Table of Contents

[ELEC 4700 Assignment 4 - Circuit Modeling 2](#_Toc100320062)

[Q1 2](#_Toc100320063)

[Q2 8](#_Toc100320064)

[Q3 8](#_Toc100320065)

[Q3 a) Formulation 9](#_Toc100320066)

[Q3 b) Programing i. 9](#_Toc100320067)

[Q3 b) ii. 10](#_Toc100320068)

[Q3 b) iii. 12](#_Toc100320069)

[Q4 Transient Circuit Simulation 14](#_Toc100320070)

[Q4 a) 15](#_Toc100320071)

[Q4 b) 15](#_Toc100320072)

[Q4 c) 15](#_Toc100320073)

[Q4 d) 15](#_Toc100320074)

[Q4 d) Input signal A: 16](#_Toc100320075)

[Q4 d) Input signal B: 18](#_Toc100320076)

[Q4 d) Input signal B (few other frequencies): 19](#_Toc100320077)

[Q4 d) Input signal C 22](#_Toc100320078)

[Q4 d) v 24](#_Toc100320079)

[Q4 d) v Comment 26](#_Toc100320080)

[Q5 Circuit with Noise 26](#_Toc100320081)

[Q5 vi 28](#_Toc100320082)

[Q5 vi. Comment 32](#_Toc100320083)

[Q5 vii 32](#_Toc100320084)

[Q5 vii Comment 35](#_Toc100320085)

[Q6 Non-linearity 35](#_Toc100320086)

[Q6 Implementation 36](#_Toc100320087)

[Helper functions 39](#_Toc100320088)

## ELEC 4700 Assignment 4 - Circuit Modeling

Student: Samuel (Wendi) Zhu

Student Number: 101088968

Date: 4/7/2022

% Clear all  
clear  
clearvars  
clearvars -global  
close all  
format shorte  
  
% Make plot pretier  
% set(0,'DefaultFigureWindowStyle','docked')  
set(0,'defaultaxesfontsize',14)  
set(0,'defaultaxesfontname','Times New Roman')  
set(0,'DefaultLineLineWidth', 1);

## Q1

Using a fixed bottled neck value and the simulation code for assignment 3, do a voltage sweep of the device from 0.1V to 10V and model the current-voltage characteristics.

% Global variables  
global Const % constants module that holds all the constants  
global x y % arrays for the current electrons positions: 1 row, colum for current position  
global xp yp % arrays for the previous electrons positions: 1 row, column for previous position  
global vx vy % arrays for current electrons velocities: 1 row, column for current velocity  
global ax ay % scalars for electron acceleration in x and y direction  
global limits % Limits for the plot  
global boxes; % matrix for the boxes: n rows, and 4 columns for [x y w h]  
% Initalize global constants  
% Electron charge  
Const.q\_0 = 1.60217653e-19; % C  
% Rest mass  
Const.m0 = 9.1093837015e-31; % KG  
% Effective mass of electrons  
Const.mn = 0.26\*Const.m0; % KG  
% Boltzmann constant  
Const.kb = 1.38064852e-23; %m^2 \* kg \* s^-2 \* K^-1  
  
% Initialize the region size 200nm X 100nm  
Region.x = 200e-9;  
Region.y = 100e-9;  
limits = [0 Region.x 0 Region.y]; % plot limit  
% Initialize the temperature  
T = 300; % K  
% Initialize the mean time between collision  
Tmn = 0.2e-12; % 0.2ps  
% Define the dimension  
L = Region.x \* 10^9; % Length in nm  
W = Region.y \* 10^9; % Width in nm  
boxLF = 0.3; % Fraction of the length of the box  
boxWF = 0.4; % Fraction of the width of the box  
Lb = boxLF\*L; % Length of the box in nm  
Wb = boxWF\*W; % Width of the box in nm  
deltaXY = 0.02\*L; % Assume deltaX = deltaY in nm  
% Calculate the dimension of solution matrix  
nx = (L/deltaXY);  
ny = (W/deltaXY);  
  
% Number of sweep points for the voltage  
nSweep = 10;  
vecVoltage = linspace(0.1, 10, nSweep); % Generate the voltage vector  
vecCurrent = zeros(1, nSweep); % vector for holding the current  
% vector for different deltaT to adjust the time to reach steady state  
vecDeltaT = linspace(12e-14,3e-14, nSweep);  
  
% Loop through the different voltages  
for iVolt = 1:length(vecVoltage)  
 % Define the voltages  
 voltageX0 = vecVoltage(iVolt); % Voltage at X=0  
 voltageX1 = 0; % Voltage at X=L  
 % Define deltaT to reach steady state sooner  
 deltaT = vecDeltaT(iVolt);  
  
 % Step 1: Calculate the E field  
 % Calculate the meshgrid  
 [X,Y] = meshgrid(linspace(0,L,nx), linspace(0,W,ny));  
 % Declare the matrix for conductivity: Sigma(y,x)  
 matrixSigma = ones(ny, nx); % Dimension: ny times nx  
 xIndexBox = ceil((L-Lb)/(2\*deltaXY)); % Find the starting x index for the box  
 LbIndexRange = ceil(Lb/deltaXY); % Index range for the length of the box  
 WbIndexRange = ceil(Wb/deltaXY); % Index range for the width of the box  
 % Assign the region for the box  
 matrixSigma(1:WbIndexRange, xIndexBox:xIndexBox+LbIndexRange) = 10^-2;  
 matrixSigma(ny-WbIndexRange:ny, xIndexBox:xIndexBox+LbIndexRange) = 10^-2;  
 % Declare the matrix for voltage V(y,x)  
 matrixV = zeros(ny, nx); % Dimension: ny times nx  
 % Declare the G matrix and F vector: GV = F  
 G = zeros(nx\*ny, nx\*ny);  
 F = zeros(nx\*ny, 1);  
 % Construct the G matrix and F vector  
 for ix = 1:nx  
 for iy = 1:ny  
 % Calculate the index  
 n = mappingEq(ix, iy, ny);  
 % Check for the boundary  
 if ix==1 || ix==nx || iy ==1 || iy==ny  
 G(n,n) = 1;  
 % Boundary condition for x  
 if ix == 1  
 F(n,1) = voltageX0; % V at x = 0  
 elseif ix == nx  
 F(n,1) = voltageX1; % and V at x = L  
 elseif iy == 1  
 nyp = mappingEq(ix, iy+1, ny); % dV/dy=0 at y=0  
 G(n,nyp) = -1;  
 elseif iy == ny  
 nym = mappingEq(ix, iy-1, ny); % dV/dy=0 at y=W  
 G(n, nym) = -1;  
 end  
 else  
 % Calculate the sigma  
 sigmaxp = (matrixSigma(iy,ix) + matrixSigma(iy,ix+1))/2;  
 sigmaxm = (matrixSigma(iy,ix) + matrixSigma(iy, ix-1))/2;  
 sigmayp = (matrixSigma(iy,ix) + matrixSigma(iy+1, ix))/2;  
 sigmaym = (matrixSigma(iy,ix) + matrixSigma(iy-1, ix))/2;  
 % Calculate mapping index  
 nxp = mappingEq(ix+1, iy, ny); % index for V(i+1,j)  
 nxm = mappingEq(ix-1, iy, ny); % index for V(i-1,j)  
 nyp = mappingEq(ix, iy+1, ny); % index for V(i,j+1)  
 nym = mappingEq(ix, iy-1, ny); % index for V(i,j-1)  
 % Setup the G matrix  
 G(n,n) = -(sigmaxp+sigmaxm+sigmayp+sigmaym)/deltaXY^2;  
 G(n, nxp) = sigmaxp/deltaXY^2;  
 G(n, nxm) = sigmaxm/deltaXY^2;  
 G(n, nyp) = sigmayp/deltaXY^2;  
 G(n, nym) = sigmaym/deltaXY^2;  
 end  
 end  
 end  
 % Solve for V from GV = F  
 V = G\F;  
 % Map back to the 2D region  
 for iMap = 1:nx\*ny  
 % Calculate the index for the 2D region  
 ix = ceil(iMap/ny);  
 iy = mod(iMap, ny);  
 if iy == 0  
 iy = ny;  
 end  
 % Assign the value  
 matrixV(iy, ix) = V(iMap);  
 end  
 % Solve the electric field  
 [Ex, Ey] = gradient(-matrixV);  
 Ex = Ex/(deltaXY \* 10^-9); % convert to V/m  
 Ey = Ey/(deltaXY \* 10^-9); % convert to V/m  
  
