## 17.M APPENDIX: MATLAB SCRIPTS

MATLAB scripts used to generate simulations for this chapter are described here. They can be downloaded for a more complete understanding. Some scripts have features that could be incorporated in others where that is found to be desirable.

### 17.M.1 Nonlinear Simulation with Noise

The MATLAB script `NLPhN` simulates the non-linear acquisition behavior of a PLL in the presence of broadband additive noise. It is similar to `NLPhP` but random noise has been added to the offset voltage $u_{off}$ in Fig. 8.21, as shown by $u_n$ in Fig. 17.M.1.

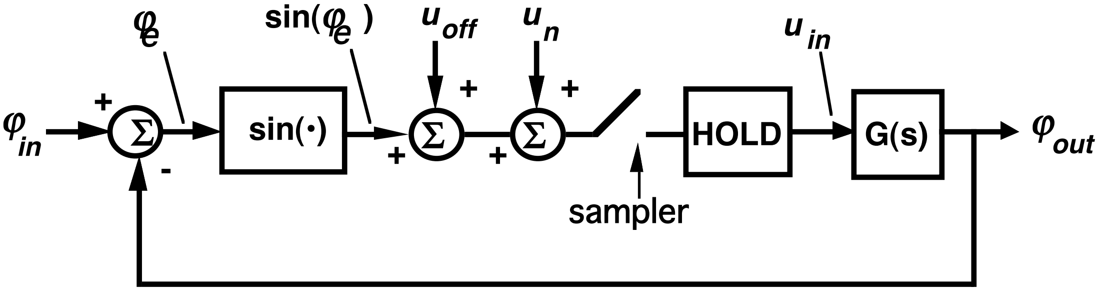

**Fig. 17.M.1 Noise Added To Model**

### 17.M.1.1 Simulating Broadband Noise

The input $u_n$ is a normally distributed random number with a mean of 0 and a variance of $\sigma_N^2$. This produces an equivalent noise power density, as seen in the loop (with noise bandwidth $B_n << f_s$) of $S_{\text{eq}} = 2\sigma_N^2 / f_s$, where $f_s$ is the sampling frequency. That is, after sampling, the noise power $\sigma_N^2$ is distributed over a band equal to half of the sampling frequency, $f_s/2$.

To see this, consider a band of noise of density $S_{\text{eq}}$ that extends from 0 to $f_s/2$. We are using one-sided noise density, on which we have standardized, so the two-sided density is $S_{\text{eq}}/2$ and extends from $-f_s/2$ to $+f_s/2$. If this band is sampled at $f_s$, it will be shifted to all multiples of $f_s$ and replicated, creating a continuous band of noise with density $S_{\text{eq}}/2$. The noise at frequency $f_x + jf_s$, where $j$ is an integer, will be the same as at $f_x$, so the replicated bands will not be independent of the original band. However, they will be attenuated by the response of the hold function, $\sin x/x$, where $x = \pi f/f_s$, and the loop will not respond to them significantly if it is sufficiently narrow ($B_n << f_s$).

This density will be multiplied by $(K_p')^2$ to give the power spectral density after the PD, $<u_{1n}^2>/df$ from Eq. (13.10). Thus, the density before the PD is

$$\sigma_N^2/(f_s/2) = <u_{1n}^2>/[(K_p')^2 df] = N_0/P_c \qquad (17.M.1)$$

and the variance for the noise generator must be set to

$$\sigma_N{}^2 = (N_0/P_c)\,(f_s/2)\ . \tag{17.M.2}$$

In terms of $\rho_{L0}$, the S/N in the (linear) loop noise bandwidth, this can be written

$$\sigma_N^2 = \frac{N_0 B_{N0}}{P_c B_{N0}} \frac{f_s}{2} = \frac{1}{\rho_{L0}} \frac{f_s}{2B_{N0}}\ . \tag{17.M.3}$$

The correlation between samples numbered $j$ and $j + k$ of $u_n$ is given by Eq. (13.22). These samples are separated by

$$T = k \text{ cycle}/f_s \tag{17.M.4}$$

and the maximum frequency of the noise band is

$$\omega_{\max} = \pi f_s/\text{cycle} \tag{17.M.5}$$

producing a correlation of

$$\frac{R(T)}{R(0)} = \frac{\sin(k\pi)}{k\pi} = 0\ , \tag{17.M.6}$$

That is, the samples are independent, so the single-source model is adequate. We can, however choose to employ dual sources, possibly for comparison.

