现代集成电路分析方法

Project 5 Solve MNA

专业: 电子科学与技术

学号:22112020002

姓名: 蔡志杰

1 题目

请设计一个程序, 用后向 Euler 法/梯形法求解电路系统的 MNA 方程 (该方程由 stamp 程序得到), 其中线性方程的求解可以选择如下几种方法: 直接法 (Project III), 稳态迭代法 (Jacobi, G-S, SOR, Project IV),GCR 迭代法 (Project V)。

$$C\dot{X} + GX = BU \tag{1}$$

$$Y = L^T X \tag{2}$$

系统的初值 X_0 通过求解 $GX_0 = BU(0)$, 也就是 DC 分析得到。

1.1 输入

输入的线性电路为 SPICE 格式的网表文件

(1) 电路方程: 以提供的 stamp 程序的输出作为本程序的输入。

(2) 部分参数: 模拟时间长度 T, 误差容限 Epi。

(3) 电路激励: 可以将源写成独立的函数, 对不同的问题相应手工修改该函数, 应至少支持 sin 以及 pulse 输入 (关于这两种输入的具体参数请参考 HSPICE 手册)。

(4) 标准结果: 以 SPICE 的模拟结果作为标准结果, 用于和程序模拟结果比较。为方便比较结果, 可以将程序的时间步长设为与 SPICE 一样, 在时间不长不一致的情况下, 则可以通过插值的方法得到在相同的时间步长下的结果。

1.2 输出结果

输出为四张图示和部分数值结果。

图 1 为输入波形。图 2 为模拟得到的输出波形。图 3 为模拟得到的输出波形和标准结 (SPICE 结果) 的比较。图 4 为整个模拟区间上的绝对误差分布。同时程序应该输出:Euler 方法的计算点数, 总模拟时间, 整个模拟区间上的最大绝对误差 (绝对值) 和均方绝对误差。

* 计算误差的采样点即为标准结果中给出的时间点。如该点不是 Euler 方法的计算点, 则模拟结果通过邻近两个计算点模拟结果的线性插值得到。

** 均方误差的定义: $MSE = \sqrt{\frac{1}{n}\sum_{i=0}^{n} E_i^2}$, 其中 E_i 为每个采样点上的误差,n 为采样点数.

1.3 测试用例

Benchmark 目录下提供三个测试用例 RLC_s3.sp,bus32bit8seg.sp 以及 bus8bit8seg.sp, 并提供 Matlab 下的 stamp 程序供构造电路矩阵。stamp 用法请参考 Benchmark 目录下的 stamp_man 文件。同时提供 read_data 程序来读取 hspice 输出 *.lis 文件中的波形数据。read_data 程序的用法请参考 benchmark 目录下的 read_data_man 文件。Stamp 程序支持 sin 以及 pulse 的输入, 读取的数据在 SRC 变量中。

1.4 提交结果

程序建议采用 MATLAB 完成, 需提交以下内容:

(1) 源程序, 应有必要的注释。

(2) 使用除 MATLAB 外其他语言的, 需要提交最终编译的可执行代码。

(3) 一份完整的说明, 主要内容包括: 主要设计思想, 程序结构, 编译的环境和方法, 运行的环境和方法, 输入的格式或方法, 以及其他需要特别说明的地方。

(4) 对测试用例的测试结果和分析。

2 算法原理

首先通过 stamp 程序获得 spice 网表对应的电路 MNA 方程。对于得到电路方程, 可以采用特征值分解的方法进行求解 (耗时) 或采用有限差分方法对其进行求解, 其中有限差分法, 包含前向欧拉 (Forward Euler)、后向欧拉 (Backward Euler) 和梯形法 (Trapezoidal Rule)。在有限差分法中又涉及到线性方程组的求解, 求解线性方程组的方法有直接法 (高斯消元法,LU 分解,QR 分解等)、迭代法 (Jacobi,Gauss-Seidel,SOR 等) 和广义共轭残差法 GCR。

2.1 偏微分方程求解

对于如下的偏微分方程, 我们介绍前向欧拉、后向欧拉和梯形法。

$$\frac{dx(t)}{dt} = Ax(t) \tag{3}$$

$$x(t_0) = x_0 \tag{4}$$

2.1.1 前向欧拉法 Forward Euler

前向欧拉法的公式如下, 使用时间上更迟的数据点做差分来逼近梯度, 只需要矩阵乘法, 而不需要求解线性方程, 速度更快, 但是精度欠佳。

$$\frac{d}{dt}x\left(t_l\right) = Ax\left(t_l\right) \cong \frac{x\left(t_{l+1}\right) - x\left(t_l\right)}{\Delta t} \tag{5}$$

$$x\left(t_{l+1}\right) \cong x\left(t_l\right) + \Delta t Ax\left(t_l\right) \tag{6}$$

