A simple SPICE backend

Saragadam R V Vishwanath
EE10B035
Department of Electrical Engineering
IITMadras

September 25, 2011

Abstract

SPICE is a circuit simulator program to solve easy as well as complicated linear and non linear circuits. A SPICE program also does a frequency sweep, a transient analysis etc. The present document will present a simple version of the SPICE program, called myspice which can solve linear circuits with bias points and can do a frequency sweep. The document will also deal with the code written in c and the method of using the program.

Introduction

The SPICE program was developed as a tool for complex circuit analysis. The program is a very poweful tool for studying how ciruits work for different conditions. The program presented here is a very simple version of the original SPICE program. It can solve circuits with linear components and DC sources or AC sources with a single frequency and no offset. The code has been written in C and was an opportunity to learn the powerful tool in C, the pointers. The program requires a net list , which is a simple text file containing the information about all the elements that are present in the ciruit to be solved. The subsequent sections will deal with the semantics of the code and the way of using the program.

Using The program

The program is to be run from the terminal from the same folder where the netlist is located. The usage is very simple. There are there modes of operation

- Circuit is a resistive network with any number of sources. In such a case, the following command is to be used

  ```
  $ ./myspice netlist.dat
  ```

- Circuit has any linear component and the whole circuit works on single frequency. Then the command is as follows

  ```
  $ ./myspice netlist.dat frequency
  ```

- A frequency sweep is required for the circuit. Then,

  ```
  $ ./myspice netlist.dat start_frequency end_frequency sampling_rate
  ```

  This case generates another file, plot_data.dat, which has the outputs along with the corresponding frequencies. This file can be used for ploting the graph of output vs frequency

Netlist

The netlist has the information about the circuit. Each row of the netlist is an element with data about the nodes between which it is connected, the dependencies if any and the elements value. The following gives the details about the format of each element

Element name	definition in the netlist
Voltage source	V_name node1 node2 value
Current source	I_name node1 node2 value
Resistance	R_name node1 node2 value
Inductor	L_name node1 node2 value
Capacitor	C_name node1 node2 value
Voltage dependent voltage source	E_name node1 node2 depnode1 depnode2 proportionality_constant
Voltage dependent current source	G_name node1 node2 depnode1 depnode2 proportionality_constant
Current controlled voltage source	H_name node1 node2 dep_voltage_source proportionality constant
Current controlled current source	F_name node1 node2 dep_voltage_source proportionality constant

There is no restriction on the node name provided it has only alpha numeric values along with underscore.One of the nodes has to be 0 for the program to solve the circuit. The value field will have constants with or without powers of 10. The valid multipliers are k, meg, m, n, p. To get voltage between two nodes node1 and node2, the following line must be included:

```
.clamp node1 node2
```

Comments can also included by starting the line with # or *.

The followin netlist is an example

```
# A simple low pass filter with capacitors and inductors
V n4 0 1
R n1 n4 4k
L n2 n1 80.96u
L n3 n2 80.96u
C 0 n2 2.485p
R 0 n3 4k
.clamp n3 0
```

The Code

The code has been completely written in c and is compiled using *gcc*. The code parses the netlist, extracts the elements and stores them in a linked list. Then the equations are solved using gauss jordhan elimination method. If it is a frequency sweep, it does the same process for the given sample rate.

Pseudocode

The pseudo code for the main function is as follows:

```
open netlist.dat
if number_of_arguements is 2:
    call solver
    print voltage difference
if number_of_arguements is 3:
    frequency = argv[2]
    call solver
    print voltage difference
if number_of_arguements is 5:
    init_frequency = argv[2]
    final_frequency = argv[3]
    num_steps = argv[4]
    diff = (final_frequency-init_frequency)/num_steps
    create file plot_data.dat
    while num_steps:
        call solver
        write frequency, voltage difference
        num_steps -= 1
        init_frequency = init_frequency + diff
```

```
        close plot_data.dat
    close netlist.dat
```

The above code for main function calls the solver function, which does the actual work. The solver function parses the netlist.dat file, extracts the element information, creates the conductance matrix , solves it and returns the voltage difference across the required nodes. The way solver works is explained in the subsequent code.

The C code

The main code consists of multiple functions for error checking, extracting values, and solving the circuit. The code initially has the all the functions and their definitions and then the final main function.

Includes

The includes has all the header files to be included for the file. *math.h* is used for calculating the magnitude of complex numbers. *complex.h* defines all the macros and the functions for handling complex numbers. This is not relevant for a resistive network , but if the circuit contains capacitors and inductors, then the complex impedences require the header file for operations.