### 17.M.1.2 Observing Cycle Skipping With `NLPhN`

Fig. 17.M.2 shows the phase output from a run of `NLPhN`. (Others are shown in Fig. 17.1.) Truncation of the phase display has been turned off so we can observe cycle skipping. Note how the phase tends to stay at a multiple of 1 cycle error for a while and then move on to another multiple except at 73 seconds where it passes through -2 cycles without pausing.

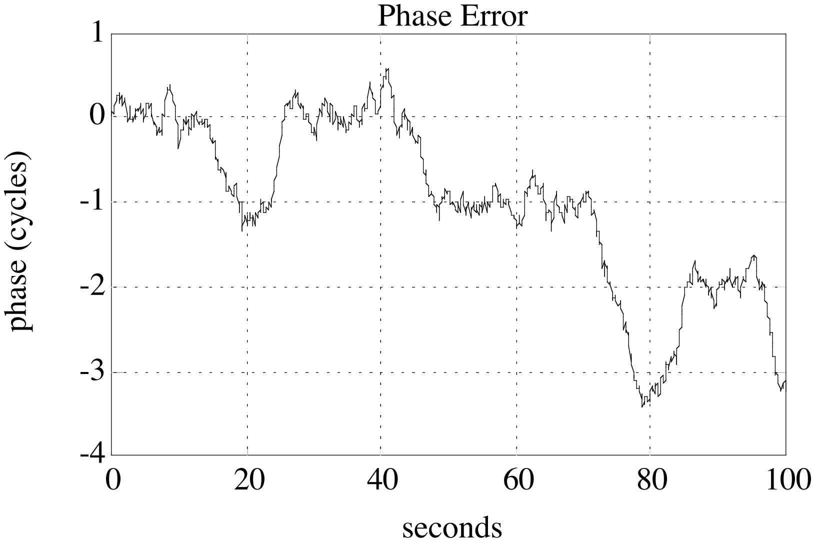

**Fig. 17.M.2 Phase in the Presence of Noise,**
**$\omega_n = 1$, $\zeta_0 = 0.707$, $\alpha_0 = 1$, S/N in $B_{n0} = 1.41$ dB**

### 17.M.1.3 Changes To `NLPhP` To Create `NLPhN`

Fig. 17.M.3 shows the significant changes to `NLPhP` that add noise to the simulation. Changes or additions are in **bold** among portions retained from `NLPhP`. The user sets either the noise-to-signal ratio in the loop bandwidth, $1/\rho_{L0} =$ `NS`, or the ratio of noise power spectral density to signal, `No`. Noise bandwidth, `Bn`, is computed from Eq. (14.10). A series of random numbers with unity variance is multiplied by the constant `Kn` to produce samples PDn of the noise voltage, $u_n$ in Fig. 17.M.1, in accordance with the theory given above. During the main simulation loop, a new sample is added to the offset each computation cycle. Output is similar to what is shown in Figs. 8.M.1 through 8.M.5 except that the plot of phase detector output has been replaced by a plot of sampler output, which includes the injected noise. There are some other differences, for example changes in graph labels, and sometimes an explanatory note or output may appear in one version and not the other.

```
%TIME RESPONSE & PHASE PLANE PLOT IN NOISE
. . . . . . . . . . . . . . .
% MODIFY PARAMETERS BELOW & GO
%******************************
% Set either of the following parameters and set the other
                 to zero.
NS = .7227; % N/S in Loop Noise Bandwidth
No = 0; % Noise Power Density, per Hz, to Signal Ratio
SR = 0; % 1 for step response (if also Phinit=Winit=0 and
                 Ap or Af not 0).,
. . . . . . . . . . . . . . .
%  op=3:  Frequency + Sampler Output vs. Time
%*******************************
fprintf('\nWn = %g rad/sec; zeta = %g; alpha =
                 %g',Wn,z,alpha)
Bn = Wn*(1/(2*z) + 2*z*alpha^2)/4; % Noise BW in Hz
fprintf('\nNoise Bandwidth = %g Hz\n',Bn)
% Compute multiplier of noise distribution having unity
                 power.
if NS == 0,
     Dn = No;
     fprintf('Noise Density to Signal Ratio = %g /Hz\n',No)
else
     Dn = NS/Bn;
     fprintf('Signal/Noise in Loop Bandwidth = %g\n',1/NS)
end
Kn = sqrt(Dn*SmpPerOut/(OutInc*2));
. . . . . . . . . . . . . . .
     NL = sin(PhIn - y); % <<<<<<<<<<<< THE NON-LINEARITY
     PDn = Kn*randn;
     PDnoise(i+1) = PDn;
     u1(1) = NL + Offset + PDn; % initial input to sampler
     u1(2) = u1(1); %  constant Hold output -> same at
                 beginning and end of dt
. . . . . . . . . . . . . . .
     if op == 3, % selecting Sampler Output vs. time
          subplot(212);
     else,
          subplot(224),
     end % if op==3
     plot(t,sin(er)+PDnoise,'+')
     title('Sampler Output')
end % if ()|()
```

