DAT116 (Mixed-signal system design) Lab 5: Nonlinearities in continuous-time circuits

Lars Svensson
larssv@chalmers.se

Version 1.9, December 10, 2018

1 Introduction

In this lab, we will tie the Cadence system and MATLAB together more intimately than before: we will use MATLAB to post-process the data produced by Cadence simulations. The integration is by no means perfect, but instead offers an example of how different tools may be stitched together to solve tasks neither tool would easily tackle on its own. (Some of the operations performed here may in fact be carried out using the Cadence Calculator instead; but MATLAB offers all the freedom of a programming language designed for general-purpose numerical computation and display, which will be utilized at the end of the session.)

This integration will require you to run MATLAB on `heffalump` rather than on your local machine. The MATLAB windows will appear in the VNC desktop.

2 Preparation

As this lab builds on the previous labs, it is assumed that you have worked through those.

3 Setup

Before launching MATLAB, it is necessary to set some environment variables in order to access the routines used to read Cadence simulation results into MAT-

LAB.

- If you use `bash` as your login shell, give the command

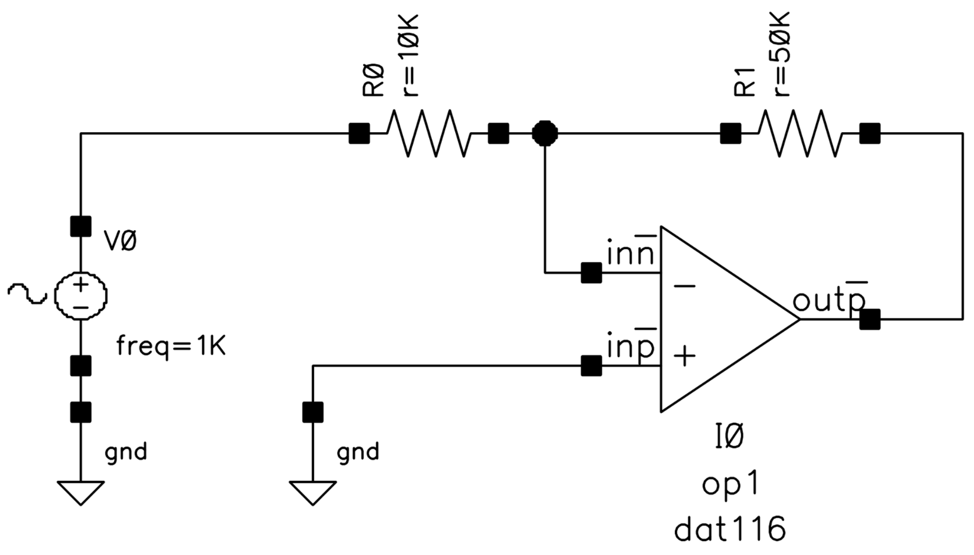
  ```
  source /usr/local/cad/course/DAT116/Y2018/cm.bash
  ```

  If you use `csh` or `tcsh`, change the last suffix to `.csh` .

4 Cadence-MATLAB data transfer

The first task is to transfer the result of a transient simulation from the Cadence system to MATLAB. We will use the simple inverting amplifier stage which we already studied in Lab 2.

- Launch Cadence and open the Library Manager tool.
- Make a copy[1] of the cell `lab2` from a few weeks ago and call it `lab5`. Verify that the copy includes both the `schematic` and `config` views.
- Open the schematic and config views of `lab5`. Verify that its schematic connectivity corresponds to the figure below.



[1]Right-click on the cell name in Library Manager; select **Copy. . .** ; enter the new cell name in the `To` pane; check the box `Update instances:`; and change the pulldown from **Of Entire Library** to **Of New Copies Only**.

- Open the properties panel of the sinewave source. Set the frequency to 10 kHz and the amplitude[2] to the variable `ampl`. Click OK to close the panel.
- Check and save the schematic view.
- Launch **ADE L** in the same way as in previous labs, and set the simulator to `ams` in accordance with the pop-up reminder.
- Before closing the dialog, make note of the `Project Directory`. All simulation files will be created in subdirectories of the project directory.
- Set up a transient analysis with a duration of 1 ms. Use the `conservative` accuracy defaults to instruct the simulator to prefer accuracy over simulation speed.
- Copy variables from the cellview into **ADE L**, and set the value of `ampl` to 0.1.
- Select the nodes at the sinewave source and the opamp output for voltage plotting.
- Choose **Simulation→Netlist and Run** to produce the input file for the simulator and to run it. A log file window opens to allow you to follow simulation progress. Verify the result in the graph window that pops up when simulation is complete. *Do not dismiss the log file window yet.*

The results of the simulation are now stored in the Cadence design database together with simulation parameter values etc. Special-purpose MATLAB routines are available to access the database; the routines need to be pointed to the right place in the filesystem though.