3a ⟨* 3a⟩≡ 3b▷

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<complex.h>
```

Constants

The constants define the maximums. $MAXBUF$ defines the maximum characters in a line of the netlist. $CHARMAX$ defines the maximum number of characters in the node name. $NODEMAX$ defines the maximum number of distinct nodes

3b ⟨* 3a⟩+≡ ◁3a 4▷

```
#define MAXBUF 256
#define CHARMAX 32
#define NODEMAX 40
```

err_val() function

The err_val function takes as arguements the string representing the value, i.e, the last field in the netlist and checks for possible type errors. Typical errors include multiple dots or no number at all or a wrong character. Returns 1 if everything is correct.

4 ⟨* 3a⟩+≡ ◁3b 5▷

```
int err_val(char *str_val)
{
        int num_flag = 0,char_flag = 0,dot_flag = 0;
        int i,len;

        char letter;
        len = strlen(str_val);

        for(i = 0;i<len;i++)
        {
                letter = *(str_val + i);
                if(letter == '.')
                {       char next_letter = *(str_val + i +1);

                        if ((char_flag != 0) || ((next_letter<'0') || (next_letter>'9')) || dot_flag!=0 )
                        {
                                printf("Invalid token \'value\'.Please correct the netlist\n ");
                                exit(3);
                        }
                        else
                                dot_flag++;
                }
                else if( (letter == '_') )
                {
                        printf("Invalid token \'value\'.Please correct the netlist\n ");
                        exit(3);
                }
                else if((letter >=48)&&(letter<=57))
                {
                        if (char_flag == 0)
                                num_flag++;
                        else
                        {
                                printf("Invalid token \'value\'.Please correct the netlist\n ");
                                exit(3);
                        }
                }

                else if ((letter>=97)&&(letter<=122))
                {
                        if (num_flag != 0)
                                char_flag++;
                        else
                        {
                                printf("Invalid token \'value\'.Please correct the netlist\n ");
                                exit(3);
                        }
                }
        }

        return 1;

}
```

err_check() function

This function checks for possible errors in one row of the netlist. Possible errors are presence of special characters. Returns the lenght of the string input if everything is correct.

5 ⟨* 3a⟩+≡ ◁4 6▷

```
int err_check(char *string)
{
        int len = strlen(string);
        int i = 0;
        char letter;
        for(i=0;i<len;i++)
        {
                letter = string[i];
                if( ((letter == 9) || (letter == 10) || (letter == 32) || (letter == 95) || (letter == 46) |
                        && (letter <=57) ) || ( (letter>=65) && (letter<=90) ) || ( (letter>=97) && (letter<=
                else
                        return i;
        }
        return len;
}
```

analyse() function

The analyse function takes in the value field and returns the float version of the value. It also handles multipliers if specified.

6 ⟨* 3a⟩+≡ ◁5 7a▷

```
float analyse(char *string)
{
        float val;
        float mult = 1;
        int len = 0;

        int val_err;
        val_err = err_val(string);
        if (val_err == 1)
        {
                val = atof(string);
                len = strlen(string);
                switch(string[len-1])
                {
                        case 'k':       mult = 1000;
                                break;
                        case 'n':       mult = 1.0/10000000000.0;
                                break;
                        case 'u':       mult = 1.0/1000000.0;
                                break;
                        case 'm':       mult = 1.0/1000.0;
                                break;
                        case 'g':       if(string[len-2] == 'e' && string[len-3] == 'm')
                                        mult = 1000000;
                                        else
                                        {
                                                printf("invalid multiplier found. Please correct the netlist\
                                                exit(2);
                                        }
                                break;
                        default:        mult = 1;
                                break;
                }
        }
        return(val*mult);

}
```

elem_analyse() function

This function takes as arguments, the first letter of the name of the element. Returns 1 if it is an independent element, 2 if it is a voltage controlled source and 3 if it is a current controlled source. Returns 0 if it is not a valid element.

7a ⟨* 3a⟩+≡ ◁6 7b▷

```
int elem_analyse(char elem,char *indep_elem,char *v_dep_elem,char *i_dep_elem)
{
        int num_elem;
        int flag = 0;

        for(num_elem = 0;num_elem<strlen(indep_elem);num_elem++)
        {
                if (elem == indep_elem[num_elem])
                        flag = 1;
        }

        for(num_elem = 0;num_elem<strlen(v_dep_elem);num_elem++)
        {
                if (elem == v_dep_elem[num_elem])
                        flag = 2;
        }

        for(num_elem = 0;num_elem<strlen(i_dep_elem);num_elem++)
        {
                if (elem == i_dep_elem[num_elem])
                        flag = 3;
        }

        return flag;
}
```

get_index() function

This function takes the node as the arguments and returns the index of the node in the main node list.

7b ⟨* 3a⟩+≡ ◁7a 8a▷

```
int get_index(char node[32],char node_list[100][32],int num_nodes)
{
        int i;
        for (i = 0;i<num_nodes;i++)
        {
                if (strcmp(node,node_list[i]) == 0)
                        return i;
        }
}
```

c_abs function

This function takes a complex number as an arguement and returns the modulus of it.

8a ⟨* 3a⟩+≡ ◁7b 8b▷