 % Step 2: Calculate the acceleration field  
 % Initialize the number of "super" electrons  
 numE = 1000;  
 % Number of simulation steps  
 numSim = 1000;  
 % Boudary mode: specular(0) or diffusive(1)  
 boundaryMode = 0;  
 % Add the boxes  
 numBox = AddObstacles(boxLF, boxWF, Region);  
 % To find the current, the following steps are performed:  
 % 1) Calculate the total area  
 areaA = Region.x \* Region.y; % m^2  
 areaA = areaA \* 100^2; % cm^2  
 % 2) Calculate the total electrons in the area assuming electron  
 % concentration is 10^15 cm-2  
 totalE = 10^15 \* areaA; % total electrons  
 % 3) Find the charge per "Super Electron", where "Super Electron" is the  
 % particle in this simulation  
 numEPerSuperE = totalE/numE; % number of electron per super electron  
 superECharge = -Const.q\_0 \* numEPerSuperE; % Charge per super electron  
 % 4) The current can be found by counting the net number of super electrons  
  
 % Initialize acceleration for each electron  
 ax = zeros(1, numE); % Acceleration in x  
 ay = zeros(1, numE); % Acceleration in y  
 % Calculate the acceleration field: a = Force/mass = q\*E/mass  
 accFieldX = -Const.q\_0 \* Ex / (Const.mn);  
 accFieldY = -Const.q\_0 \* Ey / (Const.mn);  
  
 % Add the electrons  
 AddElectrons\_WithBox(numE, Region, T, numBox);  
 % Calculate the scattering probability  
 Pscat = 1-exp(-deltaT/Tmn);  
  
 % Super electron count for current calculation  
 % Count on left side x=0. +1 flow right, -1 flow left  
 countECurrent = 0; % Hold the super electron count  
  
 % Step 3: Loop for simulation  
 for iSim = 1:numSim  
 % Store the current positions  
 xp = x;  
 yp = y;  
 % Calculate the future positions: x = x0 + vx\*t + 1/2\*ax\*t^2  
 x = x + vx \* deltaT + 1/2 \* ax \*deltaT^2;  
 y = y + vy \* deltaT + 1/2 \* ay \* deltaT^2;  
 % Calculate the future velocity: vx = ax\*t  
 vx = vx + ax\*deltaT;  
 vy = vy + ay\*deltaT;  
  
 % Reset the super electron count  
 countECurrent = 0;  
  
 % Loop through all the particles  
 for iE=1:numE  
 % flag for invalid position  
 bInvalid = false;  
 % Step 1 - Check for boundary  
 % Check for invalid x position  
 if x(iE) <= 0  
 x(iE) = Region.x; % Appear on right  
 xp(iE) = x(iE);  
 bInvalid = true;  
 % Update the electron count for current calculation  
 countECurrent = countECurrent-1; % -1 flow left  
 elseif x(iE) >= Region.x  
 x(iE) = 0; % Appear on left  
 xp(iE) = x(iE);  
 bInvalid = true;  
 % Update the electron count for current calculation  
 countECurrent = countECurrent+1; % +1 flow right  
 end  
 % Check for invalid y position  
 if y(iE) <= 0  
 bInvalid = true;  
 y(iE) = 0;  
 % Check for boundary mode  
 if boundaryMode == 0 % Specular boundary  
 vy(iE) = -vy(iE);  
 else % Diffusive boundary  
 vy(iE) = abs(sqrt(Const.kb\*T/Const.mn).\*randn()); % positive vy  
 end  
 elseif y(iE) >= Region.y  
 y(iE) = Region.y;  
 bInvalid = true;  
 % Check for boundary mode  
 if boundaryMode == 0 % Specular boundary  
 vy(iE) = -vy(iE);  
 else % Diffusive boundary  
 vy(iE) = -abs(sqrt(Const.kb\*T/Const.mn).\*randn()); % negative vy  
 end  
 end  
 % Step 2: Check for boxes  
 for iBox = 1:numBox  
 % Retrieve box info  
 boxX1 = boxes(iBox, 1);  
 boxX2 = boxes(iBox, 1)+boxes(iBox, 3);  
 boxY1 = boxes(iBox, 2);  
 boxY2 = boxes(iBox, 2)+boxes(iBox, 4);  
 % Check if the particle is inside a box  
 if (x(iE)>=boxX1 && x(iE)<=boxX2 && y(iE)>=boxY1 && y(iE) <= boxY2)  
 bInvalid = true; %Invalid position  
 % Check for x position  
 if xp(iE) <= boxX1 % Coming from left side  
 x(iE) = boxX1;  
 % Check for boundary mode  
 if boundaryMode == 0 % Specular boundary  
 vx(iE) = -vx(iE);  
 else % Diffusive boundary  
 vx(iE) = -abs(sqrt(Const.kb\*T/Const.mn).\*randn()); % negative vx  
 end  
 elseif xp(iE) >= boxX2 % Coming from right side  
 x(iE) = boxX2;  
 % Check for boundary mode  
 if boundaryMode == 0 % Specular boundary  
 vx(iE) = -vx(iE);  
 else % Diffusive boundary  
 vx(iE) = abs(sqrt(Const.kb\*T/Const.mn).\*randn()); % positive vx  
 end  
 end  
 % Check for y position  
 if yp(iE) <= boxY1 % Coming from bottom  
 y(iE) = boxY1;  
 % Check for boundary mode  
 if boundaryMode == 0 % Specular boundary  
 vy(iE) = -vy(iE);  
 else % Diffusive boundary  
 vy(iE) = -abs(sqrt(Const.kb\*T/Const.mn).\*randn()); % negative vy  
 end  
 elseif yp(iE) >= boxY2 % Coming from top  
 y(iE) = boxY2;  
 % Check for boundary mode  
 if boundaryMode == 0 % Specular boundary  
 vy(iE) = -vy(iE);  
 else % Diffusive boundary  
 vy(iE) = abs(sqrt(Const.kb\*T/Const.mn).\*randn()); % positive vy  
 end  
 end  
 % Break the loop for box  
 break;  
 end  
 end  
 % Step 3: Check for scattering  
 if ~bInvalid && Pscat > rand()  
 % Rethermalize  
 vx(iE) = sqrt(Const.kb\*T/Const.mn).\*randn();  
 vy(iE) = sqrt(Const.kb\*T/Const.mn).\*randn();  
 end  
 % Step 4: Find acceleration  
 % Find the corresponding index for the acceleration field  
 indexX = ceil(x(iE)/(deltaXY\*10^-9));  
 indexY = ceil(y(iE)/(deltaXY\*10^-9));  
 % Check for invalid index  
 if indexX <= 0  
 indexX = 1;  
 end  
 if indexY <= 0  
 indexY = 1;  
 end  
 % Assign the acceleration of the electron  
 ax(iE) = accFieldX(indexX);  
 ay(iE) = accFieldY(indexY);  
 end  
 end  
  
 % Calculate the current  
 vecCurrent(iVolt) = superECharge\*countECurrent/deltaT;  
  
end  
  
% Plot the current versus voltage characteristics  
figure(1)  
plot(vecVoltage, vecCurrent, "-b.")  
title("Current - Voltage Characteristics")  
xlabel("Voltage (V)")  
ylabel("Current (A)")  
% Do the linear fit for determining resistance value of the device  
slopeRfit = vecVoltage(:)\vecCurrent(:);  
Rfit = vecVoltage(:)\*slopeRfit;  
hold on;  
plot(vecVoltage, Rfit, "r-.")  
legend("Simulation Data", "Linear Fit Line", "Location","southeast")  
grid on  
snapnow

Chart, line chart

Description automatically generated

Figure 1 Current vs Voltage Characteristics for determining resistance value of the device

## Q2

Use a linear fit to determine the resistance value pf the device and use this value as R3

% Calculate the R3 value  
R3 = 1/slopeRfit; % Ohm  
% Print the linear fitted value  
fprintf("Resistance value from linear fit: R3 = " + R3 + " Ohms.\n")

Resistance value from linear fit: R3 = 11.6555 Ohms.