**Fig. 17.M.3 Significant Changes To `NLPhP` To Create `NLPhN`**

Partly from William F. Egan, *Phase-Lock Basics*. Reprinted with permission of John Wiley and Sons, Inc.

### 17.M.1.4 Gathering Phase Statistics and Plotting Probability

The script `erdis` can be used to plot the probability distribution function for 1001 points generated by a run of `NLPhN`. If `NLPhN` is set to produce 1001 or more values of the phase error `er` with truncation, `erdis` will produce the distribution function when run subsequently. It can be easily modified to operate on a different number of points of `er`. The curves in Fig. 17.9 used 10,001 points. We could also use the MATLAB function `hist(er)` to build a histogram of the phase error `er` after running `NLPhN`. That would be an approximation (using a finite set of data points) to the probability density function.

### 17.M.2 Generating Statistics of Output Phase and Cycle Skips

`Nstat1` and `Nstat2` simulate first- and second-order loops respectively in the presence of noise while measuring the phase variance and the rate of cycle skipping. After observing graphs showing how the statistics change with time (Fig. 17.M.4), the user can command additional simulation points. This might be done to come closer to a steady-state value, that is, to reduce variations in the accumulated averages by gathering more data.

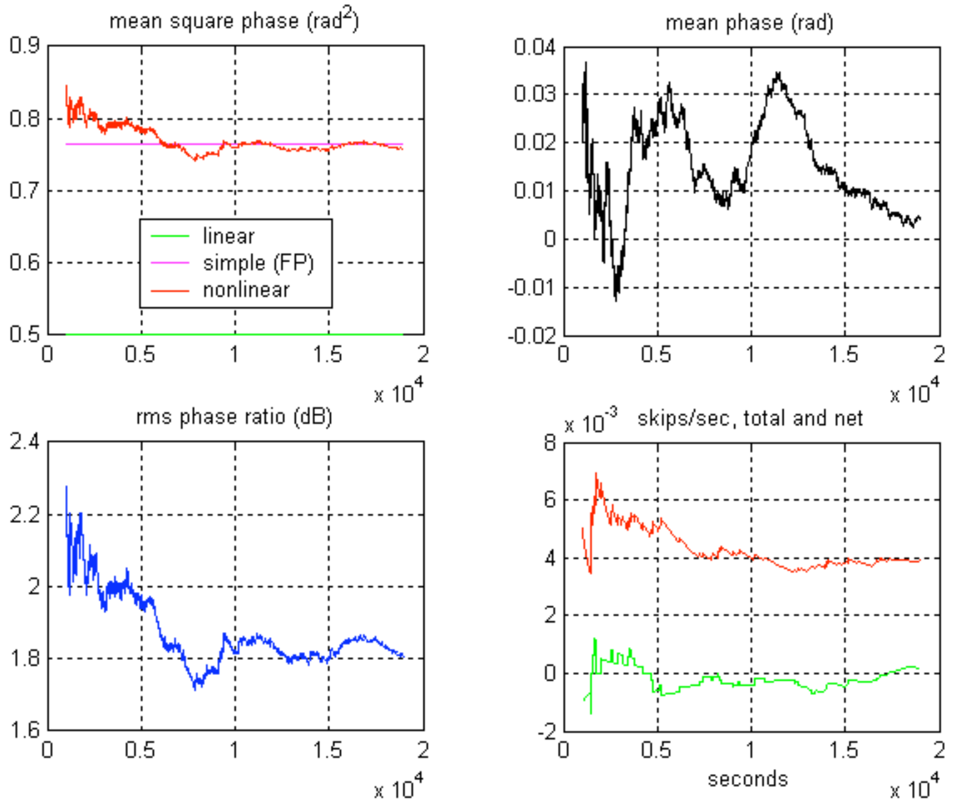

Fig. 17.M.4 Graphical output from `Nstat1`.

