现代集成电路分析方法

Project 1 Stamp

专业：电子科学与技术

学号：22112020002

姓名：蔡志杰

目录

1 设计指标

设计一个程序根据输入的线性电路网表生成相应的 MNA 方程：

$$C\dot{X} + GX = BU$$
$$Y = L^T X$$

其中 X 为包含所有节点 (普通节点 (电压变量) 和辅助节点 (电流变量)) 的列向量，$\dot{X}$ 为对应变量的导数，Y 为所有输出节点组成的列向量，U 为所有独立源组成的列向量，$C\ G\ B\ L^T$ 则为系数矩阵。

1.1 输入文件

输入的线性电路为 SPICE 格式的网表文件

(1) 能至少处理 6 种元件：R，L，C，K(互感)，V(独立电压源)，I(独立电流源)。

(2) 应能处理子电路，不用考虑全局节点 (0 节点除外)。

(3) 用一下语句生成输出变量 Y：

.PROBE V(node1) V(node2)...

其中 node1、node2 为要观察的节点的名称。(SPICE 不区分大小写)

1.2 输出结果

输出为一个或多个文件，应包含以下内容：

(1) 各矩阵或向量的大小。

(2)X 中各未知量的具体代表内容。U 中各源的名称。Y 各量的具体代表内容。

(3)$C\ G\ B\ L^T$ 应以二进制格式存储，以保证精度。

此外需要提供一个 MAT 函数能支持读取输出的文件，将其对应的矩阵存储到 MATLAB 内存中。

1.3 进一步的工作

(1) 程序运行过程和最终结果采用稀疏矩阵存储。

(2) 生成分块结构的 MNA 方程.

(3) 支持更多的元件：E(电压控制的电压源)，G(电压控制的电流源)，H(电流控制的电压源)，F(电流控制的电流源)。

2 算法原理

电路分析需要根据电路中的元件特性及其连接关系得到一系列的电路方程，进而求解得到电路的特性。电路方程都是基于基尔霍夫电压定律 (KVL)、基尔霍夫电流定律 (KCL) 和电路元件自身的电流电压方程所构建的。传统的节点电压法和回路电流法不适用于计算机求解，MNA(Modified Node Analysis) 改良节点分析则是构建了 MNA 方程，再通过分析各个元件对 MNA 方程的贡献来逐步构建起 MNA 方程。

$$C\dot{X} + GX = BU$$

$$\begin{bmatrix} C & 0 \\ 0 & L \end{bmatrix} \begin{bmatrix} \dot{V} \\ \dot{I} \end{bmatrix} + \begin{bmatrix} G & E \\ -E^T & 0 \end{bmatrix} \begin{bmatrix} V \\ I \end{bmatrix} = BU$$

上面两个方程展示了原始 MNA 方程和分块 MNA 方程，Stamp 算法的处理思路则是按顺序单独考虑每一个电路元件对 MNA 方程中矩阵的影响，这样便于计算机程序进行处理。

接下来分析一下在 Stamp 方法中各种元件对各个矩阵的影响。

2.1 电阻 R

$$i_R(t) = \frac{1}{R}\left(V_i(t) - V_j(t)\right)$$

$$[C_R] = 0 \quad [G_R] = \begin{matrix} \\ i \\ j \end{matrix} \begin{bmatrix} \begin{matrix} i & j \end{matrix} \\ \begin{matrix} 1/R & -1/R \\ -1/R & 1/R \end{matrix} \end{bmatrix} \quad [B_R] = 0 \quad [X_R] = [V_i\ V_j]^T$$

SPICE 语法：Rname Pnode Nnode Value

其中 i 和 j 分别对应着电阻正负节点 (SPICE 语法：中的 Pnode 和 Nnode) 在 MNA 矩阵中对应的序号 (后文中 i 和 j 的含义也相同)。电阻只会影响 G 矩阵。

若需要为阻引入电流变量 (当流经电阻的电流作用于受控源时)，stamp 操作如下式所示，会改变 G 矩阵。

$$[C_C] = 0 \quad \begin{matrix} i \\ j \\ k \end{matrix} \quad [G_C] = \begin{matrix} \begin{matrix} i & j & k \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & -1 & -R \end{bmatrix} \end{matrix} \quad \begin{matrix} i \\ j \\ k \end{matrix} \quad [B_C] = 0 \quad [X_C] = \begin{matrix} \begin{matrix} i & j & k \end{matrix} \\ [V_i \ V_j \ I_C]^T \end{matrix}$$

2.2 电容 C

$$i_C(t) = C\left(\dot{V}_i(t) - \dot{V}_j(t)\right)$$

$$[C_C] = \begin{matrix} \begin{matrix} i & j \end{matrix} \\ \begin{bmatrix} C & -C \\ -C & C \end{bmatrix} \end{matrix} \quad \begin{matrix} i \\ j \end{matrix} \quad [G_C] = 0 \quad [B_C] = 0 \quad [X_C] = [V_i \ V_j]^T$$