公式 1中前向欧拉的迭代公式为:

$$CX(t_{l+1}) = (C - \Delta tG)X(t_l) + \Delta tBU(t_l) \tag{7}$$

2.1.2 后向欧拉法 Backward Euler

后向欧拉法的公式如下, 使用时间上更早的数据点做差分来逼近梯度, 需要求解线性方程, 速度慢于前向欧拉, 但是精度更好。在等步长 Δt 的情况下, 可省去重复的计算步骤来实现加速。

$$\frac{d}{dt}x\left(t_{l+1}\right) = Ax\left(t_{l+1}\right) \cong \frac{x\left(t_{l+1}\right) - x\left(t_l\right)}{\Delta t} \tag{8}$$

$$x\left(t_{l+1}\right) \cong x\left(t_l\right) + \Delta t Ax\left(t_{l+1}\right) \tag{9}$$

公式 1中后向欧拉的迭代公式为:

$$(C + \Delta tG)X(t_{l+1}) = CX(t_l) + \Delta tBU(t_{l+1}) \tag{10}$$

2.1.3 梯形法 Trapezoidal Rule

梯形法的公式如下, 使差分来逼近相邻两点梯度的平均值, 计算比后向欧拉更加复杂, 但是精度更好。

$$\frac{1}{2}\left(\frac{d}{dt}x\left(t_{l+1}\right) + \frac{d}{dt}x\left(t_l\right)\right) = \frac{1}{2}\left(Ax\left(t_{l+1}\right) + Ax\left(t_l\right)\right) = \frac{x\left(t_{l+1}\right) - x\left(t_l\right)}{\Delta t} \tag{11}$$

$$x\left(t_{l+1}\right) = x\left(t_l\right) + \frac{1}{2}\Delta t A\left(x\left(t_{l+1}\right) + x\left(t_l\right)\right) \tag{12}$$

公式 1中梯形法的迭代公式为:

$$(2C + \Delta tG)X(t_{l+1}) = (2C - \Delta tG)X(t_l) + \Delta tB(U(t_{l+1}) + U_l) \tag{13}$$

2.1.4 小结

以上三种方法都是 one-step 方法, 只依赖于前一时间点的数据。前向欧拉是最简单的方法, 后向欧拉更加复杂, 而梯形法可能是最精确的。

从稳定性上看, 前向欧拉的时间步长需要足够小来保证解的稳定性 (极点不位于右半平面, 解是等幅震荡或衰减的)。后向欧拉则能在原始解稳定的情况下保证解的稳定性, 当原始解发散的时候的时候只要时间步长足够大也能保证解的衰减性。梯形法同样在解稳定时能得到稳定的解, 当解不稳定 (幅度不断增大时), 得到的解也不稳定。

2.2 线性方程求解

对于如下的线性方程组, 为了求解 x 的值, 通常有直接法 (高斯消元法,LU 分解,QR 分解等)、迭代法 (Jacobi,Gauss-Seidel,SOR 等) 和广义共轭残差法 GCR。

$$Mx = b \tag{14}$$

接下来主要迭代法和广义共轭残差法 GCR.

2.2.1 Jacobi 迭代法

将矩阵拆分成上三角、对角线和下三角矩阵, 通过移项得到迭代公式:

$$M = D - L - U \tag{15}$$

$$(D - L - U)x = b \tag{16}$$

$$Dx_k = (U + L)x_{k-1} + b \tag{17}$$

迭代速度较慢, 没有合理利用前面迭代得到的信息。

2.2.2 Gauss-Seidel 迭代法

和 Jacobi 方法相比, 利用了更多先前迭代中的结果, 能获得更快的收敛速度:

$$(D - L)x_k = Ux_{k-1} + b \tag{18}$$

2.2.3 Successive Over Relaxation 逐次超松弛法

基于 Guass-Seidel 迭代法, 进行修改来提高迭代收敛速度,

$$x_k = w\overline{x_k} + (1+w)x_{k-1} \tag{19}$$

$$\overline{x_k} = D^{-1}Lx_k + D^{-1}Ux_{k-1} + D^{-1}b \tag{20}$$

$$x_k = (D - wL)^{-1}((1-w)D + wU)x_{k-1} + w(D - wL)^{-1}b \tag{21}$$

2.2.4 广义共轭残差法 GCR

第 k 轮残差的定义为,

$$r^k \equiv b - Mx^k \tag{22}$$

第 k 轮迭代的解 x_k 定义为,

$$x^{k+1} = \sum_{i=0}^{k} \alpha_i w_i \tag{23}$$

目标是最小化残差的 2 范数, 当构成 x 的基向量关于 M 矩阵正交, 即 $(Mw_i)^T(Mw_j) = 0(i)$ 时残差的 2 范数很容易优化,

$$\min_x \|r^{k+1}\|_2^2 = \|b - \sum_{i=0}^{k} \alpha_i M w_i\| \tag{24}$$

将其转换成一个二次型目标函数, 取最优值的地方梯度为 0, 梯度对应的就是残差, 能够间接求解线性方程组。(只适用于矩阵 M 为对称正定阵)

