Relatório de

Laboratórios de Comunicações 1

- Momento 3 -

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

Neste terceiro momento de avaliação foi apresentado oralmente e executado o protótipo intermédio da solução do projecto de grupo, que no nosso caso já é uma aproximação muito grande da versão final, isto porque resolve até ao fim o problema, resolvendo e apresentando os valores pedidos.

É de notar que compreendemos por protótipo, como uma versão inicial do sistema final que está ainda em fase de testes mas com os objectivos bem defenidos. Visto o protótipo dar origem ao sistema final, este deve ser de fácil manutenção, de boa performance e fiável onde poderão ocorrer alguns erros e assim procedermos à sua rectificação e correcção.

No nosso caso concreto, o protótipo deve ser capaz, a partir de um dado circuito eléctrico, com base no método das tensões nodais, obter as tensões nos nós do circuito.

Neste relatório é apresentado o código fonte do protótipo, bem como a sua descrição, sob a forma de comentários incluídos no próprio código.

Este código foi elaborado no compilador Microsoft Visual Studio 2005.

Protótipo em Linguagem C

```
/************************************************************/
/* Nome: Projecto MIECOM 1a1s – Grupo 204                   */
/* Descricao: Calculo das tensoes nos nos de um dado circuito. */
/* Data de criacao: 24/11/2006                              */
/************************************************************/


#define _CRT_SECURE_NO_DEPRECATE 1    /* this constants are just needed by the compiler because of some */
#define _CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES 1 /* functions that seem to be old and give warnings    */
#include<math.h>                                  /* when compiling                                    */
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>
#define DIM 50

void gotoeol(FILE *in_file);  /* Declaration of the function gotoeol.      */
                              /* This function places the file pointer at  */
                              /* the end of the line.                      */

int main ()
{
   char test_str[3];                /* String that saves the first characters from a line */
   char temp_v1[20], temp_v2[20];   /* Temporary variables to save value1 and value2 */
   char *endptr;                    /* Pointer needed by the function strtod */
   int i, j, c, count, l, p, q,     /* Counter variables, used in 'for' cycles */
       line = 1,                    /* Variable to count line number of the netlist file */
       comp_n = 1,                  /* Counts the number of components in the netlist file */
       num_nodes = 0;               /* Counts the number of nodes of the given circuit...     */
                                    /* this will be needed when working with the final matrix */
   struct NL_TYPE{                  /* Structure to save the components description: */
      char c[3], r;                 /*  - for component and reference*/
      int n1, n2;                   /*  - for node 1 and node 2 */
      float v1, v2;                 /*  - for value 1 and value 2 */
   } netlist[DIM];                  /* structure array */
   FILE *nl_file;                   /* file pointer */
   char filename[255];              /* string to store the filename */
   float matrix[DIM][DIM],          /* this is the matrix of the equations */
        aux[DIM],                   /* this is an auxiliary array used to swap lines in the matrix */
        temp;                       /* variable used in the gauss-jordan transformation */

   /* program start */
   printf("\nIntroduza o nome do ficheiro: ");
   scanf("%s", filename);  /* ask for filename (including path if file is not in the same folder) */
   if ((nl_file = fopen(filename,"r")) == NULL)   /* tests if file exists */
   {
      printf("Ficheiro não encontrado.\n");   /* error message and program exits if file not found */
      _getch();
      return(1);
   }
   while(!feof(nl_file))      /* iniciates file reading and stops when the end of file is reached */
   {
      fscanf(nl_file, "%2s", test_str);  /* this reads the first two characters of each line */
      strcpy(test_str, _strupr(test_str));  /* this turns those two characters into uppercase if they are lowercase */
      if (!strcmp(test_str, "//")) /* if those characters are '//' ... */
         gotoeol(nl_file); /* ... this function ignores this line, and places the file pointer ready to */
                           /* read the next line                                                       */
      else if (!strcmp(test_str, "UR"))  /* verifies if the characters read are 'UR' (power source */
                                         /* in series with a resistance)                           */
           {
              strcpy(netlist[comp_n].c,test_str); /* copy the component characters to the structure variable c */
              fscanf(nl_file," %c %d %d %s %s", &netlist[comp_n].r, &netlist[comp_n].n1, &netlist[comp_n].n2,
temp_v1, temp_v2);     /* reads the values to the structure, except for value 1 and value 2  */
                       /* which are kept in strings....                                      */

              netlist[comp_n].v1 = (float) strtod(temp_v1, &endptr);  /* ... and here are converted to floats */
              netlist[comp_n].v2 = (float) strtod(temp_v2, &endptr);
              comp_n++;     /* this variable is incremented because one component has been read to the structure */
           }
```

```
            else if (!strcmp(test_str, "R"))  /* if test_str is 'R' ... */
                 {
                    strcpy(netlist[comp_n].c,test_str);
                    fscanf(nl_file, " %c %d %d %s", &netlist[comp_n].r, &netlist[comp_n].n1, &netlist[comp_n].n2,
temp_v1);
                    netlist[comp_n].v1 = (float) strtod(temp_v1, &endptr);
                    comp_n++;
                 }
                 else if (!strcmp(test_str, "I")) /* if test_str is 'I' ... */
