Relatório de

Laboratórios de Comunicações 1

- Momento 2 -

Ricardo Maciel, nº 50037
ricardomcl@yahoo.com

Ângelo Alves, nº 48320
angeloalves@netcabo.pt

Joana Silva, nº 50027
a50027@alunos.uminho.pt

Grupo
204

Índice

Introdução

Este projecto visa o desenvolvimento de um programa para análise de circuitos de corrente contínua que, a partir de uma descrição do circuito, permita obter as tensões nos nós.

Neste segundo momento temos presentes três etapas principais: algoritmo não refinado, algoritmo refinado e fluxograma do algoritmo refinado.

O algoritmo não refinado está escrito numa linguagem natural e serviu de base à elaboração do algoritmo refinado. Neste algoritmo estão presentes os passos mais gerais e os caminhos que decidimos escolher.

O algoritmo refinado está escrito numa linguagem natural, já muito próxima da linguagem de programação com o objectivo de mais tarde, quando for para implementar em linguagem C seja mais fácil e rápida essa passagem e assim pouparmos muito tempo.

O fluxograma é a representação gráfica do algoritmo refinado., onde todos os passos estão descritos esquematicamente.

É de notar que compreendemos por algoritmo, como um procedimento computacional bem definido que toma como parâmetro de entrada um valor (ou um conjunto de valores). Ou seja, é uma sequência de passos computacionais que transformam um input (valor ou valores de entrada) num output (valor ou valores de saída).

Este algoritmo é ser capaz de efectuar a leitura de um ficheiro com a descrição de um circuito e obter a o sistema de equações na forma matricial.

Depois de devidamente estruturado, o algoritmo será resolver as equações lineares, obtendo assim, as equações dos nós.

Para a resolução deste problema, deveremos ter conhecimentos, previamente obtidos em várias unidades curriculares. Nomeadamente:

- a unidade curricular de Análise de Circuitos, para obtenção e resolução das equações lineares, através do método das tensões nodais;
- a unidade curricular de Álgebra linear, para a obtenção das equações lineares sob a forma matricial;
- a unidade curricular de Métodos de Programação para a elaboração do algoritmo e futura implementação em código C da solução final.

Algoritmo não refinado

1. Ler Ficheiro
   1.1. Filtrar do ficheiro netlist apenas as linhas que interessam: UR, R, I
   1.2. Guardar dados de cada componente numa estrutura de apoio
2. Criar matriz dos coeficientes e termos independentes
   2.1. Calcular nº de nós do circuito
   2.2. Passar valores dos variados componentes para a matriz
3. Calcular os valores das tensões nos nós
   3.1. Passar a matriz para a forma em escada
   3.2. Passar a matriz resultante para a forma em escada reduzida
4. Apresentar resultados