$$f(x) = \frac{1}{2}x^T Mx - x^T b \tag{25}$$

$$\nabla_x f(x) = Mx - b \tag{26}$$

对目标函数 f 采用最速下降法, 将梯度作为新的探索方向, 经过分析可得如下的迭代公式:

$$\alpha_k = \frac{\left(r^k\right)^T (Mp_k)}{(Mp_k)^T (Mp_k)} \tag{27}$$

$$x^{k+1} = x^k + \alpha_k p_k \tag{28}$$

$$r^{k+1} = b - Mx^{k+1} = b - Mx^k - \alpha_k Mp_k = r^k - \alpha_k Mp_k \tag{29}$$

$$p_{k+1} = r^{k+1} - \sum_{j=0}^{k} \frac{\left(Mr^{k+1}\right)^T (Mp_j)}{(Mp_j)^T (Mp_j)} p_j \tag{30}$$

如果矩阵 M 是稀疏的, 具有 k 个特征值, 你们只需要 k 轮迭代即可收敛, 收敛速度更快。当矩阵 M 为对称矩阵时,$r^{k-1} \perp Mp_j, j < k$,$p_{k+1}$ 的正交化只需要一步操作。

$$p_{k+1} = r^{k+1} - \sum_{j=0}^{k} \frac{\left(Mr^{k+1}\right)^T (Mp_j)}{(Mp_j)^T (Mp_j)} p_j \Rightarrow p_{k+1} = r^{k+1} - \frac{\left(Mr^{k+1}\right)^T (Mp_k)}{(Mp_k)^T (Mp_k)} p_k \tag{31}$$

2.3 算法实现

2.3.1 算法流程

算法流程

- 使用提供的 stamp 动态库对 hsipce 网表进行解析得到相应的矩阵数据, 使用 hspice 进行仿真, 得到相应的波形 lis 文件, 调用 $\text{read}_data \quad lis \quad hspice \quad (main.m) \; stamp \quad (GenerateSrc.m)$

- 根据函数的输入选择是使用 Backward Euler 还是 Trapezoidal Rule 的方法来求解偏微分方程。(BackwardEuler.m, TrapezoidalRule.m)

- 在 Backward Euler 还是 Trapezoidal Rule 中根据函数的输入决定采用 GCR 还是 SOR 进行线性方程的求解。(GCR.m, SOR.m)

- 将得到的仿真结果通过线性插值得到与 hspice 相同采样时间的数据点, 并计算相应的误差 (最大绝对误差和均方绝对误差)。(main.m)

- 生成结果对比图。(report.m)

2.3.2 输入激励信号

脉冲源

语法:

Vxxx n+ n- PU<LSE> <(>v1 v2 <td <tr <tf <pw <per> > > > > <)>

Ixxx n+ n- PU<LSE> <(>v1 v2 <td <tr <tf <pw <per> > > > > <)>

其中 v1 为脉冲源的低压,v2 为高压,td 为延迟,tr 为上升时间,tf 为下降时间,pw 为脉冲宽度,per 为信号周期。下图展示了一个脉冲信号源的例子:

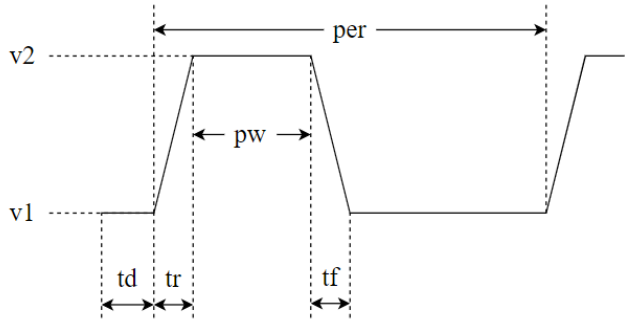

图 1: 脉冲信号源示例

正弦源

语法:

Vxxx n+ n- SIN <(> vo va <freq <td <θ <ϕ> > > > <)>

Ixxx n+ n- SIN <(> vo va <freq <td <θ <ϕ> > > > <)>

其中 vo 为直流幅度,va 为交流幅度,freq 为频率,td 为延时,θ 为衰减因子,ϕ 为相位延时。信号的公式可以如下:

$$y(t) = e^{-\theta} \cdot va \cdot sin(2\pi \cdot freq \cdot (t - td) - \phi) + vo \tag{32}$$