```
float c_abs(complex number)
{
        float real,imag;
        float abs_val;
        real = creal(number);
        imag = cimag(number);

        abs_val = sqrt(real*real+imag*imag);
        return abs_val;
}
```

Node structure definition

The structure node is the main unit which holds the information about the element parsed. The definition has the variables, a vlt

_src_num variable to hold the index of the voltage source and the pointers next and prev. These pointers help to make a doubly linked link list, which can be traversed in both direction.

8b ⟨* 3a⟩+≡ ◁8a 8c▷

```
struct node
{
        char n1[CHARMAX] ;
        char n2[CHARMAX] ;
        char n3[CHARMAX] ;
        char n4[CHARMAX] ;
        char depname[CHARMAX];
        char name[CHARMAX] ;
        complex value;

        int vlt_src_num;

        struct node* next;
        struct node* prev;
};


typedef struct node node;
```

solver() function

This is the most important function which calls other function except main function. The function takes the console inputs and the frequency as the input. Returns the voltage difference between the required nodes as a float. The function is discussed in great detail.

8c ⟨* 3a⟩+≡ ◁8b 9a▷

```
complex solver(char **argv,float frequency)
{
```

Initiation of variables

A file pointer fp is created to hold the netlist file, which is opened in read mode. The variable line will hold the parsed line. word holds the token extracted using *strtok*() and tokbuf holds the pointer for extracting the next token. The indep_elem, v_dep_elem and i_dep_elem hold the first letter of the correspondind element types.

9a ⟨* 3a⟩+≡ ◁8c 9b▷

```
FILE *fp;

fp = fopen(argv[1],"r");

char line[MAXBUF];
char *word;
char *tokbuf;
char delim[] = " \t\n";
int check_var;

char indep_elem[] = {'R','L','C','V','I','r','l','c','v','i','\0'};
char v_dep_elem[] = {'E','G','e','g','\0'};
char i_dep_elem[] = {'F','H','f','h','\0'};

int elem_type,num_nodes = 0;
int vlt_src = 0;
int cur_dep_vlt = 0;
```

Creating the start node

A start node is created to keep track of all the nodes. The old node is a temporary variable to traverse through the link list. The *clamp* array holds the two nodes between which the voltage is to be found.

9b ⟨* 3a⟩+≡ ◁9a 10▷

```
node *start;
start = (node*)malloc(sizeof(node));

start->next = NULL;
start->prev = NULL;

node *old ;
old = (node *)malloc(sizeof(node));

char clamp_nodes[2][32];
int clamp_cnt = 0;
```

Parsing loop

This loop continuously parsed the netlist file till there is a EOF. It skips parsing the line if it starts with # or $*$. If it starts with '.', it verifies if the line is *.clamp* and then parses it and stores the clamped nodes(The nodes across which voltage difference is to be found out.)

10 ⟨* 3a⟩+≡ ◁9b 11▷

```
while(fgets(line,MAXBUF-1,fp) != NULL)
{

        if (line[0] == '#'|| line[0] == '*') ;

        else if(line[0] == '.')
        {
                tokbuf = line;
                word = strtok(tokbuf,delim);
                if(strcmp(word,".clamp") == 0)
                {
                        while(word!=NULL)
                        {
                                if(clamp_cnt>1)
                                        break;
                                tokbuf = NULL;
                                word = strtok(tokbuf,delim);
                                strcpy(clamp_nodes[clamp_cnt],word);
                                clamp_cnt++;
                        }
                }
        }
```

Storing the element's information

When the parsed line contains a valid element, it start extracting tokens till there is an EOL. It checks for all possible errors. Once no errors are found, it proceeds by creating a temporaru node variable to hold the data. First, the name of the element is copied. Depending on the name, corresponding values are assigned in the node.

11 ⟨* 3a⟩+≡ ◁10 13▷

```
else
{
        int err_code;
        err_code = err_check(line);

        if (err_code != strlen(line))
        {
                printf("Invalid character %c found.Now aborting\n",line[err_code]);
                exit(2);
        }

        node *temp ;
        temp = (node*)malloc(sizeof(node));

        if(num_nodes == 0)
                temp = start;

        tokbuf = line;
        word = strtok(tokbuf, delim);

        strcpy(temp->name,word);

        elem_type = elem_analyse(temp->name[0],indep_elem,v_dep_elem,i_dep_elem);

        if(elem_type == 0)
        {
                printf("Element %s not defined\n",word);
                exit(6);
        }

        if(elem_type == 1)
        {
                int cnt = 0;
                if (temp->name[0] =='V' || temp->name[0]  == 'v' )
                {
                        temp->vlt_src_num = vlt_src;
                        vlt_src++;
                }

                while(word != NULL)
                {
                        tokbuf = NULL;
                        word = strtok(tokbuf,delim);

                        if( ( word==NULL && cnt<2 ) || ( word!=NULL && cnt>2 ) )
                        {
                                printf("Invalid number of tokens.\n");
                                exit(4);
                        }

                        switch(cnt)
                        {
                                case 0: strcpy(temp->n1,word);
                                        break;
```