Algoritmo Refinado

```
[INICIO]
1. [abrir ficheiro]
   1.1. Ler nome de ficheiro
   1.2. Verificar se o ficheiro existe
   1.3. Enquanto ficheiro não existir Fazer
          Pedir nome de ficheiro
        FimEnquanto
2. [carregar dados do ficheiro para estrutura de apoio]
   2.1. [iniciar variável contadora de componentes]
        2.1.1. ncomp ← 1
   2.2. [percorrer o ficheiro linha a linha]
        2.2.1. Enquanto não chegar ao fim do ficheiro Fazer
                 Ler test_str
                 Caso test_str Seja
                   "UR" Fazer: ncomp ← ncomp + 1
                               netlist[ncomp].c ← test_str
                               Ler netlist[ncomp].r, netlist[ncomp].n1, netlist[ncomp].n2, netlist[ncomp].v1,
netlist[ncomp].v2
                   "R" Fazer:  ncomp ← ncomp + 1
                               netlist[ncomp].c ← test_str
                               Ler netlist[ncomp].r, netlist[ncomp].n1, netlist[ncomp].n2, netlist[ncomp].v1
                   "I" Fazer:  ncomp ← ncomp + 1
                               netlist[ncomp].c ← test_str
                               Ler netlist[ncomp].r, netlist[ncomp].n1, netlist[ncomp].n2, netlist[ncomp].v1
                   "//" Fazer: Mudar para a linha seguinte
                   Outra coisa qualquer Fazer: Apresentar mensagem de erro e Terminar
                 FimCaso
                 Se test_str <> "//" Fazer
                   Se netlist[ncomp].n1 = 0 Fazer
                     Apresentar mensagem de erro e Terminar
                   FimSe
                 FimSe
               FimEnquanto
   2.3. Fechar ficheiro
3. [calcular número de nós]
   3.1. num_nodes ← 0;
   3.2. Para count ← 1,2,... até ncomp Fazer
          Se netlist[count].n1 > num_nodes Fazer
            num_nodes ← netlist[count].n1
          FimSe
          Se netlis[count].n2 > num_nodes Fazer
            num_nodes ← netlist[count].n2
          FimSe
        FimPara
4. [enviar dados para a matriz final]
   4.1 Para count ← 1, 2,... até ncomp Fazer
         Caso netlist[count].c Seja
           "UR" Fazer: Se netlist[count].n2 <> 0 Fazer
                         matrix[netlist[count].n1][netlist[count].n1] ← matrix[netlist[count].n1][netlist[count].n1]
+ (1/[netlist[count].v2)
                         matrix[netlist[count].n2][netlist[count].n2] ← matrix[netlist[count].n2][netlist[count].n2]
+ (1/[netlist[count].v2)
                         matrix[netlist[count].n1][netlist[count].n2] ← matrix[netlist[count].n1][netlist[count].n2]
– (1/[netlist[count].v2)
                         matrix[netlist[count].n2][netlist[count].n1] ← matrix[netlist[count].n2][netlist[count].n1]
– (1/[netlist[count].v2)
                         matrix[netlist[count].n1][num_nodes+1] ← matrix[netlist[count].n1][num_nodes+1] +
(netlist[count].v1/netlist[count].v2)
                         matrix[netlist[count].n2][num_nodes+1] ← matrix[netlist[count].n2][num_nodes+1] -
(netlist[count].v1/netlist[count].v2)
                       Senão
                         matrix[netlist[count].n1][netlist[count].n1] ← matrix[netlist[cont].n1][netlist[count].n1] +
(1/netlist[count].v2)
                         matrix[netlist[count].n1][num_modes+1] ← matrix[netlist[count].n1][num_modes+1] +
(netlist[count].v1/netlist[count].v2)
                       FimSe
           "R" Fazer:  Se netlist[count].n2 <> 0 Fazer
                         matrix[netlist[count].n1][netlist[count].n1] ← matrix[netlist[count].n1][netlist[count].n1]
+ (1/[netlist[count].v1)
```

```
                         matrix[netlist[count].n2][netlist[count].n2] ← matrix[netlist[count].n2][netlist[count].n2]
+ (1/[netlist[count].v1)
                         matrix[netlist[count].n1][netlist[count].n2] ← matrix[netlist[count].n1][netlist[count].n2]
– (1/[netlist[count].v1)
                         matrix[netlist[count].n2][netlist[count].n1] ← matrix[netlist[count].n2][netlist[count].n1]
– (1/[netlist[count].v1)
                       Senão
                         matrix[netlist[count].n1][netlist[count].n1] ← matrix[netlist[cont].n1][netlist[count].n1] +
(1/netlist[count].v1)
                       FimSe
           "I" Fazer:  Se (netlist[count].n1 <> 0) ∧ (netlist[count].n2 <> 0) Fazer
                        matrix[netlist[count].n1][num_nodes+1] ← matrix[netlist[count].n1][num_nodes+1] +
netlist[count].v1
                        matrix[netlist[count].n2][num_nodes+1] ← matrix[netlist[count].n2][num_nodes+1] -
netlist[count].v1
                      Senão
                        matrix[netlist[count].n1][num_nodes+1] ← matrix[netlist[count].n1][num_nodes+1] +
netlist[count].v1
                      FimSe
        FimCaso

5. [colocar matriz em escada]
   5.1. i ← 1
   5.2. [descobrir o indice do valor não nulo mais à esquerda da matriz]
        5.2.1. Para j ← 1,2,... até num_nodes+1 Fazer
                 Para l ← 1,2,... até num_nodes Fazer
                   Se matrix[l][j] <> 0 Fazer
                     Ir para o passo 5.3
                   FimSe
                 FimPara
               FimPara
   5.3. [selecionar o pivot]
        5.3.1. Se matrix[i][j] = 0 Fazer
                 aux ← matrix[i]
                 matrix[i] ← matrix[q]
                 matrix[q] ← aux
               Fim Se
   5.4. [anular elementos abaixo do pivot]
        5.4.1. Para p ← i+1,i+2,... até num_nodes Fazer
                 matrix[p] ← matrix[p] – (matrix[p][j]/matrix[i][j])*matrix[i]
               FimPara
   5.5. [está em escada?]
        5.5.1. Para l ← 1,2,... até num_nodes Fazer
                 nzeros[l] = 0
                 c ← 1
                 Enquanto (matrix[l][c] = 0) ∧ (c <= num_nodes) Fazer
                   c ← c + 1
                   nzeros[l] ← nzeros[l] + 1
                 FimEnquanto
                 Se l <> 1 Fazer
                   Se nzeros[l] <= nzero[l-1] Fazer
                    Ir para o passo 5.5.3
                   FimSe
                 FimSe
               FimPara
        5.5.2. Ir para o passo 6
        5.5.3. i ← i + 1
        5.5.4. Para j ← 1,2,... até num_nodes+1 Fazer
                 Para l ← i,i+1,... até num_nodes Fazer
                   Se matrix[l][j] <> 0 Fazer
                     Ir para o passo 5.3
                   FimSe
                 FimPara
               FimPara
6. [colocar matriz em escada reduzida]
   6.1. [encontrar pivot]
        Para i ← num_nodes, num_nodes-1,... até 1 Fazer
          Para j ← 1,2,... até num_nodes Fazer
            Se matrix[i][j] <> 0 Fazer
              Ir para o passo 6.2
            FimSe
          FimPara
```

```
        FimPara
   6.2. Se matrix[i][j] <> 1 Fazer
          matrix[i] ← (1/matrix[i][j])*matrix[i]
        FimSe
   6.3. Para p ← 1,2,... até i-1 Fazer
          matrix[p] ← matrix[p] – (matrix[p][j] * matrix[i])
        FimPara
   6.4. [está em escada reduzida?]
        6.4.1. Para l ← i-1,i-2,... até 1 Fazer
                 Para c ← 1,2,... até j Fazer
                   Se matrix[l][c] <> 0 Fazer
                     Se matrix[l][c] = 1 Fazer
                       Para l2 ← l-1,l-2,... até 1 Fazer
                         Se matrix[l2][c] <> 0 Fazer
                           Ir para o passo 6.4.3
                         FimSe
                       FimPara
                     Senão
                       Ir para o passo 6.4.3
                     FimSe
                   FimSe
                 FimPara
               FimPara
        6.4.2. Ir para o passo 7
        6.4.3. i ← l
        6.4.4. j ← c
        6.4.5. Ir para o passo 6.2
7. [Apresentar resultados]
   7.1 Para a ← 1,2,... num_nodes Fazer
         Escrever no ecrã: "U", a, " = ", matrix[a][num_nodes+1]
         Mudar de linha
       FimPara
[FIM]
```