## Q3

Report on the work done in PA 9 (PA 7).

% Declare some component values  
R1 = 1; % Ohm  
C = 0.25; % F  
R2 = 2; % Ohm  
L = 0.2; % H  
alpha = 100;  
R4 = 0.1; % Ohm  
RO=1000; % Ohm  
  
% Declare the vectors  
vectorV = zeros(9, 1); % solution vector: [N1, N2, N3, N4, N5, I1, IL, I3, I4]  
vectorF = zeros(9, 1); % F vector: F(1) = VIN

## Q3 a) Formulation

Please see the Appendix A for the differential equations that represent the network and the derivation for the C and G matrix.

% Create the C, G matrices  
% Declare the C matrix  
matrixC = [0, 0, 0, 0, 0, 0, 0, 0, 0;  
 C, -C, 0, 0, 0, 0, 0, 0, 0;  
 -C, C, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, -L, 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, 0;  
 0, 0, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, 0, 0, 0];  
  
% Declare the G matrix  
matrixG = [1, 0, 0, 0, 0, 0, 0, 0, 0;  
 1/R1, -1/R1, 0, 0, 0, 1, 0, 0, 0;  
 -1/R1, 1/R1+1/R2, 0, 0, 0, 0, 1, 0, 0;  
 0, 1, -1, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, -1, 1, 0;  
 0, 0, -1/R3, 0, 0, 0, 0, 1, 0;  
 0, 0, 0, 1/R4, -1/R4, 0, 0, 0, 1;  
 0, 0, 0, 1, 0, 0, 0, -alpha, 0;  
 0, 0, 0, -1/R4, 1/R4+1/RO, 0, 0, 0, 0];

## Q3 b) Programing i.

DC sweep input voltage from -10V to 10V and plot V0 and V3 (N3).

simStep = 21; % Simulation steps  
V1 = linspace(-10, 10, simStep); % vector for input voltages  
Vo = zeros(simStep, 1); % vector for holding the output voltage  
V3 = zeros(simStep, 1); % vector for holding the voltage at V3  
% Loop for the DC simulation  
for iSim = 1:simStep  
 % Setup the F vector  
 vectorF(1) = V1(iSim); % Stepup the input voltage  
 % Find the solution  
 vectorV = matrixG\vectorF;  
 % Save answers  
 Vo(iSim) = vectorV(5); % Save Vout  
 V3(iSim) = vectorV(3); % Save V3  
end  
% Plot the DC simulation  
figure(2)  
plot(V1, Vo, "-b.")  
hold on  
plot(V1, V3, "-r.")  
title("DC simulation")  
xlabel("Vin (V)")  
ylabel("Vout and V3 (V)")  
legend("Vout", "V3")  
grid on;  
snapnow

Chart, line chart, scatter chart

Description automatically generated

Figure 2 DC simulation for sweep the input voltage from -10 V to 10 V

## Q3 b) ii.

For the AC case, plot Vo as a function of omega also plot the gain vo/v1 in dB.

simSteps = 100; % Simulation steps  
Vin = 1; % Value for input voltage  
vectorF(1) = Vin; % Setup the input voltage  
omega = linspace(1, 100, simSteps); % vector for frequencies  
Vo = zeros(simSteps, 1); % vector store the output voltages  
V3 = zeros(simStep, 1); % vector for holding the voltage at V3  
  
% Loop for simulation  
for iSim = 1:simSteps  
 w = omega(iSim); % Retrieve the simulation frequency  
 % Construct the G+jwC matrix  
 matrixGC = matrixG + 1j\*w\*matrixC;  
 % Find the solution  
 vectorV = matrixGC\vectorF;  
 % Save answers  
 Vo(iSim) = abs(vectorV(5)); % Save Vout  
 V3(iSim) = abs(vectorV(3)); % Save V3  
end  
% Plot Vo as a function of omega  
figure(3)  
plot(omega, Vo, "-b.");  
hold on  
plot(omega, V3, "-r.");  
title("Vo as a function of omega")  
xlabel("Frequency omega (rad/s)")  
ylabel("Vout (V)")  
legend("Vout", "V3")  
grid on  
snapnow  
  
% Plot the gain Vo/V1 in dB  
figure(4)  
gain = 20.\*log10(Vo ./ Vin); % Calculate the gain in dB  
plot(omega, gain);  
title("Gain Vo/V1 in dB versus omega")  
xlabel("Frequency omega (rad/s)")  
ylabel("Gain (dB)")  
grid on  
snapnow

Chart, line chart

Description automatically generated

Figure 3 as a function of

Chart, line chart

Description automatically generated

Figure 4 Gain in dB versus

## Q3 b) iii.

For the AC case, plot the gain as function of random perturbations on C using a normal distribution with std=.05 at omega = pi. Do a histogram of the gain.

simSteps = 1000; % Simulation steps  
omega = pi;  
std = 0.05; % Standard deviation of the normal distribution  
randomC = std .\* randn(simSteps, 1)+C; % vector store the random C  
Vo = zeros(simSteps, 1); % vector store the output voltages  
Vin = 10; % Value for input voltage  
vectorF(1) = Vin; % Setup the input voltage  
  
% Plot the normal distribution of C  
nbins = 10; % Number of bins for the histogram  
figure(5)  
histogram(randomC, nbins);  
title("Distribution of C")  
xlabel("C (F)")  
ylabel("Number")  
grid on  
snapnow  
  
% Loop through the random C  
for iSim=1:simSteps  
 C = randomC(iSim); % Retrieve the C value  
 % Reconstruct the C matrix  
 matrixC = [0, 0, 0, 0, 0, 0, 0, 0, 0;  
 C, -C, 0, 0, 0, 0, 0, 0, 0;  
 -C, C, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, -L, 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, 0;  
 0, 0, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, 0, 0, 0];  
  
 % Construct the G+jwC matrix  
 matrixGC = matrixG + 1j\*omega\*matrixC;  
 % Find the solution  
 vectorV = matrixGC\vectorF;  
 % Save answers  
 Vo(iSim) = abs(vectorV(5)); % Save Vout  
end  
  
% Plot the distribution of gain  
figure(6)  
gain = Vo ./ Vin; % Calculate the gain  
histogram(gain, nbins);  
title("Distribution of Gain")  
xlabel("Vo/Vi (V/V)")  
ylabel("Number")  
grid on  
snapnow

Chart, histogram

Description automatically generated

Figure 5 Distribution of C

Chart, histogram

Description automatically generated

Figure 6 Histogram of the gain as random perturbations on C

## Q4 Transient Circuit Simulation

The circuit can be represented in the time domain as:

## Q4 a)

By inspection what type of circuit is this?

By inspection, the circuit is an LRC circuit, and the circuit models an amplifier. The circuit is linear since it does not contain any nonlinear components yet.

## Q4 b)

What sort of frequency response would you expect?

When the frequency is high, the impedance for L will be large, and the output voltage will be low. This suggest that the circuit has an overall low pass response. However, the simulation waveform of Gain versus omega shows that the gain is also relatively low at the near DC frequency. Therefore, strictly speaking, the frequency response of the circuit is bandpass response.