SPICE 语法：Cname Pnode Nnode Value <IC>

电容只对 C 矩阵有贡献。

若需要为电容引入电流变量 (当流经电容的电流作用于受控源时)，stamp 操作如下式所示，会改变 C 和 G 矩阵。

$$[C_C] = \begin{matrix} \begin{matrix} i & j & k \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & -1 & 0 \end{bmatrix} \end{matrix} \quad \begin{matrix} i \\ j \\ k \end{matrix} \quad [G_C] = \begin{matrix} \begin{matrix} i & j & k \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1/C \end{bmatrix} \end{matrix} \quad \begin{matrix} i \\ j \\ k \end{matrix} \quad [B_C] = 0 \quad [X_C] = \begin{matrix} \begin{matrix} i & j & k \end{matrix} \\ [V_i \ V_j \ I_C]^T \end{matrix}$$

2.3 电感 L

$$L\dot{i_L}(t) = C\left(\dot{V}_i(t) - \dot{V}_j(t)\right)$$

$$[C_L] = \begin{matrix} \begin{matrix} i & j & k \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & L \end{bmatrix} \end{matrix} \quad \begin{matrix} i \\ j \\ k \end{matrix} \quad [G_L] = \begin{matrix} \begin{matrix} i & j & k \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix} \end{matrix} \quad \begin{matrix} i \\ j \\ k \end{matrix} \quad [B_L] = 0 \quad [X_L] = \begin{matrix} \begin{matrix} i & j & k \end{matrix} \\ [V_i \ V_j \ I_L]^T \end{matrix}$$

SPICE 语法：Iname Pnode Nnode Value <IC>

为了描述电感的电流需要引入一个额外的变量,在 MNA 中对应一个辅助节点 (aux_node)，k 为电感引入的辅助节点的序号。电感对 C 和 G 矩阵均有贡献。

2.4 互感 K

$$\begin{cases} v_1(t) = L_1\frac{di_1(t)}{dt} + M_{12}\frac{di_2(t)}{dt} \\ v_2(t) = M_{12}\frac{di_1(t)}{dt} + L_2\frac{di_2(t)}{dt} \end{cases}$$

$$\text{Coupling} \quad \text{coefficient} \qquad k = \frac{M}{\sqrt{L_1 L_2}} \quad (M_{12} = M_{21} = M)$$

$$[C_K] = \begin{matrix} \\ i \\ j \end{matrix}\begin{bmatrix} \overset{i}{0} & \overset{j}{-M} \\ -M & 0 \end{bmatrix} \quad [G_K] = 0 \quad [B_K] = 0 \quad [X_K] = \begin{bmatrix} \overset{i}{I_{L1}} & \overset{j}{I_{L2}} \end{bmatrix}^T$$

SPICE 语法：Kname Inductor1 Inductor2 Value_of_k <IC>

这里的 i 和 j 分别指电感 $L1$ 和 $L2$ 对应的辅助节点的序号。互感只对 C 矩阵有贡献。

2.5 独立电流源 I_{src}

$$[C_{CS}] = 0 \quad [G_{CS}] = 0 \quad [B_{CS}] = [-I_{CS}\ I_{CS}]^T \quad [X_{CS}] = [V_i\ V_j]^T$$

SPICE 语法：Iname Pnode Nnode <Type> Value

独立电流源只对 B 矩阵有贡献，在 stamp 的过程中需要给每个电流源在矩阵 U 中确定输入源的序号，独立电流源对 B 矩阵贡献的位置将取决于电源的序号以及电源正负节点对应的序号。SPICE 语法中的 Type 对应着独立电流源的种类 (如交流或直流)，实际中独立电流源和电压源的种类更多，语法也更加复杂，这里默认解析上面这种语法，若没提供种类则默认直流，否则种类后只跟一个数值。(假设电流源的电流的方向为从正节点流到负节点，即对正节点为流出，对负节点为流入)

2.6 独立电压源 V_{src}

$$V_{src} = (V_i - V_j)$$

$$[C_{VS}] = 0 \quad [G_{VS}] = \begin{matrix} \\ i \\ j \\ k \end{matrix}\begin{bmatrix} \overset{i}{0} & \overset{j}{0} & \overset{k}{1} \\ 0 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix} \quad [B_{VS}] = \begin{bmatrix} \overset{i}{0} & \overset{j}{0} & \overset{k}{E_{VS}} \end{bmatrix}^T \quad [X_{VS}] = \begin{bmatrix} \overset{i}{V_i} & \overset{j}{V_j} & \overset{k}{I_{VS}} \end{bmatrix}^T$$