```
                        case 1: strcpy(temp->n2,word);
                                break;
                        case 2:         if(frequency != 0)
                                        {
                                                if(temp->name[0] == 'R'||temp->name[0
                                                        temp->value = analyse(word)
                                                else if(temp->name[0] == 'L'|| temp->
                                                        temp->value = I*frequency*ana
                                                else if (temp->name[0] == 'C' || temp
                                                        temp->value = 1/(I*frequency*
                                                else
                                                        temp->value =analyse(word);
                                        }

                                        else
                                                temp->value = analyse(word);
                                break;
                        default:
                                break;

                }
                cnt++;
        }
}

if(elem_type == 2)
{
        int cnt = 0;

        if (temp->name[0] == 'E' || temp->name[0] == 'e')
        {
                temp->vlt_src_num = vlt_src;
                vlt_src++;
        }


        while(word != NULL)
        {
                tokbuf = NULL;
                word = strtok(tokbuf,delim);

                if( ( word==NULL && cnt<4 ) || ( word!=NULL && cnt>4 ) )
                {
                        printf("Invalid number of tokens.\n");
                        exit(4);
                }

                switch(cnt)
                {
                        case 0:         strcpy(temp->n1,word);

                                        break;
                        case 1:         strcpy(temp->n2,word);
                                        break;
                        case 2:         strcpy(temp->n3,word);
                                        break;
                        case 3:         strcpy(temp->n4,word);
                                        break;
                        case 4:         temp->value = analyse(word) ;
                                        break;
```

```
                                default:
                                        break;
                        }
                        cnt++;
                }

        }

        if(elem_type == 3)
        {
                int cnt = 0;

                if (temp->name[0] == 'H' || temp->name[0] == 'h')
                {
                        vlt_src++;
                }

                while(word != NULL)
                {
                        tokbuf = NULL;
                        word = strtok(tokbuf,delim);

                        if( ( word==NULL && cnt<3 ) || ( word!=NULL && cnt>3 ) )
                        {
                                printf("Invalid number of tokens.\n");
                                exit(4);
                        }

                        switch(cnt)
                        {
                                case 0: strcpy(temp->n1,word);
                                        break;
                                case 1:         strcpy(temp->n2,word);
                                        break;
                                case 2: strcpy(temp->depname,word);
                                        break;
                                case 3: temp->value = analyse(word) ;
                                        break;
                                default:
                                        break;
                        }
                        cnt++;
                }
        }
```

Going to the next node

Once the line is parsed, the present node is stored in old node, the old node's next is pointed to temp and the prev is pointed to the old node.

13 ⟨* 3a⟩+≡ ◁11 14▷

```
                temp->prev = old;
                temp->next = NULL;
                old->next = temp;
                old = temp;

                num_nodes++;

        }
}
```

Extracting unique nodes

Once the whole file is parsed, the unique nodes are extracted. This is done by traversing the whole linked list and the extracted nodes are compared in the previously extracted node. It is added if it is not previously extracted

14 ⟨* 3a⟩+≡ ◁13 15a▷

```
char nodes[NODEMAX][32];

num_nodes = 0;
strcpy(nodes[0],start->n1);
strcpy(nodes[1],start->n2);
num_nodes += 2;

int i = 0,flag_n1 = 1,flag_n2 = 1;
old = start->next;

do
{
        flag_n1 = 1;
        flag_n2 = 1;
        for (i = 0 ; i<num_nodes;i++)
        {
                if ( strcmp(nodes[i],old->n1) == 0)
                        flag_n1 = 0;
        }

        if(flag_n1)
        {
                strcpy(nodes[num_nodes],old->n1);
                num_nodes++;
        }

        for (i = 0 ; i<num_nodes;i++)
        {
                if ( strcmp(nodes[i],old->n2) == 0)
                        flag_n2 = 0;
        }

        if(flag_n2)
        {
                strcpy(nodes[num_nodes],old->n2);
                num_nodes++;
        }

        old = old->next;
}while(old != NULL);
```

Declaring the conductance matrix

The dimension of the conduction matrix is (num_nodes +vlt_src)x(num_nodes + vlt_src +1). The matrix is initialised to all zeros to avoid any garbage values.

15a ⟨* 3a⟩+≡ ◁14 15b▷

```
int dimension = num_nodes+vlt_src +1;
complex conduct_mat[dimension-1][dimension];

int j;

for(i = 0;i<dimension-1;i++)
        for(j=0;j<dimension;j++)
                conduct_mat[i][j] = 0+0*I;


old = start;

int eqn_cnt = 0;
```

Creating the matrix

The creation of matrix is as follows. A node is taken, and the linked list is completely traversed. If the first node or the second node of the element matches with the present node, then the matrix is updated by adding values to the corresponding columns of the present row. If the present node is 0, an equation of the form $v_k = 0$ is created.

15b ⟨* 3a⟩+≡ ◁15a 16▷