Either a single noise source or two sources (Fig. 13.5) can be used. This enables us to show that the single source is adequate for these broadband noise simulations.

`Nstat1` can also be set to produce a time response. Because there is no filter in the first-order loop, the matrix functions that are used in `NLPhP` and `NLPhN` are not necessary in `Nstat1`. [Information about simulating higher-order loops without matrix manipulation can be found in Egan (2000, p. 538)].

### 17.M.3 Obtaining mean time to first cycle skip.

`Skip1` and `Skip2i` provide mean time to first cycle skip statistics for first-order and second-order loops. The latter is designed to iterate three loop parameters, providing a sequence of outputs. This is most useful for long unattended runs and does not provide the opportunity for the user to modify or read parameters on the run or to lengthen the simulation interactively. Thus, each of these scripts has a different set of features, including the simulation of differing loop types. New scripts (e.g., a `Skip2`, simulating a type-2 loop interactively or a version of `Skip1` with iterated variables) can be created by using these as guides.

### 17.M.4 Simulating Phase Modulation

`Sig1i` and `SigOff2i` simulate the effects of phase modulation at a rate too high to be followed by the loop, using a square wave of phase, for first- and second-order loops, respectively. Both scripts are iterative but only `SigOff2i` writes to a file (`runrec.txt` in MATLAB `work` directory). Output data is normally available from the MATLAB Command Window after a series of iterated runs but writing to a file can be more convenient, especially if a long run should be interrupted by some problem that causes the Command-Window data to be lost.

Both scripts give the number of cycles skipped and the mean phase but `Sig1i` does not give an output variance if there is a finite offset (or no noise). However, if there is additive noise, `SigOff2i` gives both the mean square output phase and its sample variance (i.e., an unbiased estimate of the variance of the underlying data) in the presence of an offset. `SigOff2i` can also produce time responses.

`Sig2` also simulates a second-order loop but is not iterative and has some restrictions like `Sig1i`. (Note: Some scripts use `sig2` as a variable so typing `sig2` might cause that variable to display if it has been left in the MATLAB environment. Therefore, it may be necessary to type `clear` first or to use the proper script name `Sig2` to execute the script.)

### 17.M.5 Controlling Iteration

Scripts whose names end in "i.m" iterate between two and four variables using "for loops." The initial and final values and step size may be given or an array of values may be used for the iterated (loop counter) variable in the loop control line. (Some such arrays that have been commented out can still be seen.) Use of a single value for the iterated variable effectively defeats the loop. An iterated variable *x* can be changed to another variable *y* by the following procedure. Change *x* to *y* in the loop control line. Remove the comment designator % from before the line below that sets *x* (thus making it active)) and give *x* the desired value in that line. Place a comment designator % before the line that had set *y* (to make it inactive).

### 17.M.6 Control of Interactive Simulations

If a negative number is entered when requesting additional points from `Nstat1` or `Nstat2`, program execution is paused and commands can be entered, so some of the simulation control parameters can be changed or data can be read. When the word `return` is typed and entered, the program continues and additional points, equal in number to the absolute value of the negative number that had been entered, are computed. This sort of control is common to scripts that are interactive.

As an example, when execution is paused, we might set `strtplt` to a value greater than 1 (e.g., by typing `strtplt = 1000`) to inhibit the initial points, with their large excursions, from being plotted, allowing the plot scale to expand vertically in subsequent plots. (The effect of this feature can be observed in Fig. 17.M.4.) Or, we might read some parameter's value before continuing the simulation. For example, typing `Fsum(last)` at the point where Fig. 17.M.4 is being displayed, would produce `ans = 0.0039`, the last value of the red curve in the lower-right plot. Of course, greater familiarity with the script is required to use these features.