Descrição do algoritmo

Este algoritmo utiliza como memória de apoio para os dados lidos a partir do ficheiro netlist, uma estrutura de variáveis associadas a um vector. Cada elemento desse vector (que é chamado de *netlist*), tem associadas 5 variáveis que vão guardar os dados relativos a cada componente. Essas variáveis são, para cada elemento do vector *netlist*, as seguintes:

- *c*, para o tipo de componente: UR, R ou I;
- *n1* e *n2* para os nós entre os quais o componente está ligado;
- *v1* para o valor do componente, para o caso das resistências (R) e fontes de corrente (I);
- *v2* que guarda o valor da resistência, para o caso das fontes de tensão em série com uma resistência (UR).

Para calcular os valores das tensões nos nós, que é o resultado pretendido, utilizamos apenas uma matriz, a matriz das equações, a qual é referenciada por *matrix* no nosso algoritmo. Nela serão guardados, através de um método explicado mais à frente, os valores dos coeficientes e dos termos independentes. Aqui, cada linha corresponderá às equações de cada nó e cada coluna guarda os coeficientes de cada incógnita (U_1, U_2, U_3, etc…). Consideramos que com este método, os resultados pretendidos seriam obtidos de uma maneira mais simples e eficaz, pois iremos manipular apenas uma matriz, utilizando os métodos aprendidos nas aulas de Álgebra Linear B.

A leitura do ficheiro netlist foi feita com uma abordagem linha a linha, ou seja, em cada passo lemos todos os valores relativos a um componente, para as variáveis da estrutura mencionada acima. A variável *test_str* guarda apenas a primeira expressão de cada linha, de modo a que se possa depois decidir que operações vão ser executadas. No caso de *test_str* ser igual a "//" o resto é ignorado, passando para a linha seguinte. pois para cada componente existe pelo menos uma operação que difere entre eles, que é a atribuição da variável c (que guarda os caracteres "UR", "R" ou "I", conforme o caso). A variável *ncomp* é incrementada sempre que são lidos os valores de um componente. O seu objectivo é contar o número de componentes lidos, dado esse que nos vai ser útil finda a leitura deste ficheiro, pois será usada finda a leitura do ficheiro. Este consiste na determinação do número de nós, recorrendo, para o efeito, a uma variável *num_nodes* (inicializada a zero) e a um ciclo limitado pela variável *ncomp*. Dentro deste ciclo, em cada iteração, *num_nodes* tomará o valor de *n1* ou *n2* caso estes tenham valores numericamente maiores que *num_nodes*.

Em seguida passamos os dados guardados na estrutura *netlist* para a matriz das equações. Para isso usamos o ciclo que incrementa a variável *count* até ao valor de *ncomp* (nº de componentes guardados na estrutura), e percorre toda a estrutura, linha a linha. Dentro é efectuado o teste com a variável *c* (tipo de componente), de modo a decidir que operações efectuar, operações essas que consistem na utilização de *n1* e *n2* como índices da matriz das equações, ou seja, as posições dos *v1* e *v2* nessa matriz. O método utilizado para esse posicionamento é o seguinte:

- Para o caso das fontes de tensão (UR), se *n2* for diferente de 0, são usados os dois nós (*n1* e *n2*) da seguinte forma: o inverso do valor da resistência (1/*v2*) é colocado nas posições (*n1,n1*) e (*n2,n2*) somando ao que já lá existir, e nas posições (*n1,n2*) e (*n2,n1*) subtraindo ao que já lá existir. Para a coluna dos termos independentes (num_nodes+1) vai o resultado de *v1/v2*, na linha *n1*, a somar, ou na linha *n2*, a subtrair. Se *n2* for 0 é apenas usado *n1*, ou seja, o inverso de *v2* vai ser somado ao valor que se encontrar na posição (*n1*,*n1*) e na posição (*num_nodes+1,n1*)(isto é, na coluna dos termos independentes, na linha *n1*);
- Para o caso das resistências (R), é usado o mesmo raciocínio só que não vai ser utilizada a coluna dos termos independentes, ou seja, não vai haver posicionamento na coluna *num_nodes+1* e em vez de *v2* será apenas usado *v1*, como é óbvio.
- Para o caso das fontes de tensão (*I*), se *n1* for diferente de 0, o seu valor (*v1*) será somado ao valor que existir na linha *n1* da coluna dos termos independentes, ou seja, vai para a posição (*num_nodes+1*,*n1*) e será subtraído ao que existir na linha *n2* da mesma coluna.

Este método surgiu da análise de variados circuitos de exemplo.