```
for(i = 0;i<num_nodes;i++)
{
        old = start;

        int node1,node2;
        char present_node[32] ;
        strcpy(present_node,nodes[i]);

        if(strcmp(present_node,"0") == 0)
        {
                node1= get_index("0",nodes,num_nodes);
                conduct_mat[eqn_cnt][node1] += 1;
                eqn_cnt++;
                continue;
        }
```

Creating equations for elements

Once the zero node equation is formed, the other element's equations are also formed. The values are added to the corresponding element of the current row. If it is a voltage source, the unknown current is added. The auxilary equations are dealt seperately.

16 ⟨* 3a⟩+≡ ◁15b 19▷

```
do
{

        if(strcmp(old->n1,present_node) == 0)
        {
                node1 = get_index(old->n1,nodes,num_nodes);
                node2 = get_index(old->n2,nodes,num_nodes);

                if(old->name[0] == 'i' || old->name[0] == 'I')
                        conduct_mat[eqn_cnt][dimension-1] += old->value;

                if(old->name[0] == 'r' || old->name[0] == 'R')
                {

                        conduct_mat[eqn_cnt][node1] += 1.0/old->value;
                        conduct_mat[eqn_cnt][node2] -= 1.0/old->value;
                }

                if((old->name[0] == 'l' || old->name[0] == 'L') &&(frequency != 0))
                {

                        conduct_mat[eqn_cnt][node1] += 1.0/old->value;
                        conduct_mat[eqn_cnt][node2] -= 1.0/old->value;
                }

                if((old->name[0] == 'c' || old->name[0] == 'C') && (frequency != 0))
                {

                        conduct_mat[eqn_cnt][node1] += 1.0/old->value;
                        conduct_mat[eqn_cnt][node2] -= 1.0/old->value;
                }

                if(old->name[0] == 'V' || old->name[0] == 'v')
                        conduct_mat[eqn_cnt][num_nodes + old->vlt_src_num] += 1;


                if(old->name[0] == 'G' || old->name[0] == 'g')
                {
                        int node3,node4;

                        node3 = get_index(old->n3,nodes,num_nodes);
                        node4 = get_index(old->n4,nodes,num_nodes);

                        conduct_mat[eqn_cnt][node3] += old->value;
                        conduct_mat[eqn_cnt][node4] -= old->value;
                }

                if(old->name[0] == 'E' || old->name[0] == 'e')
                        conduct_mat[eqn_cnt][num_nodes + old->vlt_src_num] += 1;

                if(old->name[0] == 'F' || old->name[0] == 'f')
                {
                        node *temp;
                        temp = start;
```

```
                do
                {
                        int cmp_chk = strcmp(old->depname,temp->name);
                        if(cmp_chk == 0)
                                conduct_mat[eqn_cnt][num_nodes + temp->vlt_src_num] -

                        temp = temp->next;
                }while(temp!= NULL);
        }


}

if(strcmp(old->n2,present_node) == 0)
{
        node1 = get_index(old->n1,nodes,num_nodes);
        node2 = get_index(old->n2,nodes,num_nodes);

        if(old->name[0] == 'i' || old->name[0] == 'I')
                conduct_mat[eqn_cnt][dimension-1] -= old->value;

        if(old->name[0] == 'r' || old->name[0] == 'R')
        {

                conduct_mat[eqn_cnt][node1] -= 1.0/old->value;
                conduct_mat[eqn_cnt][node2] += 1.0/old->value;
        }

        if((old->name[0] == 'l' || old->name[0] == 'L') && (frequency != 0))
        {

                conduct_mat[eqn_cnt][node1] -= 1.0/old->value;
                conduct_mat[eqn_cnt][node2] += 1.0/old->value;
        }

        if(old->name[0] == 'c' || old->name[0] == 'C' && (frequency != 0))
        {

                conduct_mat[eqn_cnt][node1] -= 1.0/old->value;
                conduct_mat[eqn_cnt][node2] += 1.0/old->value;
        }

        if(old->name[0] == 'V' || old->name[0] == 'v')
                conduct_mat[eqn_cnt][num_nodes + old->vlt_src_num] -= 1;


        if(old->name[0] == 'G' || old->name[0] == 'g')
        {
                int node3,node4;

                node3 = get_index(old->n3,nodes,num_nodes);
                node4 = get_index(old->n4,nodes,num_nodes);

                conduct_mat[eqn_cnt][node3] -= old->value;
                conduct_mat[eqn_cnt][node4] += old->value;
        }

        if(old->name[0] == 'E' || old->name[0] == 'e')
                conduct_mat[eqn_cnt][num_nodes + old->vlt_src_num] -= 1;
```

```
                        if(old->name[0] == 'F' || old->name[0] == 'f')
                        {
                                node *temp;
                                temp = start;

                                do
                                {
                                        int cmp_chk = strcmp(old->depname,temp->name);
                                        if(cmp_chk == 0)
                                                conduct_mat[eqn_cnt][num_nodes + temp->vlt_src_num] -