## Q4 c)

The derivation for the formulation in time domain is shown as follow:

Solving MNA Time Domain (implicit):

Let define , then

The derived equation is:

Where .

## Q4 d)

Write a Matlab program that can simulate the circuit. Simulate for 1s and use 1000 steps.

simTime = 1; % Simulate for 1 second  
simSteps = 1000; % Use 1000 steps  
% Calculate the deltaT  
deltaT = simTime/simSteps; % second/step  
% Declare the time vector  
vecTime = linspace(0,1,simSteps);

## Q4 d) Input signal A:

A step that transitions from 0 to 1 at 0.03s.

iii. Plot the input Vin and output V as the simulation progresses.

iv. After the simulation is completed, use the fft() and fftshift() functions to plot the frequency of the input and output signals.

% Declare the input for a step from 0 to 1 at 0.03s  
vecInputV = zeros(1, simSteps);  
vecInputV(0.03\*simSteps:simSteps) = 1;  
% Hold the output vector  
vecOutputV = zeros(1, simSteps);  
  
% Initialize the V and F vector  
vectorV = zeros(9, 1); % solution vector: [N1, N2, N3, N4, N5, I1, IL, I3, I4]  
vectorF = zeros(9, 1); % F vector: F(1) = VIN  
% Construct the A matrix  
matrixA = matrixC/deltaT + matrixG;  
  
% Loop through the simulation  
for iSim = 1:simSteps  
 % Update the F vector  
 vectorF(1) = vecInputV(iSim);  
 % Update the V vector  
 vectorV = matrixA^-1 \* (matrixC \* vectorV / deltaT + vectorF);  
 % Save the output voltage  
 vecOutputV(iSim) = vectorV(5);  
  
% % Plot the input Vin and output V as the simulation progresses  
% figure(7)  
% plot(vecTime(1:iSim), vecInputV(1:iSim), "-r.") % Vin versus time  
% hold on  
% plot(vecTime(1:iSim), vecOutputV(1:iSim), "-b.") % Vo versus time  
% hold off  
% title("Transient simulation for a step input")  
% xlabel("Time (s)")  
% ylabel("Voltage (V)")  
% legend("Vin versus time", "Vo versus time", "Location", "southeast")  
% grid on  
% % Pause for a while  
% pause(0.002)  
end  
  
% Plot of completed transient simulation for step input  
figure(7)  
% Time domain plot  
subplot(1,2,1)  
plot(vecTime, vecInputV, "-b.") % Vin versus time  
hold on  
plot(vecTime, vecOutputV, "-r.") % Vo versus time  
hold off  
title("Transient simulation for a step input")  
xlabel("Time (s)")  
ylabel("Voltage (V)")  
legend("Vin versus time", "Vo versus time")  
grid on  
% Frequency domain plot (fft)  
subplot(1,2,2)  
% Calculate sampling frequency  
Fs = 1/deltaT;  
df = Fs/length(vecInputV);  
vecFreqPlot = -Fs/2:df:Fs/2-df; % Create the frequency vector for plot  
vecOmega = 2\*pi\*vecFreqPlot; % Calculate the omega vector  
fftVin = 20\*log10(abs(fftshift(fft(vecInputV)))/simSteps); % Input fft in dB  
plot(vecOmega, fftVin, "-b.") % Plot the input fft  
hold on  
fftVo = 20\*log10(abs(fftshift(fft(vecOutputV)))/simSteps); % Output fft in dB  
plot(vecOmega, fftVo, "-r.") % Plot the output fft  
hold off  
title("FFT of the step input")  
xlabel("Omega (rad/s)")  
ylabel("V (dB)")  
legend("Vin versus time", "Vo versus time")  
grid on  
snapnow

Chart, histogram

Description automatically generated

Figure 7 Transient simulation and FFT for a step input

## Q4 d) Input signal B:

The input signal is a sin(2\*pi\*f\*t) function with f=1/0.03 Hz. Try few other frequencies.

iii. Plot the input Vin and output V as the simulation progresses.

iv. After the simulation is completed, use the fft() and fftshift() functions to plot the frequency of the input and output signals.

% Declare the input for a sin(2\*pi\*f\*t) with freq=1/0.03 Hz  
freq = 1/0.03; % Hz  
vecInputV = sin(2\*pi\*freq\*vecTime);  
% Hold the output vector  
vecOutputV = zeros(1, simSteps);  
  
% Initialize the V and F vector  
vectorV = zeros(9, 1); % solution vector: [N1, N2, N3, N4, N5, I1, IL, I3, I4]  
vectorF = zeros(9, 1); % F vector: F(1) = VIN  
% Construct the A matrix  
matrixA = matrixC/deltaT + matrixG;  
  
% Loop through the simulation  
for iSim = 1:simSteps  
 % Update the F vector  
 vectorF(1) = vecInputV(iSim);  
 % Update the V vector  
 vectorV = matrixA^-1 \* (matrixC \* vectorV / deltaT + vectorF);  
 % Save the output voltage  
 vecOutputV(iSim) = vectorV(5);  
end  
  
% Plot of completed transient simulation for sinusoidal input with f = 1/0.03 Hz  
figure(8)  
subplot(1,2,1)  
plot(vecTime, vecInputV, "-b.") % Vin versus time  
hold on  
plot(vecTime, vecOutputV, "-r.") % Vo versus time  
hold off  
title("Transient simulation for sinusoidal input with f = 1/0.03 Hz")  
xlabel("Time (s)")  
ylabel("Voltage (V)")  
legend("Vin versus time", "Vo versus time")  
grid on  
% Frequency domain plot (fft)  
subplot(1,2,2)  
% Calculate sampling frequency  
Fs = 1/deltaT;  
df = Fs/length(vecInputV);  
vecFreqPlot = -Fs/2:df:Fs/2-df; % Create the frequency vector for plot  
vecOmega = 2\*pi\*vecFreqPlot; % Calculate the omega vector  
fftVin = 20\*log10(abs(fftshift(fft(vecInputV)))/simSteps); % Input fft in dB  
plot(vecOmega, fftVin, "-b.") % Plot the input fft  
hold on  
fftVo = 20\*log10(abs(fftshift(fft(vecOutputV)))/simSteps); % Output fft in dB  
plot(vecOmega, fftVo, "-r.") % Plot the output fft  
hold off  
title("FFT of the sinusoidal input with f = 1/0.03 Hz")  
xlabel("Omega (rad/s)")  
ylabel("V (dB)")  
legend("Vin versus time", "Vo versus time")  
grid on  
snapnow

Chart, histogram

Description automatically generated

Figure 8 Transient simulation and the FFT for sinusoidal input with

## Q4 d) Input signal B (few other frequencies):

iii. Plot the input Vin and output V as the simulation progresses.

iv. After the simulation is completed, use the fft() and fftshift() functions to plot the frequency of the input and output signals.