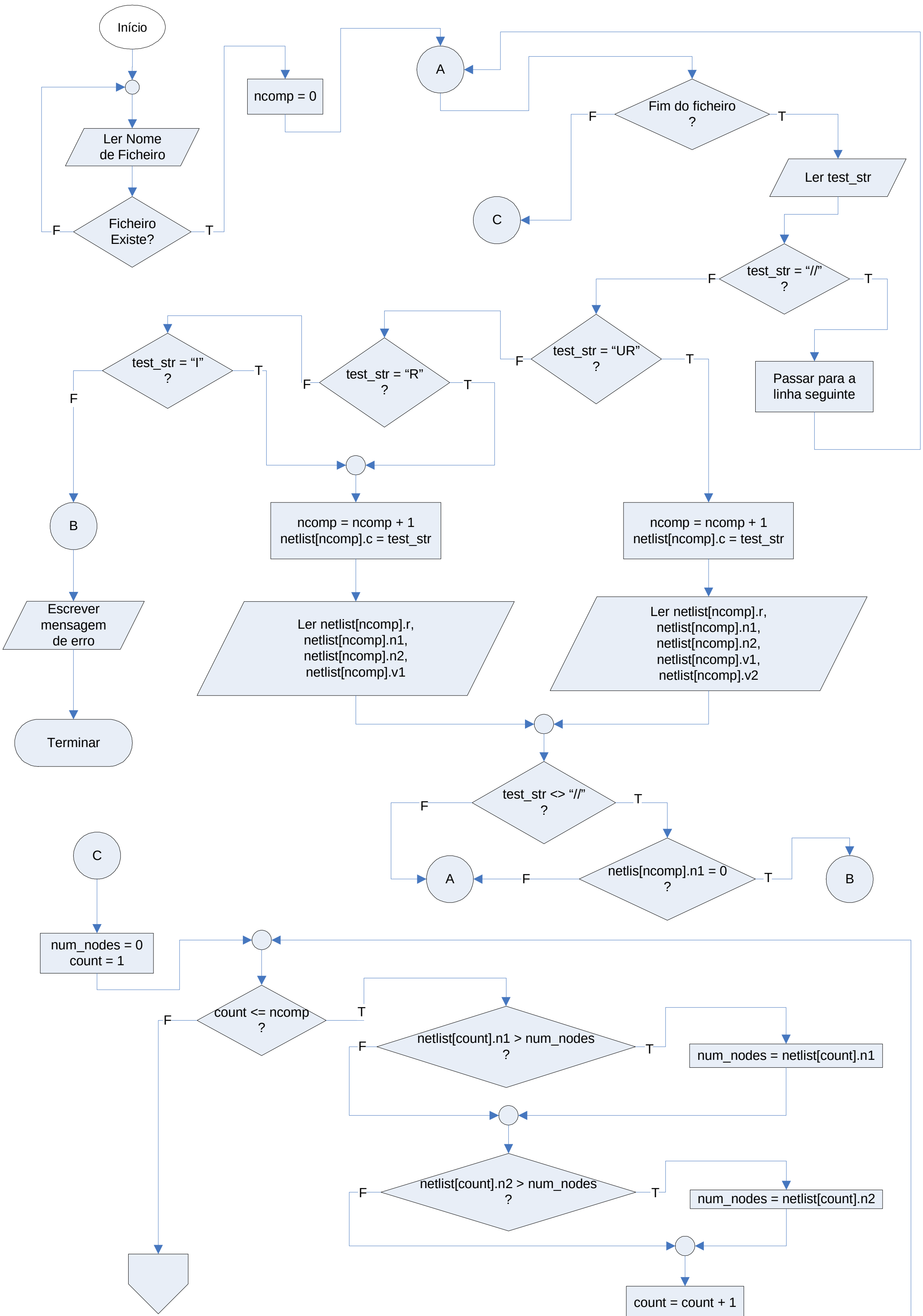
O passo seguinte consiste na adaptação dos algoritmos que nos foram apresentados nas aulas de Álgebra Linear B, os quais descrevem as operações a ser usadas para a obtenção da forma em escada reduzida da matriz das equações. A coluna *num_nodes+1* dessa matriz resultante vai conter os valores das tensões dos vários nós.