                                        temp = temp->next;
                                }while(temp!= NULL);
                        }

                }

                old = old->next;

        }while(old != NULL);

        eqn_cnt++;
}
```

The auxilary equations

Once all the nodes are traversed, the auxilary equations are formed. If it is an independent voltage source, 3 columns are updated, corresponding to the nodes and the constants row. If it is a voltage dependent source, 4 columns are updated, corresponding to the nodes. If it is a current controlled source, then the columns corresponding to nodes and the column corresponding to the dependent voltage source is updated.

19 ⟨* 3a⟩+≡ ◁16 20▷

```
int vlt_src_cnt = 0;
old = start;

do
{

        if(old->name[0] == 'V' || old->name[0] == 'v')
        {
                int node1 = get_index(old->n1,nodes,num_nodes);
                int node2 = get_index(old->n2,nodes,num_nodes);

                conduct_mat[num_nodes + vlt_src_cnt][node1] += 1;
                conduct_mat[num_nodes + vlt_src_cnt][node2] -= 1;

                conduct_mat[num_nodes + vlt_src_cnt][dimension-1] += old->value;
                vlt_src_cnt++;
        }

        if(old->name[0] == 'E' || old->name[0] == 'e')
        {
                int node1 = get_index(old->n1,nodes,num_nodes);
                int node2 = get_index(old->n2,nodes,num_nodes);
                int node3 = get_index(old->n3,nodes,num_nodes);
                int node4 = get_index(old->n4,nodes,num_nodes);

                conduct_mat[num_nodes + vlt_src_cnt][node1] += 1;
                conduct_mat[num_nodes + vlt_src_cnt][node2] -= 1;
                conduct_mat[num_nodes + vlt_src_cnt][node3] -= old->value;
                conduct_mat[num_nodes + vlt_src_cnt][node4] += old->value;

                vlt_src_cnt++;

        }

        if(old->name[0] == 'H' || old->name[0] == 'h')
        {
                int node1 = get_index(old->n1,nodes,num_nodes);
                int node2 = get_index(old->n2,nodes,num_nodes);

                node *temp;
                temp = start;

                do
                {

                        int cmp_chk = strcmp(temp->name,old->depname);
                        if (cmp_chk == 0)
                        {
                                conduct_mat[num_nodes + vlt_src_cnt][node1] += 1+0*I;
                                conduct_mat[num_nodes + vlt_src_cnt][node2] -= 1+0*I;

                                conduct_mat[num_nodes + vlt_src_cnt][num_nodes + temp->vlt_src_num] -
                        }
```

```
                        temp = temp->next;
                }while(temp != NULL);
        }
        old = old->next;
}while(old != NULL);


int k = 0;
```

Checking for empty I_{ii}

There will be cases when the element I_{ii} is zero. One example being voltage sources in series. In such a case, it searches for a simalar row in which the I_{ji} element is non zero and adds that row to the i^{th} row. This ensures that the matrix is solvable.

20 ⟨* 3a⟩+≡ ◁19 21a▷

```
for(i=0;i<dimension-1;i++)
{
        if(conduct_mat[i][i] == (0+0*I))
        {
                for(j = 0;j<dimension-1;j++)
                {
                        if(conduct_mat[j][i] !=0+0*I)
                                break;
                }


                for(k = 0;k<dimension;k++)
                {
                        conduct_mat[i][k] +=conduct_mat[j][k];
                }
        }
}
```

Forming upper triangle using Gaussian elimination

Once the matrix is ready, the matrix is converted into an upper triangle matrix. This is done using the Gaussian method. For a given row j , the elements upto $j-1$ column is made zero. This is repeated for all the rows except the first. The first is left intact. After the matrix is an upper triangle matrix, it checks if any element I_{ii} is zero. If so, it is an indeterminate system and it quits.

21a ⟨* 3a⟩+≡ ◁20 21b▷

```
for(i=1;i<dimension-1;i++)
{
        for(j=0;j<i;j++)
        {
                if(conduct_mat[i][j] == 0)
                        continue;
                complex mult_factor = conduct_mat[i][j]/conduct_mat[j][j];

                for(k=j;k<dimension;k++)
                {
                        conduct_mat[i][k]  -= mult_factor*conduct_mat[j][k] ;
                }

        }
}

for(i = 0;i<dimension-1;i++)
{
        if(conduct_mat[i][i] == 0)
        {
                printf("Indeterminate system.Analysis will stop.\n");
                exit(8);
        }
}
```

Generating solution

Now, the matrix is solved from the last row. The solution obtained upto $(i-1)^{th}$ row is used for solving i^{th} row. The generated solutions are stored in a complex type matrix.

21b ⟨* 3a⟩+≡ ◁21a 22▷