SPICE 语法：Vname Pnode Nnode <Type> Value

为了描述独立电压源的电流需要引入一个辅助节点，k 则对应了这个辅助节点的序号。独立电压源对 G 和 B 矩阵都有贡献，同样为了确定独立电压源对矩阵 B 的贡献位置需要获得电源在矩阵 U 中的序号。(假设电压源的电流变量的方向为从负节点流到正节点，即对正节点为流入，对负节点为流出，之后的受控电压源的电流方向也按照这样定义)

2.7 电流控制的电流源 $F(CCCS)$

$$I_{CCCS} = F \cdot I_{Ctrl}$$

$$[C_F] = 0 \quad [G_F] = \begin{matrix} \\ i \\ j \\ k \end{matrix} \begin{matrix} \begin{matrix} i & j & k \end{matrix} \\ \begin{bmatrix} 0 & 0 & F \\ 0 & 0 & -F \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad [B_F] = 0 \quad [X_F] = \begin{matrix} [\end{matrix} \begin{matrix} i & j & k \\ V_i & V_j & I_{Ctrl} \end{matrix} \begin{matrix}]^T \end{matrix}$$

SPICE 语法：Fname Pnode Nnode Vcontrol Value

在 SPICE 语法中的 Vcontrol 通常是一个元件的名称，以流过这个元件的电流作为控制电流，当这个元件本身已经引入了相应的电流变量和辅助节点时 (如电感 L 和独立电流源 V)，k 则对应了这个辅助节点的序号，但若控制元件没本身没有辅助节点 (如电阻 R 和电容 C) 则需要为其引入相应的电流变量和辅助节点 (这里增加一个辅助节点即电流变量是为了运算的方便，实际上对于 R/C 也可以直接用现有的节点来表示电流，但是加入新的节点可以使得在处理的阶段更加有条理，为电阻和电容增加辅助节点的方法可以参见章节 2.1和章节 2.2)，k 同样为这个辅助节点的序号。电流控制的电流源对 G 矩阵有贡献。

2.8 电流控制的电压源 $H(CCVS)$

$$V_{CCVS} = V_{CCVSP} - V_{CCVSN} = H \cdot I_{Ctrl}$$

$$[C_H] = 0 \quad [G_H] = \begin{matrix} \\ i \\ j \\ k \\ l \end{matrix} \begin{matrix} \begin{matrix} i & j & k & l \end{matrix} \\ \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & -H \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad [B_H] = 0 \quad [X_H] = \begin{matrix} [\end{matrix} \begin{matrix} i & j & k & l \\ V_i & V_j & I_{CCVS} & I_{Ctrl} \end{matrix} \begin{matrix}]^T \end{matrix}$$

SPICE 语法：Hname Pnode Nnode Vcontrol Value

为了描述电流控制电压源的电流，需要为电流控制电压源引入相应的电流变量和辅助节点，节点序号为 k。同样控制电流也有相应的辅助节点，其对应序号为 l(当控制元件没有电流变量 (辅助节点时) 同样也需要为其增加一个辅助变量)。电流控制的电压源对 G 矩阵有贡献。

2.9 电压控制的电流源 $G(VCCS)$

$$I_{VCCS} = F \cdot V_{Ctrl} = G \cdot (V_{CtrlP} - V_{CtrlN})$$

$$[C_G] = 0 \quad [G_G] = \begin{matrix} \\ i \\ j \\ k \\ l \end{matrix} \begin{matrix} \begin{matrix} i & j & k & l \end{matrix} \\ \begin{bmatrix} 0 & 0 & G & -G \\ 0 & 0 & -G & G \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad [B_G] = 0 \quad [X_G] = \begin{matrix} \begin{matrix} i & j & k & l \end{matrix} \\ [V_i \ V_j \ V_{CtrlP} \ V_{CtrlN}]^T \end{matrix}$$

SPICE 语法：Gname Pnode Nnode PControlNode NControlnode Value

式中 k 和 l 分别对于控制的正负节点即 PControlNode NControlnode。电流控制的电压源对 G 矩阵有贡献。

2.10 电压控制的电压源 $E(VCVS)$

$$V_{VCCS} = V_{VCCSP} - V_{VCVSN} = E \cdot V_{Ctrl} = E \cdot (V_{CtrlP} - V_{CtrlN})$$

$$[C_E] = 0 \quad [G_E] = \begin{matrix} \\ i \\ j \\ k \\ l \\ m \end{matrix} \begin{matrix} \begin{matrix} i & j & k & l & m \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & -E & E & 0 \end{bmatrix} \end{matrix} \quad [B_E] = 0 \quad [X_E] = \begin{matrix} \begin{matrix} i & j & k & l & m \end{matrix} \\ [V_i \ V_j \ V_{CtrlP} \ V_{CtrlN} \ I_{VCVS}]^T \end{matrix}$$