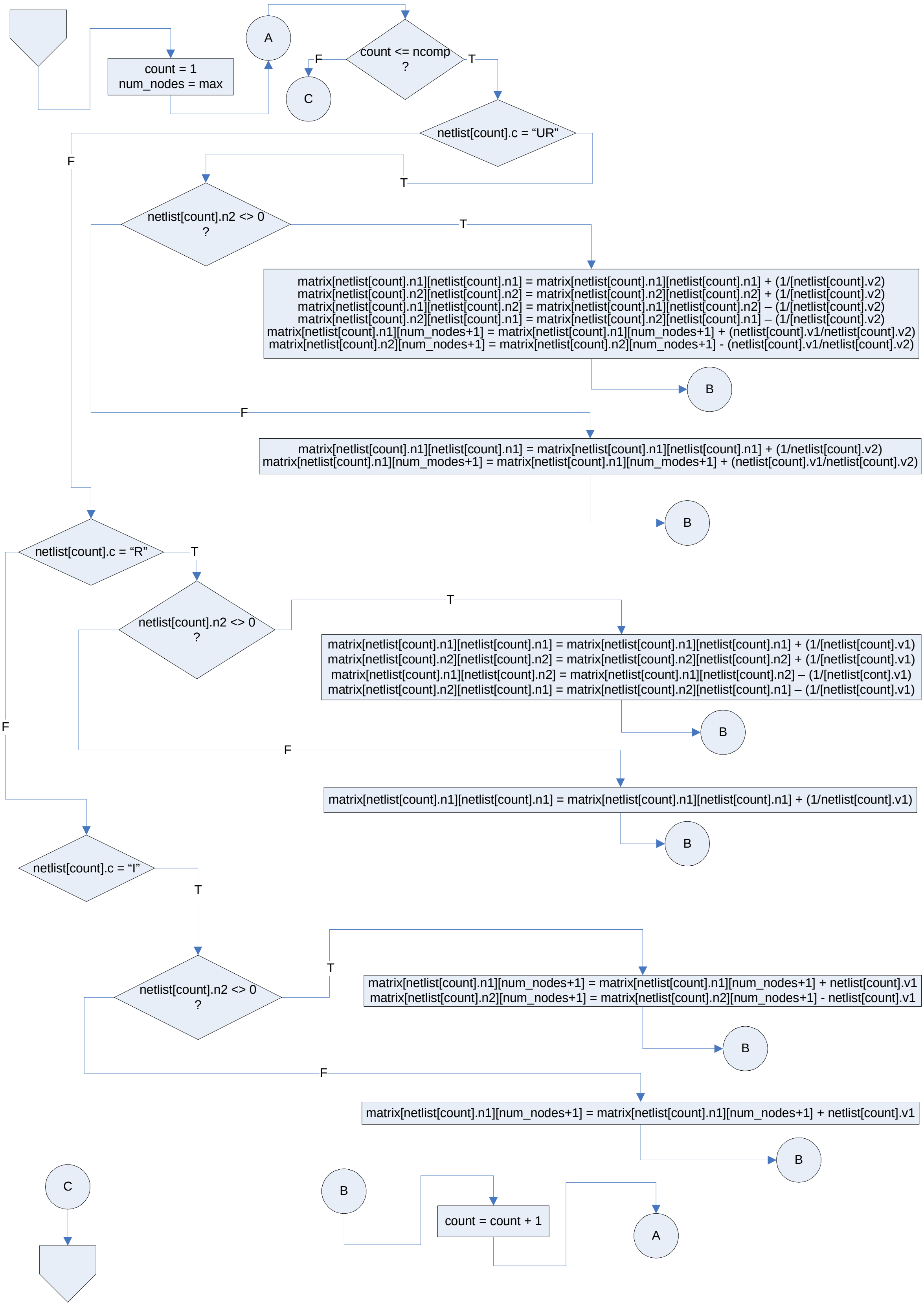
Primeiramente inicializamos a 1 a variável *i*, que nesta fase vai correspondes ao índice da linha 1 da matriz. Em seguida, para descobrir o índice do valor não nulo, mais à esquerda da matriz, recorremos a dois ciclos. Um para incrementar *c* (que corresponde ao índice das colunas) e outro para *l* (linhas). Ambos os ciclos usam a variável *ncomp* como limite, com a diferença de no ciclo de *l*, esta ser somada de 1. Isto quer dizer que, como *ncomp* é o número de nós, ou seja, o número de incógnitas, isso implica que as colunas usadas para guardar os coeficientes são limitadas por *ncomp*, sendo *ncomp+1* a coluna dos termos independentes. E esta é a vantagem em usarmos uma só matriz para ambos os tipos de dados (coeficientes e termos independentes). Voltando agora à explicação do ciclo em questão, este vai percorrer todas as linhas, coluna a coluna, ficando este interrompido quando for encontrado um valor diferente de zero. Ficam assim guardados os índices da linha (*l*) e da coluna (*j*) onde foi encontrado esse valor. Seguidamente testamos se o valor da primeira linha dessa coluna *j* é igual a zero. Se for, trocamos a linha *i* com a linha *l*, com a ajuda do vector *aux*. Depois, para anularmos (passar a zero) todos os elementos da coluna *j* abaixo da linha *i*, usamos outro ciclo que incrementa a variável p desde *i+1* (a linha abaixo) até *num_nodes* (a última linha) e efectua a operação indicada, isto é, subtrai a todos os elementos da linha *p*, o produto dos elementos da linha *i*, pela divisão entre o elemento que queremos anular (elemento da linha *p,* coluna j) *e* o pivot (elemento da linha i, coluna j). Concluído o ciclo, é altura de verificar se a matriz que resultou destas operações, já está na forma em escada. Para isso utilizamos a definição, que diz que para uma matriz estar na forma em escada, o número de elementos nulos que precedem o pivot aumenta de linha para linha até que, possivelmente, sobrem apenas linhas nulas. E é isso que é feito no passo seguinte. Aqui é usado um vector (*nzeros*) que vai guardando para cada linha o número de zeros que encontra, até ao primeiro valor não nulo dessa linha. Em cada iteração do ciclo que incrementa o índice da linha (*l*), é efectuada uma comparação entre o número de zeros de duas linhas adjacentes (*l* e *l-1*). Se o número de zeros da linha acima (*l*-1) for superior ou igual ao da linha actual (*l*), o ciclo acaba e fica determinado que a matriz ainda não se encontra na forma em escada. Nesse caso a variável *i* é incrementada, é utilizado o método já descrito acima, para descobrir o índice não nulo mais à esquerda da matriz, mas desta vez partindo da linha *i*. Caso contrário o ciclo continua e se, quando *l* atingir *num_nodes* (for atingida a última linha), a condição se mantiver falsa, a matriz está na forma em escada, sendo agora necessário transformá-la na forma em escada reduzida.