% Declare the frequencies  
vecFreq = [1, 10, 35]; % Hz  
  
% Loop through the frequencies  
for iFreq = 1:length(vecFreq)  
 % Declare the input for a sin(2\*pi\*f\*t)  
 freq = vecFreq(iFreq); % Hz  
 vecInputV = sin(2\*pi\*freq\*vecTime);  
 % Hold the output vector  
 vecOutputV = zeros(1, simSteps);  
  
 % Initialize the V and F vector  
 vectorV = zeros(9, 1); % solution vector: [N1, N2, N3, N4, N5, I1, IL, I3, I4]  
 vectorF = zeros(9, 1); % F vector: F(1) = VIN  
 % Construct the A matrix  
 matrixA = matrixC/deltaT + matrixG;  
  
 % Loop through the simulation  
 for iSim = 1:simSteps  
 % Update the F vector  
 vectorF(1) = vecInputV(iSim);  
 % Update the V vector  
 vectorV = matrixA^-1 \* (matrixC \* vectorV / deltaT + vectorF);  
 % Save the output voltage  
 vecOutputV(iSim) = vectorV(5);  
 end  
  
 % Plot of completed transient simulation for sinusoidal input with f  
 figure(8+iFreq)  
 subplot(1,2,1)  
 plot(vecTime, vecInputV, "-b.") % Vin versus time  
 hold on  
 plot(vecTime, vecOutputV, "-r.") % Vo versus time  
 hold off  
 title("Transient simulation for sinusoidal input with f = "+freq+" Hz")  
 xlabel("Time (s)")  
 ylabel("Voltage (V)")  
 legend("Vin versus time", "Vo versus time")  
 grid on  
 % Frequency domain plot (fft)  
 subplot(1,2,2)  
 % Calculate sampling frequency  
 Fs = 1/deltaT;  
 df = Fs/length(vecInputV);  
 vecFreqPlot = -Fs/2:df:Fs/2-df; % Create the frequency vector for plot  
 vecOmega = 2\*pi\*vecFreqPlot; % Calculate the omega vector  
 fftVin = 20\*log10(abs(fftshift(fft(vecInputV)))/simSteps); % Input fft in dB  
 plot(vecOmega, fftVin, "-b.") % Plot the input fft  
 hold on  
 fftVo = 20\*log10(abs(fftshift(fft(vecOutputV)))/simSteps); % Output fft in dB  
 plot(vecOmega, fftVo, "-r.") % Plot the output fft  
 hold off  
 title("FFT of the sinusoidal input with f = "+freq+" Hz")  
 xlabel("Omega (rad/s)")  
 ylabel("V (dB)")  
 legend("Vin versus time", "Vo versus time")  
 grid on  
 snapnow  
end

Chart, line chart

Description automatically generated

Figure 9 Transient simulation and the FFT for sinusoidal input with

Chart

Description automatically generated

Figure 10 Transient simulation and the FFT for sinusoidal input with

Chart, histogram

Description automatically generated

Figure 11 Transient simulation and the FFT for sinusoidal input with

As the frequency increases, the transient simulation plot is more crowded, and the two spikes in the FFT plot is separated further. This is expected because the Fourier Transform of a sine function is two Direc Delta function as shown in follow:

Therefore, as the frequency increases, the two spiles in the FFT plot are separated further. It should also be noted that if the frequency is too large without increasing the sampling rate (decreasing ), there will be more error in the waveform.

## Q4 d) Input signal C

A gaussian pulse with a magnitude of 1, std dev. of 0.03s and a delay of 0.06s

iii. Plot the input Vin and output V as the simulation progresses.

iv. After the simulation is completed, use the fft() and fftshift() functions to plot the frequency of the input and output signals.

% Declare the input for a guassian pulse  
stddev = 0.03; % std dev. of 0.03s  
pulseDelay = 0.06; % delay pf 0.06s  
vecInputV = exp(-((vecTime-pulseDelay)/stddev).^2/2);  
% Hold the output vector  
vecOutputV = zeros(1, simSteps);  
  
% Initialize the V and F vector  
vectorV = zeros(9, 1); % solution vector: [N1, N2, N3, N4, N5, I1, IL, I3, I4]  
vectorF = zeros(9, 1); % F vector: F(1) = VIN  
% Construct the A matrix  
matrixA = matrixC/deltaT + matrixG;  
  
% Loop through the simulation  
for iSim = 1:simSteps  
 % Update the F vector  
 vectorF(1) = vecInputV(iSim);  
 % Update the V vector  
 vectorV = matrixA^-1 \* (matrixC \* vectorV / deltaT + vectorF);  
 % Save the output voltage  
 vecOutputV(iSim) = vectorV(5);  
end  
  
% Plot of completed transient simulation for a gaussian pulse  
figure(12)  
subplot(1,2,1)  
plot(vecTime, vecInputV, "-b.") % Vin versus time  
hold on  
plot(vecTime, vecOutputV, "-r.") % Vo versus time  
hold off  
title("Transient simulation for Gaussian pulse input")  
xlabel("Time (s)")  
ylabel("Voltage (V)")  
legend("Vin versus time", "Vo versus time")  
grid on  
% Frequency domain plot (fft)  
subplot(1,2,2)  
% Calculate sampling frequency  
Fs = 1/deltaT;  
df = Fs/length(vecInputV);  
vecFreqPlot = -Fs/2:df:Fs/2-df; % Create the frequency vector for plot  
vecOmega = 2\*pi\*vecFreqPlot; % Calculate the omega vector  
fftVin = 20\*log10(abs(fftshift(fft(vecInputV)))/simSteps); % Input fft in dB  
plot(vecOmega,fftVin, "-b.") % Plot the input fft  
hold on  
fftVo = 20\*log10(abs(fftshift(fft(vecOutputV)))/simSteps); % Output fft in dB  
plot(vecOmega,fftVo, "-r.") % Plot the output fft  
hold off  
title("FFT of Gaussian pulse input")  
xlabel("Omega (rad/s)")  
ylabel("V (dB)")  
legend("Vin versus time", "Vo versus time")  
grid on  
snapnow

Chart, line chart

Description automatically generated

Figure 12 Transient simulation and the FFT for Gaussian pulse input

## Q4 d) v

Increase the time step and see what happens. Comment.

The simulation steps are decreased to increase the time step ().

simTime = 1; % Simulate for 1 second  
simSteps = 100; % Decrease the simulation steps to 100 steps  
% Calculate the deltaT  
deltaT = simTime/simSteps; % second/step  
% Declare the time vector  
vecTime = linspace(0,1,simSteps);  
  
% Declare the input for a guassian pulse  
stddev = 0.03; % std dev. of 0.03s  
pulseDelay = 0.06; % delay pf 0.06s  
vecInputV = exp(-((vecTime-pulseDelay)/stddev).^2/2);  
% Hold the output vector  
vecOutputV = zeros(1, simSteps);  
  
% Initialize the V and F vector  
vectorV = zeros(9, 1); % solution vector: [N1, N2, N3, N4, N5, I1, IL, I3, I4]  
vectorF = zeros(9, 1); % F vector: F(1) = VIN  
% Construct the A matrix  
matrixA = matrixC/deltaT + matrixG;  
  
% Loop through the simulation  
for iSim = 1:simSteps  
 % Update the F vector  
 vectorF(1) = vecInputV(iSim);  
 % Update the V vector  
 vectorV = matrixA^-1 \* (matrixC \* vectorV / deltaT + vectorF);  
 % Save the output voltage  
 vecOutputV(iSim) = vectorV(5);  
end  
  
% Plot of completed transient simulation for step input  
figure(13)  
% Time domain plot  
subplot(1,2,1)  
plot(vecTime, vecInputV, "-b.") % Vin versus time  
hold on  
plot(vecTime, vecOutputV, "-r.") % Vo versus time  
hold off  
title("Transient simulation for a Guassian input for "+ simSteps + " steps")  
xlabel("Time (s)")  
ylabel("Voltage (V)")  
legend("Vin versus time", "Vo versus time")  
grid on  
% Frequency domain plot (fft)  
subplot(1,2,2)  
% Calculate sampling frequency  
Fs = 1/deltaT;  
df = Fs/length(vecInputV);  
vecFreqPlot = -Fs/2:df:Fs/2-df; % Create the frequency vector for plot  
vecOmega = 2\*pi\*vecFreqPlot; % Calculate the omega vector  
fftVin = 20\*log10(abs(fftshift(fft(vecInputV)))/simSteps); % Input fft in dB  
plot(vecOmega,fftVin, "-b.") % Plot the input fft  
hold on  
fftVo = 20\*log10(abs(fftshift(fft(vecOutputV)))/simSteps); % Output fft in dB  
plot(vecOmega,fftVo, "-r.") % Plot the output fft  
hold off  
title("FFT of the Guassian input for "+simSteps+" steps")  
xlabel("Omega (rad/s)")  
ylabel("V (dB)")  
legend("Vin versus time", "Vo versus time")  
grid on  
snapnow

Chart, line chart, histogram

Description automatically generated

Figure 13 Transient simulation and the FFT for Gaussian pulse input with 100 simulation steps (increased )

## Q4 d) v Comment

After increasing the time steps (decreases simulation steps), the time domain waveform contains less points, and the frequency domain plot contains less detail. This is because the sampling rate is lower.