- Starting from the project directory that you made note of above, explore the directory structure of the results files. Descend into directories named after your top-level cell and after the simulator used; stop when you find a directory called `psf`.
- Make note of the whole directory path (starting with "/") leading to this directory; example: `/a/b/c/simulation/optest/ams/config/psf` .

The MATLAB utility files which you can find on PingPong are also available on `heffalump` and have been added to the MATLAB search path. One function,

[2] A reminder: there is both an `AC magnitude` field and an `Amplitude` field. The former value is used only in AC simulations. Here, you need to set the latter one!

`cds2sig.m`, will convert voltage waveforms generated by Cadence simulations to the format expected by `sigview` etc.

- Launch MATLAB from the command line in a shell window where you sourced the setup file according to instructions in Section 3 above. Select a working directory for this lab.
- If you ran the setup script correctly (as described in Section 3), you should now have access to a MATLAB function called `cds_srr`. Verify that this is so by typing the function name at the MATLAB command line. An `undefined function or variable` message means that you don't have access.
- In MATLAB, give the command `cds_srr` with one string argument enclosed in single quotes: the directory path you made note of above. (Make sure to specify a *complete path* that starts with a `/`, not with a `.` or a ∼.) Example:

  ```
  cds_srr('/a/b/c/simulation/optest/ams/config/psf')
  ```

  The command returns labels for all the data available at the indicated point in the database. Note that `'tran-tran'` is one of these labels.
- Give the same command again, but add `'tran-tran'` *as a second string argument.* Example:

  ```
  cds_srr('/a/b/c/simulation/optest/ams/config/psf', ...
          'tran-tran')
  ```

  Again labels for the available data are returned; these labels include the names of the circuit waveforms which were saved during your simulations.
- Give the same command a third time, but add the name of the output signal as a third string argument. Example:

  ```
  cds_srr('/a/b/c/simulation/optest/ams/config/psf', ...
          'tran-tran', 'optest.net010')
  ```

  A structure is returned which includes arrays of time and signal values.
- Assign the result of the last `cds_srr` command to a MATLAB variable name, such as `foo`.

The simulation waveform has now been imported into MATLAB. A quick plot acts as a sanity check.

- Plot the just-imported signal:

  ```
  plot(foo.time, foo.V) ;
  ```

  Verify that the signal has the same amplitude and frequency as in the Cadence plot window.

To be able to use `sigspectrum` on the imported signal, we must adapt its format to what `sigspectrum` expects. The routine `cds2sig` performs the conversion[3]. In order to do so, it needs two extra parameters: the duration of the time window to convert, and the number of sample intervals in that time window. The duration should be equal to a number of full cycles of the original signal; just like `sigspectrum`, the conversion routine selects the *trailing end* of the signal for conversion, in order to avoid any initial transient.

The number of sample intervals in the window should be a power of two, as that is what `sigspectrum` expects. Increasing this parameter far beyond the number of points in the signal you just imported will not improve accuracy or resolution.

- Create a signal `bar` from `foo`, using 1024 points distributed across the 8 last cycles of the signal:

  ```
  bar = cds2sig(foo, 0.8e-3, 1024) ;
  ```

- Call `sigview` with `bar` as an argument. Verify the frequency and the number of cycles included.
- Call `sigspectrum` with `bar` as an argument. Inspect the plot window. Does it correspond to your expectations?

Compared with previous spectra seen during the labs, you may find that the "noise floor" is unexpectedly high. Circuit simulators such as Spectre (which does the actual calculations during the `ams` simulations here) use *tolerance parameters*

[3]It should be noted here that the signal produced by the Cadence simulation is almost universally *not* uniformly sampled. To be accepted by `sigspectrum`, the signal must therefore be *interpolated*: new values are computed for a set of uniformly spaced times. This operation has limited accuracy and may affect the spectrum noise floor.

to control the tradeoff between accuracy and simulation speed. Smaller tolerances improve accuracy at an execution-time cost and typically produce more time points.