SPICE 语法：Ename Pnode Nnode PControlNode NControlnode Value

式中 k 和 l 分别对于控制的正负节点即 PControlNode NControlnode，为了表示电压控制电压源的电流变量，引入新的辅助节点，m 为其序号。电流控制的电压源对 G 矩阵有贡献。

3 算法实现

为了实现从 SPICE 网表生成 MNA 矩阵的过程，整个算法可以分为三大步骤：

- 对输入的网表文件进行解析，得到相应的器件信息 (如器件的连接关系、器件的值等)，同时能够递归地解析嵌套的子电路。解析完网表文件后进行预处理 (增加相应的电流变量和辅助节点，确定独立电流、电压源的序号)
- 对每个器件进行 stamp 操作，完成电路的 MNA 方程。
- 将得到的 MNA 矩阵输出为二进制格式的数据。

$$C\dot{X} + GX = BU$$
$$Y = L^T X$$

为了提高输出文件的可读性，$C\ G\ B\ L^T$ 存储的为浮点数，$X\ Y\ U$ 存储的为字符串，对应了相应的变量名。同时增加了一个和列向量 U 大小相同的列向量 I 存储 U 向量中输入对应的数值 (如独立电压/流源的电压/流)。

接下来将分别对各个步骤以及一些底层的实现进行介绍。

3.1 解析 SPICE 网表

基于原有的 parser 框架进行修改和完善，增加了解析子电路和四种受控源的部分。

3.1.1 解析电路元件和输出节点

对于电路元件，相应的 SPICE 语法按照章节 2中介绍的进行解析。为了方便后续分块矩阵的实现，将普通节点 (电压节点) 和辅助节点 (电流变量) 分开存储。在解析过程中就先对普通节点进行编号，当所有元件都已经解析完，所有辅助节点都以及生成好后，再对辅助节点按照元件的类别的顺序生成序号，这样能保证辅助节点的序号都在普通节点之后。

对于输出节点的解析，SPICE 的语法是 .PROBE V(node1) V(node2) ...。其中 node1、node2 为要观察的节点名称 (通常为普通节点)，只要解析到所有要观察的节点就能得到输出矩阵 Y 的变量。

3.1.2 解析子电路

在 SPICE 中子电路的定义语法为：

.SUBCKT SUBNAME N1 N2 N3 ...

Element statements

.

.

.ENDS SUBNAME

子电路的调用方法为：

Xname N1 N2 N3 ... SUBNAME

其中 SUBNAME 为子电路的名称，N1、N2、N3... 为子电路的接口，Xname 则为实例化子电路的名称。由于子电路的定义可能在实例化之后，而且子电路还可能出现嵌套调用的情况 (即子电路中包含别的子电路的实例)，因此为解析子电路单独构建了一个解析函数打包在 Subckt 类下。该类只有一个实例化函数 instantiate，每当解析到 SPICE 中的实例化语句时会进行调用 (以 X 开头的语句)。实例化函数的运行流程是先根据实例化语句获取实例化子电路的名称、接口节点名称以及子电路名称，然后根据子电路名称从网表文件中查找子电路的定义语句，然后解析子电路对应语句中的电路元件。为了避免重名，子电路中电路元件的名称后面会加上 # 和子电路实例名。同时对于子电路中的节点，如果为子电路的端口节点 (即定义在.SUBCKT SUBNAME ... 这一行的节点名) 则将其名字对应到实例化子电路的接口节点名称中，同时 0 节点作为全局节点若识别到也会直接映射到序号 0，此外的节点为子电路的内部节点，需要添加新的节点，并为其分配序号，同时节点的命名也是节点原先名字加上 # 和子电路实例名。这里对子电路的操作会影响到一些电路数据 (如新增的节点，新增的元件)，为了函数调用时方便，将算法操作过程中所需的一些变量都打包到一个 db 类中 (包含 MNA 的各个矩阵变量、元件信息、节点信息等)，调用子电路解析函数时只需要将 db 类的引用作为参数传递进去。同时在子电路解析过程中如果再次遇到子电路实例化语句则会递归调用 instantiate 函数，继续解析，以实现解析嵌套子电路的功能。

3.1.3 元件预处理

在解析网表文件的过程中，为了实现分块矩阵 (节点矩阵 X 的有序性) 暂时没有直接为辅助节点确定序号，同时还有一些元件也需要后续处理一下 (增加额外的辅助节点，绑定相关电感等) 这些操作都将在元件预处理中进行。

由于预处理中需要遍历某一类型的元件，在解析过程中为各种元件独立设置了一个 map

来存储其对应的元件索引 (一个从字符串映射到整数集合的 map，字符串为元件的类型如"R"、"C"，整数集合则为这一类型的元件索引)，接下来遍历所有的电感、独立电压源、电压控制的电压源和电流控制的电压源，为其绑定额外节点的序号。然后遍历独立电源为其绑定电源的序号。接着遍历所有互感，为其绑定两个相应的电感、辅助节点和计算互感值。最后遍历电流控制的电压源和电流控制的电流源，确定其控制源是否存在电流变量，存在则之间绑定控制源的辅助节点序号，不存在则要为其增加新的辅助节点，然后再绑定辅助节点序号。

