10.M APPENDIX: MATLAB SIMULATIONS

10.M.1 Higher-Order Responses Using MATLAB

While it is difficult to solve for the transient response for third- and higher-order loops using classical or transform techniques, we can easily obtain their responses using MATLAB. The program `Ord3spcl` will produce any of the curves in Fig. 10.A.4 , 10.A.6, 10.A.7 and 10.A.8 and more. It uses Eq. (10.A.8) and the error response 1-*H* derived from it. As before, the ramp error response is obtained by taking the step response of the integral of 1-*H* and the integration is performed simply by shifting the elements of the numerator vector.

While `Ord3spcl` is instructive, in that we can see how MATLAB can be used to produce the same results that were obtained in Section 10.A, we would often like to get the transient and frequency responses for higher-order loops without finding *H* explicitly. During the design process, we will usually work with the open-loop singularities and gain so we would like to obtain closed-loop responses given the open loop poles and zeros and gain. The program `Open2cls` does this. It begins by specifying vectors containing the coefficients of power of *s* (descending order, as before) for the open-loop poles and zeros. Then it uses a MATLAB function `zp2tf` to convert these vectors to the numerator and denominator vectors we have used before — except these are for the open-loop transfer function *G*. However, we can easily obtain the closed-loop *H* and 1-*H* from them as follows.

$$H=\frac{\texttt{out}}{\texttt{den}}=\frac{G}{1+G}=\frac{\texttt{nopen}/\texttt{dopen}}{1+\texttt{nopen}/\texttt{dopen}}=\frac{\texttt{nopen}}{\texttt{dopen}+\texttt{nopen}}, \qquad (10.M.1)$$

$$1-H=1-\frac{\texttt{nopen}}{\texttt{dopen}+\texttt{nopen}}=\frac{\texttt{dopen}}{\texttt{dopen}+\texttt{nopen}}, \qquad (10.M.2)$$

where

$$G=\frac{\texttt{nopen}}{\texttt{dopen}}. \qquad (10.M.3)$$

As a starting point, `Open2cls` has been written to apply directly to the special third-order loop described in Section 10.A. A gain value is employed that places unity gain at the geometric mean of the zero and added pole, as in Section 10.A. We can interactively

chose *r* and then can, if we wish, apply a multiplier to the gain to shift it relative to the nominal value. The resulting closed loop time and frequency responses can be plotted as can the open-loop response. By varying the gain multiplier we can observe the effects of inaccuracies in setting up the optimum loop described in Section 10.A.

As before, the numerator vector elements of 1-*H* are shifted to the right to obtain the ramp response but, since we do not have the numerator explicitly (having derived it from *G*), we use matrix multiplication to accomplish the shift.

We can modify `Open2cls` to obtain responses corresponding to arbitrary open-loop configurations. `Open2c4` is an example of such a modification to accommodate a fourth-order loop with two poles at zero frequency and two other real poles, `pole1` and `pole2`, plus one zero. The finite poles are input interactively, as is the zero indirectly. The matrix that is used for integration in obtaining the ramp response is increased in size; a 5×5 matrix is required to shift the coefficients of s^0 through s^4. The time and frequency ranges have been set for a particular set of frequencies, to be given below, so the `t` and `w` vectors will require modification for different loop bandwidths.

Here is a suggestion for the beginning of an investigation of the properties of the kind of loop represented by `Open2c4`. Execute `Open2c4` and enter `r1z` = 4 followed by `pole1` = 1000. (We are entering the negative of the pole frequencies for simplicity — for reasons of stability we certainly cannot have poles with positive real parts). This will set the zero (= `pole1/r1z`) to 250. (Frequency units are in rad/sec.) Enter `pole2` = 4000 and a gain constant of 0 dB. Enter *b* for plot type and thus obtain a Bode plot. Note the frequency where excess phase shift is the least and the gain change necessary to give 0 dB there, thus giving maximum phase margin. (You may want to right click on the plot and choose "grid," since a grid has not been commanded by the script.) Retain the same `r1z`, `pole1` and `pole2` as before by using the "return" or "enter" key in response to a request for value. Change the gain to the value that will give maximum phase margin and then check the Bode plot to insure that unity gain occurs at minimum phase lag. At the same time note the gain margin (at -180° phase). Then check the error response and observe the degree of overshoot. Now change the gain to give about 3 to 5 dB of gain margin and observe how the overshoot changes.

Now change the `pole2` frequency to 1000, placing the two poles on top of each other. Adjust gain for maximum phase margin again and compare the overshoot to what

previously occurred at maximum phase margin.

`Open2c4x` is similar to `Open2c4` but the zero frequency is entered explicitly. Step through the default values by pressing the return or enter key. You should get the same defaults as in `Open2c4`, which are just slightly different from what you entered above. Now try entering a pair of complex poles corresponding to a 2-pole (1 complex pair) Butterworth low-pass filter. For the first pole, enter 1414+1414*i and for the second enter 1414-1414*i. Compare the resulting performance to the case with two poles at 2000 rad/sec. Both filters have the same high-frequency gain so they will be equally desirable for that parameter where it is important (e.g., many frequency synthesizer applications). Compare Bode plots. Which can give the greatest phase margin?

10.M.2 Simulation of the ADPLL Using MATLAB

The program `ADPLL` simulates the loop in Fig. 10.26. It will plot N_{out} and show where the output was sampled, as a result of the input transition, to produce a new value of N_2'. It will also show instantaneous frequency versus time and the average frequency for the simulation. Try Example 10.1 using `ADPLL.` Use `Phin` = 0.5 and `Tmaximum` = E-4 (10^{-4} seconds). Set the initial values of `Nout` and `N2` to their theoretical steady-state values, which will be displayed when the program is run and which should minimize the initial transient. Note how the sampled values of `Nout` vary above and below the theoretical steady-state value. Compare the average output frequency to the input frequency. You might want to try a longer simulation to see if the difference is reduced.

Then use an offset of `Noff` = 100 with an input frequency of 600 kHz. (Use `q` = 4, `nv` = 12, `Fclock` = 8.192E6.) Set the initial values of `Nout` and `N2` to the given steady-state values. Now retain those initial values and halve the input frequency. (This simulates stepping the input to 300 kHz after a steady-state lock at 600 kHz.) You should obtain a false lock. The program will also compute new steady-state values of `Nout` and `N2` that correspond to the 300 kHz-input frequency. Change the initial values to those final values and note that a correct lock is obtained. You might want to experiment with how far off the initial conditions can be without producing false lock. How do the instantaneous frequency excursions about the average change with input frequency?

Note that the minimum value of f_{in} for stability, to keep C <2 in Eq. (10.32), is 256 kHz. Set `Tmaximum` = 2E-4 and try an input frequency of 250 kHz, setting the initial

conditions at steady-state values. Is the response stable? Is the average output frequency correct? Increase the input frequency to 260 kHz. (Leave the other conditions alone.) This meets our stability criterion but what do you see? — perhaps the results of quantization effects. Try 270 kHz. That should be better. You might repeat this with `Noff` = 0. This puts the sampled value of `Nout` closer to mid-range and allows you to watch the instability grow.