## Q5 Circuit with Noise

In Q5, a current source in parallel with R3 is added to model the thermal noise generated in the resistor R3. A capacitor Cn in parallel with the resistor is also added to bandwidth limit the noise.

Please see Appendix B for the derivation and formulation of the matrices and equations.

simTime = 1; % Simulate for 1 second  
simSteps = 1000; % Use 1000 steps  
% Calculate the deltaT  
deltaT = simTime/simSteps; % second/step  
% Declare the time vector  
vecTime = linspace(0,1,simSteps);  
  
% Declare the capacitor Cn  
Cn = 0.00001;  
magIn = 0.001; % Magnitude for In  
  
% Declare the input for a guassian pulse  
stddev = 0.03; % std dev. of 0.03s  
pulseDelay = 0.06; % delay pf 0.06s  
vecInputV = exp(-((vecTime-pulseDelay)/stddev).^2/2);  
% Hold the output vector  
vecOutputV = zeros(1, simSteps);  
  
% Generate the vector for noise current  
vecIn = magIn\*randn(1, simSteps);  
  
% Declare the vectors  
vectorV = zeros(10, 1); % solution vector: [N1, N2, N3, N4, N5, I1, IL, I3, I4, In]  
vectorF = zeros(10, 1); % F vector: F(1) = VIN, F(10) = In  
  
% Declare the C matrix  
matrixC = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0;  
 C, -C, 0, 0, 0, 0, 0, 0, 0, 0;  
 -C, C, 0, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, -L, 0, 0, 0;  
 0, 0, Cn, 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, 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, 0];  
  
% Declare the G matrix  
matrixG = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0;  
 1/R1, -1/R1, 0, 0, 0, 1, 0, 0, 0, 0;  
 -1/R1, 1/R1+1/R2, 0, 0, 0, 0, 1, 0, 0, 0;  
 0, 1, -1, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, -1, 1, 0, 1;  
 0, 0, -1/R3, 0, 0, 0, 0, 1, 0, 0;  
 0, 0, 0, 1/R4, -1/R4, 0, 0, 0, 1, 0;  
 0, 0, 0, 1, 0, 0, 0, -alpha, 0, 0;  
 0, 0, 0, -1/R4, 1/R4+1/RO, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 1];  
  
% Construct the A matrix  
matrixA = matrixC/deltaT + matrixG;  
  
% Loop through the simulation  
for iSim = 1:simSteps  
 % Update the F vector for Vin and In  
 vectorF(1) = vecInputV(iSim);  
 vectorF(10) = vecIn(iSim);  
 % Update the V vector  
 vectorV = matrixA^-1 \* (matrixC \* vectorV / deltaT + vectorF);  
 % Save the output voltage  
 vecOutputV(iSim) = vectorV(5);  
end  
  
% Plot of completed transient simulation for a Gaussian input  
figure(14)  
% Time domain plot  
subplot(1,2,1)  
plot(vecTime, vecInputV, "-b.") % Vin versus time  
hold on  
plot(vecTime, vecOutputV, "-r.") % Vo versus time  
hold off  
title("Transient simulation for a Gaussian input with noise")  
xlabel("Time (s)")  
ylabel("Voltage (V)")  
legend("Vin versus time", "Vo versus time")  
grid on  
% Frequency domain plot (fft)  
subplot(1,2,2)  
% Calculate sampling frequency  
Fs = 1/deltaT;  
df = Fs/length(vecInputV);  
vecFreqPlot = -Fs/2:df:Fs/2-df; % Create the frequency vector for plot  
vecOmega = 2\*pi\*vecFreqPlot; % Calculate the omega vector  
fftVin = 20\*log10(abs(fftshift(fft(vecInputV)))/simSteps); % Input fft in dB  
plot(vecOmega, fftVin, "-b.") % Plot the input fft  
hold on  
fftVo = 20\*log10(abs(fftshift(fft(vecOutputV)))/simSteps); % Output fft in dB  
plot(vecOmega, fftVo, "-r.") % Plot the output fft  
hold off  
title("FFT of the Gaussian input with noise")  
xlabel("Omega (rad/s)")  
ylabel("V (dB)")  
legend("Vin versus time", "Vo versus time")  
grid on  
snapnow

Chart, histogram

Description automatically generated

Figure 14 Transient simulation and the FFT for Gaussian pulse input with noise

## Q5 vi

Vary Cn to see how the bandwidth changes. Comment on your results

% Create an array for three different Cn  
arrCn = [0.000002, 0.0002, 0.02];  
  
% Simulation loop for different Cn  
for iCn = 1:length(arrCn)  
 % Retrieve the corresponding Cn  
 Cn = arrCn(iCn);  
 magIn = 0.001; % Magnitude for In  
  
 % Declare the input for a guassian pulse  
 stddev = 0.03; % std dev. of 0.03s  
 pulseDelay = 0.06; % delay pf 0.06s  
 vecInputV = exp(-((vecTime-pulseDelay)/stddev).^2/2);  
 % Generate the vector for noise current  
 vecIn = magIn\*randn(1, simSteps);  
 % Hold the output vector  
 vecOutputV = zeros(1, simSteps);  
  
 % Declare the vectors  
 vectorV = zeros(10, 1); % solution vector: [N1, N2, N3, N4, N5, I1, IL, I3, I4, In]  
 vectorF = zeros(10, 1); % F vector: F(1) = VIN, F(10) = In  
  
 % Declare the C matrix  
 matrixC = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0;  
 C, -C, 0, 0, 0, 0, 0, 0, 0, 0;  
 -C, C, 0, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, -L, 0, 0, 0;  
 0, 0, Cn, 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, 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, 0];  
  
 % Declare the G matrix  
 matrixG = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0;  
 1/R1, -1/R1, 0, 0, 0, 1, 0, 0, 0, 0;  
 -1/R1, 1/R1+1/R2, 0, 0, 0, 0, 1, 0, 0, 0;  
 0, 1, -1, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, -1, 1, 0, 1;  
 0, 0, -1/R3, 0, 0, 0, 0, 1, 0, 0;  
 0, 0, 0, 1/R4, -1/R4, 0, 0, 0, 1, 0;  
 0, 0, 0, 1, 0, 0, 0, -alpha, 0, 0;  
 0, 0, 0, -1/R4, 1/R4+1/RO, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 1];  
  
 % Construct the A matrix  
 matrixA = matrixC/deltaT + matrixG;  
  
 % Loop through the simulation  
 for iSim = 1:simSteps  
 % Update the F vector for Vin and In  
 vectorF(1) = vecInputV(iSim);  
 vectorF(10) = vecIn(iSim);  
 % Update the V vector  
 vectorV = matrixA^-1 \* (matrixC \* vectorV / deltaT + vectorF);  
 % Save the output voltage  
 vecOutputV(iSim) = vectorV(5);  
 end  
  