3.2 电路元件实现

为了简化元件的表示，将元件共有的属性提取成一个抽象基类 Device(具有元件名称 _name、正负节点序号 _p/nnode、元件的值 _value、辅助节点序号 _aux_node 和名字 _aux_name、是否具有辅助节点的标志位 _exist_aux_node(同时包含相应成员变量的设置 set 和访问 get 接口)，以及 stamp 函数 (纯虚函数))。为了方便受控源的实现，为其也建立了一个基类 CtrlSrc，以 Device 为基类，额外增加了控制元件的正负节点 _ctrl_p/nnode，以及控制元件的名称 _ctrl_name，同样也包含设置和访问相应成员变量的接口。

为了表示每一个元件，为每一种类别的元件都创建一个 C++ 类，电阻 Resistor、电容 Capacity、电感 Inductor、互感 Mutual、独立电流源 Isrc 和独立电压源 Vsrc 则是直接继承与 Device 基类，其中互感需要额外增加成员变量：互感的值 _mut_value(可通过章节 2.4中的公式计算得到)、两个电感的名字 _ind1/2 和两个电感对于的辅助节点的序号 _aux_node_ind1/2。对于独立电流源和电压源，则需要增加表示电源类型的成员变量 _type 和电源对于的序号 _src_idx(在 U 矩阵中的序号)。

电压控制的电压源 Vcvs、电压控制的电流源 Vccs、电流控制的电压源 Ccvs 和电流控制的电流源 Cccs 则继承与 CtrlSrc 基类。其中电流控制的电压源和电流源需要增加以下的成员变量：控制元件的名字 _ctrl_name、控制元件的辅助节点序号 _ctrl_aux_node、控制节点的值 _ctrl_value(电阻就为阻值，电容就为容值，用于为之前没有辅助节点的电阻和电容创造辅助节点所需的方程，体现在 MNA 系数矩阵中，对矩阵的修改方法详见章节 2.1和章节 2.2) 以及控制元件是否存在辅助节点的标志位 _exist_ctrl_aux(不存在则需在 stamp 操作中为辅助节点进行相应的操作，创造辅助节点的步骤会在解析后的元件预处理中进行)。

各个元件的 stamp 操作则按照章节 2中描述的进行。

3.3 矩阵实现

由于 MNA 的系数矩阵具有稀疏的特点，因此可以实现稀疏矩阵更能节省空间，所有对于 MNA 的稀疏矩阵需要实现的功能主要有基于底层稀疏矩阵的数据结构，实现对矩阵某一元素访问 get 并加减 add 的功能，同时为了更方便地获得矩阵的某一部分 (打印出相应的分块矩阵) 还实现了根据根据子矩阵的其实坐标和子矩阵的大小，从矩阵中通过切片获得子矩阵的接口 getSubMatrix。

为了输出的方便，为矩阵提供了几个输出的接口 output(普通输出，会按照矩阵大小，将完整的矩阵打印出来，没有元素的地方会用 0 填补)，sparse_output(输出矩阵的稀疏新式即每行只包括对应元素的 (row, col) value) 和 binary_output(将矩阵按照二进制形式打印出来，开头打印三个数字，分别对应矩阵的总行数、总列数以及非零元素个数，后续每行包括每个非零元素的行、列和值这三个数，打印出的文件存储于.dat 文件中，方便 matlab 函数读取)。

此外每个矩阵还有一些成员变量：矩阵的大小 (总行列数)、矩阵的名字 (便于打印的时候验证)。

3.3.1 稀疏存储

虽然原始代码中建议用十字链表来实现矩阵的稀疏存储，但是考虑到链表这一结构访问的元素的速度还不够快，因此本算法采用 c++ 标准库中的 map 进行实现，矩阵底层数据的类型为 std::map<int, std::map<int,T>> 其中 T 为矩阵的模板变量，可以存储不同类型的矩阵，由于 map 底层的实现是红黑树，在数据存储上稀疏程度与链表接近，同时红黑树的查找时间复杂度为 O(log(N))(十字链表是 O(N)，二维数组是 O(1))，其中 N 为该行/列的非 0 元素数量，算是在内存和运行时间上取得了一个较好的权衡。