2.3.3 线性方程组求解

不论是使用 GCR 还是 SOR 来求解线性方程组都需要确定一个误差容限, 本实验中将误差容限 $errorThres$ 的设置放到顶层函数接口, 同时误差的标准为两次迭代间解 x 值的变化的 2 范数, 即:

$$||x_{t+1} - x_t|| < errotThres \tag{33}$$

此外迭代的轮数也要进行限制,GCR 迭代的最大轮数即为矩阵的行/列数,SOR 的最大迭代轮数设置成 1000 轮。

2.3.4 输出

为了将仿真结果与 hspice 的结果进行对比, 首先需要通过插值法得到自己仿真结果在 hspice 采样时间上的值, 然后求解最大绝对误差和均方绝对误差。同时为了方便访问结果, 使用 save 命令将仿真的数据存储到.mat 文件中。对于有多个输出节点的电路, 为了方便观察选取其中一个的波形进行绘制。

2.4 编译与运行

程序基于 Matlab 2014a(32 位) 平台, 将整个解析网表文件、MNA 方程求解和结果输出的过程都打包在 main 函数中,main 函数的接口为 $main(caseName, methodName, algorithmName, startTime, endTime, stepNum, errorThres)$. 其中 caseName(RCL_s3, RCL_p3,bus8bit8seg,bus32seg16),methodName(BE,TR),algorithmName(SOR,G 分别对于测试用例名、常微分方程求解方法 (后向欧拉和梯形法), 以及线性方程求解方法。startTime 和 endTime 为记录仿真结果的时间区间,stepNum 为这一区间的采样数,errorThres 为误差阈值。下面给出一个运行示例:

$$main('RLC_s3','BE','GCR',0,2*1e-2,1000,1e-9) \tag{34}$$

2.5 结果与分析

2.5.1 实验结果

2.5.1.1 RLC_s3(正弦激励信号)

仿真设置: 起始时间 0s, 终止时间 0.02s, 采样点数 1000, 误差容限 $1e^{-9}$, 下面展示了几种算法得到的仿真结果:

后向欧拉 (BE)+GCR:

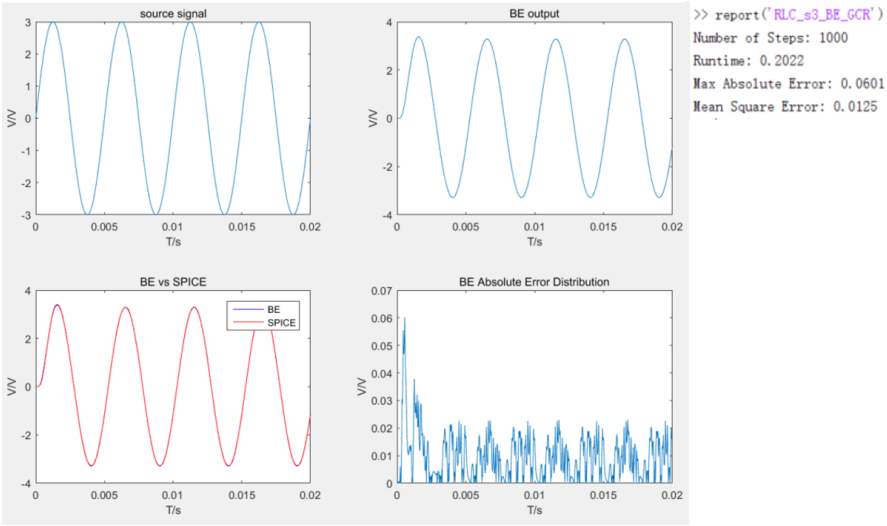

图 2: RLC_s3_BE_GCR 实验结果

梯形法 (TR)+GCR:

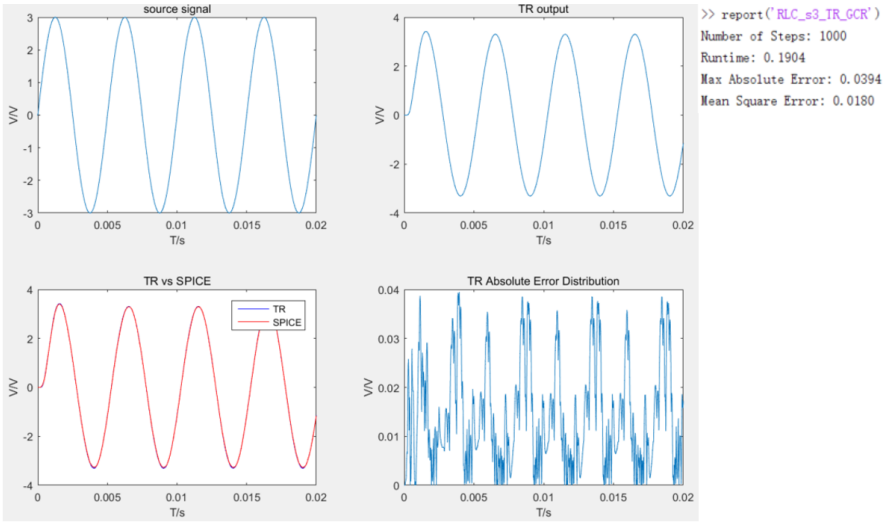

图 3: RLC_s3_TR_GCR 实验结果

后向欧拉 (BE)+SOR:

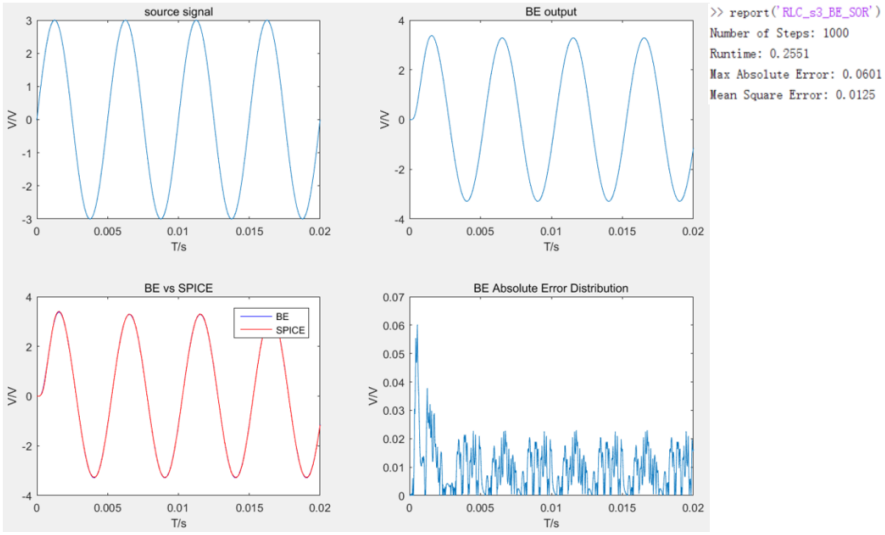

图 4: RLC_s3_BE_SOR 实验结果

梯形法 (TR)+SOR:

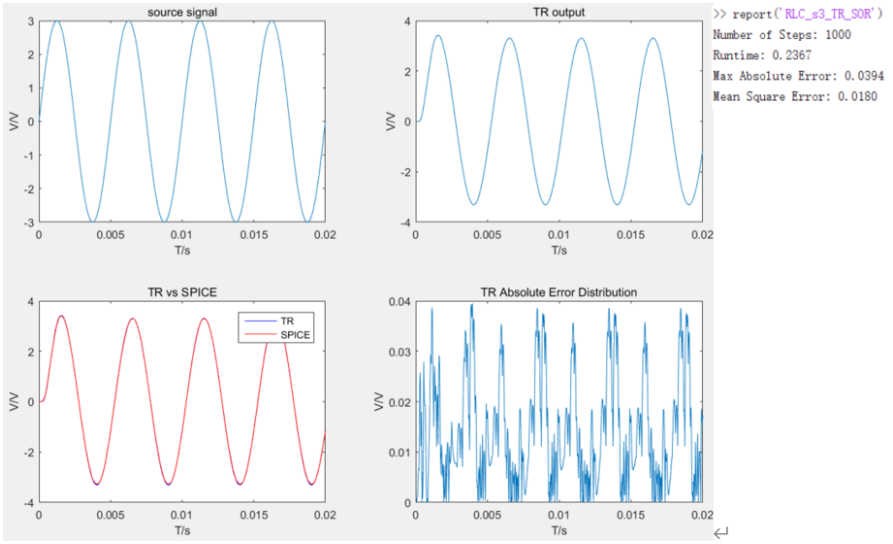

图 5: RLC_s3_TR_SOR 实验结果

2.5.1.2 RLC_p3(脉冲激励信号)

仿真设置: 起始时间 0s, 终止时间 0.02s, 采样点数 1000, 误差容限 $1e^{-9}$, 下面展示了几种算法得到的仿真结果:

后向欧拉 (BE)+GCR:

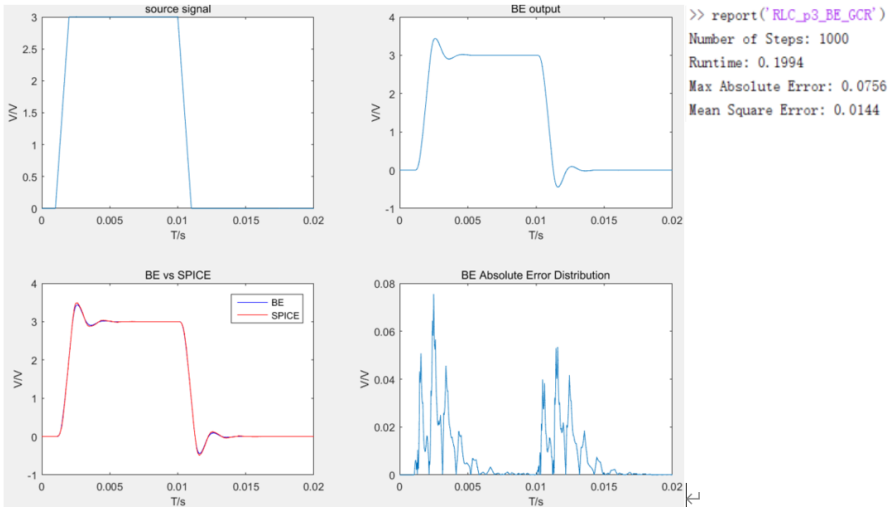

图 6: RLC_p3_BE_GCR 实验结果

梯形法 (TR)+GCR:

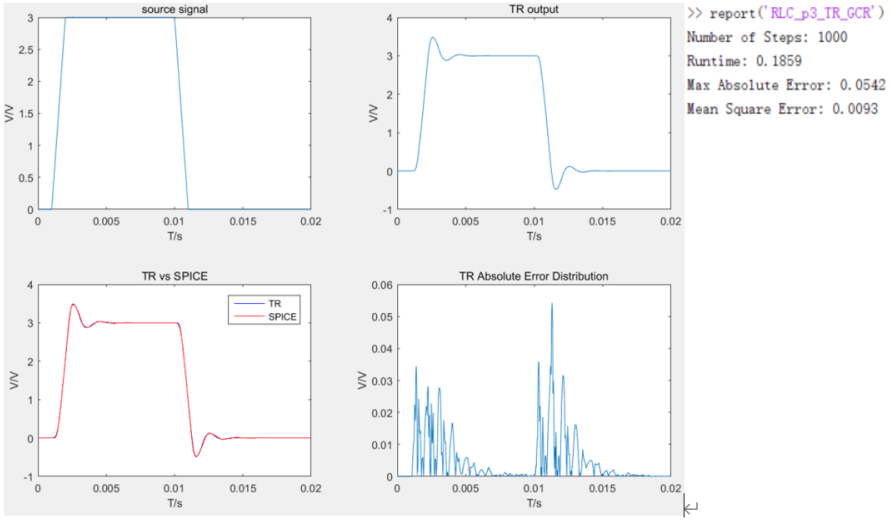

图 7: RLC_p3_TR_GCR 实验结果

后向欧拉 (BE)+SOR:

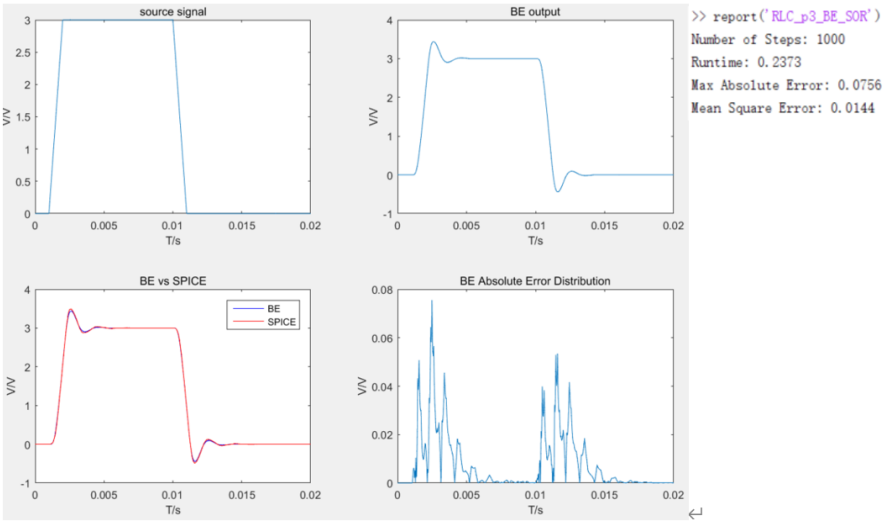

图 8: RLC_p3_BE_SOR 实验结果

梯形法 (TR)+SOR:

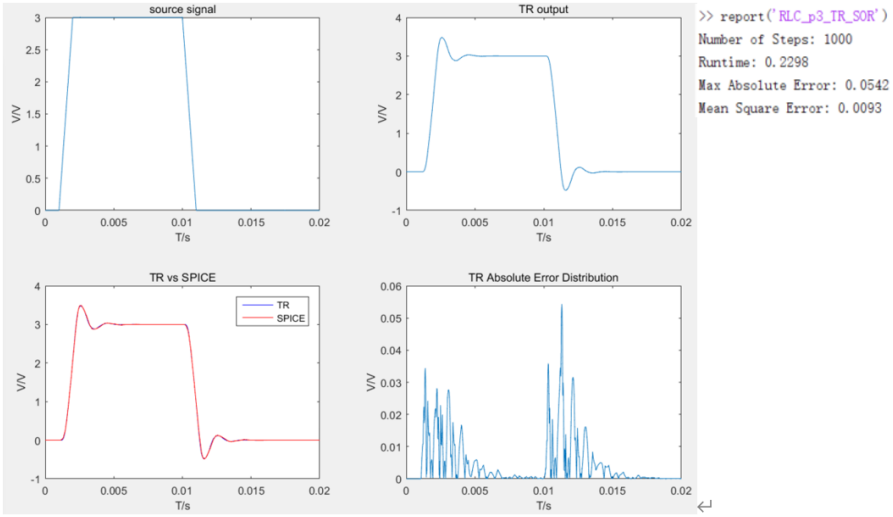

图 9: RLC_p3_TR_SOR 实验结果

2.5.1.3 bus8bit8seg(脉冲激励信号)

仿真设置: 起始时间 0s, 终止时间 $1e^{-9}$s, 采样点数 1000, 误差容限 $1e^{-9}$, 下面展示了几种算法得到的仿真结果:

后向欧拉 (BE)+GCR:

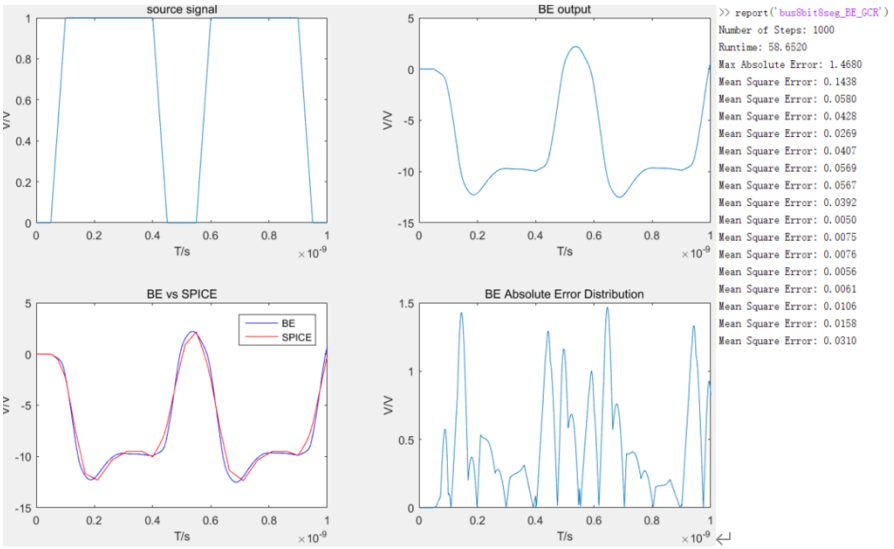

图 10: bus8bit8seg_BE_GCR 实验结果

梯形法 (TR)+GCR:

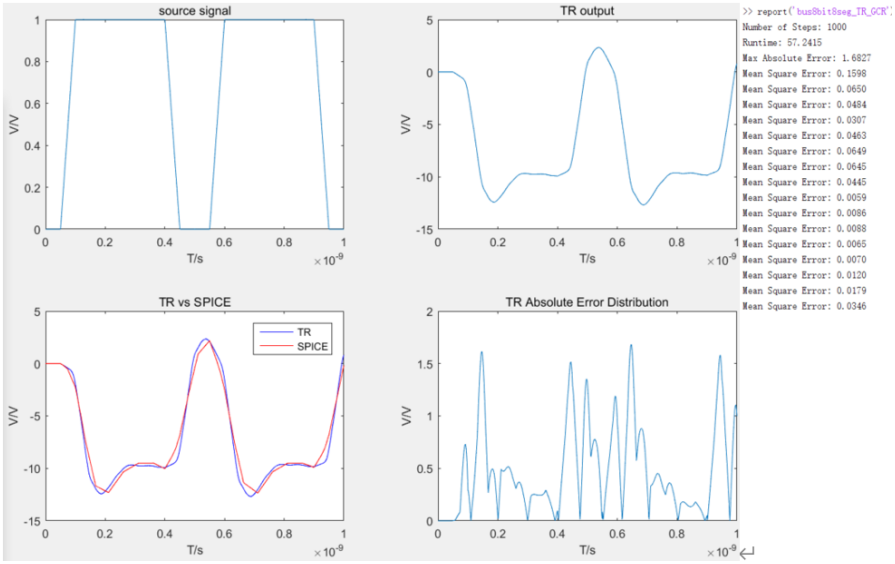

图 11: bus8bit8seg_TR_GCR 实验结果

2.5.1.4 bus32seg16(脉冲激励信号)