 % Plot of completed transient simulation for step input  
 figure(14+iCn)  
 % Time domain plot  
 subplot(1,2,1)  
 plot(vecTime, vecInputV, "-b.") % Vin versus time  
 hold on  
 plot(vecTime, vecOutputV, "-r.") % Vo versus time  
 hold off  
 title("Transient simulation for a Gaussian input with noise for Cn = "+Cn+" F")  
 xlabel("Time (s)")  
 ylabel("Voltage (V)")  
 legend("Vin versus time", "Vo versus time")  
 grid on  
 % Frequency domain plot (fft)  
 subplot(1,2,2)  
 % Calculate sampling frequency  
 Fs = 1/deltaT;  
 df = Fs/length(vecInputV);  
 vecFreqPlot = -Fs/2:df:Fs/2-df; % Create the frequency vector for plot  
 vecOmega = 2\*pi\*vecFreqPlot; % Calculate the omega vector  
 fftVin = 20\*log10(abs(fftshift(fft(vecInputV)))/simSteps); % Input fft in dB  
 plot(vecOmega, fftVin, "-b.") % Plot the input fft  
 hold on  
 fftVo = 20\*log10(abs(fftshift(fft(vecOutputV)))/simSteps); % Output fft in dB  
 plot(vecOmega, fftVo, "-r.") % Plot the output fft  
 hold off  
 title("FFT of the Gaussian input with noise for Cn = "+Cn+" F")  
 xlabel("Omega (rad/s)")  
 ylabel("V (dB)")  
 legend("Vin versus time", "Vo versus time")  
 grid on  
 snapnow  
end

Chart

Description automatically generated

Figure 15 Transient simulation and the FFT for Gaussian pulse input with noise for

Chart, line chart, histogram

Description automatically generated

Figure 16 Transient simulation and the FFT for Gaussian pulse input with noise for

Chart, line chart

Description automatically generated

Figure 17 Transient simulation and the FFT for Gaussian pulse input with noise for

## Q5 vi. Comment

The bandwidth decreases as Cn increases. As Cn increases, and the output waveforms become smoother, which suggest that the effect of the noise is reduced. However, the transient output waveform for a large Cn value will also contain oscillations.

## Q5 vii

Vary the time step and see how that changes the simulation.

In the following code, the simulation steps are varied, which effectively varied the time step since deltaT = simTime/simSteps, and simTime is 1 second.

simTime = 1; % Simulate for 1 second  
  
% Create an array of simulation steps.  
arrSimSteps = [100, 10000];  
% Declare the capacitor Cn  
Cn = 0.00001;  
magIn = 0.001; % Magnitude for In  
  
% Simulation loops for different simulation steps  
for iSimSteps = 1:length(arrSimSteps)  
 % Retrieve the simulation step  
 simSteps = arrSimSteps(iSimSteps);  
 % Calculate the deltaT  
 deltaT = simTime/simSteps; % second/step  
 % Declare the time vector  
 vecTime = linspace(0,1,simSteps);  
  
 % Declare the input for a guassian pulse  
 stddev = 0.03; % std dev. of 0.03s  
 pulseDelay = 0.06; % delay pf 0.06s  
 vecInputV = exp(-((vecTime-pulseDelay)/stddev).^2/2);  
 % Generate the vector for noise current  
 vecIn = magIn\*randn(1, simSteps);  
 % Hold the output vector  
 vecOutputV = zeros(1, simSteps);  
  
 % Declare the vectors  
 vectorV = zeros(10, 1); % solution vector: [N1, N2, N3, N4, N5, I1, IL, I3, I4, In]  
 vectorF = zeros(10, 1); % F vector: F(1) = VIN, F(10) = In  
  
 % Declare the C matrix  
 matrixC = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0;  
 C, -C, 0, 0, 0, 0, 0, 0, 0, 0;  
 -C, C, 0, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, -L, 0, 0, 0;  
 0, 0, Cn, 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, 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, 0];  
  
 % Declare the G matrix  
 matrixG = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0;  
 1/R1, -1/R1, 0, 0, 0, 1, 0, 0, 0, 0;  
 -1/R1, 1/R1+1/R2, 0, 0, 0, 0, 1, 0, 0, 0;  
 0, 1, -1, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, -1, 1, 0, 1;  
 0, 0, -1/R3, 0, 0, 0, 0, 1, 0, 0;  
 0, 0, 0, 1/R4, -1/R4, 0, 0, 0, 1, 0;  
 0, 0, 0, 1, 0, 0, 0, -alpha, 0, 0;  
 0, 0, 0, -1/R4, 1/R4+1/RO, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 1];  
  
 % Construct the A matrix  
 matrixA = matrixC/deltaT + matrixG;  
  
 % Loop through the simulation  
 for iSim = 1:simSteps  
 % Update the F vector for Vin and In  
 vectorF(1) = vecInputV(iSim);  
 vectorF(10) = vecIn(iSim);  
 % Update the V vector  
 vectorV = matrixA^-1 \* (matrixC \* vectorV / deltaT + vectorF);  
 % Save the output voltage  
 vecOutputV(iSim) = vectorV(5);  
 end  
  
 % Plot of completed transient simulation for step input  
 figure(18+iSimSteps)  
 % Time domain plot  
 subplot(1,2,1)  
 plot(vecTime, vecInputV, "-b.") % Vin versus time  
 hold on  
 plot(vecTime, vecOutputV, "-r.") % Vo versus time  
 hold off  
 title("Transient simulation for a step input with noise for simulation steps = "+simSteps)  
 xlabel("Time (s)")  
 ylabel("Voltage (V)")  
 legend("Vin versus time", "Vo versus time")  
 grid on  
 % Frequency domain plot (fft)  
 subplot(1,2,2)  
 % Calculate sampling frequency  
 Fs = 1/deltaT;  
 df = Fs/length(vecInputV);  
 vecFreqPlot = -Fs/2:df:Fs/2-df; % Create the frequency vector for plot  
 vecOmega = 2\*pi\*vecFreqPlot; % Calculate the omega vector  
 fftVin = 20\*log10(abs(fftshift(fft(vecInputV)))/simSteps); % Input fft in dB  
 plot(vecOmega, fftVin, "-b.") % Plot the input fft  
 hold on  
 fftVo = 20\*log10(abs(fftshift(fft(vecOutputV)))/simSteps); % Output fft in dB  
 plot(vecOmega, fftVo, "-r.") % Plot the output fft  
 hold off  
 title("FFT of the step input with noise for simulation steps = "+simSteps)  
 xlabel("Omega (rad/s)")  
 ylabel("V (dB)")  
 legend("Vin versus time", "Vo versus time")  
 grid on  
 snapnow  
  
end

Chart, line chart, histogram

Description automatically generated

Figure 18 Transient simulation and the FFT for Gaussian pulse input with noise for simulation steps = 100 (large )

Chart, histogram

Description automatically generated

Figure 19 Transient simulation and the FFT for Gaussian pulse input with noise for simulation steps = 10000 (small )

## Q5 vii Comment

The waveforms for large (low simulation steps) have less detail because the sampling rate is low, and the waveforms for small (higher simulation steps) have more detail because the sampling rate is high. The waveforms for the small show more details on the noise.

## Q6 Non-linearity

If the voltage source on the output stage described by the transconductance equation V = alpha\*I3 was instead modeled by V = alpha\*I3 + beta\*I3^2 + gamma\*I3^3, what would need to change in your simulator?

To model the non-linearity, we can add a B(V) vector in our matrix equation. The B(V) vector models the non-linearity of the circuit: . The detail derivation is shown in the Appendix C. In each of the time step, we can use a loop implementing the Newton Raphson method to solve the non-linear matrix equation. The non-linear matrix equation is shown as follow:

To solve the non-linear matrix equation, the following steps are performed in a loop until is less than the tolerance:

1. Construct the B(V) vector to model the non-linearity of the circuit
2. Compute
3. Compute the Jacobian matrix:
4. Calculate the H matrix:
5. Calculate
6. Update
7. Check whether the tolerance is met or not

## Q6 Implementation

The following code shows the implementation of the non-linearity.

simTime = 1; % Simulate for 1 second  
simSteps = 1000; % Use 1000 steps  
% Calculate the deltaT  
deltaT = simTime/simSteps; % second/step  
% Declare the time vector  
vecTime = linspace(0,1,simSteps);  
% Tolerance  
tol = 0.01;  
  
% Transconductance parameters  
alpha = 100;  
beta = 500;  
gamma = 1000;  
  