3.3.2 分块矩阵

由于前面解析和预处理过程以及将节点按照类别进行排序 (电压节点在前，辅助节点在后)，同时辅助节点也有按照类型进行排序，因此直接按照完整矩阵进行切割即可得到相应的分块矩阵。因此只要实现子矩阵获取功能即可得到分块矩阵。(经分析当电流控制的电压源的控制源有辅助节点时，G 矩阵右下角的子矩阵都不为全零矩阵。)

$$\begin{bmatrix} C & 0 \\ 0 & L \end{bmatrix} \begin{bmatrix} \dot{V} \\ \dot{I} \end{bmatrix} + \begin{bmatrix} G & E \\ -E^T & 0 \end{bmatrix} \begin{bmatrix} V \\ I \end{bmatrix} = BU$$

3.3.3 Matlab 读取矩阵

对输出的二进制文件，编写了 matlab 函数 read_matrix 来读取 C++ 程序输出的二进制文件，读取的时候先用一个 3xn 的矩阵来存储数据 (其中 n 为矩阵非 0 元素的个数，每一列为非零元素的行坐标、列坐标和值)，然后再用 matlab 的 sparse 函数将其转换成稀疏矩阵。

4 编译运行与结果

为了使得本次项目的结构更加清晰，将原始的源文件加头文件的格式换成了 hpp 的文件格式，每一个类都对应一个 hpp 文件，同时编写了 CMakeLists.txt 来方便文件的编译。

4.0.1 编译运行

编译方法：

cd ./project/src

mkdir build

cd ./build

cmake ..

make

运行方法：

./MNA infile.sp

其中 infile.sp 为输入的 SPICE 网表，将会在当前目录自动输出五份文件，分别为 infile_binary.dat(输出的二进制矩阵数据)，infile_full.txt(输出的完整矩阵，用于观察)，infile_sparse.txt(输出的稀疏矩阵,用于观察),infile_sub_matrix.txt(输出的分块矩阵) infile_zero.txt(输出分块矩阵中对应全零的矩阵用于验证是否满足全 0 的要求)。

编译环境为 Linux 系统，gcc 版本为 9.4.0。

4.0.2 运行结果

运行结果如下图所示，由于 Bus 和 Tree 两个例子的规模太大没办法展示完全，故以 Test 和 RLC 两个例子为例。(为了方便展示将输出文件中的各个矩阵截图后拼接了一下，原始数据可以参见/output/RLC_s3_full.txt 和/output/Test_0_full.txt)。对 RLC 和 Test 网表进行了手动验算，发现结果与矩阵输出一致。