```
complex solution[dimension-1];
for(i = 0;i<dimension-1;i++)
        solution[i] = 0+0*I;

solution[dimension-2] = (conduct_mat[dimension-2][dimension-1]/conduct_mat[dimension-2][dimension-2])

for(i=dimension-3;i>=0;i-)
{
        complex subtractor = 0+0*I;
        for (j=dimension-2;j>i;j-)
        {
                subtractor += conduct_mat[i][j]*solution[j];
        }
        solution[i] = ((conduct_mat[i][dimension-1]) - subtractor)/conduct_mat[i][i];
}

complex vlt1,vlt2,vlt_diff;
```

Clamp voltage

Once the solutions are obtained, the voltages corresponding to the clamp nodes is retrieved and the difference is returned. This ends the work for the solver loop.

22 ⟨* 3a⟩+≡ ◁21b 23▷

```
        vlt1 = solution[get_index(clamp_nodes[0],nodes,num_nodes)];
        vlt2 = solution[get_index(clamp_nodes[1],nodes,num_nodes)];
        vlt_diff = vlt2-vlt1;

        return vlt_diff;

        fclose(fp);

}
```

main() function.

The main function is very small since, it only calls the solver function. If there are only two arguements, it prints the voltage difference, at $0Hz$. If there are 3 arguements, it prints the voltage difference at the given frequency. If there are 5 arguements, it creates a *.dat* file and saves the voltage difference and the corresponding frequencies.

23 ⟨* 3a⟩+≡ ◁22

```
int main(int argc,char **argv)
{
        float frequency;
        float strt_freq,end_freq,step_size;
        int steps;
        complex voltage = 0+0*I;

        FILE *data;

        printf("Program: Myspice\n");

        data = fopen("plot_data.dat","w");
        printf("opened netlist: %s\n",argv[1]);

        if(argc<=1)
        {
                printf("Control file missing.\n");
                printf("Usage: ./executable controlfile\n");
                printf("Now aborting\n");
                exit(1);
        }

        else if(argc == 2)
        {
                printf("DC analysis chosen\n");
                voltage = solver(argv,0);
                printf("voltage at requested node is %f\n",c_abs(voltage));
        }
        else if(argc == 3)
        {
                printf("Single frequency analysis at %s\n",argv[2]);
                frequency = atof(argv[2]);
                voltage = solver(argv,frequency);
                printf("Voltage at requested node at %f frequency is %f",frequency,c_abs(voltage));
        }

        else if(argc == 5)
        {
                printf("Frequency sweep.\n");

                strt_freq = atof(argv[2]);
                end_freq = atof(argv[3]);
                steps = atof(argv[4]);
                step_size = (end_freq-strt_freq)/steps;

                int i =0;
                for (i =0;i<steps;i++)
                {
                        frequency = strt_freq+step_size*i;
                        voltage = solver(argv,frequency);
                        fprintf(data,"%f %f\n",frequency,c_abs(voltage));
                }

                fclose(data);
```

```
                printf("data file created as plot_data.dat\n");
                printf("thank you for using myspice.use 'python plot.py plot_data.dat'for getting a plot of t

        }

}
```

Results

The code was checked for multiple net lists and the following results were obtained

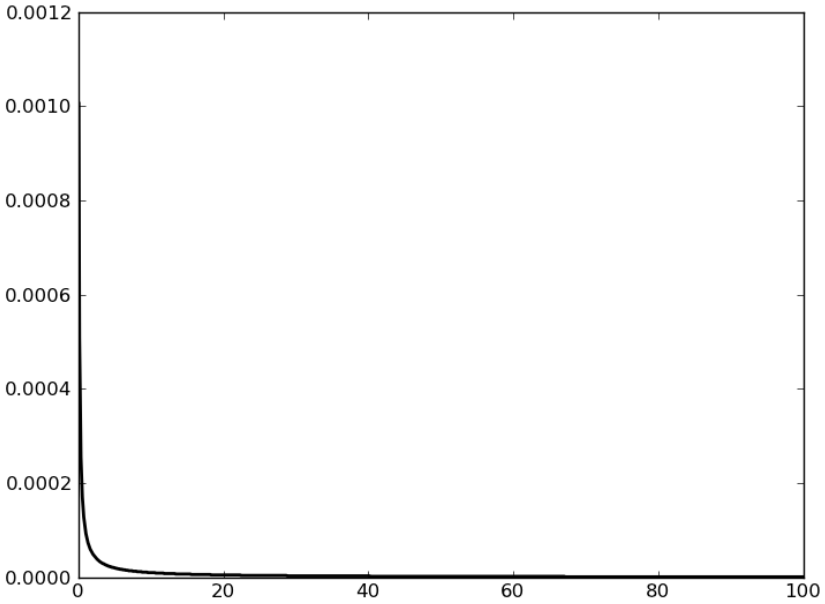
- A very simple netlist having only resistive elements was checked.
  - The netlist is as follows:

```
# Simple resistive network
V n1 0 5
R n2 n1 1
R n2 0 1
R n2 0 1
.clamp n2 0
```

- A very steep low pass filter.
  - The netlist is :