                      {
                         strcpy(netlist[comp_n].c,test_str);
                         fscanf(nl_file, " %c %d %d %s", &netlist[comp_n].r, &netlist[comp_n].n1, &netlist[comp_n].n2,
temp_v1);
                         netlist[comp_n].v1 = (float) strtod(temp_v1, &endptr);
                         comp_n++;
                      }
                      else
                      {
                         printf("Invalid data on line %d.\n",line);
                         _getch();     /* displays error message and exits program if it encounters */
                         return (1);   /* other character in the beginning of lines                 */
                      }
      if ((netlist[comp_n-1].c[0] != 'I') && (netlist[comp_n-1].n1 == 0))
      {                                                          /* this condition tests if node1 is zero */
         printf("\n\nNo 1 nao pode ser zero. (linha %d)\n", line);  /* sends error message end exits program */
         _getch();                                               /* if its true, but just in case of      */
         return(1);                                              /* UR and R.                             */
      }
      if ((comp_n > 1) && (netlist[comp_n-1].n1 == netlist[comp_n-1].n2))
      {
         printf("\n\nNo 1 nao pode ser igual a No 2. (linha %d)\n", line);
         _getch();          /* displays error message and exits if node 1 and node 2 are the same */
         return(1);
      }
      line++;  /* this variable is just used in the messages above, to tell where the error was found */
   }
   fclose(nl_file); /* closes the file for reading, because the loading of values is done */

   for (count=1; count < comp_n; count++)
   {
      if (netlist[count].n1 > num_nodes)     /* calculate the number of nodes of the circuit. */
         num_nodes = netlist[count].n1;      /* num_nodes is initialized with 0 (when it is declared) and on each */
      if (netlist[count].n2 > num_nodes)     /* iteration of the cycle is compared with the nodes variables and   */
         num_nodes = netlist[count].n2;      /* keeps its values when they are greater.                           */
   }

   for (i=0; i<DIM; i++)          /* this is to fill the matrix and the aux array with zeros */
   {
      for (j=0; j<DIM; j++)
         matrix[i][j] = 0.0;
      aux[i] = 0.0;
   }
   for (count=1; count<comp_n; count++) /* this cycle runs through structure array of components with the count */
                                        /* variable, and loads the values the matrix.                          */
   {
      if (!strcmp(netlist[count].c,"UR"))  /* if c is 'UR' ... */
         if (netlist[count].n2 != 0)
                     /* this tests to the nodes are necessary because they are used as indexes of the matrix */
                     /* where the values will be placed (summed or subtracted).                              */
         {   /* the following two instructions will sum the inverse of the resistance values (v2) */
             /* to the values stored in the diagonal of the matrix                                */
            matrix[netlist[count].n1][netlist[count].n1] += (1/netlist[count].v2);
            matrix[netlist[count].n2][netlist[count].n2] += (1/netlist[count].v2);
            /* the following two will subtract them to the values stored outside the diagonal */
            matrix[netlist[count].n1][netlist[count].n2] -= (1/netlist[count].v2);
            matrix[netlist[count].n2][netlist[count].n1] -= (1/netlist[count].v2);
            /* this will sum the value v1/v2 (tension source divided by resistance) to the value stored */
            /* in the column of the independent terms (num_nodes+1), in the line given by node 1 (n1)   */
            matrix[netlist[count].n1][num_nodes+1] += (netlist[count].v1/netlist[count].v2);
            /* this will subtract the same value to the value placed in the same column, but in the given */
            /* by node 2 (n2)                                                                             */
            matrix[netlist[count].n2][num_nodes+1] -= (netlist[count].v1/netlist[count].v2);
         }
         else
```

```
            /* if node2 is zero, only node1 will be used as index of the matrix.                    */
            /* therefore placing the same values described above (1/v2 and v1/v2) only in the diagonal */
            /* and the independant terms column, respectively.                                      */
      {
            matrix[netlist[count].n1][netlist[count].n1] += (1/netlist[count].v2);
            matrix[netlist[count].n1][num_nodes+1] += ((netlist[count].v1)/(netlist[count].v2));
      }
      else if (netlist[count].c[0] == 'R') /* if c is 'R', the same criteria applies here, except that only */
                                           /* v1 is used and there are no operations to num_nodes+1 column.    */
            if (netlist[count].n2 != 0)
            {
               matrix[netlist[count].n1][netlist[count].n1] += (1/netlist[count].v1);
               matrix[netlist[count].n2][netlist[count].n2] += (1/netlist[count].v1);
               matrix[netlist[count].n1][netlist[count].n2] -= (1/netlist[count].v1);
               matrix[netlist[count].n2][netlist[count].n1] -= (1/netlist[count].v1);
            }
            else
               matrix[netlist[count].n1][netlist[count].n1] += (1/netlist[count].v1);
         else if (netlist[count].c[0] == 'I') /* if c is 'I', the same method is applied, but it only places */
                                              /* the values in the num_nodes+1 column.                      */
               if ((netlist[count].n1 != 0) && (netlist[count].n2 !=0))
               {
                  matrix[netlist[count].n1][num_nodes+1] += netlist[count].v1;
                  matrix[netlist[count].n2][num_nodes+1] -= netlist[count].v1;
               }