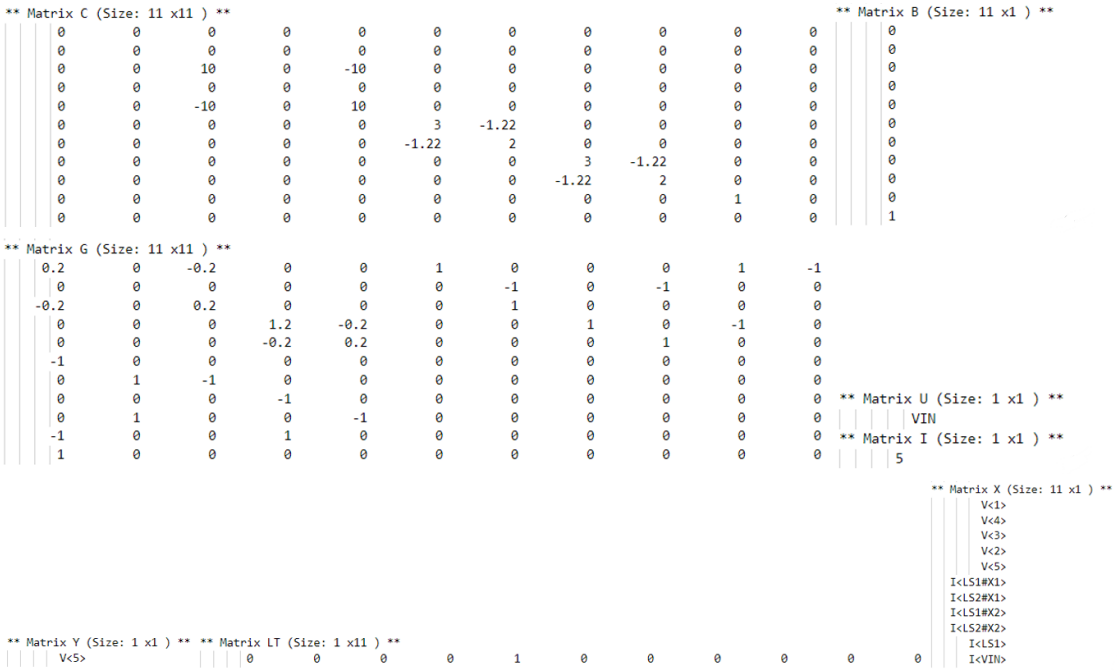

图 1: Test_0 全矩阵

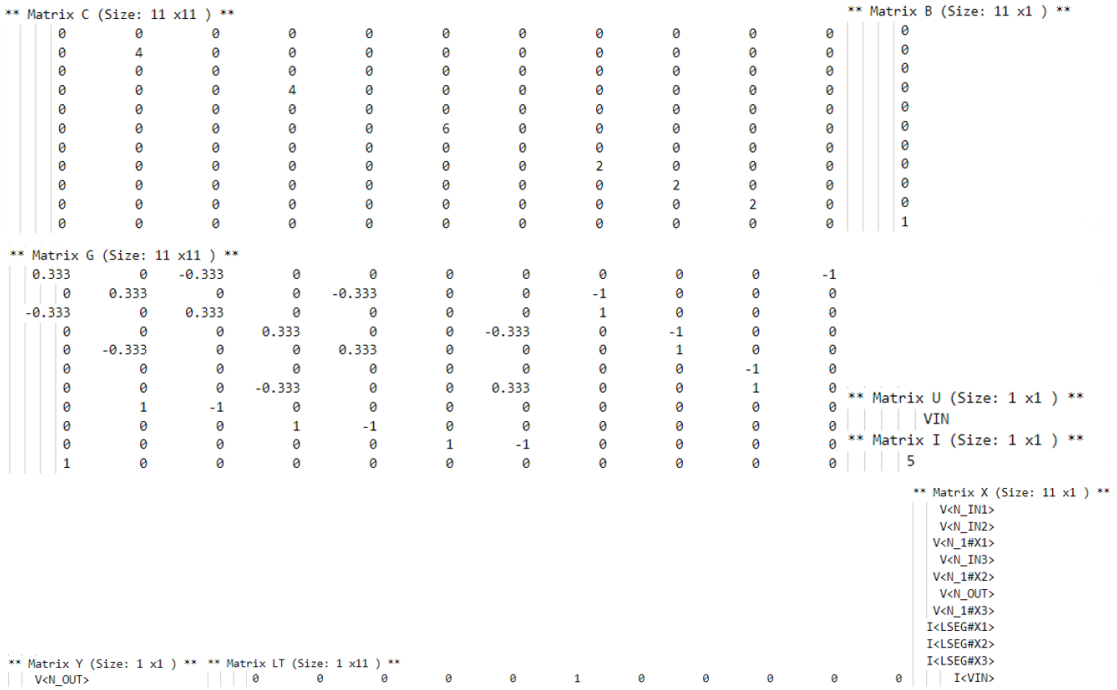

图 2: RLC_s3 全矩阵

下图为使用编写的 read_matrix 函数，在 matlab 中读取的稀疏矩阵的结果，经过验证稀疏矩阵的数据与完整矩阵的数据一致。

```
Test_C =
   (3,3)     10.0000
   (5,3)    -10.0000
   (3,5)    -10.0000
   (5,5)     10.0000
   (6,6)      3.0000
   (7,6)     -1.2247
   (6,7)     -1.2247
   (7,7)      2.0000
   (8,8)      3.0000
   (9,8)     -1.2247
   (8,9)     -1.2247
   (9,9)      2.0000
  (10,10)     1.0000
Test_G =
   (1,1)      0.2000
   (3,1)     -0.2000
   (6,1)     -1.0000
  (10,1)     -1.0000
  (11,1)      1.0000
   (7,2)      1.0000
   (9,2)      1.0000
   (1,3)     -0.2000
   (3,3)      0.2000
   (7,3)     -1.0000
   (4,4)      1.2000
   (5,4)     -0.2000
   (8,4)     -1.0000
  (10,4)      1.0000
   (4,5)     -0.2000
   (5,5)      0.2000
   (9,5)     -1.0000
   (1,6)      1.0000
   (2,7)     -1.0000
   (3,7)      1.0000
   (4,8)      1.0000
   (2,9)     -1.0000
   (5,9)      1.0000
  (1,10)      1.0000
  (4,10)     -1.0000
  (1,11)     -1.0000
Test_B =
  (11,1)      1
Test_LT =
   (1,5)      1
```

图 3: Test_0 稀疏矩阵

```
RLC_C =
   (2,2)      4
   (4,4)      4
   (6,6)      6
   (8,8)      2
   (9,9)      2
  (10,10)     2
RLC_G =
   (1,1)      0.3333
   (3,1)     -0.3333
  (11,1)      1.0000
   (2,2)      0.3333
   (5,2)     -0.3333
   (8,2)      1.0000
   (1,3)     -0.3333
   (3,3)      0.3333
   (8,3)     -1.0000
   (4,4)      0.3333
   (7,4)     -0.3333
   (9,4)      1.0000
   (2,5)     -0.3333
   (5,5)      0.3333
   (9,5)     -1.0000
  (10,6)      1.0000
   (4,7)     -0.3333
   (7,7)      0.3333
  (10,7)     -1.0000
   (2,8)     -1.0000
   (3,8)      1.0000
   (4,9)     -1.0000
   (5,9)      1.0000
   (6,10)    -1.0000
   (7,10)     1.0000
   (1,11)    -1.0000
RLC_B =
  (11,1)      1
RLC_LT =
   (1,6)      1
```