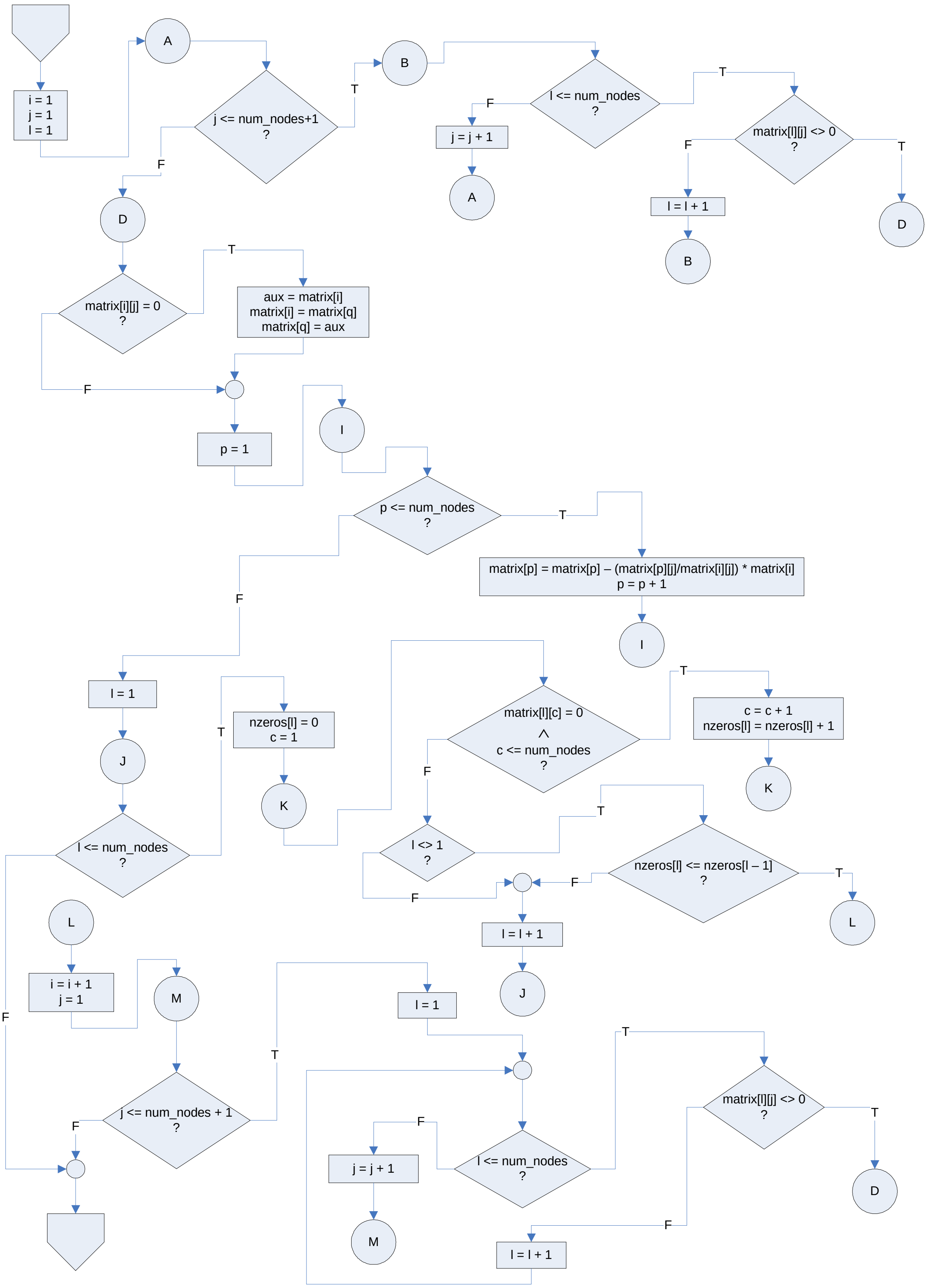
Vamos agora percorrer a matriz de baixo para cima, no ciclo que se segue, para encontrar o elemento pivot. Para isso percorremos linha a linha todos os elementos até encontrar um que seja diferente de zero. Isso acontecendo, passamos para o passo seguinte, que consiste na verificação desse elemento. Se for deferente de 1 (um) teremos de multiplicar todos os elementos dessa linha (neste caso é indicada pela variável *i*), pelo seu inverso, de modo a que esse elemento (*matrix[i][j]*) passe também a ser 1 (um). Em seguida anulamos todos os elementos acima do pivot, recorrendo mais uma vez a um ciclo que percorra as linhas, de cima até a linha anterior. Em cada iteração do ciclo efectuamos para cada elemento de cada linha, a sua subtracção com o produto do elemento que queremos anular (dado por *matriz[p][j]*) pelo elemento pivot, que agora é 1 (um). Findo o ciclo, está na altura de verificar se a matriz já se encontra na forma em escada reduzida. A definição diz que todos os elementos pivot tem de ser iguais a 1 (um) e todos elementos acima desses pivots têm de ser nulos (diferentes de zero). Para isso usamos, mais uma vez, dois ciclos. Um que percorre as linhas e outro as colunas. Quando encontrar um elemento diferente de zero (*matrix[l][c]<>0*), testa também se é igual a 1 (um). Se não for, os ciclos são interrompidos, *i* toma o valor de *l*, *j* toma o valor de c e o programa continua a execução em 6.2. Se for igual a 1 (um), vai verificar com a ajuda de outro ciclo que percorre com *l2*, a coluna em questão (*c*), de baixo para cima (a começar na linha acima), a existência de algum elemento diferente de zero. Se não encontrar, o ciclo continua. Quando *l* atinge o valor 1 (a linha superior da matriz), o ciclo acaba, e se elemento pivot dessa linha for igual a 1 (um), considera-se também que a matriz está em escada reduzida.