% Declare the capacitor Cn  
Cn = 0.00001;  
magIn = 0.001; % Magnitude for In  
  
% Declare the input for a sin(2\*pi\*f\*t)  
freq = 1/0.03; % Hz  
vecInputV = sin(2\*pi\*freq\*vecTime);  
% Generate the vector for noise current  
vecIn = magIn\*randn(1, simSteps);  
% Hold the output vector  
vecOutputV = zeros(1, simSteps);  
  
% Declare the vectors  
vectorV = zeros(10, 1); % solution vector: [N1, N2, N3, N4, N5, I1, IL, I3, I4, In]  
vectorF = zeros(10, 1); % F vector: F(1) = VIN, F(10) = In  
  
% Declare the C matrix  
matrixC = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0;  
 C, -C, 0, 0, 0, 0, 0, 0, 0, 0;  
 -C, C, 0, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, -L, 0, 0, 0;  
 0, 0, Cn, 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, 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, 0];  
  
% Declare the G matrix  
matrixG = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0;  
 1/R1, -1/R1, 0, 0, 0, 1, 0, 0, 0, 0;  
 -1/R1, 1/R1+1/R2, 0, 0, 0, 0, 1, 0, 0, 0;  
 0, 1, -1, 0, 0, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, -1, 1, 0, 1;  
 0, 0, -1/R3, 0, 0, 0, 0, 1, 0, 0;  
 0, 0, 0, 1/R4, -1/R4, 0, 0, 0, 1, 0;  
 0, 0, 0, 1, 0, 0, 0, -alpha, 0, 0;  
 0, 0, 0, -1/R4, 1/R4+1/RO, 0, 0, 0, 0, 0;  
 0, 0, 0, 0, 0, 0, 0, 0, 0, 1];  
  
  
% Loop through the simulation  
for iSim = 1:simSteps  
 % Update the F vector for Vin and In  
 vectorF(1) = vecInputV(iSim);  
 vectorF(10) = vecIn(iSim);  
  
 % Hold the old vectorV  
 oldVectorV = vectorV;  
 % Boolean indicating whether the tolerance is meet to end while loop  
 bTolMeet = false;  
 % Loop to find V vector  
 while ~bTolMeet  
 % Construct the B vector  
 vectorB = zeros(10,1);  
 vectorB(8) = -beta\*vectorV(8)^2 - gamma\*vectorV(8)^3;  
 % Update the f(Vn) vector  
 fVn = (matrixC/deltaT + matrixG)\*vectorV - matrixC/deltaT\*oldVectorV + vectorB - vectorF;  
 % Construct the J matrix  
 matrixJ = zeros(10,10);  
 matrixJ(8, 8) = -2\*beta\*vectorV(8) - 3\*gamma\*vectorV(8)^2;  
 % Calculate the H matrix  
 matrixH = matrixC/deltaT + matrixG + matrixJ;  
 % Calculate the deltaV vector  
 vecDeltaV = - matrixH^-1 \* fVn;  
 % Update the V vector  
 vectorV = vectorV + vecDeltaV;  
 % Check whether the tolerance is met or not  
 bTolMeet = abs(vecDeltaV) <= tol;  
 end  
  
 % Save the output voltage  
 vecOutputV(iSim) = vectorV(5);  
end  
  
% Plot of completed transient simulation for step input  
figure(21)  
% Time domain plot  
subplot(1,2,1)  
plot(vecTime, vecInputV, "-b.") % Vin versus time  
hold on  
plot(vecTime, vecOutputV, "-r.") % Vo versus time  
hold off  
title("Transient simulation for a sinusoidal input with noise and non-linearity")  
xlabel("Time (s)")  
ylabel("Voltage (V)")  
legend("Vin versus time", "Vo versus time")  
grid on  
% Frequency domain plot (fft)  
subplot(1,2,2)  
% Calculate sampling frequency  
Fs = 1/deltaT;  
df = Fs/length(vecInputV);  
vecFreqPlot = -Fs/2:df:Fs/2-df; % Create the frequency vector for plot  
vecOmega = 2\*pi\*vecFreqPlot; % Calculate the omega vector  
fftVin = 20\*log10(abs(fftshift(fft(vecInputV)))/simSteps); % Input fft in dB  
plot(vecOmega,fftVin, "-b.") % Plot the input fft  
hold on  
fftVo = 20\*log10(abs(fftshift(fft(vecOutputV)))/simSteps); % Output fft in dB  
plot(vecOmega,fftVo, "-r.") % Plot the output fft  
hold off  
title("FFT of a sinusoidal input with noise and non-linearity")  
xlabel("Omega (rad/s)")  
ylabel("V (dB)")  
legend("Vin versus time", "Vo versus time")  
grid on  
snapnow

Chart, bar chart, histogram

Description automatically generated

Figure 20 Transient simulation and the FFT for sinusoidal input with noise and non-linearity

The nonlinearity of the circuit is shown as the varying amplitude in transient simulation waveform, and the nonlinearity of the circuit is shown as spikes in the FFT waveform.

## Helper functions

The following functions are helper functions used in the main code

% Helper function for mapping index  
% @param iRow = i index for the row  
% jRow = j index for the column  
% ny = size of the y  
function [n] = mappingEq(iRow, jCol, ny)  
 n = jCol + (iRow - 1) \* ny;  
end % End mappingEq  
  
  
% Helper function to add the obstacles  
% @ param boxLF = length of the box in fraction of region.x  
% boxWF = width of the box in fraction of region.y  
% region = region.x and region.y  
function [numBox] = AddObstacles(boxLF, boxWF, region)  
global boxes % Matrix for holding the boxes  
% Find the x, y, w, h for the bottom box  
xbb = region.x/2 - region.x\*boxLF/2;  
ybb = 0;  
wbb = region.x\*boxLF;  
hbb = region.y \* boxWF;  
% Find the x, y, w, h for the upper box  
xub = region.x/2 - region.x \* boxLF/2;  
yub = region.y \* (1-boxWF);  
wub = region.x \* boxLF;  
hub = region.y \* boxWF;  
% Create the boxes  
boxes = [xbb ybb wbb hbb;  
 xub yub wub hub];  
% Return number of boxes  
numBox = height(boxes);  
end % End AddObstacles  
  
  
% This function add a bunch of electrons in a given region randomly for Q3  
% @param numE = number of electrons  
% region = region for the electrons  
% T = temperature in Kelvin  
% numBox = number of boxes  
function AddElectrons\_WithBox(numE, region, T, numBox)  
global Const % Constants  
global x y % arrays for current electron positions  
global xp yp % arrays for previous electron positions  
global vx vy % arrays for current electron velocities  
global boxes % Matrix for the boxes position  
  
% Create the arrays for electrons locations  
x = rand(1, numE) \* region.x;  
y = rand(1, numE) \* region.y;  
  
% Loop through the electrons to make sure that no electrons inside obstacles  
for iE = 1:numE  
 % Flag to indicate whether inside box  
 insideBox = true;  
 while (insideBox)  
 insideBox = false;  
 % Loop through the boxes  
 for iBox = 1:numBox  
 % Check for invalid electrons position  
 if (x(iE)>boxes(iBox, 1) && x(iE)<(boxes(iBox, 1)+boxes(iBox, 3)) ...  
 && y(iE)>boxes(iBox, 2) && y(iE) < (boxes(iBox, 2)+boxes(iBox, 4)))  
 insideBox = true;  
 break;  
 end  
 end  
 if (insideBox)  
 % Regenerate position  
 x(iE) = rand() \* region.x;  
 y(iE) = rand() \* region.y;  
 end  
 end  
end  
% Create the arrays for previous electron positions  
xp = x;  
yp = y;  
% Create helper arrays for velocity distrubution  
vx = sqrt(Const.kb\*T/Const.mn).\*randn(1, numE);  
vy = sqrt(Const.kb\*T/Const.mn).\*randn(1, numE);  
end % End AddElectrons\_WithBox

[*Published with MATLAB® R2021b*](https://www.mathworks.com/products/matlab)