile,
  '8 ------------------------  END OF INPUT FILE  ------------------------');
{$I-}
Close(InFile);
{$I+}
If IOResult<>0 Then
        begin
        case WSInfo.Study of
            id_LoadFlow  : StrCopy( SS, InName);
            id_TransStab : StrCopy( SS, TSInName);
          end;
        MessageBox (HWindow, SS
                ,'Closing File Error',mb_Ok+mb_ICONExclamation);
        Exit;
        end;
strcopy( Name, 'NOTEPAD.EXE ');
case WSInfo.Study of
 id_LoadFlow :  strcat( Name, InName);
 id_TransStab : strcat( Name, TSInName);
 end;
WinExec( Name, SW_SHOW);
End;
```