               else
                  matrix[netlist[count].n1][num_nodes+1] += netlist[count].v1;
}

/* gauss-jordan transformation */
i = 1; q = 1;     /* initialization of the variables used to run through the matrix */
while (i <= num_nodes)  /* the instructions inside is executed until it reaches the last line of the matrix */
{
   for (j = q; j <= num_nodes+1; j++)  /* cycles to run the matrix and search for the first non-zero element */
   {                                   /* j is for the columns... */
      for (l = i; l <= num_nodes; l++) /* ...and l is for lines.  */
         if (matrix[l][j] != 0)     /* if it's found, it exits the cycle, with l and j variables */
            break;                  /* keeping its values, wich will be used again, later.       */

      if (matrix[l][j] != 0)    /* this is needed to fully exit the cycles */
         break;
   }
   /* find pivot element */
   if (matrix[i][j] == 0)  /* if the element in line i of column j (with j being the column where  */
                           /* the non-zero element was found in the previous cycle)...             */
   {
      for (count = 1; count <= num_nodes+1; count++)
      {
         aux[count] = matrix[i][count];         /* ... line i is swap with line l (wich is the line */
         matrix[i][count] = matrix[l][count];   /* where the non-zero element was found before).    */
         matrix[l][count] = aux[count];
      }                                        /* after this the pivot element is in the position (i,j) */
   }
   /* eliminate elements bellow pivot */
   for   (p = i+1; p <= num_nodes; p++)   /* here the elements below the pivot will be turned zero */
   {
      temp = matrix[p][j];       /* temp variable will hold the value of the element that will be      */
      matrix[p][j] = 0.0;        /* turned zero, because this will be needed in the instruction below  */
      for (count = j+1; count <= num_nodes+1; count++)
         matrix[p][count] = matrix[p][count] - ((temp / (matrix[i][j])) * matrix[i][count]);
                     /* this instruction changes all elements of each line, based on the element */
   }                 /* that was to be turned zero and the pivot                                  */
   i++; q++;   /* this will set the the index of the lines (i) and columns (q) to the next line/column */
}
i--;  /* this is necessary because in the last cycle of the previous while block i was incremented */
      /* after the last line was reached. so it must be decremented in order to set i to the       */
      /* last line of the matrix, again.                                                           */

/* now its time to turn the pivot elements to 1 and null the elements above it */
while (i >= 1) /* cycle terminates once i reaches the top line */
{
   /* find the pivot element */
   for (l = i; l >= 1; l--) /* l runs the lines (from the position set by i) until it reaches the top line */
   {
```

```
      for (c = 1; c <= num_nodes; c++) /* c runs through the columns */
         if (matrix[l][c] != 0)   /* when it finds the pivot (the first non-zero element of that line) */
            break;                /* it stops the cycle.                                               */

         if (matrix[l][c] != 0)   /* this is needed to fully exit the cycles. */
            break;
      }

      if (matrix[l][c] != 1) /* if the pivot is not 1 ... */
      {
         temp = matrix[l][c];   /* ... temp takes its value... */
         for (count = 1; count <= num_nodes+1; count++)   /* ... this is will run through the columns. */
            if (count == c)              /* when it reaches the column of the pivot ... */
               matrix[l][count] = 1.0;   /* ...it explicitely turns it 1.          */
            else                                          /* all the other elements of that line    */
               matrix[l][count] = (1/temp) * matrix[l][count]; /* will be tranformed by this instruction */
      }
      /* the next cycle will null the elements above the pivot */
      for (p = l - 1; p >= 1; p--) /* this will run through the lines from above the pivot to the first */
      {
         temp = matrix[p][c]; /* temp will take the value of the element above the pivot */
         for (count = 1; count <= num_nodes+1; count++) /* this will run through the columns */
            if (count == c)             /* when it reaches the columns of the pivot ... */
               matrix[p][count] = 0.0;  /* ... it is explicitely turned to zero. */
            else                                          /* all the other elements will be tranformed */
               matrix[p][count] -= (temp * matrix[l][count]); /* by this instruction.                    */
      }
      i--; /* line index is decreased */
   }
   /* presentation of the final results wich are*/
   /* the elements of the column num_nodes+1 */
   printf ("\n\nValores das tensoes:\n\n");
   for (i = 1; i <= num_nodes; i++)    /* i runs through the lines... */
      printf ("\t\tU%d: %3.2f V\n", i, fabs(matrix[i][num_nodes+1]));  /* ...and each values is presented. */

   _getch();
} /* end of program */

void gotoeol(FILE *in_file) /* this function is used to ignore the comment lines in the netlist file */
{
   char test_ch; /* this variable will store each character read from the line */
   do
      test_ch = fgetc(in_file);          /* this cycle runs through the line ... */
   while((test_ch != 10) || (test_ch == EOF)); /* until it reaches the character with ASCII code 10 (wich is */
}                                              /* line feed character), meaning that it reached the end of   */
                                               /* the line or the end of file (character code -1, or EOF).   */
                                               /* this will make the next fscanf function start reading the  */
                                               /* elements of the next line.                                 */
```