仿真设置: 起始时间 0s, 终止时间 $1e^{-9}$s, 采样点数 1000, 误差容限 $1e^{-9}$, 下面展示了几种算法得到的仿真结果:

后向欧拉 (BE)+GCR:

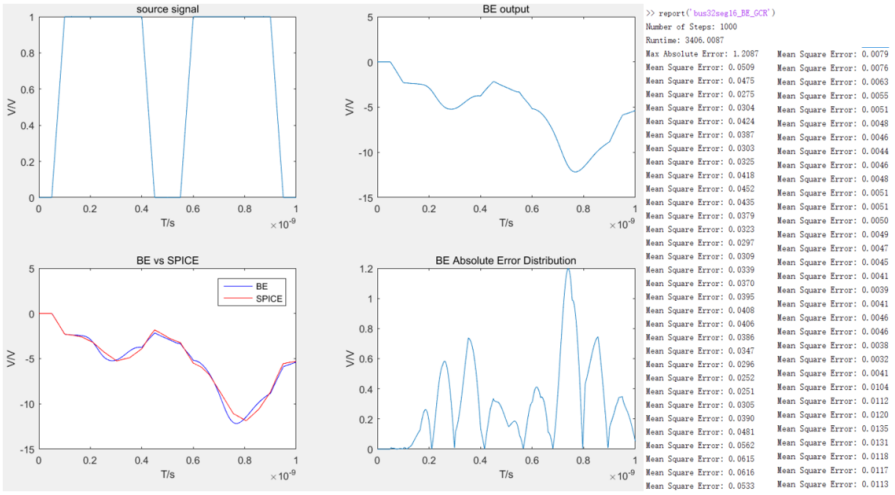

图 12: bus32seg16_BE_GCR 实验结果

梯形法 (TR)+GCR:

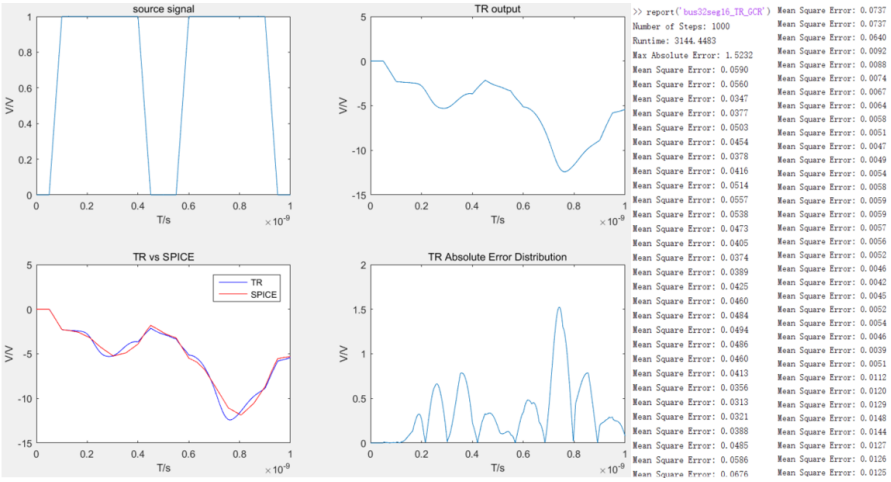

图 13: bus32seg16_TR_GCR 实验结果

2.5.2 结果分析

从上述结果中可以看出在给定的电路已经仿真条件下使用后向欧拉法和梯形法得到的误差相差不大, 运行时间上也没有显著的差异, 分析原因可能是电路本身都是稳定的, 同时由于实现过程中 Matlab 底层对矩阵不同运算处理的优化上不同。此外 SOR 和 GCR 两种求解线性方程组的方法在 RLC 的例子上表现相当, 但是 SOR 的收敛轮数会大于 GCR 的收敛轮数, 此外还尝试使用了 matlab 自带的线性方程求解方法 (
, 于 golden.m 文件中), 得到的结果与 GCR,SOR 相近, 同时运行时间明显更快, 可见 matlab 自带的线性方程求解器的高效。

2.6 文件说明

src 目录中的文件结构

- main.m: 包含整个 MNA 方程的求解框架和流程。
- BackwardEuler.m: 实现后向欧拉求解法。
- TrapezoidalRule.m: 实现梯形法。
- GCR.m: 实现 GCR 算法。
- SOR.m: 实现 SOR 算法。
- Golden.m: 使用 matlab 自带线性方程求解器进行求解。

- GenerateSrc.m: 生成激励信号。
- report.m: 分析数据并绘制对比图。