```
# A simple low pass filter with capacitors and inductors
V n4 0 1
R n1 n4 4k
L n2 n1 80.96u
L n3 n2 80.96u
C 0 n2 2.485p
R 0 n3 4k
.clamp n3 0
```

  - The plot obtained is:

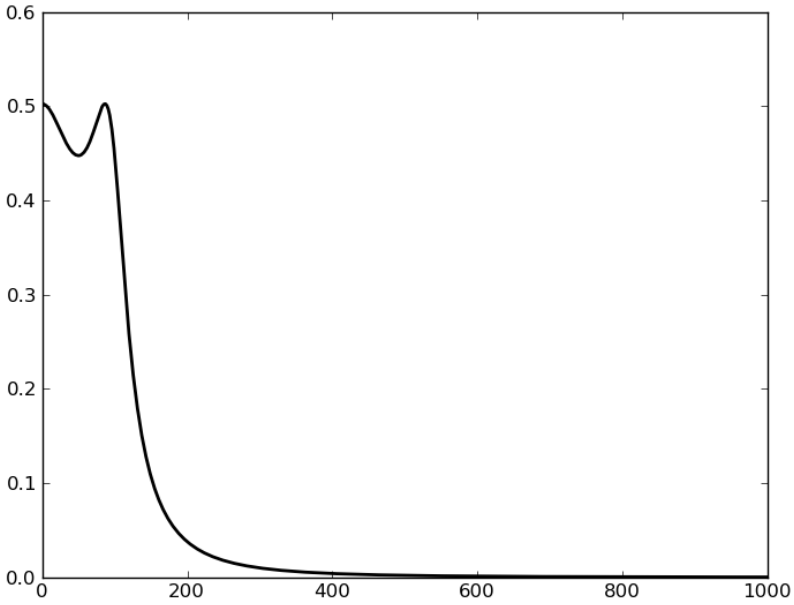


- A low pass filter with an opamp:
  - The netlist is :

```
V in 0 1
R 1 in 4
```

```
L 2 1 80.96m
C 0 2 2.485m
L 3 2 80.96m
R m 3 4
R 0 m 2meg
R out m 4
R 0 out 1k
E 7 0 m 0 1000
R out 7 10
.clamp out 0
```

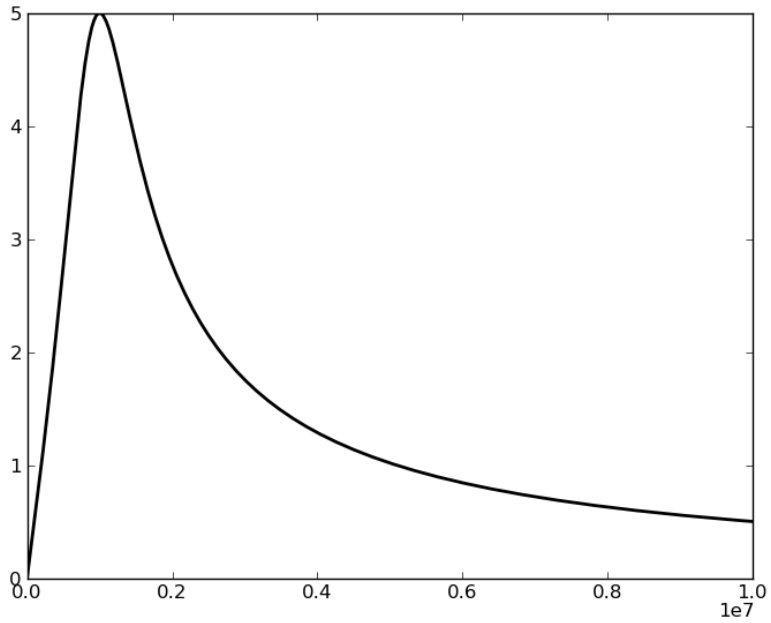
- The plot obtained is:



- A band pass filter
  - The netlist:

```
V n1 0 5
C n2 n1 1u
R n3 n2 1
L 0 n3 1u
```

  - Plot obtained



The netlist with opamp asserts the power of the program.

Drawbacks

The following are the drawbacks:

1. The program does not check pivoting of the matrix. Hence a large noise can also be induced
2. The program cannot solve if there are parallel voltage sources
3. The program does not solve a DC analysis if there are capacitors and inductors.
4. It is only limited to linear sources.

Conclusion

The program effectively exploits the power of C and the pointers in C. The program is a very simple version of the SPICE program which can solve very complicated circuits provided there are only linear elements. The program does a frequency analysis with much efficiency, which is asserted by the plot of the low pass filter and the low pass filter wiht op amp.

References

- Tutorial notes by Dr.Harishankar Ramachandran at $http://www.ee.iitm.ac.in/moodle$
- Ted jensen lectures on pointers at $pw1.netcom.com/\ tjensen/ptr/pointers.htm$
- Tutorial on pointers on Stanford University at $cslibrary.stanford.edu/106/$