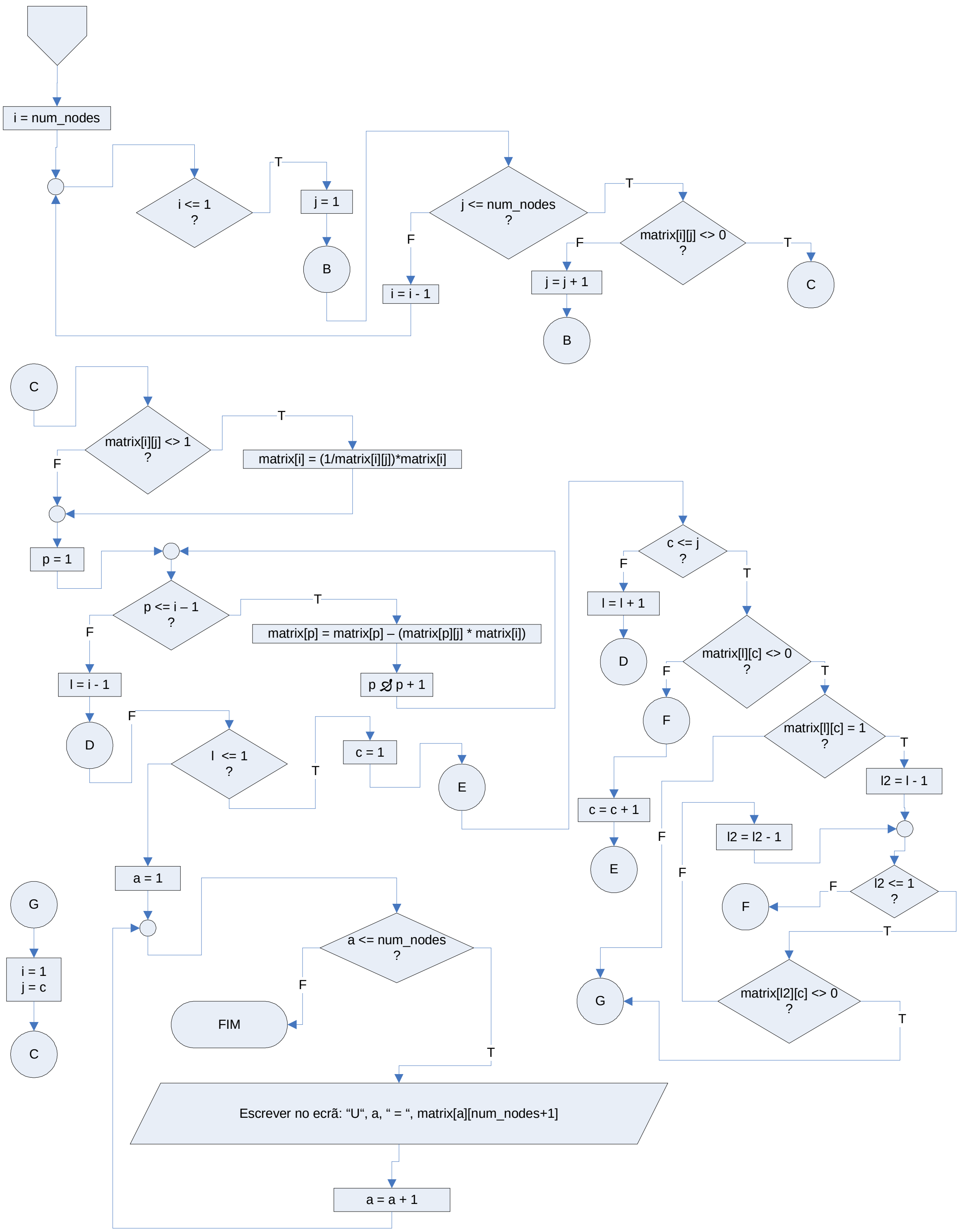
Posto isto, resta apresentar os resultados que se encontram na coluna *num_nodes+1*, usando para o efeito, mais um ciclo que percorre as linhas dessa coluna com a variável *a*.

Fluxograma


Início
Ler Nome de Ficheiro
Ficheiro Existe?
F
T
ncomp = 0
A
Fim do ficheiro ?
F
T
Ler test_str
C
test_str = "//" ?
F
T
Passar para a linha seguinte
test_str = "UR" ?
F
T
test_str = "R" ?
F
T
test_str = "I" ?
F
T
B
Escrever mensagem de erro
Terminar
ncomp = ncomp + 1
netlist[ncomp].c = test_str
Ler netlist[ncomp].r, netlist[ncomp].n1, netlist[ncomp].n2, netlist[ncomp].v1
ncomp = ncomp + 1
netlist[ncomp].c = test_str
Ler netlist[ncomp].r, netlist[ncomp].n1, netlist[ncomp].n2, netlist[ncomp].v1, netlist[ncomp].v2
test_str <> "//" ?
F
T
A
netlis[ncomp].n1 = 0 ?
F
T
B
C
num_nodes = 0
count = 1
count <= ncomp ?
F
T
netlist[count].n1 > num_nodes ?
F
T
num_nodes = netlist[count].n1
netlist[count].n2 > num_nodes ?
F
T
num_nodes = netlist[count].n2
count = count + 1