- Inspect the simulation log file window that was mentioned on page 3. Go to the bottom of the window and make note of the `"Number of accepted tran steps"` and the CPU time in `"Intrinsic tran analysis time"`. Dismiss the log window.
- In **ADE L**, select **Simulation→Options→Analog(Spectre)...** A parameter dialog window opens. Find the options `reltol` and `vabstol` and give each of them the new value of `1e-9`. Click OK to accept the change and dismiss the window.
- Run the simulation again. Make note of the timestep count and the CPU analysis time as before.
- Read the opamp output node waveform into MATLAB as before. Save the converted version of the waveform *in another variable*:

  ```
  baz = cds2sig(foo, 0.8e-3, 1024) ;
  ```

- Plot spectra of both `bar` and `baz` in the same `sigspectrum` window:

  ```
  sigspectrum(bar, baz) ;
  ```

  Save the plot for your report.

What has happened to the “noise floor”?

What has happened to the simulation time and to the length of the result vector?

- Use `sigspectrum` to extract the numerical component values for the `bar` and `baz` spectra. Calculate SNRs based on these values as you did in Lab 1. Do the SNR values agree with the shifted noise floor?
- Use the special argument `'linf'` to make `sigspectrum` use a linear frequency scale. It is now possible to see the component at frequency 0, that is, the output DC offset voltage[4]. How does the magnitude of the DC component compare to the rest of the noise floor?

[4]The output DC offset voltage is caused by the opamp input bias currents, which are drawn from different source resistances (ground, i.e. 0, and $R_0//R_1$).

- Recompute the SNR values for the two cases, this time ignoring the DC component. You may have to take care not to lose too much precision when calculating the SNR! Comment on the correspondence with the noise floors in the two cases.

Does the higher accuracy make a practical difference in these SNR calculations?

Continue to use the tighter tolerances for the rest of this lab session.

5 SNR as function of input level

In most circuits and systems, the signal-to-noise ratio depends on the signal level. It is therefore important to be able to estimate the SNR as a function of the input signal level. We will now run a series of transient simulations while sweeping the input amplitude, and calculate the SNR based on each of these simulations.

- In the **ADE L** window, select **Tools→Parametric analysis...** A **Parametric Analysis** window appears as in Lab 2, allowing you to specify a variable to sweep and what values it should take.
- In the `Variable Name` box, enter the name of the variable that controls the input amplitude (`ampl` in this case).
- Enter a variable range from `1e-3` to `10` to sweep the amplitude from 1 mV to 10 V. Use a logarithmic sweep with a total of 20 steps.
- Select **Analysis→Start All**. Simulation progress is tracked with a status bar. When all simulations have been completed, all output waveforms are plotted in the Cadence waveform window (which is likely to be very crowded. . .).

What happens to the output signal at high input levels? What is the maximum value you see at the output?

- Go back to the schematic window in Cadence. Select the opamp and open its property window (**Edit→Properties→Object**, or `[q]`). The `VLIMN` and `VLIMP` properties set the maximum and minimum output voltages reachable with ±15 V supplies. Verify that these values correspond to the extreme values seen in the plot. Do not change the values; just close the property window.

- In MATLAB, repeat the call to `cds_srr` to read in the Cadence waveforms.

Note that `time` and `V` are now *two-dimensional arrays* rather than the vectors returned in the non-parameterized case. An extra vector containing the parameter values is also returned.

- Inspect the contents of the extra vector in MATLAB. Make a copy of this vector; a good name for the copy might be `ampl`.
- In MATLAB, repeat the call to `cds2sig` with the just-read waveforms as an argument.
- Repeat the call to `sigspectrum` in the same way as before, with the result of the `cds2sig` as an argument. A colorful plotted figure appears with spectra for all simulation runs.
- Repeat the `sigspectrum` call again and assign its output to a variable. As with the voltage waveforms, the result is a two-dimensional array, where each column contains the spectral values for one of the simulations.