Análise Crítica

A elaboração do protótipo foi até agora a parte mais difícil/trabalhosa do projecto, visto termo-nos deparado com algumas dificuldades, nomeadamente passar o sistema da netlist para uma matriz e depois resolve-la para a forma de escada e escada reduzida (método Gauss-Jordan). Devido a problemas com os arredondamentos, dado que o computador tem um limite no que toca às casa decimais com as quais vai trabalhar, em alguns cálculos, o resultado obtido não era exactamente o que seria suposto. Então tivemos de explicitamente atribuir os valores correctos aos elementos em questão, para posteriormente aqueles elementos não causarem problemas.

Outra dificuldade apareceu quando chegámos à parte do método de Gauss-Jordan, pois no algoritmo utilizamos alguns 'saltos' e como isso não é permitido no nosso código, tivemos de efectuar algumas modificações. Posto isto, o código do programa em C não está neste momento 100% fiel ao algoritmo.

Para passar para escada, pusemos o programa a correr a matriz de forma a encontrar na primeira linha o elemento não nulo mais à esquerda, fixando-o como o elemento pivot. Encontrado o elemento pivot, anulamos todos os elementos da coluna abaixo dele. Passamos então para a linha abaixo e procuramos o elemento não nulo mais à esquerda, fixando este como novo elemento pivot e anulamos todos os elementos abaixo deste. Repetimos este processo até não existirem mais elementos pivot, estando neste momento a matriz em escada.

Tendo a matriz em escada tivemos que a passar a escada reduzida. Para tal pegamos no último elemento da matriz e fixamo-lo como pivot e de seguida anulamos todos os elementos da coluna que estão acima dele. Depois passamos para a linha acima e procuramos o elemento não nulo mais à esquerda, passando este a ser o novo pivot. Repetimos o processo até chegarmos ao ponto de não haverem mais elementos pivot. Encontrado o último pivot temos que o passar a 1, obtendo assim, uma matriz em escada reduzida.

No nosso protótipo não consideramos nó1=0 porque pensámos que tal não seria possível de acontecer, mas verificamos na apresentação aos docentes que afinal o nó1 poderia assumir o valor de zero, ou seja, que a fonte pode ser ligada de forma inversa. Visto isto vamos proceder a esta

pequena alteração no nosso programa e fazer com que ele resolva este tipo de casos (caso de o nó1 ser igual a zero), visto que já resolve os casos restantes.

Conclusão

Com a ajuda dos métodos aprendidos nas várias unidades curriculares envolvidas no desenvolvimento deste projecto, conseguimos obter um protótipo intermédio já capaz de obter as tensões nos nós a partir da descrição de um circuito eléctrico.

O nosso protótipo já realiza as tarefas todas pedidas no enunciado do projecto logo está muito próximo da versão final. Apenas não o é devido ao ajuste que temos de fazer em relação ao nó1 de UR e R poder assumir o valor de zero, problema esse que irá ser prontamente resolvido.

De referir que durante a elaboração do código em linguagem C alguns passos do algoritmo foram modificados visto o grupo ter encontrado maneiras mais fáceis e rápidas de resolver alguns problemas no calculo matricial.

Também concluímos que este processo de conversão para linguagem C foi um dos processos mais complicados de todo o projecto devido ao facto de haver alguns pormenores que na construção do algoritmo não previámos que fossem aparecer, mas que com empenho conseguimos resolver e passar os obstáculos que foram aparecendo ao longo da construção do mesmo.