图 4: RLC_s3 稀疏矩阵

下图为 RLC 例子的分块矩阵输出结果，为了方便对对比同样将矩阵从输出文件中进行截图和拼接 (原始数据可参见/output/RLC_s3_sub_matrix.txt)，从结果可以看出分块矩阵达到了预期的效果，同时与完整的矩阵结果相同，并且全零矩阵区域在各个例子中都为 0。

```
** Matrix subC (Size: 7 x7 ) **
     0     0     0     0     0     0     0
     0     4     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     4     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     6     0
     0     0     0     0     0     0     0
** Matrix zero1 (Size: 7 x4 ) **
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
** Matrix zero2 (Size: 4 x7 ) **
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
     0     0     0     0     0     0     0
** Matrix subL (Size: 4 x4 ) **
     2     0     0     0
     0     2     0     0
     0     0     2     0
     0     0     0     0
** Matrix subG (Size: 7 x7 ) **
 0.333     0  -0.333     0     0     0     0
     0 0.333     0     0  -0.333     0     0
-0.333     0 0.333     0     0     0     0
     0     0     0 0.333     0     0  -0.333
     0 -0.333     0     0 0.333     0     0
     0     0     0     0     0     0     0
     0     0     0  -0.333     0     0 0.333
** Matrix subE (Size: 7 x4 ) **
     0     0     0    -1
    -1     0     0     0
     1     0     0     0
     0    -1     0     0
     0     1     0     0
     0     0    -1     0
     0     0     1     0
** Matrix subV (Size: 7 x1 ) **
V<N_IN1>
V<N_IN2>
V<N_1#X1>
V<N_IN3>
V<N_1#X2>
V<N_OUT>
V<N_1#X3>
** Matrix subI (Size: 4 x1 ) **
I<LSEG#X1>
I<LSEG#X2>
I<LSEG#X3>
  I<VIN>
** Matrix sub-E^T (Size: 4 x7 ) **
     0     1    -1     0     0     0     0
     0     0     0     1    -1     0     0
     0     0     0     0     0     1    -1
     1     0     0     0     0     0     0
** Matrix zero3 (Size: 4 x4 ) **
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

图 5: Matlab 读取二进制文件的矩阵结果

由于两个大例子 Tree 和 Bus 未能展示完整的矩阵，下图展示了 matlab 读取 c++ 输出二进制文件后的矩阵大小。由于给定的例子中没有受控源，故暂时无法验证其正确性.

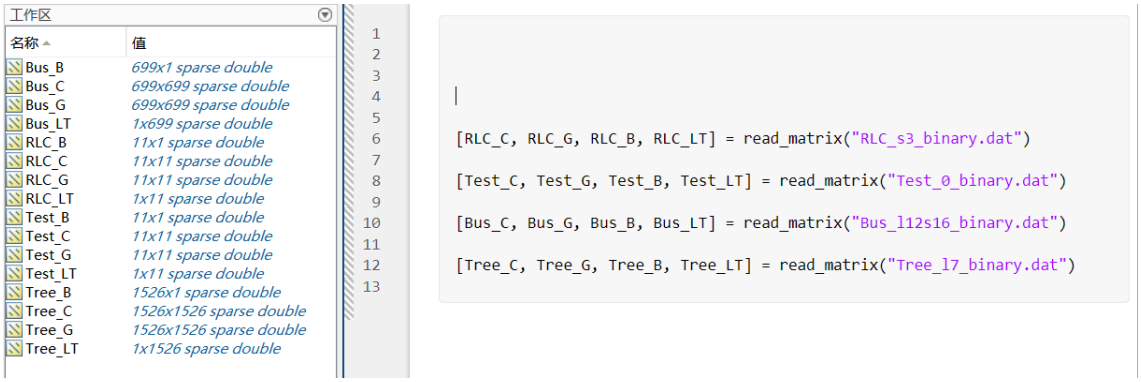

图 6: Matlab 读取二进制文件的矩阵结果

5 文件说明

本次项目目录下主要有一下几个子目录：

- benchmark/：里面为输入的网表文件
- doc/：编写的项目报告
- output/：输出文件
- src/：项目的源代码
  - db/ ：项目基本的数据类，包括各个元件的 hpp 文件以及子电路的 hpp 文件
  - utils/ ：项目用到的一些功能函数，如矩阵 hpp 文件、解析需要用到的函数以及打 log 的文件。
  - CMakeList.txt：cmake 配置文件
  - main.cpp：主函数
  - stamp.hpp：stamp 函数
  - read_matrix.mlx：matlab 读取二进制文件的函数
  - test.mlx：matlab 测试代码