Calculation of SNR values for each column value is simplifed by the behavior of the MATLAB functions `max` and `sum`: with a two-dimensional array argument, they produce a vector containing the results for each column, that is, for each separate spectrum. Thus, the following expression returns a vector of SNR values for the spectrum array `arf`:

```
10*(log10(max(arf)) - log10(sum(arf) - max(arf)))
```

- Extract the SNR values as above.
- Open a new figure window in MATLAB (with the command `figure`). Plot the SNR values as a function of the parameter value vector `ampl` which you saved above. Use the function `semilogx` rather than `plot` to get a logarithmic axis for the parameter values, and give the extra argument `'-+'` to highlight the discrete values. Save the plot for your report.

What happens to the SNR when the output signal enters the clipping region?

6 Soft nonlinearity

As you saw above, hard signal clipping causes severe SNR degradation when the levels of several harmonics rise and approach the level of the fundamental tone;

compare with quantizer clipping in Lab 3! This mode of operation is therefore generally avoided[5], but subtler nonlinearities can still affect the SNR of an amplifier. In this next experiment, you will replace the `op1` amplifier with its sibling `op2`, which includes a gradual nonlinearity (implemented with a *tanh* function).

- In the `lab5` schematics window, select the opamp component and open its properties window (using `[q]`). Change the contents of the `Cell Name` field from `op1` to `op2`. Accept the changes and close the window by clicking **OK**.
- Check and save the schematic view.
- To generate a netlist (without running a single analysis according to the listing in **ADE L**), select **Simulation→Netlist→Create**.
- Go back to the **Parametric Analysis** window. Reduce the upper limit of the sweep range to 2.5 to avoid clipping, start the analysis, and wait for it to complete.
- Generate an SNR diagram as in the previous section, and include it in your lab report.

How does the SNR depend on the signal level when the amplifier has a soft nonlinearity such as this one? Compare and contrast with the previous diagram.

A non-linearity affects the harmonic content of the signal; the level of each harmonic depends on the signal level. We will now plot the powers of the fundamental and the first nine harmonics separately, as functions of the input level. Still another MATLAB function, `sigharms.m`, is provided for this purpose. It takes three arguments: an array of spectral component powers as provided by `sigspectrum`; the number of full cycles of the fundamental included in the spectal analysis; and an array of the parameter values used in sweeping (which will be used on the X axis of the plot).

- Assuming that the spectrum component array which you produced with `sigspectrum` above is called `arf`, give the following MATLAB command:

  ```
  sigharms(arf, 8, foo.ampl) ;
  ```

  A plot window opens and shows the levels of fundamental and harmonics as a function of the sweep parameter (the input amplitude in this case). You need this plot for your report!

[5]The most common exception seems to be guitar amplifers. . .

Which one is the dominating harmonic in the diagram? What does that observation tell you about the nonlinearity?

Compare the rate of increase of the harmonic levels with the input amplitude. Do higher-order harmonics rise faster or slower with the amplitude? Why?

As a final experiment, we will reduce the open-loop gain of the opamp and observe the effect on the SNR and on the levels of the harmonics.

- Again, select the opamp component in the schematic window and open its properties window. Reduce the open-loop gain parameter `AOLDC` to 20 dB. Accept the change, save the schematic, generate a netlist, and run the parametric simulation as before.
- Read the simulation results into MATLAB, convert the waveforms to the `sigspectrum` format, generate an array of spectral components, and create a plot of the harmonic levels as functions of the input level. Save the plot for your report.

What has happened to the levels of the harmonics when the open-loop gain changed? Why?

- Generate a plot of the SNR values as a function of the input level, as before.

How has the SNR level changed? Why?

7 Wrap-up

After completing this lab session, you should be able to do the following:

- Set up a parametric simulation of a circuit including macro models in Cadence.
- Transfer the results of transient simulations from Cadence to MATLAB for further analysis.
- Extract data for and produce plots which show the SNR and the harmonic levels as functions of the input level of a circuit.
- Qualitatively describe typical SNR and harmonic-level diagrams as functions of signal level.

Reflection questions

- You have seen that non-linearities of active components will generate harmonics when a single-tone sinewave is applied to the component. You have also seen that the harmonics depend on the signal level.

  Consider the implications of these observations on the problem of active-filter design, as studied in Lab 4. The filters studied there were implemented as cascade sequences of second-order sections; but no time was spent on deciding the place in the sequence of each section (first, second, etc). Given what you have seen here, how can the choice of the sequence affect the overall filter SNR?

- In one reflection question in Lab 2, you were asked to consider using several amplifier stages to reach a specific gain accuracy. Is there a similar case to be made for distortion? Discuss.