count = 1
num_nodes = max
A
count <= ncomp
?
F
T
C
netlist[count].c = “UR”
F
T
netlist[count].n2 <> 0
?
T
matrix[netlist[count].n1][netlist[count].n1] = matrix[netlist[count].n1][netlist[count].n1] + (1/[netlist[count].v2)
matrix[netlist[count].n2][netlist[count].n2] = matrix[netlist[count].n2][netlist[count].n2] + (1/[netlist[count].v2)
matrix[netlist[count].n1][netlist[count].n2] = matrix[netlist[count].n1][netlist[count].n2] – (1/[netlist[count].v2)
matrix[netlist[count].n2][netlist[count].n1] = matrix[netlist[count].n2][netlist[count].n1] – (1/[netlist[count].v2)
matrix[netlist[count].n1][num_nodes+1] = matrix[netlist[count].n1][num_nodes+1] + (netlist[count].v1/netlist[count].v2)
matrix[netlist[count].n2][num_nodes+1] = matrix[netlist[count].n2][num_nodes+1] - (netlist[count].v1/netlist[count].v2)
B
F
matrix[netlist[count].n1][netlist[count].n1] = matrix[netlist[count].n1][netlist[count].n1] + (1/netlist[count].v2)
matrix[netlist[count].n1][num_modes+1] = matrix[netlist[count].n1][num_modes+1] + (netlist[count].v1/netlist[count].v2)
B
netlist[count].c = “R”
T
netlist[count].n2 <> 0
?
T
matrix[netlist[count].n1][netlist[count].n1] = matrix[netlist[count].n1][netlist[count].n1] + (1/[netlist[count].v1)
matrix[netlist[count].n2][netlist[count].n2] = matrix[netlist[count].n2][netlist[count].n2] + (1/[netlist[count].v1)
matrix[netlist[count].n1][netlist[count].n2] = matrix[netlist[count].n1][netlist[count].n2] – (1/[netlist[cont].v1)
matrix[netlist[count].n2][netlist[count].n1] = matrix[netlist[count].n2][netlist[count].n1] – (1/[netlist[count].v1)
B
F
F
matrix[netlist[count].n1][netlist[count].n1] = matrix[netlist[count].n1][netlist[count].n1] + (1/netlist[count].v1)
B
netlist[count].c = “I”
T
netlist[count].n2 <> 0
?
T
matrix[netlist[count].n1][num_nodes+1] = matrix[netlist[count].n1][num_nodes+1] + netlist[count].v1
matrix[netlist[count].n2][num_nodes+1] = matrix[netlist[count].n2][num_nodes+1] - netlist[count].v1
B
F
matrix[netlist[count].n1][num_nodes+1] = matrix[netlist[count].n1][num_nodes+1] + netlist[count].v1
B
C
B
count = count + 1
A


i = 1
j = 1
l = 1
A
j <= num_nodes+1
?
T
F
B
l <= num_nodes
?
F
j = j + 1
A
T
matrix[l][j] <> 0
?
F
T
l = l + 1
B
D
D
matrix[i][j] = 0
?
T
aux = matrix[i]
matrix[i] = matrix[q]
matrix[q] = aux
F
p = 1
I
p <= num_nodes
?
T
matrix[p] = matrix[p] – (matrix[p][j]/matrix[i][j]) * matrix[i]
p = p + 1
I
F
l = 1
J
l <= num_nodes
?
T
nzeros[l] = 0
c = 1
K
matrix[l][c] = 0
^
c <= num_nodes
?
T
c = c + 1
nzeros[l] = nzeros[l] + 1
K
F
l <> 1
?
T
nzeros[l] <= nzeros[l – 1]
?
T
L
F
F
l = l + 1
J
L
i = i + 1
j = 1
M
j <= num_nodes + 1
?
T
F
F
l = 1
T
matrix[l][j] <> 0
?
T
D
F
j = j + 1
M
l <= num_nodes
?
l = l + 1


i = num_nodes
i <= 1
?
T
j = 1
B
j <= num_nodes
?
F
i = i - 1
T
matrix[i][j] <> 0
?
F
j = j + 1
B
T
C
C
matrix[i][j] <> 1
?
T
matrix[i] = (1/matrix[i][j])*matrix[i]
F
p = 1
p <= i – 1
?
T
matrix[p] = matrix[p] – (matrix[p][j] * matrix[i])
F
l = i - 1
D
F
l <= 1
?
T
c = 1
E
c <= j
?
F
l = l + 1
D
T
matrix[l][c] <> 0
?
F
F
T
matrix[l][c] = 1
?
T
l2 = l - 1
c = c + 1
E
F
l2 = l2 - 1
F
l2 <= 1
?
F
F
T
matrix[l2][c] <> 0
?
T
G
a = 1
a <= num_nodes
?
F
FIM
T
Escrever no ecrã: “U“, a, “ = “, matrix[a][num_nodes+1]
a = a + 1
G
i = 1
j = c
C

Análise Crítica

Durante a elaboração deste relatório deparámo-nos com algumas dificuldades, nomeadamente com a obtenção das tensões nos nós do circuito pelo método das tensões nodais, visto esta parte da matéria não estar bem cimentada.

No início do processo estávamos a pensar percorrer o ficheiro *netlist*, tratando-o como se fosse uma matriz, mas como reparámos que assim seria mais complicado e mais trabalhoso, decidimos optar por outra estratégia, decidindo assim tratar o ficheiro linha a linha, ou melhor dizendo, componente a componente, carregando para a memória de uma só vez, todos os parâmetros relativos a cada componente.

Optámos também por fazer um algoritmo num nível de refinamento já bastante elevado para ser mais fácil a sua implementação em linguagem C.

Para a resolução da matriz e assim obter os valores das tensões dos nós tínhamos dois métodos: o método de Gauss e o método de Gauss-Jordan. O método que decidimos adoptar foi o segundo, pois, embora seja um método mais complicado que o primeiro, estamos mais familiarizados com esta técnica, fruto das aulas de Álgebra Linear B. Para além disso, em termos de raciocínio algorítmico é mais simples e tínhamos mais informações sobre este método.

Conclusão

Analisando todos os caminhos percorridos para a realização deste relatório, podemos concluir que será possível a sua implementação em linguagem C e que o programa conseguirá calcular as tensões dos nós, qualquer que seja o circuito apresentado. Para que tal fosse possível, tivemos o apoio de vários métodos, que foram cima descritos. Foi também com base nestes métodos, que concluímos que esta seria a melhor forma de elaboração do algoritmo.

O nosso algoritmo apresenta um nível de refinamento bastante elevado, pois entendemos que esta seria a melhor maneira para depois podermos efectuar a sua implementação em linguagem C.